RAIRO-Theor. Inf. Appl. 43 (2009) 625–651
DOI: 10.1051/ita/2009010

Available online at:
www.rairo-ita.org

INDUCTIVE COMPUTATIONS ON GRAPHS DEFINED BY CLIQUE-WIDTH EXPRESSIONS [*]

Frédérique Carrère[1]

Abstract. Labelling problems for graphs consist in building distributed data structures, making it possible to check a given graph property or to compute a given function, the arguments of which are vertices. For an inductively computable function D, if G is a graph with n vertices and of clique-width at most k, where k is fixed, we can associate with each vertex x of G a piece of information (bit sequence) $\mathrm{lab}(x)$ of length $O(\log^2(n))$ such that we can compute D in constant time, using only the labels of its arguments. The preprocessing can be done in time $O(h.n)$ where h is the height of the syntactic tree of G. We perform an inductive computation, without using the tools of monadic second order logic. This enables us to give an explicit labelling scheme and to avoid constants of exponential size.

Mathematics Subject Classification. 68R10, 90C35.

1. Introduction

Many problems can be solved efficiently on the class of graphs that are structured in some way, by means of tree-decompositions of bounded width to take a well-known example. Clique-width is a graph parameter based on the expression of graphs by means of graph operations. Classes of graphs of bounded clique-width have been investigated in [3,9,23]. These graphs can be defined from trees by particular mappings (called monadic second order transductions).

The method of inductive computations on trees representing the structure of a graph extends to the computation of numerical functions on graphs like

Keywords and phrases. Terms, graphs, clique-width, labeling schemes, inductive computation.

[*] *This work has been supported by the ANR project GRAAL.*

[1] Laboratoire Bordelais de Recherche en Informatique, Université Bordeaux 1, CNRS, France.
carrere@labri.fr

distance or chromatic number. MS logic, where MS stands for Monadic second order, can be considered as a specification language for such problems and functions [7,8,13,16,26]. The proofs are based on the translation of MS formula into tree-automata. This fundamental tool is also useful for building labelling schemes. Labelling schemes are useful for routing (via distance labelling), see [19] and [10], but more generally for queries expressible in MS logic on graphs of bounded clique-width, see [11]. The general theorem based on MS logic suffers a major drawback which is the size of constants. MS logic and the corresponding automata can be avoided by direct constructions, as done in [10] for a problem which is in the scope of [11]. In this article, we follow the same idea of direct constructions avoiding MS logic.

We are interested in labelling problems for graphs. These problems consist in building a distributed data structure: some data (a label) is attached to each vertex of the graph, but there are no centralised data. The aim is to attach an information of size as small as possible to each vertex, such that one can compute from the labels a given function on the vertices, like the distance. This is useful to solve routing problems in networks, where the information must be distributed because of the global size of the network. Note that by using the best algorithms for general graphs, which compute the distance for all pairs of nodes in time $O(|V|+|E|)$ (V the set of vertices, E the set of edges), we could label each vertex with the distances to all others. Thus, with a preprocessing time $O(|V|+|E|)$, we would obtain labels of size $O(|V|.\log(|V|))$, from which we can get the distances in constant time. The aim of the labelling problems is to get labels of size as small as possible. Most labelling schemes use labels of logarithmic size.

The labelling problems originate from finding implicit representations of graphs [27]. An overview of labelling problems on graphs can be found in [20]. For the graphs which can be represented by syntactic trees, labelling problems also relate to a result of Harel and Tarjan [24] which provides, after some precomputation, a data structure such that any query for the nearest common ancestor between vertices can be processed in constant time. Gavoille *et al.* in [21] show that the minimal size of the labels to compute the distances for bounded tree-width graphs is $O(\log^2(n))$, where $n = |V|$. Courcelle and Vanicat in [11] give a fundamental result concerning labelling problems on graphs of clique-width at most k: if G is a graph of clique-width at most k, if $f(u_1,...,u_p)$ is a monadic second-order optimisation function on vertices (like the distance, with $p = 2$), one can associate with each vertex u of G a piece of information of size $O(\log^2(n))$ such that one can compute $f(u_1,...,u_p)$ in time $O(\log^2(n))$. The preprocessing time is $O(n\log^2(n))$.

This result shows the existence of relatively short labels from which the distance can be computed with a good time complexity. But it uses very powerful logical tools, that are hard to implement because of huge constants in the size of the automata ($O(2^{2^{\cdot^{\cdot^{\cdot^{2^k}}}}})$ in the case of clique-width at most k) which are inherent to the logical method [17]. The height of the tower of exponentials depends on the number of alternating quantifications in the formula. So it remains worth of interest to find easily implementable algorithms which construct labellings for graphs of bounded clique-width. Gavoille and Paul in [19] use the split decomposition of graphs to find explicit labelling schemes for the distance. Their algorithm applies to the class of distance-hereditary graphs (distance-hereditary graphs have clique-width at most 3).

In this paper, we use inductive computation for graphs of clique-width at most k. Let G be such a graph with n vertices and with syntactic tree t. We introduce a notion of system of inductive functions, or *SIF*. A *SIF* is a set of three functions $\{D_0, D_1, D_2\}$ of respective arities 0, 1 and 2 which satisfy some relations, that are inductive with respect to the tree under consideration. Considering RAM model with unit time cost (for reasonable arithmetic operations like $+$ and $*$), we prove that the functions of a *SIF* are computable in constant time from a labelling scheme of G.

We give an effective algorithm which for any *SIF* first computes a labelling scheme of G in time $O(h.n)$ where h is the height of the syntactic tree t. Thus if t is balanced, the preprocessing time of the algorithm is $O(n.\log(n))$. It associates with each vertex of the graph a label of size $c.k^2\log^2(n)$, where the constant c depends on the computed functions ($c = 4$ if D_2 is the distance, $c = 8$ if D_2 is the number of distinct shortest paths, $c = 4p^2$ if D_2 is the number of constrained shortest paths, when the deterministic automaton recognising the words labelling the paths has p states). This algorithm is easy to implement. It works for directed or undirected graphs, as well as for weighted graphs.

When the labelling scheme is built, we can then compute the functions of the *SIF* in constant time using only the local information stored in the labels, as follows. From the labels of two vertices x and y of G, we can find in constant time a node s in t, called a separator-node for x and y, which enables us to cut the tree t in three parts, with x and y in different parts. Then we can find in the labels of x and y some relevant values of the functions of the *SIF* restricted to each of these three parts of t. An important difference between the general algorithm using tree automata and our algorithm is the following: knowing the values of the functions restricted to the subtree rooted in s, the tree automaton must then compute the functions bottom-up along the branch from s to the root of t. This means at least $O(\log(n))$ operations, assuming that the syntactic tree t is balanced. Our algorithm uses the fact that some relevant values of the functions of the *SIF* are precomputed in the upper part of the tree, called the context of s, and stored in the labels of x and y. Then the inductive relations which the *SIF* satisfies permit us to achieve the computation in constant time, without any assumption on the height of t. The constants are of size $c'.k^2$, where the constant c' depends on the computed functions ($c' = 20$ for the distance or for the number of shortest paths, $c' = 20p^2$ for the constrained shortest paths, when the deterministic automaton has p states).

The paper is organised as follows. In the first section, we recall the notions of term and context. In the second section, we present the suitable inductive relations, which enable us to compute the functions on terms in constant time, using a piece of information which will be attached to each leaf of the term. In the third section, we present the operations for graphs of clique width k and the notion of context graph, analogous to the notion of context for terms. In the fourth and fifth section, we present four applications for particular functions on graphs: the computation of the distance, the computation of the number of distinct shortest paths, the computation of the length of the shortest paths avoiding a set of vertices, the computation of the length of constrained shortest paths in the case of graphs with labelled edges (constrained paths are paths the labels of which form words belonging to a given language).

2. Terms and contexts

We shall deal with terms (or trees), as the graphs we are interested in can be built from algebraic expressions and any algebraic expression can be represented as a tree (see Sect. 4.1 for the construction of graphs of clique-width k from a k-expression).

Let F be a set of binary operation symbols. Let C be the set $\{1, 2, \ldots, k\}$. $T(F, C)$ is the set of *well-formed terms* over the sets F and C. There is a classical bijection between the terms of $T(F, C)$ and the "well-labelled" trees with labels in F and C and we shall deal with terms or trees indifferently.

Let t be a tree, the height $h(t)$ will denote the maximum length of a branch of t and $|t|$ will denote the size (number of nodes) of t. Let a be an integer. We say that t is a-balanced iff $h(t) \leq a.\log(n)$, where n is the number of nodes of t ($n \geq 2$). The node s' is a left [resp. right] descendant of a node s if it is either the left [resp. right] son of s or a descendant of the left [resp. right] son of s.

Let t be a tree in $T(F, C)$. Let s be a node of t. The *context* of s in t is the tree obtained by replacing s in t with a specific node u without successor (all the descendants of s are deleted). If s is the root of t, the context of s is the trivial context reduced to one node u. The *contexts* are trees of $T(F, C \cup \{u\})$, containing a unique occurrence of the constant u, where $u \notin C$ and u is a label of a leaf. The set of all the contexts will be denoted $\mathrm{Ctxt}(F, C)$.

Let t be a tree in $T(F, C)$. Any internal node s of t separates t in three parts: its left subtree t_1, its right subtree t_2, and its *context* c. The triple (c, t_1, t_2) will be called the s-3*cut of* t. If s has label f, t is the result of the substitution of u by $f(t_1, t_2)$ in c. This is denoted $t = c[f(t_1, t_2)/u]$ or $t = c[f(t_1, t_2)]$ for short. Let c be a context in $\mathrm{Ctxt}(F, C)$. Let s be an ancestor of the unique node u of c. Let f be the label of s. If u is a left (resp. right) descendant of s, then s separates c in two contexts c_1, c_2 and a tree t such that $c = c_1[f(c_2, t)]$ (resp. $c = c_1[f(t, c_2)]$). The triple (c_1, c_2, t) is called the s-3*cut of* c. Note that if s is not an ancestor of u then replacing s with an occurrence of u in c would not give a context (two occurrences of u).

If x and y are two leaves of the tree t (resp. of the context c), s is a *separator* of x and y in t (resp. in c) iff none of the elements of the s-3cut contains both x and y.

The result of the substitution of the node u of a context by a tree is a tree. The result of the substitution of the node u of a context by a context is a new context. So one can define two binary operations on terms and contexts: for any $c \in \mathrm{Ctxt}(F,C)$ and $t \in T(F,C)$, $c \bullet t = c[t/u]$ belongs to $T(F,C)$, for any $c, c' \in \mathrm{Ctxt}(F,C)$, $c \circ c' = c[c'/u]$ belongs to $\mathrm{Ctxt}(F,C)$.

3. Inductive functions on terms

The inductive computation is used in [3] to solve problems on graphs such as maximum cardinality independent set, minimum cardinality dominating set, Hamiltonian path. We shall use this technique for systems of inductive functions on terms.

We are interested in a set $\mathcal{S}$ of numerical functions of the form $D(t, x_1, \ldots, x_p)$, where $t \in T(F,C)$ and $x_1, \ldots, x_p$ are leaves of t. The values of these functions can be vectors of different lengths. Our aim is to put some information on each leaf, depending on t and on that leaf, such that one can compute $D(t, x_1, \ldots, x_p)$ just from the information, hopefully of small size, put on $x_1, \ldots, x_p$. An important hypothesis is that the numerical functions that we deal with are also defined on the set of contexts $\mathrm{Ctxt}(F,C)$.

Let f be an integer function. An *f-labelling scheme* of a term t denotes a mapping *lab* from the set $L(t)$ of leaves of t to $\{0,1\}^*$ such that, for every leaf x, the length of the word $\mathrm{lab}(x)$ is at most $f(|t|)$. We say that a mapping $D(t, x_1, \ldots, x_p)$ is computable from a labelling scheme *lab* of t if there exists a computable function ϕ such that $D(t, x_1, \ldots, x_p) = \phi(\mathrm{lab}(x_1), \ldots, \mathrm{lab}(x_p))$.

We prove that a labelling scheme exists for the functions in $\mathcal{S}$ if they satisfy some relations, called inductive, on the trees t of the form $f(t_1, t_2)$ or of the form $c_1 \bullet t_1$, as well as on the contexts c of the form $c_1 \circ c_2$, where t_1, t_2 are terms and c_1, c_2 are contexts. We consider unit cost RAM model. Then the inductive relations permit us to compute in constant time a function $D(t, x_1, \ldots, x_p)$ or $D(c, x_1, \ldots, x_p)$ from the values of some $D'(t_i, x_j, \ldots, x_q)$ or $D'(c_i, x_j, \ldots, x_q)$ with D' in $\mathcal{S}$, $1 \leq i \leq 2$, and $1 \leq j \leq q \leq p$.

We do not detail the most general case here. We restrict to $p = 3$ to simplify the notations. The use of sets of size 3 is sufficient for the applications that we want to treat, for example the distance between two vertices of a graph of clique-width k, or the number of distinct shortest paths between two vertices. Note that one will clearly obtain similar results with a system of p functions having as arguments a term t and respectively $0, 1, \ldots, p-1$ leaves of t.

Definition 3.1. Let $\{D_0(t), D_1(t,x), D_2(t,x,y)\}$ be a set of three functions, with parameters $t \in T(F,C) \cup \mathrm{Ctxt}(F,C)$ and x, y leaves of t, and with values in some (possibly different) sets $\mathbb{N}^q$. We call $\{D_0, D_1, D_2\}$ a *system of inductive functions* on T(F,C), or *SIF*, iff one has:

1. for each $f \in F$, there exist functions $\phi_f, \phi'_f, \phi''_f, \psi_f, \psi'_f$ and ξ_f such that,

for any term $t = f(t_1, t_2)$, $t_1, t_2 \in T(F, C)$, the following holds:

$$D_2(\ f(t_1,t_2),x,y) \ = \begin{cases} \phi_f(D_1(t_1,x),\ D_1(t_2,y),\ D_0(t_1),\ D_0(t_2)), \\ \qquad\qquad\qquad \text{for any } x \in t_1 \text{ and } y \in t_2, \\ \phi'_f\ (D_2(t_1,x,y),\ \ D_1(t_1,x),\ D_1(t_1,y),\ D_0(t_1),\ D_0(t_2)), \\ \qquad\qquad\qquad \text{for any } x, y \in t_1,\ x \neq y \\ \phi''_f\ (D_2(t_2,x,y),\ \ D_1(t_2,x),\ D_1(t_2,y),\ D_0(t_1),\ D_0(t_2)), \\ \qquad\qquad\qquad \text{for any } x, y \in t_2,\ x \neq y \end{cases}$$

$$D_1(\ f(t_1,t_2),x) \ = \begin{cases} \psi_f(\ D_1(t_1,x),\ D_0(t_1),\ D_0(t_2)), & \text{for any } x \in t_1, \\ \psi'_f(\ D_1(t_2,x),\ D_0(t_1),\ D_0(t_2)), & \text{for any } x \in t_2, \end{cases}$$

$$D_0(\ f(t_1,t_2)) = \xi_f(\ D_0(t_1),\ D_0(t_2));$$

2. there exist functions $\phi_\bullet, \phi'_\bullet, \phi''_\bullet, \psi_\bullet, \psi'_\bullet$ and $\xi_\bullet$ such that, for any term $t = c_1 \bullet t_2$, with $c_1 \in \mathrm{Ctxt}(F, C)$, $t_2 \in T(F, C)$, the following holds:

$$D_2(\ c_1 \bullet t_2,x,y) = \begin{cases} \phi_\bullet\ (D_1(c_1,x),\ D_1(t_2,y),\ D_0(c_1),\ D_0(t_2)), \\ \qquad\qquad\qquad \text{for any } x \in c_1 \text{ and } y \in t_2, \\ \phi'_\bullet(D_2(c_1,x,y),\ \ D_1(c_1,x),\ D_1(c_1,y),\ D_0(c_1),\ D_0(t_2)), \\ \qquad\qquad\qquad \text{for any } x, y \in c_1,\ x \neq y \\ \phi''_\bullet(D_2(t_2,x,y),\ \ D_1(t_2,x),\ D_1(t_2,y),\ D_0(c_1),\ D_0(t_2)), \\ \qquad\qquad\qquad \text{for any } x, y \in t_2,\ x \neq y \end{cases}$$

$$D_1(\ c_1 \bullet t_2,x) = \begin{cases} \psi_\bullet(\ D_1(c_1,x),\ D_0(c_1),\ D_0(t_2)), & \text{for any } x \in c_1, \\ \psi'_\bullet(\ D_1(t_2,x),\ D_0(c_1),\ D_0(t_2)), & \text{for any } x \in t_2, \end{cases}$$

$$D_0(\ c_1 \bullet t_2) = \xi_\bullet(\ D_0(c_1),\ D_0(t_2));$$

3. there exist functions $\phi_\circ, \phi'_\circ, \phi''_\circ, \psi_\circ, \psi'_\circ$ and $\xi_\circ$ such that, for any term $t = c_1 \circ c_2$, with $c_1, c_2 \in \mathrm{Ctxt}(F, C)$, the following holds:

$$D_2(\ c_1 \circ c_2,x,y) = \begin{cases} \phi_\circ(D_1(c_1,x),\ D_1(c_2,y),\ D_0(c_1),\ D_0(c_2)), \\ \qquad\qquad\qquad \text{for any } x \in c_1 \text{ and } y \in c_2, \\ \phi'_\circ(D_2(c_1,x,y),\ \ D_1(c_1,x),\ D_1(c_1,y),\ D_0(c_1),\ D_0(c_2)), \\ \qquad\qquad\qquad \text{for any } x, y \in c_1,\ x \neq y \\ \phi''_\circ(D_2(c_2,x,y),\ \ D_1(c_2,x),\ D_1(c_2,y),\ D_0(c_1),\ D_0(c_2)), \\ \qquad\qquad\qquad \text{for any } x, y \in c_2,\ x \neq y \end{cases}$$

$$D_1(\ c_1 \circ c_2,x) = \begin{cases} \psi_\circ(\ D_1(c_1,x),\ D_0(c_1),\ D_0(c_2)), & \text{for any } x \in c_1, \\ \psi'_\circ(\ D_1(c_2,x),\ D_0(c_1),\ D_0(c_2)), & \text{for any } x \in c_2, \end{cases}$$

$$D_0(\ c_1 \circ c_2) = \xi_\circ(\ D_0(c_1),\ D_0(c_2)).$$

Note that the results that we obtain for *SIF*, as defined in Definition 3.1, remain true for functions D_0, D_1, D_2 taking values in some semiring.

We are interested in relations, like the ones detailed in Definition 3.1, involving numerical functions ($\phi_f, \phi'_f, \phi''_f, \psi_f, \psi'_f$ and ξ_f) which are computable in constant time using unit cost RAM model. Then the inductive relations which the *SIF* satisfies enable us to compute bottom-up the values $\{D_0(t), D_1(t,x), D_2(t,x,y)\}$

for all leaves of t. The complexity will be better than the one of general algorithms if the tree is a-balanced, with a small integer a.

Proposition 3.2. *Let $\{D_0, D_1, D_2\}$ be a system of inductive functions on $T(F, C)$, such that the numerical functions ϕ_f, ϕ'_f, ψ_f and ξ_f can be computed in constant time.*

1. *One can compute $D_0(t)$ in time $O(n)$, for any term t of size n.*
2. *One can compute $\{D_1(t, x),\ x \in t\}$ in time $O(h.n)$, for any term $t \in T(F, C)$ of size n and of height h.*
3. *One can compute $\{D_2(t, x, y),\ x, y \in t\}$ in time $O(h.n^2)$, for any term $t \in T(F, C)$ of size n and of height h.*

Proof. By hypothesis the computation time of the numerical functions ϕ_f, ϕ'_f, ψ_f and ξ_f is constant. Let t be a term of $T(F, C)$ of size n. The computation time of D_0 or D_1 is obvious. Then, for each couple of leaves of the tree, having computed D_0 and D_1, the value of the function D_2 will be recomputed at each common ancestor of the two leaves, that is at most h times. So the computation time of the function D_2 is $O(h.n^2)$. $\square$

3.1. Labelling scheme for inductive functions on terms

For any graph G of clique-width at most k, given as $\mathrm{val}(t)$ for some syntactic tree $t \in T(F, C)$, labellings of the leaves of the syntactic tree t yield labellings of vertices of the graph G. Courcelle and Vanicat in [11] proved the following fundamental theorem.

Theorem 3.3 (Courcelle and Vanicat)**.** *For every graph G of clique-width at most k, given as $\mathrm{val}(t)$ for some $t \in T(F, C)$, for any MS-optimisation function f on graphs, one can compute in time $O(n \log^2(n))$ a $\log^2(n)$-labelling scheme of G from which one can compute any value of f in time $O(\log^2(n))$.*

The distance is precisely an MS-optimisation function on graphs (MS stands for monadic second order logic). In the case of the distance, Courcelle and Vanicat show that a time $O(\log(n))$ is sufficient to compute it from the set of labels. Their proof uses powerful logical tools. But these tools are not easy to implement and they necessarily induce great constants of exponential size.

Rather than using the framework of monadic second order logic, which leads to constants of size $2^{2^{\cdot^{\cdot^{\cdot 2^k}}}}$, we give an algorithm that performs a direct computation of the functions, from the inductive relations which they fulfil. This is a practical algorithm, easy to implement, which also holds for weighted or directed graphs: the computing time of the labels will be $O(h.n)$, and the inductive functions can be computed in constant time from the labelling scheme. In the case of graphs of clique-width at most k, the constant will be of size $c.k^2$ for a small c, as we will see in Sections 5, 6,7 ($c \leq 10$) and 8 ($c \leq 5p^2$).

We assume that the graph G is given with its decomposition tree. For $k \leq 3$, such a decomposition tree can be found in time $O(n^2m)$, if G admits one (see [5]). Graphs with clique-width at most 2 are exactly Cographs. Some well-known families of graphs have clique-width at most 3: distance-hereditary graphs, $P4$-sparse graphs. For $k > 3$, Hlinený and Oum recently give an $O(n^3)$ algorithm (see [25]) which for any graph G either outputs a decomposition tree of width less than 2^{3k+2} or confirms that the clique-width of G is larger than k. We can use this result to compute inductive functions on such graphs, although the decomposition is not optimal. It has been proved that it is NP-hard to find graph clique-width (see [16]).

Main Theorem. *For any term t of $T(F, C)$ of size n, for any system of inductive functions $\{D_0, D_1, D_2\}$ on $T(F, C)$, one can compute in time $O(h.n)$ an $O(\log^2(n))$ labelling scheme of t, from which one can compute $D_2(t, x, y)$ for any vertices x and y in constant time.*

Corollary 3.4. *For any a-balanced term t of $T(F, C)$, for any system of inductive functions $\{D_0, D_1, D_2\}$ on $T(F, C)$, one can compute in time $O(n \log(n))$ an $O(\log^2(n))$ labelling scheme of t, from which one can compute $D_2(t, x, y)$ for any vertices x and y in constant time.*

Proof of Main Theorem. Let t be a term of $T(F, C)$ of size $n = 2p + 1$. Let $\{D_0, D_1, D_2\}$ be a *SIF* on $T(F, C)$.

Inductive 3-partitioning of the tree

The notion of *cut-nodes* is introduced in [11] to transform a syntactic tree into another balanced one. Here we use the *cut-nodes* to distribute information in the labels. It is defined for trees and contexts. Let t be either a tree or a context. If t is reduced to one node, this unique node is the cut-node of t. Otherwise a cut-node of t is a node s such that at least two elements of the s-3cut of t are of size less than $|t|/2 + 1$. Then each element of the s-3cut contains itself a new cut-node and the process can go on until one obtains trees or contexts reduced to one node.

As we inductively cut t and the resulting trees or contexts in 3 parts, we can index the cut-nodes with words on the alphabet $\{0, 1, 2\}$ as follows. Let s_ϵ be the cut-node of t and $\{c_0, t_1, t_2\}$ be the s_ϵ-3cut of t, then s_0 denotes the cut-node of c_0, s_1 denotes the cut-node of t_1, s_2 denotes the cut-node of t_2.

Then each t_i, $i = 1, 2$, provided that it is not reduced to one node, is partitioned into three parts, giving three new cut-nodes: s_{i0}, s_{i1} and s_{i2}. The context c_0, provided that it is not reduced to one node, can be partitioned too. After this first refinement, one obtains a global partition of t in at most nine blocks. Each block contains a cut-node s_{ij}, $0 \leq i, j \leq 2$ and one refines this partition again. The ith refinement gives rise to a partition of t in at most 3^{i+1} blocks, each of them containing a cut-node indexed with a word of length at most $i + 1$. The process ends when one obtains blocks reduced to one node.

An example is shown in Figures 1 and 2. □

Fact 3.1. *All the blocks obtained by partitioning t are terms or contexts.*

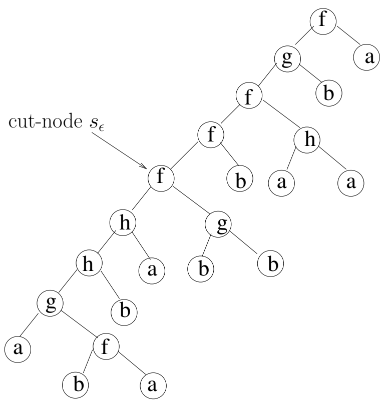


FIGURE 1. A tree of $T(F, C)$.

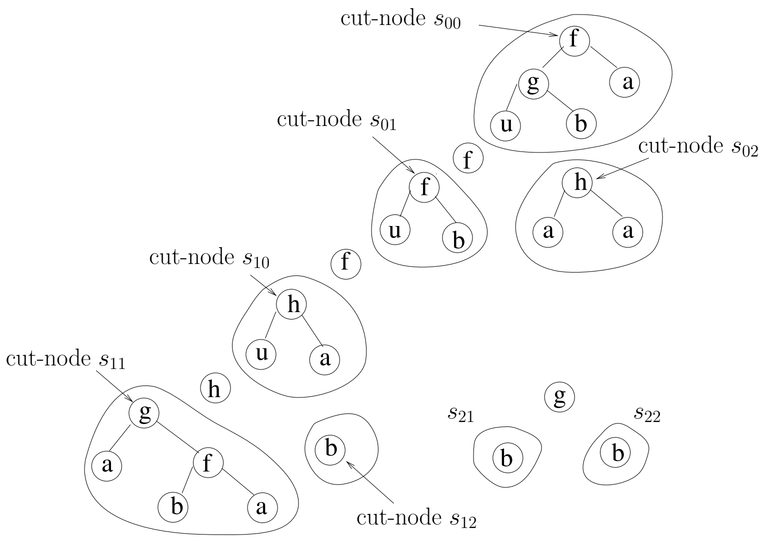


FIGURE 2. Partitioning of the tree of Figure 1.

This is a consequence of the choice of the cut-node in the contexts: the cut-node s of a context c is always an ancestor of u, thus it determines an s-3cut of c containing two contexts and a tree. This is the reason why each block is partitioned in three parts (for further details, see [11]). A dichotomous partitioning would not work as shown in Figure 3.

Choice of the relevant cut-nodes for a leaf x

Let x be a leaf of t. One can inductively construct a sequence of cut-nodes $\text{sep}(x) = \{s_\epsilon, s_i, \ldots, s_m = x\}$, $i \in \{0, 1, 2\}$, $m \in \{0, 1, 2\}^*$, called the *separator*

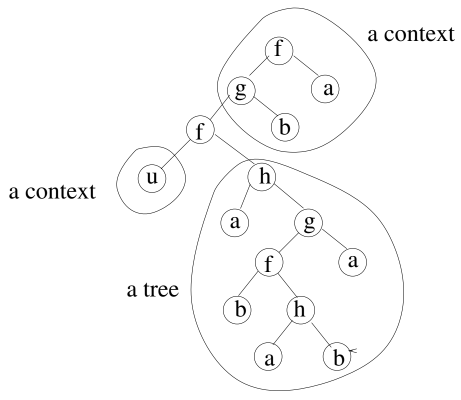


FIGURE 3. If the cut-node s with label h is not an ancestor of u.

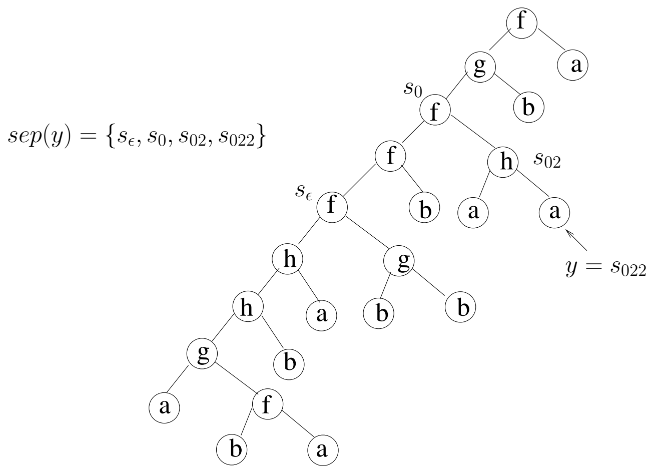


FIGURE 4. The separator sequence of the node y.

sequence of x, and a sequence of terms or contexts $\{t_i, \dots, t_m\}$ such that for all prefixes w and wi of m, $i \in \{0, 1, 2\}$, t_{wi} is the element of the s_w-3cut of t_w which contains x. Then s_{wi} is the cut-node of t_{wi}. Note that the cut-nodes in sep(x) are not necessarily ancestors of x: see Figure 4.

Fact 3.2. *The length of* m *is at most* $3\log(n)$*.*
The proof is given in [11].

Construction of the label of a leaf x

Let $\{c_0, t_1, t_2\}$ be the s_ϵ-3cut of t. Let us set $C_\epsilon = c_0$, $A_\epsilon = t_1$ and $B_\epsilon = t_2$.

If $w \neq \epsilon$, each cut-node s_w of sep(x) cuts a block of some refinement of the initial partition of t. But it also cuts the whole tree t: let $\{C_w, A_w, B_w\}$ be the s_w-3cut of t. We write in the label of x each s_w of sep(x), followed by the values of the functions D_0, D_1, on C_w, A_w and B_w. As $t = C_w[f(A_w, B_w)]$, f labelling s, we will then be able to compute D_2 on t in two induction steps: one for $t' = f(A_w, B_w)$ followed by one for $C_w \bullet t'$. Let $T_w(x)$ denotes the element of $\{A_w, B_w, C_w\}$ which contains x, then the label of x is:

$$\begin{aligned}\mathrm{Lab}(x) = \{ \quad & s_\epsilon, D_1(T_\epsilon(x), x), D_0(A_\epsilon), D_0(B_\epsilon), D_0(C_\epsilon), s_i, D_1(T_i(x), x), D_0(A_i),\\ & D_0(B_i), D_0(C_i), \ldots, s_m, D_1(T_m(x), x),\ D_0(A_m), D_0(B_m), D_0(C_m)\},\end{aligned}$$

with $i \in \{0,1,2\}$ and $m \in \{0,1,2\}^*$.

Fact 3.3. *The set of values* $\{D_1(T_w(x)); x$ *leaf of* t, w *such that* $s_w \in \mathrm{sep}(x)\}$ *can be inductively computed in time* $O(h(t).n)$.

It follows from Proposition 3.2.

Fact 3.4. *For any two leaves, x and y of t, there exists a unique cut-node in* $\mathrm{Lab}(x) \cap \mathrm{Lab}(y)$, *which is a* separator node *for x and y in t.*

Proof. Let m be the word such that the last element of the separator sequence of x is s_m. Let m' be the word such that the last element of the separator sequence of y is $s_{m'}$. Let w be the longest common prefix of m and m'. Then x and y belong to t_w. Let $\{t_{w0}, t_{w1}, t_{w2}\}$ be the s_w-3cut of t_w. Then x and y do not belong to the same block, t_{w0}, t_{w1} or t_{w2}. Otherwise, if x and y both belong to t_{wi}, then s_{wi} will be the next element of both $\mathrm{sep}(x)$ and $\mathrm{sep}(y)$ and w would not be the longest common prefix of m and m'; so s_w is a separator-node for x and y in t_w.

Then it is also a separator-node of x and y in t: $t = C_w[f(A_w, B_w)]$, with x and y not in the same part C_w, A_w or B_w. □

Evaluating inductive functions in constant time

After the preprocessing of the tree to compute the labels, the value of the function D_2 for a couple (x, y) of leaves can be computed in constant time as follows.

As shown in [18], we can code (in polynomial time) the labels of x and y on $O(\log(n))$ bits, if we are interested in finding the longest common prefix of $\mathrm{sep}(x)$ and $\mathrm{sep}(y)$ in constant time. The last node s_w of this prefix is the unique separator-node of x and y in t, which belongs to $\mathrm{Lab}(x) \cap \mathrm{Lab}(y)$ (see Fact 3.4). Then the next lemma gives the formula to compute $D_2(x, y)$ in constant time, using the Definition 3.1 and the numerical values associated with s_w in $\mathrm{Lab}(x)$ and $\mathrm{Lab}(y)$.

Lemma 3.5. *Let t be a term of $T(F, C)$ and $\{D_0, D_1, D_2\}$ be a* SIF. *For each f in F, there exists a function Φ_f such that if x and y are leaves of t, if the unique separator node s_w of x and y belonging to* $\mathrm{Seq}(x) \cap \mathrm{Seq}(y)$ *is labelled by f, if $\{C_w, A_w, B_w)\}$ is the s_w-3cut of t, and we let $T_w(x)$ (resp. $T_w(y)$) denotes the element of $\{A_w, B_w, C_w\}$ which contains x (resp. y), then one has:*
$D_2(t, x, y) = \Phi_f(D_1(T_w(x), x), D_1(T_w(y), y), D_0(C_w), D_0(A_w), D_0(B_w))$.

Proof. Since $\{C_w, A_w, B_w)\}$ is the s_w-3cut of t, one has:
$t = C_w[f(A_w, B_w)] = C_w \bullet f(A_w, B_w)$, with f labelling s_w.

We will assume that the leaf x of t appears on the left of the leaf y. Then there are three cases:

First case. Assume that $x \in A_w$ and $y \in B_w$.
So $T_w(x) = A_w$ and $T_w(y) = B_w$. By the definition of a *SIF*, one has:
$D_2(\, f(A_w, B_w), x, y) = \phi_f(\, D_1(A_w, x),\ D_1(B_w, y),\ D_0(A_w),\ D_0(B_w)\,)$,
$D_1(\, f(A_w, B_w), x) = \psi_f(\, D_1(A_w, x),\ D_0(A_w),\ D_0(B_w))$,
$D_1(\, f(A_w, B_w), y) = \psi'_f(\, D_1(B_w, y),\ D_0(A_w),\ D_0(B_w))$,
$D_0(\, f(A_w, B_w)) = \xi_f(\, D_0(A_w),\ D_0(B_w))$.
Since $t = C_w \bullet f(A_w, B_w)$, with C_w a context and $f(A_w, B_w)$ a subtree, one has:
$D_2(\, t, x, y) = \phi_\bullet(\, D_2(f(A_w, B_w), x, y),\ D_1(f(A_w, B_w), x),\ D_1(f(A_w, B_w), y),$
$D_0(C_w),\ D_0(f(A_w, B_w))\,)$.
Let $\pi_i : \mathbb{N}^5 \longrightarrow \mathbb{N}$, $1 \leq i \leq 5$, be the ith projection. In this case Φ_f is:
$\phi_\bullet(\, \phi_f(\pi_1, \pi_2, \pi_4, \pi_5),\ \psi_f(\pi_1, \pi_4, \pi_5),\ \psi'_f(\pi_2, \pi_4, \pi_5),\ \pi_3,\ \xi_f(\pi_4, \pi_5)\,)$
So $D_2(\, t, x, y)$ can be computed in constant time from $L(x)$ and $L(y)$.

Second case. Assume that $x \in C_w$ and $y \in A_w$.
So $T_w(x) = C_w$ and $T_w(y) = A_w$. By the definition of a *SIF*, one has:
$D_1(\, f(A_w, B_w), y) = \psi_f(\, D_1(A_w, y),\ D_0(A_w),\ D_0(B_w))$,
$D_0(\, f(A_w, B_w)) = \xi_f(\, D_0(A_w),\ D_0(B_w))$.
Since $t = C_w \bullet f(A_w, B_w)$, with C_w a context and $f(A_w, B_w)$ a subtree, one has:
$D_2(\, t, x, y) = \phi_\bullet(\, D_1(C_w, x),\ D_1(f(A_w, B_w), y),\ D_0(C_w),\ D_0(f(A_w, B_w))\,)$.
In this case Φ_f is:
$\phi_\bullet(\, \pi_1,\ \psi_f(\pi_2, \pi_4, \pi_5),\ \pi_3,\ \xi_f(\pi_4, \pi_5)\,)$.

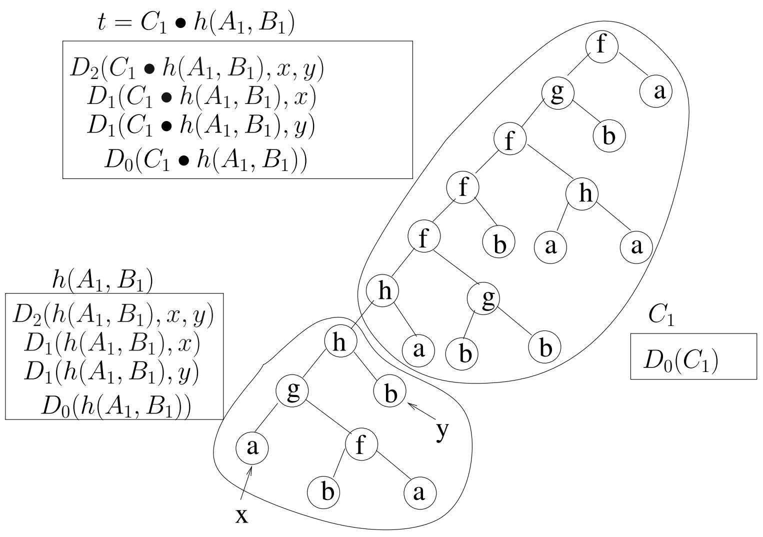
Third case. Assume that $x \in C_w$ and $y \in B_w$.
So $T_w(x) = C_w$ and $T_w(y) = B_w$. By the definition of a *SIF*, one has:
$D_1(\, f(A_w, B_w), y) = \psi'_f(\, D_1(B_w, y),\ D_0(A_w),\ D_0(B_w))$,
$D_0(\, f(A_w, B_w)) = \xi_f(\, D_0(A_w),\ D_0(B_w))$.
Since $t = C_w \bullet f(A_w, B_w)$, with C_w a context and $f(A_w, B_w)$ a subtree, one has:
$D_2(\, t, x, y) = \phi_\bullet(\, D_1(C_w, x),\ D_1(f(A_w, B_w), y),\ D_0(C_w),\ D_0(f(A_w, B_w))\,)$.
In this case Φ_f is:
$\phi_\bullet(\, \pi_1,\ \psi'_f(\pi_2, \pi_4, \pi_5),\ \pi_3,\ \xi_f(\pi_4, \pi_5)\,)$
So $D_2(\, t, x, y)$ can be computed in constant time from $L(x)$ and $L(y)$. □

4. Application to graphs

Let k be an integer. A k-graph is a graph whose vertices have labels in $\{1, 2, \ldots, k\}$.

Let $G = \langle V_G, E_G, \gamma_G \rangle$ be a simple, undirected (resp. directed) and connected k-graph, where V_G is the set of *vertices*, E_G is the set of *edges* (resp. directed *edges*), and γ_G is a labelling function which maps V_G into $\{1, 2, \ldots, k\}$. For each $i \in \{1, 2, ..., k\}$, we denote by $S_i(G)$, or S_i if there is no ambiguity, the set of vertices with label i.

For each vertex x of G, the distance $d_G(x, S_i)$ is the minimum length of a path from x to a vertex of S_i in G. If S_i is empty or if no such path exists, $d_G(x, S_i)$



$t = C_1 \bullet h(A_1, B_1)$
$D_2(C_1 \bullet h(A_1, B_1), x, y)$
$D_1(C_1 \bullet h(A_1, B_1), x)$
$D_1(C_1 \bullet h(A_1, B_1), y)$
$D_0(C_1 \bullet h(A_1, B_1))$
$h(A_1, B_1)$
$D_2(h(A_1, B_1), x, y)$
$D_1(h(A_1, B_1), x)$
$D_1(h(A_1, B_1), y)$
$D_0(h(A_1, B_1))$
C_1
$D_0(C_1)$
x
y

FIGURE 5. Computation of D_2.

is infinite. We denote by $d_G(x)$ the vector of distances $(d_G(x, S_1)$,$d_G(x, S_2)$,..., $d_G(x, S_k))$. For all $i, j \in \{1, 2, ..., k\}$, the distance $d_G(S_i, S_j)$ is the minimum length of a path from a vertex of S_i to a vertex of S_j in G.

In the directed case, we shall use the notation $\overrightarrow{d}_G$ to pointed out that the paths are directed. Note that there are two vectors for any vertex x of G: $\overrightarrow{d}_G(x)$ will denote the vector containing the lengths of the directed shortest paths from x to the sets S_i, and $\overrightarrow{d'}_G(x)$ the vector containing the lengths of the directed shortest paths from the sets S_i to x.

We apply our main theorem to the particular case of distances and shortest paths in graphs of clique-width at most k. The graphs of clique-width at most k are particular k-graphs given by algebraic expressions.

4.1. Graphs of clique-width at most k

We recall here the usual definition of the set of graphs of clique-width at most k. Let CW_k be the set of graphs of clique-width at most k.

- A single vertex with label in $\{1, \dots, k\}$ belongs to CW_k,
- if G_1 and G_2 belong to CW_k, $G = G_1 \oplus G_2$, the disjoint union of G_1 and G_2, belongs to CW_k;
- if G_1 belongs to CW_k, $G = add_{\alpha,\beta}(G_1)$, the graph obtained by adding to G_1 all edges between vertices labelled α and vertices labelled β (with $\alpha \neq \beta$), belongs to CW_k;
- if G_1 belongs to CW_k, $G = ren_{\alpha,\beta}(G_1)$, the graph obtained by changing the label α into β at each occurrence of α in G_1, belongs to CW_k.

In the directed case, the operation $add_{\alpha,\beta}$ will add directed edges from S_α to S_β.

Thus a graph of clique-width at most k is the evaluation of a term built with the above operations. But to apply our main theorem, we need to deal with binary operations. These binary operations are defined in the next paragraph.

4.2. Binary operations on graphs

We will use the following binary operations. For any $R \subset \{1, \ldots, k\}^2$, for any relabelling $f : L \longrightarrow L$, let us denote by $\otimes_{R,f}(G_1, G_2)$ the binary operation consisting of applying first to the graph $G_1 \oplus G_2$ all $add_{\alpha,\beta}$-operations, for (α, β) in R, then applying all $ren_{i,f(i)}$-operations, for i in $\{1, \ldots, k\}$.

It has been proved in [9] that, at the cost of multiplying the number of labels by 2 (dealing thus with graphs of clique-width at most $2k$), one can suppose that the $\oplus$ operation only applies on two graphs having disjoint sets of labels. Then, as in the operations used by Wanke in [22], each operation $add_{\alpha,\beta}$, for (α, β) in R, will add edges between two disjoint graphs.

Let F be the set $\{\otimes_{R,f};\ R \subset \{1, \ldots, k\}^2, f : L \longrightarrow L\}$ of binary operations. There exists a mapping val which maps the set of terms $T(F, C)$ to the set of graphs CW_k.

4.3. Operation of substitution

We first need to define a notion of "context graph". A "context graph" is the value of a context term in $\mathrm{Ctxt}(F, C)$ by a mapping val'.

The *context graphs* are special graphs of clique-width k, containing a unique occurrence of specific vertices $u_1, \ldots, u_k$. The elementary context graph I_k consists of k vertices, $u_1, \ldots, u_k$, such that $S_i(I_k) = \{u_i\}$, for any i, $1 \leq i \leq k$.

The set CT_k of all context graphs can be inductively constructed as follows:

- I_k belongs to CT_k;
- if G belongs to CT_k and H belongs to CW_k, $f(G, H)$ and $f(H, G)$ belong to CT_k with $f \in F$.

We now define an operation of substitution of a graph of CW_k in a context graph. This operation is a generalisation of the usual substitution. A graph H of clique-width k has a partition of the vertices in k subsets $S_1(H), S_2(H), \ldots, S_k(H)$. These subsets will be respectively substituted for the special vertices $u_1, \ldots, u_k$ of the context graph.

For each i, $1 \leq i \leq k$, let $N_G(u_i)$ be the set of neighbors of u_i in G. The idea is to replace each u_i by $S_i(H)$, joining all vertices of $N_G(u_i)$ to all vertices of $S_i(H)$, but without deleting edges of H. We call this operation a *k-substitution*. This operation takes as first argument a context graph, and as second argument a graph of clique-width k. The resulting graph is a graph of clique-width k. The *k-substitution* is formally defined below.

Definition 4.1. Let G be a context graph. Let $u_1, u_2, \ldots, u_k$ be the special vertices of G. For each i, $1 \leq i \leq k$, let $\tilde{N}_G(u_i)$ be the set of neighbors of u_i

which do not belong to $\{u_1, u_2, \ldots, u_k\}$. Let H be a graph of clique-width k. The vertices of H are partitioned into $S_1(H), S_2(H), \ldots, S_k(H)$.

We denote $G[H/u_1, u_2, \ldots, u_k]$ the graph $\langle V_{G'}, E_{G'}, \gamma_{G'}\rangle$, where

- $V_{G'} = V_G - \{u_1, u_2, \ldots, u_k\} \cup S_1(H) \cup S_2(H) \ldots \cup S_k(H)$,
- $\begin{array}{ll} E_{G'} = & E_G - [\,\cup_{i=1}^{i=k}\{u_i\} \times N_G(u_i)\,] \,\cup E_H \,\cup\, [\,\cup_{i=1}^{i=k} S_i(H) \times \tilde{N}_G(u_i)\,] \\ & \cup\, [\,\cup_{(u_i,u_j)\in E_G} S_i(H) \times S_j(H)\,], \end{array}$
- $\gamma_{G'}(x) = \gamma_G(x)$, for any $x \in V_G - \{u_1, u_2, \ldots, u_k\}$, $\gamma_{G'}(x) = \gamma_G(u_i)$, for any $x \in S_i(H)$, $1 \leq i \leq k$.

Note that this operation of substitution of graphs could be simulated with the clique-width operations using labels in $\{1, \ldots, k'\}$ with $k' = O(k * 2^k)$. We introduce this operation of k-substitution to avoid constants of exponential size, and we will use it to prove that there exists some *SIF* to compute the functions which we are interested in, like the distance or the number of distinct shortest paths.

Let us define the mapping val$'$ which maps the elements of $\mathrm{Ctxt}(F, C) \cup T(F, C)$ to elements of $CT_k \cup CW_k$.

For any $t \in T(F, C)$, we set $\mathrm{val}'(t) = \mathrm{val}(t)$. For the elementary context, we set $\mathrm{val}'(u) = I_k$ (note that the context graph $\mathrm{val}'(u)$ is not connected, but if G is a connected graph of clique-width k, substituting G for I_k using the k-substitution gives a connected graph). For any operation $\otimes_{R,f}$ of F, for any well-formed term $t' = \otimes_{R,f}(t, c)$ (resp. $t' = \otimes_{R,f}(c, t)$) of $\mathrm{Ctxt}(F, C)$, we set $\mathrm{val}'(t') = \otimes_{R,f}(\mathrm{val}'(t), \mathrm{val}'(c))$ (resp. $\mathrm{val}'(t') = \otimes_{R,f}(\mathrm{val}'(c), \mathrm{val}'(t))$).

Then the value of a term resulting from the substitution of a term t for a context c is the graph obtained by the k-substitution of $\mathrm{val}(t)$ in the context graph $\mathrm{val}'(c)$.

Lemma 4.2. *For any c, c' in* $\mathrm{Ctxt}(F, C)$ *and t in* $T(F, C)$*, one has:*
$\mathrm{val}(c \bullet t) = \mathrm{val}'(c)[\mathrm{val}(t)/u_1, u_2, \ldots, u_k]$,
$\mathrm{val}(c \circ c') = \mathrm{val}'(c)[\mathrm{val}'(c')/u_1, u_2, \ldots, u_k]$.

This can easily be proved by induction on c.

For convenience we shall decompose the k-substitution in k elementary steps, that we shall call "partial k-substitutions": let H be a graph whose vertices are partitioned in k sets. For any $i < k$, $G[H/u_1, u_2, \ldots, u_i]$ denotes the graph obtained by substituting $S_j(H)$ to u_j, only for the ith first indexes $1 \leq j \leq i$. A *partial k-substitution* is formally defined as follows:

Definition 4.3. Let G be a context graph. Let $u_1, u_2, \ldots, u_k$ be the special vertices of G. For each i, $1 \leq i \leq k$, let $\tilde{N}_G(u_i)$ be the set of neighbors of u_i which do not belong to $\{u_1, u_2, \ldots, u_k\}$. Let H be a graph of clique-width k. For any $i < k$, we denote $G[H/u_1, u_2, \ldots, u_i]$ the graph $\langle V_{G'}, E_{G'}, \gamma_{G'}\rangle$, where

- $V_{G'} = V_G - \{u_1, u_2, \ldots, u_i\} \cup S_1(H) \cup S_2(H) \ldots \cup S_k(H)$,
- $\begin{array}{ll} E_{G'} = & E_G - [\,\cup_{j=1}^{j=i}\{u_j\} \times N_G(u_j)\,] \,\cup E_H \,\cup\, [\,\cup_{j=1}^{j=i} S_j(H) \times \tilde{N}_G(u_j)\,]; \\ & \cup\, [\,\cup_{(u_j,u_h)\in E_G, 1\leq j,h\leq i} S_j(H) \times S_h(H)\,], \\ & \cup\, [\,\cup_{(u_j,u_h)\in E_G, 1\leq j\leq i<h} S_j(H) \times \{u_h\}\,]; \end{array}$

– $\gamma_{G'}(x) = \gamma_G(x)$, for any $x \in V_G - \{u_1, u_2, \ldots, u_i\}$,
$\gamma_{G'}(x) = \gamma_G(u_j)$, for any $x \in S_i(H)$, $1 \leq j \leq i$.

5. First application: length of the shortest paths

We will assume that each arithmetic operation can be evaluated in one unit of time. We show that one can find a *SIF* $\{D_0, D_1, D_2\}$ such that $D_2(t, x, y)$ is the length of the shortest paths between x and y in the graph $\mathrm{val}(t)$. Then we obtain by our main theorem a labelling scheme for a graph G of bounded clique width, such that one can compute D_2 in constant time from labels of its arguments.

Let t be a term in $T(F, C) \cup \mathrm{Ctxt}(F, C)$ and let G be the graph or context graph given as $\mathrm{val}(t)$ or $\mathrm{val}'(t)$. We define the three following numerical functions:

– $D_2(t, x, y) = d_G(x, y)$;

– $D_1(t, x) = \begin{cases} d_G(x), & \text{if } t \in T(F, C), \\ (d_G(x), d_G(x, u_1), \ldots, d_G(x, u_k)), & \text{if } t \in \mathrm{Ctxt}(F, C); \end{cases}$

– $D_0(t) = \begin{cases} (d_G(S_1,\ S_1), d_G(S_1, S_2), \ldots, d_G(S_k, S_k)), & \text{if } t \in T(F, C), \\ (d_G(S_1,\ S_1), d_G(S_1, S_2), \ldots, d_G(S_k, S_k),\ d_G\ (u_1, S_1), d_G(u_1, S_2), \\ \qquad \ldots, d_G(u_k, S_k)), & \text{if } t \in \mathrm{Ctxt}(F, C). \end{cases}$

We want to show that the functions D_0,D_1,D_2 form a *SIF* on $T(F, C)$.

We first show that $d_G(x, y)$, $d_G(x, S_i)$ and $d_G(S_i, S_j)$ satisfy inductive relations for any basic clique-width-k operation, $\oplus$, $add_{\alpha,\beta}$ and $ren_{\alpha,\beta}$. Each binary operation $\otimes_{R,L}$ of F, $R, L \subset \{1, \ldots, k\}^2$, is a composition of (at most $2k^2 + 1$) basic clique-width-k operations. So any function which satisfies inductive relations for $\oplus$, $add_{\alpha,\beta}$ and $ren_{\alpha,\beta}$, will also satisfy inductive relations for any operation of F.

The functions of a *SIF* must also satisfy inductive relations on contexts. This will be the case since the operations $\bullet$ and $\circ$ on trees and contexts correspond to k-substitutions for graphs and context-graphs (see Lem. 4.2). We will that the lengths of the shortest paths (between vertices or sets S_i) satisfy inductive relations for any partial-k-substitution.

1. Case $G = G_1 \oplus G_2$
Neither the length of the shortest paths between vertices of G_1 (resp. G_2), nor the length of the shortest paths between the sets S_α in the same subgraph, G_1 or G_2, will change. If x belongs to G_1 then $d_G(x, S_\gamma)$ will be infinite for any $S_\gamma \subset G_2$. This holds for the directed case too.

2. Case $G = add_{\alpha,\beta}(H)$
We assume that $\alpha \neq \beta$. We can compute the length of the shortest paths between two vertices as follows:

Lemma 5.1. *If $x, y \in V_H$, $x \neq y$, then* $d_G(x, y) = \min(d_H(x, y),\ d_H(x, S_\alpha) + 1 + d_H(y, S_\beta),\ d_H(x, S_\beta) + 1 + d_H(y, S_\alpha),\ d_H(x, S_\alpha) + 2 + d_H(y, S_\alpha),\ d_H(x, S_\beta) + 2 + d_H(y, S_\beta))$.

Proof. We consider the shortest paths between x and y in G. There are three cases.
First case. The shortest paths between x and y in G can be paths wholly contained in H (containing eventually edges from S_α to S_α). Then $d_G(x,y) = d_H(x,y)$.

Second case. The shortest paths between x and y in G can contain one new edge. Then they can be decomposed in a path between x and S_α in H, a new edge between S_α and S_β and a path between S_β an y in H (exchanging α and β gives a symmetric case). So $d_G(x,y)$ is the sum of the length of the shortest paths from x to S_α in H and the length of the shortest paths from y to S_β in H, plus one. It remains true if x belongs itself to S_α, which implies that $d_H(x,S_\alpha) = 0$, and (or) if y belongs to S_β, which implies that $d_H(y,S_\beta) = 0$.

Third Case. If $\mathrm{Card}(S_\alpha) \geq 2$, since the graph between S_α to S_β is complete, there can be no more than two new edges in a shortest path between x and y in G (if not, the path could be shortened). Any shortest path between x and y in G, which contains two new edges, can be decomposed in a path between x and S_α in H, a new edge from S_α to S_β followed by a new edge from S_β to S_α and a path between S_β and y (exchanging α and β give a symmetric case). In this case, note that the path from x to S_α and from y to S_α are necessarily disjoint, otherwise the global path would not be a shortest path. Then $d_G(x,y)$ is the sum of the length of the shortest paths from x to S_α and from y to S_α plus two.

If S_α is reduced to a singleton $\{z\}$ (or symmetrically if $\mathrm{Card}(S_\beta) = 1$), there are no shortest paths crossing S_α twice. But in this case one has $d_H(x,S_\alpha) = d_H(x,z)$ and $d_H(y,S_\alpha) = d_H(y,z)$, so $d_H(x,y) \leq d_H(x,z)+d_H(y,z) < d_H(x,S_\alpha)+ 2 + d_H(y,S_\alpha)$. The equality remains true, the minimum is not $d_H(x,S_\alpha) + 2 + d_H(y,S_\alpha)$.

If x itself belongs to S_α (or symmetrically that y belongs to S_β) then $d_H(x,S_\alpha) = 0$. As one has $d_H(x,S_\beta) + 1 + d_H(y,S_\alpha) \geq 2 + d_H(y,S_\alpha)$ ($d_H(x,S_\beta) \geq 1$) and $d_H(x,S_\beta) + 2 + d_H(y,S_\beta)) \geq 1 + d_H(y,S_\beta)$, the equality of Lemma 5.1 becomes $d_G(x,y) = \min(d_H(x,y),\, 1 + d_H(y,S_\beta),\, 2 + d_H(y,S_\alpha))$ (1) which is true. It remains true if x is the only element of S_α, because in this case $d_H(y,S_\alpha)$ is $d_H(x,y)$. If x belongs to S_α and y belongs to S_β, then the equality (1) gives $d_G(x,y) = \min(d_H(x,y), 1, 2 + d_H(y,S_\alpha)) = 1$.

So the length of the shortest paths between two vertices of G can be computed in constant time.

In the directed case, the result for the length of the shortest paths is obvious. We need to compute also the length of a minimum cycle containing u: $mcl_G(u)$ (it will be useful for the operation of substitution). □

Lemma 5.2. *Let $x, y \in V_H$, $x \neq y$.*
$\overrightarrow{d}_G(x,y) = \min(\overrightarrow{d}_H(x,y)\ ,\ \overrightarrow{d}_H(x,S_\alpha) + 1 + \overrightarrow{d}_H(y,S_\beta)\ )$.
$mcl_G(u) = \min(mcl_H(u)\ ,\ \overrightarrow{d}_H(x,S_\alpha) + 1 + \overrightarrow{d}_H(x,S_\beta)\ )$.

So the length of the shortest paths between two vertices of G can be computed in constant time in the directed or undirected case.

The length of the shortest paths between a vertex x of G and any set S_ρ, $1 \leq \rho \leq k$, can be obviously computed in constant time, by similar inductive relations, in the directed and undirected case.

3. Case $G = ren_{\alpha,\beta}(H)$

This operation does not modify the length of the shortest paths between vertices, but it modifies the sets S_α and S_β and the vectors $d_G(x)$. The new vector $d_G(x)$ can be easily computed in constant time for each vertex x of G as follows:

Lemma 5.3. *Let $x \in V_H$, one has $d_G(x, S'_\beta) = \min(d_H(x, S_\alpha), d_H(x, S_\beta))$, $d_G(x, S'_\alpha) = \infty$ and for any $\rho \neq \alpha, \beta$, $d_G(x, S'_\rho) = d_H(x, S_\rho)$.*

This holds even if x is in S_α or in S_β, because in this case either $d_H(x, S_\alpha)$ or $d_H(x, S_\beta)$ is zero, and so is $d_G(x, S'_\beta)$.

The same result holds for the directed case.

4. Case $G' = G[H/u_1, u_2, \ldots, u_k]$

We compute the length of the shortest paths in a graph obtained by a k-substitution of a graph H in a context graph G.

We can compute $d_G(x, y)$ for two vertices x and y of $G' = G[H/u_1, u_2, \ldots, u_k]$ in k steps: we computes the length of the shortest paths after substituting $S_1(H)$ for u_1, then after substituting $S_2(H)$ for u_2, ... and finally after the k-substitution of H for $u_1, \ldots, u_k$.

Lemma 5.4. *Let $G = \mathrm{val}'(c)$ be a context graph. Let H be a (context-)graph. Assume that the sets $S_i(H)$, $1 \leq i \leq k$, are non empty. Let $G_0 = G$ and for each i, $1 \leq i \leq k$, $G_i = G[H/u_1, u_2, \ldots, u_i]$. Then $V_{G_i} = V_{G_{i-1}} - \{u_i\}$ and for any $x, y \in V_{G_i}$, the following holds:*

$$\begin{aligned} d_{G_i}(x,y) = \min \quad & (d_{G_{i-1}}(x,y),\ d_{G_{i-1}}(x,u_i) + d_{G_{i-1}}(y,S_i(H)),\ d_{G_{i-1}}(y,u_i) + \\ & d_{G_{i-1}}(x,S_i(H)),\ d_{G_{i-1}}(x,S_i(H)) + 2 + d_{G_{i-1}}(y,S_i(H))\ ). \end{aligned}$$

Proof.

1. We perform the first partial substitution to obtain the graph $G_1 = G[H/u_1]$. For any vertices x and y in G_1, we consider the new shortest paths between x and y resulting from the substitution of u_1 by $S_1(H)$. Let $N_G(u_1)$ be the set of neighbors of u_1 in G. Recall that $V_{G_1} = V_G - \{u_1\} \cup V_H$ and that the new edges of G_1 belong to $N_G(u_1) \times S_1(H)$.
   First case. If x and y belong to V_G, then the new shortest paths will contain two necessarily consecutive new edges $e_1, e_2 \in N_G(u_1) \times S_1(H)$, but there existed a path of same length in G with two edges of $N_G(u_1) \times \{u_1\}$ instead of e_1, e_2. So the length of the shortest paths in G_1 is the same as in G.
   Second case. If x and y belong to V_H, then the new shortest paths will contain two necessarily consecutive new edges $e_1, e_2 \in N_G(u_1) \times S_1(H)$, this gives the result.

Third case. If x belongs to V_G and y belongs to V_H, then the new shortest paths will contain only one new edge of $N_G(u_1) \times \{u_1\}$ (otherwise they will not be shortest paths) this gives the result.

2. Assume that the result holds for any j, $1 \leq j \leq i-1$. We now perform the partial substitution $G_i = G[H/u_1, u_2, \ldots, u_i]$. Let $N_{G_{i-1}}(u_i)$ be the set of neighbors of u_i in G_{i-1}. The new edges of G_i belong to $N_{G_{i-1}}(u_i) \times S_i(H)$. For any vertices x and y of G_i, we consider the shortest paths between x and y resulting from the partial substitution of H for $u_1, u_2, \ldots, u_i$. Let P be such a path. If P already exists in G_{i-1}, then $d_{G_i}(x,y) = d_{G_{i-1}}(x,y)$. Otherwise P contains new edges belonging to $N_{G_{i-1}}(u_i) \times S_i(H)$. It cannot contain more than two new edges otherwise it will not be a shortest path. If it contains two new edges $e_1, e_2 \in N_G(u_1) \times S_1(H)$, they are necessarily consecutive, for the same reason. If this 2-path $\{e_1, e_2\}$ has its extremities in $N_G(u_1)$, then a 2-path $\{e'_1, e'_2\}$ crossing u_i existed in G_{i-1}. So it does not change the length of the shortest paths. If the 2-path $\{e_1, e_2\}$ has its extremities in $S_i(H)$, it may gives a new value of the length of the shortest paths, joined with a shortest path form x to $S_i(H)$, and a shortest path from $S_i(H)$ to y. This gives the result. □

Note that if p sets $S_i(H)$ are empty, a $(k-p)$-substitution is sufficient.

In the directed case, we have the following result:

Lemma 5.5. *Let $G = \mathrm{val}'(c)$ be a context graph. Let H be a [context-]graph. Assume that the sets $S_i(H)$, $1 \leq i \leq k$, are non empty. Let $G_0 = G$ and for each i, $1 \leq i \leq k$, $G_i = G[H/u_1, u_2, \ldots, u_i]$. Let $mcl_{G_{i-1}}(u_i)$ be the length of a minimum cycle containing u_i in G_{i-1}. For any $x, y \in V_{G_i}$, one has*

$$\begin{aligned}\overrightarrow{d}_{G_i}(x,y) = \min(&\overrightarrow{d}_{G_{i-1}}(x,y),\ \overrightarrow{d}_{G_{i-1}}(x,u_i) + \overrightarrow{d}_{G_{i-1}}(S_i(H),y),\ \overrightarrow{d}_{G_{i-1}}(y,u_i)\\ &+ \overrightarrow{d}_{G_{i-1}}(S_i(H),x),\ \overrightarrow{d}_{G_{i-1}}(x,S_i(H)) + mcl_{G_{i-1}}(u_1) + \overrightarrow{d}_{G_{i-1}}(S_i(H),y)\ ).\end{aligned}$$

The result comes from the fact that every directed cycle containing u_i in G_{i-1} gives rise, after the partial substitution of H for $u_1, u_2, \ldots, u_i$, to a directed path between any couple of vertices of $S_i(H)$. Such a path will give a new path between the vertices x and y of H.

So the length of the shortest paths between two vertices after a partial k-substitution can be computed in constant time. In the directed case, one need to compute the length of the shortest (oriented) cycle containing the substituted vertex in the context graph. This can obviously be done, with similar inductive relations as above.

The length of the shortest paths between a vertex x of G and any set S_ρ, $1 \leq \rho \leq k$, can be obviously computed in constant time, by similar inductive relations, in the directed and undirected case.

5.1. A system of inductive functions

Using the preceding lemmas, one can show that the system of functions $\{D_2, D_1, D_0\}$, defined at the beginning of the section, is a *SIF*.

We give the detailed formulas for the functions $\phi_f, \phi'_f, \phi''_f$, ψ_f, ψ'_f and ξ_f for any f in F in Appendix. These formulas are a straightforward consequence of the results of the preceding paragraph, applied to a composition of $2 * k^2 + 1$ basic clique-width-k operations. Thus, from the preceding results, the functions involved in the definition of the *SIF* $\{D_2, D_1, D_0\}$, can be computed in constant time with constants of size $20k^2$ (some of the functions $\phi_f, \phi'_f, \phi''_f$, ψ_f, ψ'_f and ξ_f compute the minimum of five data and five additions).

We let to the reader the case of $\phi_\bullet, \phi'_\bullet, \phi''_\bullet$, $\psi_\bullet, \psi'_\bullet$ and $\xi_\bullet$ (resp. $\phi_\circ, \phi'_\circ, \phi''_\circ$, $\psi_\circ, \psi'_\circ$ and $\xi_\circ$) which are very similar.

6. Second application: number of distinct shortest paths

We can easily apply a similar algorithm to compute the number of distinct shortest paths between two vertices. Let t be a term in $T(F, C) \cup \mathrm{Ctxt}(F, C)$ and let G be the graph or context graph such that $\mathrm{val}'(t) = G$. Let x and y be vertices of G. The function $\eta_G(x, y)$ will denote the number of distinct shortest paths between x and y in G. In the same way, $\eta_G(x, S_\alpha)$ will denote the number of distinct shortest paths between x and S_α, with the convention that if $x \in S_\alpha$ then $\eta_G(x, S_\alpha) = 1$ and if $\eta_G(x, S_\alpha) \geq 2$ then necessarily S_α has cardinality more than two (the extremities of the paths in S_α must be distinct). The function $\eta_G(S_\alpha, S_\beta)$ will denote the number of distinct shortest paths between S_α and S_β and similarly if $\eta_G(S_\alpha, S_\beta) \geq 2$ then necessarily S_α and S_β have cardinality more than two.

Case 1. $G = H \oplus J$

The number of distinct shortest paths between two vertices or between a vertex and a set S_γ will not changed after this operation.

Case 2. $G = add_{\alpha,\beta}(H)$

Recall that we can assume that no edges from S_α to S_β already exist in H. Thus, every shortest path containing an edge between S_α and S_β will be a new path.

Let *equal*(a, b) be the function from $\mathbb{N} \times \mathbb{N}$ into $\{0, 1\}$ which is 1 iff $a = b$.

Lemma 6.1. *Let $x, y \in V_H$. The number of shortest paths between x and y is*

$$\begin{aligned}\eta_G(x, y) = {} & \mathrm{equal}(d_G(x, y), d_H(x, y)) \;.\; \eta_H(x, y)\\ & + \mathrm{equal}(d_G(x, y), d_H(x, S_\alpha) + 1 + d_H(y, S_\beta)) \;.\; \eta_H(x, S_\alpha).\eta_H(y, S_\beta)\\ & + \mathrm{equal}(d_G(x, y), d_H(x, S_\beta) + 1 + d_H(y, S_\alpha)) \;.\; \eta_H(x, S_\beta).\eta_H(y, S_\alpha)\\ & + \mathrm{equal}(d_G(x, y), d_H(x, S_\alpha) + 2 + d_H(y, S_\alpha)) \;.\; \eta_H(x, S_\alpha).|S_\beta|.\eta_H(y, S_\alpha)\\ & + \mathrm{equal}(d_G(x, y), d_H(x, S_\beta) + 2 + d_H(y, S_\beta))) \;.\; \eta_H(x, S_\beta).|S_\beta|.\eta_H(y, S_\beta).\end{aligned}$$

Proof. In the case where $d_G(x,y) = d_H(x,y)$, the distinct shortest paths between x and y in H are still distinct shortest paths between x and y in G.

In the case where $d_G(x,y) = d_H(x,S_\alpha)+1+d_H(y,S_\beta)$, every couple of shortest paths from x to S_α and from y to S_β in H produces a new shortest path between x and y in G.

In the case where $d_G(x,y) = d_H(x,S_\alpha)+2+d_H(y,S_\alpha)$, every couple of shortest paths from x to S_α and from y to S_α in H produces as many distinct shortest paths in G as there are vertices in S_β. Note that the paths from x to S_α and from y to S_α have necessarily disjoint extremities in S_α (otherwise there would exist a shortest path in H).

If x belongs to S_α (resp. y belongs to S_β), as by convention, in this case, $\eta_G(x,S_\alpha) = 1$ (resp. $\eta_G(y,S_\beta) = 1$), the result remains true.
So the number of distinct shortest paths between two vertices of G can be computed in constant time.

The number of shortest paths between a vertex x of G and any set S_ρ, $1 \le \rho \le k$, can be obviously computed in constant time, by similar inductive relations, in the directed and undirected case. □

Case 3. $G = ren_{\alpha,\beta}(H)$

This operation does not modify the number of shortest paths between two vertices. Obviously it only modifies the number of shortest paths between a vertex x and the set S_α and S_β.

Case 4. $G' = G[H/u_1,u_2,\ldots,u_k]$

We can compute the distance between two vertices of the new graph G' as follows:

Lemma 6.2. *Let $G = \mathrm{val}'(c)$ be a context graph. Let H be a (context-)graph. Assume that the sets $S_i(H)$, $1 \le i \le k$, are non empty. Let $G_0 = G$ and for each i, $1 \le i \le k$, $G_i = G[H/u_1,u_2,\ldots,u_i]$. Then $V_{G_i} = V_{G_{i-1}} - \{u_i\}$ and for any $x,y \in V_{G_i}$, the following holds:*

$$\begin{aligned}
\eta_{G_i}(x,y) = {} & \mathrm{equal}(d_{G_i}(x,y), d_{G_{i-1}}(x,y)) \,.\, \eta_{G_{i-1}}(x,y) + \\
& + \mathrm{equal}(d_{G_i}(x,y), d_{G_{i-1}}(x,u_i) + d_{G_{i-1}}(y,S_i(H))) \,.\, \eta_{G_{i-1}}(x,u_i).\eta_{G_{i-1}}(y,S_i(H)) \\
& + \mathrm{equal}(d_{G_i}(x,y), d_{G_{i-1}}(y,u_i) + d_{G_{i-1}}(x,S_i(H))) \,.\, \eta_{G_{i-1}}(y,u_i).\eta_{G_{i-1}}(x,S_i(H)) \\
& + \mathrm{equal}(d_{G_i}(x,y), d_{G_{i-1}}(x,S_i(H)) + 2 + d_{G_{i-1}}(y,S_i(H))) \,.\, \eta_{G_{i-1}}(x,S_i(H)) \\
& \quad \cdot |N_{G_{i-1}}(u_i)| \,.\, \eta_{G_{i-1}}(y,S_i(H)).
\end{aligned}$$

Proof. In the case $d_{G_i}(x,y) = d_{G_{i-1}}(x,y)$, the distinct shortest paths between x and y in G_{i-1} are still distinct shortest paths between x and y in G_i.

In the case where $d_{G_i}(x,y) = d_{G_{i-1}}(x,u_i) + d_{G_{i-1}}(y,S_i(H))$, every couple of shortest paths from x to u_i and from y to $S_i(H)$ in G_{i-1} produces a new shortest path in G (replacing the edge between u_i and one of his neighbor by an edge between this neighbor and $S_i(H)$).

In the case where $d_{G_i}(x,y) = d_{G_{i-1}}(x, S_i(H)) + 2 + d_{G_{i-1}}(y, S_i(H))$, every couple of shortest paths from x to $S_i(H)$ and from y to $S_i(H)$ in G_{i-1} produces as many distinct shortest paths in G_i as there are vertices in $N_{G_{i-1}}(u_i)$ (the number of vertices in $N_G(u_i)$ can easily be inductively computed from the decomposition tree of G and one can deduce from it the value of $N_{G_{i-1}}(u_i)$). Note that the shortest paths from x to $S_i(H)$ and from y to $S_i(H)$ have necessarily disjoint extremities in $S_i(H)$ (otherwise there would exist a shortest path in G_{i-1}).

If x belongs to $S_i(H)$ (resp. y belongs to $S_i(H)$), as by convention, in this case, $\eta_G(x, S_i(H)) = 1$ (resp. $\eta_G(y, S_\beta) = 1$), the result remains true. So the number of distinct shortest paths between two vertices after a k-substitution can be computed in constant time. □

6.1. A system of inductive functions

Using the preceding lemmas, a system of inductive functions can then be given for the computation of the number of distinct shortest paths between two vertices, in the same way as for the computation of the length of the shortest paths between two vertices in the preceding paragraph.

As the operations of F (occurring in the term representing the graph) are compositions of at most $2k^2 + 1$ basic clique-width-k operations, a step of induction can be done in constant time, with constants of size $20k^2$.

7. Third application: length of shortest paths avoiding a set of vertices

Let t be a term in $T(F, C)$ and let G be the graph val(t). Let X be a fixed set of vertices of G. We say that a path avoids X if it contains no vertices of X.

If we replace in the set of inductive functions of Section 5 the length of the shortest paths, by the length of the shortest paths avoiding X, we obtain a system of inductive functions $\{D_0, D_1, D_2\}$ such that $D_2(t, x, y)$ is the length of the shortest paths avoiding X, between x and y in G. Then, by our main theorem, there exists a labelling scheme for a graph G of bounded clique width, such that for any fixed set X of vertices, one can compute the length of the shortest paths avoiding X, between any two vertices in constant time.

8. Fourth application: formal language constrained shortest paths

In this section, we deal with graphs with labelled edges. Let A be the finite set of letters labelling the edges of the graphs. We replace operations on graphs, $add_{\alpha,\beta}$ with $\alpha, \beta \in \{1, \ldots, k\}$ $(\alpha \neq \beta)$, by operations $add_{\alpha,\beta,a}$ with $\alpha, \beta \in \{1, \ldots, k\}$ and $a \in A$. The operation $add_{\alpha,\beta,a}$ add edges labelled with a. Since the graphs constructed with these operations have labelled edges, we will say that a word w

of A^* labels a path P if w is the concatenation of the letters labelling the edges of P.

Given a language L over A, the problem of formal language constrained shortest paths consists of finding a shortest path P in a graph G, with the additional constraint that the label of P (*i.e.* the word of letters that label its edges) belongs to L.

Our result applies in the case of a regular language L. If L is regular, there exists a deterministic automaton recognising L. Let Q be the set of states of this automaton and p be the cardinality of Q. For any (i, j) with $1 \leq i, j \leq p$, let $L_{(i,j)}$ be the language accepted by the automaton from the state q_i to the state q_j. Replace in the set of inductive functions used to compute the length of the shortest paths, any distance between x (resp S_α) and y (resp S_β) by a $p \times p$ vector such that the (i, j) component is the length of the shortest paths labelled with a word of $L_{(i,j)}$. We obtain a set of inductive functions. Then, by our main theorem, there exists a labelling scheme for a graph G of bounded clique-width, such that one can compute the constrained shortest paths between any two vertices in constant time, for any regular language of constraints.

9. Conclusion

The paper presents an efficient algorithm which, for a given system of inductive functions, builds a labelling scheme for graphs of clique-width at most k. It is then possible to compute the functions of the system in constant time from the labelling scheme.

It can be used to compute the length of the shortest paths, the number of distinct shortest paths, the length of the shortest paths avoiding a fixed set X of vertices, the length of constrained shortest paths, as well as other inductive functions on graphs. It avoids logical tools and thus huge constants, that appear in the use of automata and MS formula. The constants here are of size $c.k^2$ where the constant c depends on the computed functions ($c = 20$ for the length of the shortest paths or for the number of distinct shortest paths, $c = 20p^2$ for the constrained shortest paths, when the automaton has p states).

The values of the functions in the *SIF* are either numbers or vectors of numbers. Once the precomputation has been done, any numerical value or component of the vectors of the *SIF* can be computed in constant time. The algorithm applies as well to directed or weighted graphs, which were not treated in the paper by Courcelle and Vanicat (see [11]).

The set of functions belonging to some *SIF* is not contained in the set of MS definable optimisation functions. We give a system of inductive functions to compute the number of distinct shortest paths, which is not an MS definable optimisation function. On the other hand, we conjecture that the set of MS definable optimisation functions is not contained in the set of functions belonging to some *SIF*. The existence of p disjoint paths, $p \geq 2$ (not necessarily shortest paths) between two vertices is an MS definable Boolean query, but no system of inductive functions is known to compute this query (for example if G is the graph $add_{\alpha,\beta}(H)$, we can not know how many new paths in G intersect the previous paths in H).

The case of functions with alternating computations of min and max remains an open problem. Neither the logical approach of Courcelle and Vanicat nor the inductive approach generally apply to such functions.

Acknowledgements. I thank B. Courcelle and the referees for their critical reading and their numerous useful comments.

Appendix. Inductive relations to compute the length of the shortest paths

Inductive relations for the term $f(t_1, t_2)$ with f in F and t_1, t_2 in $T(F, C)$. Let $f = \otimes_{R,L}$ be an operation of F, with $L = \{(\alpha_1, \beta_1)(\alpha_2, \beta_2), \ldots, (\alpha_p, \beta_p)\}$. We assume that α_i is a label occurring in the left argument of f and β_i is a label occurring in the right argument of f, for any $1 \le i \le p$. We must show that there exist functions $\phi_f, \phi'_f, \phi''_f$ (resp. ψ_f, ψ'_f, resp. ξ_f) such that D_2 (resp. D_1, resp. D_0) fulfill the conditions of Definition 3.1.

Recall that $D_1(t_1, x)$, $D_1(t_2, y)$ are vectors of length k. Let π'_α denote the projection on the αth component of a vector of length k. Recall that $D_0(t_1)$, $D_0(t_2)$ are vectors of length k^2. We will index these vectors by couples (i, j), $1 \le i, j \le k$. Let $\pi''_{\alpha,\beta}$ denote the projection on the (α, β) component of a vector of length k^2.

To compute the functions $\phi_f, \phi'_f, \phi''_f$ (resp. ψ_f, ψ'_f, resp. ξ_f) we will proceed step by step, computing new functions $\phi_i, \phi'_i, \phi''_i$ (resp. ψ_i, ψ'_i, resp. ξ_i) after adding the edges $S_{\alpha_i} \times S_{\beta_i}$. The case $i = 1$ needs a special treatment because the edges are added between two disjoint graphs.

Case $i = 1$.

ϕ_1 will have four arguments, $D_1(t_1, x)$, $D_1(t_2, y)$, $D_0(t_1)$, $D_0(t_2)$. So we will need the projections $\pi_j^{(4)} : \mathbb{N}^4 \longrightarrow \mathbb{N}$, $1 \le j \le 4$.

ϕ'_1 and ϕ''_1 will have five arguments, $D_2(t_2, x, y)$, $D_1(t_2, x)$, $D_1(t_2, y)$, $D_0(t_1)$, $D_0(t_2)$. So we will need the projections $\pi_j^{(5)} : \mathbb{N}^5 \longrightarrow \mathbb{N}$, $1 \le j \le 5$.

- If $x \in t_1$, $y \in t_2$, one defines $\phi_1 = \pi'_{\alpha_1}(\pi_1^{(4)}) + 1 + \pi'_{\beta_1}(\pi_2^{(4)})$.
- If $x, y \in t_1$, one defines $\phi'_1 = \min(\pi_1^{(5)}, \pi'_{\alpha_1}(\pi_2^{(5)}) + 2 + \pi'_{\alpha_1}(\pi_3^{(5)}))$.
- If $x, y \in t_2$, one defines $\phi''_1 = \min(\pi_1^{(5)}, \pi'_{\beta_1}(\pi_2^{(5)}) + 2 + \pi'_{\beta_1}(\pi_3^{(5)}))$.

ψ_1 (resp. ψ'_1) will have three arguments, $D_1(t_1, x)$, $D_0(t_1)$, $D_0(t_2)$ (resp. $D_1(t_2, y)$, $D_0(t_1)$, $D_0(t_2)$). So we will need the projections $\pi_j^{(3)} : \mathbb{N}^3 \longrightarrow \mathbb{N}$, $1 \le j \le 3$.

- If $x \in t_1$, let ψ_1 be the vector function with jth component: $(\psi_1)_j = \begin{cases} \min(\pi'_j(\pi_1^{(3)}), \pi'_{\alpha_1}(\pi_1^{(3)}) + 2 + \pi''_{\alpha_1,j}(\pi_2^{(3)})), & \text{if } j \text{ labels } t_1, \\ \pi'_{\alpha_1}(\pi_1^{(3)}) + 1 + \pi_{\beta_1,j}(\pi_3^{(3)}), & \text{if } j \text{ labels } t_2. \end{cases}$

– If $x \in t_2$, let ψ'_1 be the vector function with jth component:

$$(\psi'_1)_j = \begin{cases} \min(\pi'_j(\pi_1^{(3)}), \pi'_{\alpha_1}(\pi_1^{(3)}) + 2 + \pi''_{\alpha_1,j}(\pi_3^{(3)})), & \text{if } j \text{ labels } t_1, \\ \pi'_{\alpha_1}(\pi_1^{(3)}) + 1 + \pi_{\beta_1,j}(\pi_2^{(3)}), & \text{if } j \text{ labels } t_2. \end{cases}$$

ξ_1 will have two arguments, $D_0(t_1)$, $D_0(t_2)$. So we will need the projections $\pi_j^{(2)} : \mathbb{N}^2 \longrightarrow \mathbb{N}$, $1 \leq j \leq 2$. Let ξ_1 be the vector function having k^2 components, the (r,s)-component being:

$$(\xi_1)_{r,s} = \begin{cases} \pi''_{r,\alpha_1}(\pi_1^{(2)}) + 1 + \pi''_{\beta_1,s}(\pi_2^{(2)}), & \text{if } r \text{ labels } t_1,\ s \text{ labels } t_2, \\ \min(\pi''_{r,s}(\pi_1^{(2)}), \pi''_{r,\alpha_1}(\pi_1^{(2)}) + 2 + \pi''_{\alpha_1,s}(\pi_1^{(2)})), & \text{if } r \text{ and } s \text{ label } t_1, \\ \min(\pi''_{r,s}(\pi_2^{(2)}), \pi''_{r,\beta_1}(\pi_2^{(2)}) + 2 + \pi''_{\beta_1,s}(\pi_2^{(2)})), & \text{if } r \text{ and } s \text{ label } t_2. \end{cases}$$

Case $i > 1$.

– After adding the edges $S_{\alpha_i} \times S_{\beta_i}$, for any i, $2 \leq i \leq p$, the function ϕ_i, ϕ'_i and ϕ''_i are obtained by applying the next function g_i to the relevant tuple:
$(\phi_{i-1}, \psi_{i-1}, \psi'_{i-1})$, if $x \in t_1$, $y \in t_2$,
$(\phi'_{i-1}, \psi_{i-1}, \psi_{i-1})$, if $x, y \in t_1$,
$(\phi''_{i-1}, \psi'_{i-1}, \psi'_{i-1})$, if $x, y \in t_2$.
For any i, $2 \leq i \leq p$, let $g_i = \min(\pi_1^{(3)}, \pi'_{\alpha_i}(\pi_2^{(3)}) + 1 + \pi'_{\beta_i}(\pi_3^{(3)}), \pi'_{\beta_i}(\pi_2^{(3)}) + 1 + \pi'_{\alpha_i}(\pi_3^{(3)}), \pi'_{\alpha_i}(\pi_2^{(3)}) + 2 + \pi'_{\alpha_i}(\pi_3^{(3)}), \pi'_{\beta_i}(\pi_2^{(3)}) + 2 + \pi'_{\beta_i}(\pi_3^{(3)}))$.

– For any i, $2 \leq i \leq p$, the functions ψ_i and ψ'_i are obtained by applying the next function h_i to the relevant tuple:
$(\psi_{i-1}, \xi_{i-1}, \xi'_{i-1})$, if $x \in t_1$,
$(\psi'_{i-1}, \xi_{i-1}, \xi'_{i-1})$, if $x \in t_2$.
For any i, $2 \leq i \leq p$, let h_i be the vector function with jth component:
$(h_i)_j = \min(\pi'_j(\pi_1^{(3)}), \pi'_{\alpha_i}(\pi_1^{(3)}) + 1 + \pi''_{\beta_i,j}(\pi_2^{(3)}), \pi'_{\beta_i}(\pi_1^{(3)}) + 1 + \pi''_{\alpha_i,j}(\pi_2^{(3)}), \pi'_{\alpha_i}(\pi_1^{(3)}) + 2 + \pi''_{\alpha_i,j}(\pi_2^{(3)}), \pi'_{\beta_i}(\pi_1^{(3)}) + 2 + \pi''_{\beta_i,j}(\pi_3^{(3)}))$.

– For any i, $2 \leq i \leq p$, the function ξ_i is obtained by applying the next function m_i to ξ_{i-1}. Let m_i be the vector function having k^2 components, the (r,s)-component being:
$(m_i)_{r,s} = \min(\pi''_{r,s}, \pi''_{r,\alpha_i} + 1 + \pi''_{\beta_i,s}, \pi''_{r,\beta_i} + 1 + \pi''_{\alpha_i,s}, \pi''_{r,\alpha_i} + 2 + \pi''_{\alpha_i,s}, \pi''_{r,\beta_i} + 2 + \pi''_{\beta_i,s})$.

Then the functions $\phi_f, \phi'_f, \phi''_f$ are respectively equal to $\phi_p, \phi'_p, \phi''_p$, because the renamings occurring in f have no effect on these functions.

The renamings affect the functions ψ_f and ψ'_f and ξ_f.

Let $R = \{(\gamma_1, \rho_1)(\gamma_2, \rho_2), \ldots, (\gamma_p, \rho_q)\}$.
For any i, $2 \leq i \leq q$, let h'_i be the vector function with ρ_ith component: $(h'_i)_{\rho_i} = \min(\pi'_{\gamma_i}, \pi'_{\rho_i})$, γ_ith component: $(h'_i)_{\gamma_i} = \infty$, the other components being the identity. Then one has:

$$\psi_f = h'_q(\ldots(h'_2(h'_1(\psi_p))\ldots) \text{ and } \psi'_f = h'_q(\ldots(h'_2(h'_1(\psi'_p))\ldots).$$

For any i, $2 \leq i \leq q$, let (m'_i) be the vector function having k^2 components, the (α, ρ_i)- and the (ρ_i, α)-components being: $(m'_i)_{\alpha,\rho_i} = (m'_i)_{\rho_i,\alpha} = \min(\pi''_{\gamma_i,\alpha}, \pi'_{\rho_i,\alpha})$, the (α, γ_i)- and the (γ_i, α)-components being: $(m'_i)_{\alpha,\gamma_i} = (m'_i)_{\gamma_i,\alpha} = \infty$, the other components being the identity.

Then one has $\xi'_f = m'_q(\ldots(m'_2(m'_1(\xi'_p))\ldots)$.

References

[1] S. Arnborg, J. Lagergren, D. Seese, Easy problems for tree-decomposable graphs. *J. Algor.* **12** (1991) 308–340.

[2] H. Bodlaender, Treewidth: Algorithmic techniques and results, in *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science. Lect. Notes Comput. Sci.* **1295** (1997) 19–36.

[3] R.B. Borie, R.G. Parker, C.A. Tovey, Algorithms on Recursively Constructed Graphs. *CRC Handbook of Graph Theory* (2003) 1046–1066.

[4] S. Chaudhuri, C.D. Zaroliagis, Optimal parallel shortest paths in small treewidth digraphs, in: *Proceedings 3rd Annual European Symposium on Algorithms. Lect. Notes Comput. Sci.* **979** (1995) 31–45.

[5] D.G. Corneil, M. Habib, J.M. Lanlignel, B.A. Reed, U. Rotics, Polynomial time recognition algorithm of clique-width ≤ 3 graphs, *LATIN'00. Lect. Notes Comput. Sci.* **1776** (2000) 126–134.

[6] B. Courcelle, Clique-width of countable graphs: a compactness property. *Discrete Math.* **276** (2003) 127–148.

[7] B. Courcelle, J.A. Makowsky, U. Rotics, On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Appl. Math.* **108** (2001) 23–52.

[8] B. Courcelle, M. Mosbah, Monadic second-order evaluations of tree-decomposable graphs. *Theoret. Comput. Sci.* **109** (1993) 49–82.

[9] B. Courcelle, S. Olariu, Upper bounds to clique-width of graphs. *Discrete Appl. Math.* **101** (2000) 77–114.

[10] B. Courcelle, A. Twigg, Compact forbidden-set routing, in: *STACS'07. Lect. Notes Comput. Sci.* **4393** (2007) 37–48.

[11] B. Courcelle, R. Vanicat, Query efficient implementations of graphs of bounded clique-width. *Discrete Appl. Math.* **131** (2003) 129–150.

[12] C. Demetrescu, G.F. Italiano, a new approach to dynamic all pairs shortest paths, in *Proceedings of. the 35. th. Annual ACM Symposium on the Theory of Computing* (2003) 159–166.

[13] R.G. Downey, M.R. Fellows, *Parametrized Complexity.* Springer Verlag (1999).

[14] J. Engelfriet, G. Rozenberg, Node replacement graph grammars, in *Handbook of Graph Grammars and Computing by Graph Transformation, Foundations*, Vol. **1**, edited by G. Rozenberg. World Scientific (1997) 1–94.

[15] M.R. Fellows, F.A. Rosamond, U. Rotics, S. Szeider, Clique-width minimization is NP-hard. *Proceedings of. the 38. th. Annual ACM Symposium on the Theory of Computing* (2006) 354–362.

[16] J. Flum, M. Grohe, *Theory of parametrized complexity.* Springer Verlag (2006).

[17] M. Frick, M. Grohe, The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic* **130** (2004) 3–31.

[18] C. Gavoille, M. Katz, Nir A. Katz, C. Paul, D. Peleg, Approximate distance labeling schemes, *ESA'01. Lect. Notes Comput. Sci.* **2161** (2001) 476–488.

[19] C. Gavoille, C. Paul, Distance labeling scheme and split decomposition. *Discrete Math.* **273** (2003) 115–130.

[20] C. Gavoille, D. Peleg, Compact and localized distributed data structures. *J. Distrib. Comput.* **16** (2003) 111–120.

[21] C. Gavoille, D. Peleg, S. Pérennes, R. Raz, Distance labeling in graphs. *J. Algor.* **53** (2004) 85–112.

[22] E. Wanke, k-NLC graphs and polynomial algorithms. *Disc. Appl. Math.* **54** (1994) 251–266.

[23] F. Gurski, E. Wanke, Vertex disjoint paths on clique-width bounded graphs, *LATIN'04. Lect. Notes Comput. Sci.* **2978** (2004) 119–128.

[24] D. Harel, R. Tarjan, Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* **13** (1984) 338–355.

[25] P. Hlinený, S. Oum, Finding Branch-Decompositions and Rank-Decompositions. *SIAM J. Comput.* **38** (2008) 1012–1032.

[26] D. Seese, Interpretability and tree automata: A simple way to solve algorithmic problems on graphs closely related to trees, in *Tree Automata and Languages*, edited by M. Nivat, A. Podelski. North-Holland (1992) 83–114.

[27] J.P. Spinrad, *Efficient Graph Representations.* American Mathematical Society (2003).

Communicated by C. Choffrut.
Received June 25, 2007. Accepted February 21st, 2009.