k-Chordal Graphs: from Cops and Robber to Compact Routing via Treewidth

Adrian Kosowski, Bi Li, Nicolas Nisse, Karol Suchan

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

k-Chordal Graphs: from Cops and Robber to Compact Routing via Treewidth

Adrian Kosowski — Bi Li — N. Nisse — Karol Suchan

N° 7888

February 2012

Domaine

ISSN 0249-6399 ISRN INRIA/RR--7888--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

k-Chordal Graphs: from Cops and Robber to Compact Routing via Treewidth *

Adrian Kosowski† , Bi Li‡§ , N. Nisse‡ , Karol Suchan¶‖

Thème :
Équipe-Projet Mascotte

Rapport de recherche n° 7888 — February 2012 — 13 pages

Abstract: *Cops and robber games* concern a team of cops that must capture a robber moving in a graph. We consider the class of k-chordal graphs, i.e., graphs with no induced cycle of length greater than k, $k \geq 3$. We prove that $k-1$ cops are always sufficient to capture a robber in k-chordal graphs. This leads us to our main result, a new structural decomposition for a graph class including k-chordal graphs.

We present a quadratic algorithm that, given a graph G and $k \geq 3$, either returns an induced cycle larger than k in G, or computes a *tree-decomposition* of G, each *bag* of which contains a dominating path with at most $k-1$ vertices. This allows us to prove that any k-chordal graph with maximum degree Δ has treewidth at most $(k-1)(\Delta-1)+2$, improving the $O(\Delta(\Delta-1)^{k-3})$ bound of Bodlaender and Thilikos (1997). Moreover, any graph admitting such a tree-decomposition has hyperbolicity $\leq \lfloor \frac{3}{2}k \rfloor$.

As an application, for any n-node graph admitting such a tree-decomposition, we propose a *compact routing scheme* using routing tables, addresses and headers of size $O(\log n)$ bits and achieving an additive stretch of $O(k \log \Delta)$. As far as we know, this is the first routing scheme with $O(\log n)$-routing tables and small additive stretch for k-chordal graphs.

Key-words: Treewidth, chordality, hyperbolicity, compact routing, cops and robber games.

* Partially supported by programs Fondap and Basal-CMM, Anillo ACT88 and Fondecyt 11090390 (K.S.), FP7 STREP EULER (B.L.,N.N.).
† CEPAGE, INRIA, LaBRI, Talence, France
‡ MASCOTTE, INRIA, I3S(CNRS/Univ. Nice Sophia Antipolis), France firstname.lastname@inria.fr
§ CAS & AAMS, Beijing, China
¶ FIC, Universidad Adolfo Ibáñez, Santiago, Chile
‖ WMS, AGH - University of Science and Technology, Krakow, Poland

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

k-Chordal Graphs: from Cops and Robber to Compact Routing via Treewidth

Résumé : Nous présentons un algorithme quadratique qui, étant donné un graphe G et un entier $k \geq 3$, certifie que G contient un cycle induit de longueur $> k$, ou calcule une décomposition arborescente de G dont chaque "sac" induit un k-caterpillar (graphe qui contient un chemin dominant, de longueur au plus $k-2$). Entre autre, ce résultat implique que les graphes k-cordaux (sans cycle induit de longueur $> k$) de maximum degree Δ ont une largeur arborescente $O(k \cdot \Delta)$, ce qui améliore la borne $\Delta(\Delta-1)^{k-3}$ de Bodlaender et Thilikos (1997). De plus, l'hyperbolicité d'un graphe admettant une telle décomposition est $\leq \lfloor \frac{3}{2}k \rfloor$. Pour tout graphe qui admet une telle décomposition, nous proposons un algorithme de routage compact utilisant des adresses, en-têtes et tables de routage de taille $O(k \log n)$ bits et de stretch $O(k \cdot \log \Delta)$. Au passage, nous montrons que $k-1$ policiers sont suffisants pour capturer un voleur dans un graphe k-cordal.

Mots-clés : Cordalité, hyperbolicité, décomposition arborescente, routage compact, un peu de gendarmes et voleur.

1 Introduction

Because of the huge size of real-world networks, an important current research effort concerns exploiting their structural properties for algorithmic purposes. Indeed, in large-scale networks, even algorithms with polynomial-time in the size of the instance may become unpractical. Therefore, it is important to design algorithms depending only quadratically or linearly on the size of the network when its topology is expected to satisfy some properties. Among these properties, the *chordality* of a graph is the length of its longest induced (i.e., chordless) cycle. The (Gromov) *hyperbolicity* of a graph reflects how the metric (distances) of the graph is close to the metric of a tree. More precisely, a graph has hyperbolicity $\leq \delta$ if, for any $u, v, w \in V(G)$ and for any shortest paths P_{uv}, P_{vw}, P_{uw} between these three vertices, any vertex in P_{uv} is at distance at most δ from $P_{vw} \cup P_{uw}$ [**?**]. Intuitively, in a graph with small hyperbolicity, any two shortest paths between the same pair of vertices are close to each other. Several recent works take advantage of such structural properties of large-scale networks for algorithm design (e.g., routing [**?**,**?**]). Indeed, Internet-type networks have a so-called high clustering coefficient (see e.g. [**?**,**?**]), leading to the existence of very few long chordless cycles, whereas their low (logarithmic) diameter implies a small hyperbolicity [**?**].

Another way to study tree-likeness of graphs is by *tree-decompositions*. Introduced by Robertson and Seymour [**?**], such decompositions play an important role in design of efficient algorithms. Roughly speaking, a tree-decomposition maps each vertex of a graph to a subtree of the *decomposition tree* in a way that the subtrees assigned to adjacent vertices intersect [**?**, **?**]. The nodes of the decomposition tree are called *bags*, and the size of a bag is the number of vertices assigned to it (assigned subtrees intersect the bag). The *width* of a tree-decomposition is the maximum size over its bags, and the *treewidth* of a graph is the smallest width over its tree-decompositions. By using dynamic programming based on a tree-decomposition, many NP-hard problems have been shown to be linear time solvable for graph with bounded treewidth [**?**]. In particular, there are linear-time algorithms to compute an optimal tree-decomposition of a graph with bounded treewidth [**?**, **?**]. However, from the practical point of view, this approach has several drawbacks. First, all above-mentioned algorithms are linear in the size of the graph but (at least) exponential in the treewidth. Moreover, due to the high clustering coefficient of large-scale networks, their treewidth is expected to be large [**?**]. Hence, to face these problems, it is important to focus on the structure of the bags of the tree-decomposition, instead of trying to minimize their size. For instance, several works study the diameter of the bags [**?**, **?**]. In this work, we consider tree-decompositions in which each bag admits a particular small dominating set. Such decompositions turn out to be applicable to a large family of graphs (including k-chordal graphs).

1.1 Our results

Our results on tree decomposition are inspired by a study of the so called *cops and robber games*. The aim of such a game is to capture a robber moving in a graph, using as few cops as possible. This problem has been intensively studied in the literature, allowing for a better understanding of the structure of graphs [**?**].

Outline of the paper. We start by presenting our results for the cops and robber problem in Section 2. Next, using these results, in Section 3 we provide a new type of efficiently computable tree-decomposition which we call *good tree decomposition*. Our tree decomposition turns out to be applicable to many real-world graph classes (including k-chordal graphs), and has several algorithmic applications. Finally, we focus on the applications of this decomposition to the *compact routing problem*, a research area in which tree decompositions have already proved useful [**?**]. The objective of compact routing is to provide a scheme for finding a path from a sender node to a known destination, taking routing decisions for the packet at every step using only very limited information stored at each node. In Section 4, we show how to use our tree decomposition to minimize the additive stretch of the routing scheme (i.e., the difference between the length of a route computed by the scheme and that of a shortest path connecting the same pair of nodes) in graphs admitting with k-good tree-decomposition for any given integer $k \geq 3$ (including k-chordal graphs), assuming logarithmic size of packet headers and routing tables stored at each node.

The necessary terminology concerning cops and robber games, tree decompositions, and compact routing, is introduced in the corresponding sections.

Main contributions. Our main contribution is the design of a quadratic algorithm that, given a n-node graph G and an integer $k \geq 3$, either returns an induced cycle of length at least $k+1$ in G or computes a tree-decomposition of G with each bag having a dominating path of order $\leq k-1$. More precisely, each bag of our tree-decomposition contains a chordless path with at most $k-1$ vertices, such that any vertex in the bag is either in the path or adjacent to some vertex of the path. In the case when G admits such a decomposition, this ensures that G has treewidth at most $(k-1)(\Delta-1)+2$ (where Δ is the maximum degree), tree-length at most k and hyperbolicity at most $\lfloor \frac{3}{2}k \rfloor$. In particular, this shows that the treewidth of any k-chordal graph is upper-bounded by $O(k \cdot \Delta)$, improving the exponential bound of [**?**]. The proposed algorithm is mainly derived from our proof of the fact that $k-1$ cops are sufficient to capture a robber in k-chordal graphs (generalizing some results in [**?**, **?**]).

Our tree-decomposition may be used efficiently for solving problems using dynamic programming in graphs of small chordality and small maximum degree. In particular, we present a compact routing scheme that uses our tree-decomposition and that achieves an additive stretch $\leq 2k(\lceil \log \Delta \rceil + \frac{5}{2}) - 5$ with routing tables, addresses and message headers of $O(\max\{k \cdot \log \Delta, \log n\})$ bits. An earlier approach of Dourisboure achieved stretch $k+1$, but with routing tables of size $O(\log^2 n)$.

1.2 Related Work

Chordality and hyperbolicity. Chordality and hyperbolicity are both parameters measuring the tree-likeness of a graph. Some papers consider relations between them [**?**, **?**]. In particular, the hyperbolicity of a k-chordal graph is at most k, but the gap may be arbitrary large (take a $3 \times n$-grid). The seminal definition of hyperbolicity is the following. A graph G is δ-hyperbolic provided that for any vertices $x, y, u, v \in V(G)$, the two larger of the three sums $d(u,v)+d(x,y), d(u,x)+d(v,y)$ and $d(u,y)+d(v,x)$ differ by at most 2δ [**?**]. This definition is equivalent to the one we use in this paper, using so called *thin triangles*, up to a constant ratio. No algorithm better than the $O(n^4)$-brute force algorithm (testing all 4-tuples in G) is known to compute hyperbolicity of n-node graphs. The problem of computing the chordality of a graph G is NP-complete since it may be related to computing a longest cycle in the graph obtained from G after subdividing all edges once. Finding the longest induced path is $W[2]$-complete [**?**] and the problem is Fixed Parameter Tractable in planar graphs [**?**]. It is coNP-hard to decide whether an n-node graph G is k-chordal for $k = \Theta(n)$ [**?**].

Treewidth. It is NP-complete to decide whether the treewidth of a graph G is at most k [**?**]. For chordal graphs, cographs [**?**], circular arc graphs [**?**], chordal bipartite graphs [**?**] and etc., the treewidth problem is polynomially solvable. Bodlaender and Thilikos proved that the treewidth of a k-chordal graph with maximum degree Δ is at most $\Delta(\Delta-1)^{k-3}$ which implies that treewidth is polynomially computable in the class of graphs with chordality and maximum degree bounded by constants [**?**]. They also proved that the treewidth problem is NP-complete for graphs with small maximum degree [**?**].

Compact routing. In [**?**], a universal name-independent routing scheme with stretch linear in k and $n^{1/k} polylog(n)$ space is provided. There are weighted trees for which every name-independent routing scheme with space less than $n^{1/k}$ requires stretch at least $2k+1$ and average stretch at least $k/4$ [**?**]. Subsequently, the interest of the scientific community was turned toward specific properties of graphs. Several routing schemes have been proposed for particular graph classes: e.g., trees [**?**], bounded growth [**?**], bounded doubling dimension [**?**, **?**], excluding a fixed graph as a minor [**?**, **?**], etc. The best compact routing scheme in k-chordal graphs (independent from the maximum degree) is due to Dourisboure and achieves a stretch of $k+1$ using routing tables of size $O(\log^2 n)$ bits [**?**]. A routing scheme achieving stretch $k-1$ with a distributed algorithm for computing routing tables of size $O(\Delta \log n)$ bits has been proposed in [**?**].

2 A detour through Cops and Robber games

Let us formally define a cops' strategy to capture a robber. Given a graph G, a player starts by placing $k \geq 1$ cops on some vertices of G, then a visible robber is placed on one vertex of G. Alternately, the cop-player may move each cop along one edge, and then the robber can move to an adjacent vertex. The robber is captured if, at some step, a cop occupies the same vertex.

Aigner and Fromme introduced the notion of *cop-number* of a graph G, i.e., the fewest number of cops required to capture a robber in G, denoted by $cn(G)$ [**?**]. A long standing conjecture due to Meyniel states that $cn(G) = O(\sqrt{n})$ for any n-node graph G [**?**]. To tackle this question, many researchers have focused on particular graph classes and provided many nice structural results (see the recent book [**?**]). For any n-node graph G, $cn(G) = O(\frac{n}{2^{(1-o(1))\sqrt{\log n}}})$ [**?**, **?**], $cn(G) \leq 3$ in any planar graph G [**?**], $cn(G) \leq 3 + \frac{3}{2}g$ in any graph G with genus at most g [**?**], $cn(G) = O(m)$ in any graph G excluding a m-edge graph as a minor [**?**], etc. Bounded hyperbolicity graphs have been considered in [**?**]. The cop number of graphs with minimum degree d and smallest induced cycle (girth) at least $8t-3$ is known to be $\Omega(d^t)$ [**?**]. Strangely, few is known when related to the largest induced cycle (chordality): in [**?**], it is shown that $cn(G) \leq 3$ any 2-connected 5-chordal graph G. In this section, we consider the class of k-chordal graphs.

Theorem 1. *Let $k \geq 3$. For any k-chordal connected graph G, $cn(G) \leq k-1$, and there exists a strategy where all $k-1$ cops always occupy a chordless path.*

Proof. Let $v \in V$ be any vertex and place all cops at it. Then, the robber chooses a vertex. Now, at some step, assume that the cops are occupying $\{v_1, \cdots, v_i\}$ which induce a chordless path, $i \leq k-1$, and it is the turn of the cops (initially $i = 1$). Let $N = \cup_{j \leq i} N[v_j]$, if the robber occupies a vertex in N, it is captured during the next move. Else, let $R \neq \emptyset$ be the connected component of $G \setminus N$ occupied by the robber. Finally, let S be the set of vertices in N that have some neighbor in R. Clearly, while R is not empty, then so does S.

Now, there are two cases to be considered.

- If $N(v_1) \cap S \subseteq \cup_{1<j\leq i} N[v_j]$. This case may happen only if $i > 1$. Then, "remove" the cop(s) occupying v_1. That is, the cops occupying v_1 go to v_2. Symmetrically, if $N(v_i) \cap S \subseteq \cup_{1 \leq j<i} N[v_j]$, then the cops occupying v_i go to v_{i-1}. Then, the cops occupy a shorter chordless path while the robber is still restricted to R.

- Hence, there is $u \in (N(v_1) \cap S) \setminus (\cup_{1<j\leq i} N[v_j])$ and $v \in (N(v_i) \cap S) \setminus (\cup_{1 \leq j<i} N[v_j])$. First, we show that this case may happen only if $i < k-1$. Indeed, otherwise, let P be a shortest path between such u and v with all internal vertices in R (possibly, P is reduced to an edge). Such a path exists by definition of S. Then, $v_1, \cdots, v_i, v, P, u$ is a chordless cycle of length at least $i+2$. Since G is k-chordal, this implies that $i+2 \leq k$.

  Then, one cop goes to $v = v_{i+1}$ while all the vertices in $\{v_1, \cdots, v_i\}$ remain occupied. Since $v \in S$, it has some neighbor in R, and then, the robber is restricted to occupy R' the connected component of $G \setminus (N \cup N[v])$ which is strictly contained in R.

Therefore, proceeding as described above strictly reduces the area of the robber (i.e., R) after $< k$ steps and then the robber is eventually captured. □

Note that previous Theorem somehow extends the model in [**?**] where the authors consider the game when two cops always reminding at distance at most 2 from each other must capture a robber.

Now, we improve a bit the previous theorem in case of 4-chordal graphs.

Theorem 2. *For any 4-chordal connected graph G, then $cn(G) \leq 2$ and there always exists a winning strategy for the cops such that they are always at distance at most one from each other.*

Proof. Initially, place the cops on any two adjacent vertices. At some step of the strategy, let us assume that the cops are on two adjacent vertices a and b (or $a = b$) and it is the turn of the cops. If the robber stands at some vertex in $N = N[a] \cup N[b]$, then it is captured during the next move. Hence, let C be the connected component of $G \setminus N$ where the robber stands. Let $S \subseteq N$ be the set of the vertices adjacent to a or b and at least one vertex of C, i.e., S is an inclusion-minimal separator between $\{a,b\}$ and C.

We will prove that there is $z \in \{a,b\}$ and a vertex c in $S \cap N(z)$, such that, $S \subset N[z] \cup N[c] = N'$. Since $c \in S$, $N(c) \cap V(C) \neq \emptyset$. Hence, if the cops move from a,b to c,z, which can be done in one step, then the robber is constrained to occupy a vertex of C' where C' is the connected component of $G \setminus N'$ which is strictly contained in C. Note that, C' is a proper subgraph of C. Iterating such moves, the robber will eventually be captured.

It remains to prove the existence of $z \in \{a,b\}$ and $c \in S \cap N(z)$, such that, $S \subseteq N[z] \cup N[c]$.

- If there is $z \in \{a,b\}$ such that $S \subseteq N[z]$, then any vertex in $N(z) \cap S$ satisfies the requirements.
- Else, let $c \in S \setminus N(b)$ (such a vertex exists because otherwise we would be in the previous case). Clearly, $S \cap N(a) \subseteq N[a] \cup N[c]$. Now, let $x \in S \setminus N(a)$. By definition of S, there is a path P from x to c in C. Moreover, all internal vertices of P are at distance at least two from a and b, c is not adjacent to b and x is not adjacent to a. Hence, considering the cycle a,b,x,P,c, there must be an edge between x and c because G is 4-chordal. Therefore, $S \subseteq N[a] \cup N[c]$.

The bound provided by this theorem is tight because of the cycle with 4 vertices. □

Theorem 1 relies on chordless paths P in G such that $N[P]$ is a separator of G. In next section, we show how to adapt this to compute particular tree-decompositions.

3 Structured Tree-decomposition

In this section, we present our main contribution, that is, an algorithm that, given a n-node graph G and an integer $k \geq 3$, either returns an induced cycle of length at least $k+1$ in G or computes a tree-decomposition of G with interesting structural properties. First, we need some definitions.

A *tree-decomposition* of a graph $G = (V,E)$ is a pair $(\{X_i | i \in I\}, T = (I,M))$, where T is a tree and $\{X_i | i \in I\}$ is a family of subsets, called bags, of vertices of G such that (1) $V = \cup_{i \in I} X_i$; (2) $\forall \{uv\} \in E$ there is $i \in I$ such that $u,v \in X_i$; and (3) $\forall v \in V$, $\{i \in I | v \in X_i\}$ induces a (connected) subtree of T. The *width* of a tree-decomposition is the size (minus 1) of its largest bag and its *ℓ-width* is the maximum diameter of the subgraphs induced by the bags. The *treewidth* denoted by $tw(G)$, resp., *tree-length* denoted by $tl(G)$, of a graph G is the minimum width, resp., ℓ-width, over all possible tree-decompositions of G [**?**, **?**].

Let $k \geq 2$. A k-caterpillar is a graph that has a dominating set, called *backbone*, which induces a chordless path of order at most $k-1$. That is, any vertex of a k-caterpillar either belongs to the backbone or is adjacent to a vertex of the backbone. A tree-decomposition is said to be *k-good* if each of its bags induces a k-caterpillar.

Theorem 3. *There is a $O(m^2)$-algorithm that takes a m-edge graph G and an integer $k \geq 3$ as inputs and:*

- *either returns an induced cycle of length at least $k+1$;*
- *or returns a k-good tree-decomposition of G;*

Proof. The proof is by induction on $|V(G)| = n$. We prove that either we find an induced cycle larger than k, or for any chordless path $P = \{v_1, \ldots, v_i\}$ with $i \leq k-1$, there is a k-good tree-decomposition for G with one bag containing $N_G[P]$. Obviously, it is true if $|V(G)| = 1$. Now we assume that it is true for any graph G with n' nodes, $1 \leq n' < n$, and we show it remains true for n-node graphs.

Let G be a connected n-node graph, $n > 1$. Let $P = \{v_1, \ldots, v_i\}$ be any chordless path with $i \leq k-1$ and let $N = N_G[P]$, $N_j = N_G[v_j]$ for $j = 1, \ldots, i$ and $G' = G \setminus N$. There are three cases to be considered:

Case 1. $G' = \emptyset$. In this case, we have $G = N$. The desired tree-decomposition consists of one node, corresponding to the bag N.

Case 2. G' is disconnected. Let $C_1, \ldots, C_r$, $r \geq 2$, be the connected components of G' For any $j \leq r$, let G_j be the graph induced by $C_j \cup N$. Note that any induced cycle in G_j, $j \leq r$, is an induced cycle in G. By the induction hypothesis, either there is an induced cycle $\mathcal{C}$ larger than k in G_j, then $\mathcal{C}$ is also an induced cycle larger than k in G, or our algorithm computes a k-good tree-decomposition TD_j of G_j with one bag X_j containing N. To obtain the k-good tree-decomposition of G, we combine the TD_j's, $j \leq r$, by adding a bag $X = N$ adjacent to all the bags X_j for $j = 1, \ldots, r$. It is easy to see that this tree-decomposition satisfies our requirements.

Case 3. G' is connected. We consider the order of the path $P = \{v_1, \ldots, v_i\}$. In the following proof, first we prove that if the order of path P, $i = k-1$, then we can find either an induced cycle larger than k or the required tree-decomposition for G. Subsequently, we prove it is also true for path with length $i < k-1$ by reversed induction on i. More precisely, if $i < k-1$, either we find directly the desired cycle or tree-decomposition, or we show that there exists a vertex v_{i+1} such that $P' = P \cup \{v_{i+1}\}$ is a chordless path with order $i+1$. By reverse induction on i we can find either an induced cycle larger than k or a k-good tree-decomposition of G with one bag containing $N_G[P'] \supseteq N_G[P]$.

(a) If $i = k-1$, then we consider the following two cases.

- Assume first that there is $u \in N_G(P) \cup \{v_1, v_i\}$ (in particular, $u \notin P \setminus \{v_1, v_i\}$) such that $N_G(u) \subseteq N_G[P \setminus \{u\}]$. Let $\tilde{G} = G \setminus u$. Then $\tilde{G}$ is a graph with $n' = n-1$ vertices. By the induction hypothesis on $n' < n$, the algorithm either finds an induced cycle larger than k in $\tilde{G}$, then it is also the one in G; Otherwise our algorithm computes a k-good tree-decomposition $\widetilde{TD}$ of $\tilde{G}$ with one bag $\tilde{X}$ containing $N_{\tilde{G}}[P \setminus \{u\}]$. To obtain the required tree-decomposition of G, we just add vertex u into the bag $\tilde{X}$. The tree-decomposition is still k-good.
- Otherwise, there exist two distinct vertices $v_0 \in N_G(v_1)$ and $v_{i+1} \in N_G(v_i)$ and there are vertices $u_1, u_2 \in V(G')$ (possibly $u_1 = u_2$) such that $\{v_0, u_1\} \in E(G)$ and $\{v_{i+1}, u_2\} \in E(G)$. If $\{v_0, v_{i+1}\} \in E(G)$, $P \cup \{v_0, v_{i+1}\}$ is an induced cycle with $k+1$ vertices. Otherwise, let Q be a shortest path between u_1 and u_2 in G' (Q exists since G' is connected). So $P \cup \{v_{i+1}, u_2\} \cup Q \cup \{u_1, v_0\}$ is an induced cycle with at least $k+1$ vertices in G.

(b) If $i < k-1$, we proceed by reverse induction on i. Namely, assume that, for any chordless path Q with $i+1$ vertices, our algorithm either finds an induced cycle larger than k in G or computes a k-good tree-decomposition of G with one bag containing $N[Q]$. Note that the initialization of the induction holds for $i = k-1$ as described in case (b). We show it still holds for a chordless path with i vertices. We consider the following two cases.

- Either there is $u \in N_G(P) \cup \{v_1, v_i\}$ (in particular, $u \notin P \setminus \{v_1, v_i\}$) such that $N_G(u) \subseteq N_G[P \setminus \{u\}]$. That is, we are in the same case as the first item of (a). We proceed as above and the result holds by induction on n.
- Or there is $w \in N_G(v_1) \cup N_G(v_i) \setminus P$ such that $P \cup \{w\}$ is chordless (i.e., w is a neighbor of v_1 or v_i but not both). Therefore, we apply the induction hypothesis (on i) on $P' = P \cup \{w\}$. By the assumption on i, either our algorithm returns an induced cycle larger than k or it computes a k-good tree-decomposition of G with one bag containing $N_G[P'] \supseteq N_G[P]$.

To conclude, we describe the algorithm and study its complexity. Let G be a m-edge n-node graph with maximum degree Δ. Roughly, the algorithm proceeds by steps. At each step, one vertex is considered and the step takes $O(m)$ time. We prove that at each step (but the initial step), at least one edge will be *considered* and that all edges are considered at most once. This implies a time-complexity of $O(m^2)$ for the algorithm.

The algorithm starts from an arbitrary vertex $v \in V(G)$ and computes the connected components $C_1, \cdots, C_j$ of $G \setminus N[v]$ ($j \geq 1$) in time $O(m)$. We start with the k-good tree-decomposition for the induced graph of $N[v]$ in G that consists of a bag $B = N[v]$ adjacent to, for any $i \leq j$, each bag $B_i = \{v\} \cup \{w \in N(v) \ : \ N(w) \cap C_i \neq \emptyset\}$. This takes time $O(m)$.

Now, at some step of the strategy, assume that we have built a k-good tree-decomposition $(T, \mathcal{X})$ of a connected subgraph G_0 of G. Let $C_1, \cdots, C_j$ ($j \leq 1$) be the connected components of $G \setminus G_0$, and, for any $i \leq j$, let S_i be the set of the vertices of G_0 that are adjacent to some vertex of C_i. Assume finally that, for any $i \leq j$, there is a leaf bag $B_i \supset S_i$ of $(T, \mathcal{X})$ where $P_i = B_i \setminus S_i$ is a chordless path dominating B_i.

For any $e \in E(G)$, we say that $e = \{x, y\}$ is *alive* if there is $i \leq j$ such that $x \in S_i \cup C_i$ and $y \in C_i$. Note that, if an edge is alive, such an integer i is unique. An edge that is not alive is said *dead*. Note also that, after the initial step, all edges in the bag B are dead and other edges are alive.

The next step consists of the following. Choose any $i \leq j$ and let w be any vertex of S_i such that $Q = P_i \cup \{w\}$ is a chodless path. Note that by definition of S_i, there is at least one edge from w to C_i and that such an edge is alive before this step. We add the bag $B' = Q \cup B_i \cup (N(w) \cap C_i)$ adjacent to B_i. If Q is larger than k, by the above proof, the algorithm finds a large cycle. Otherwise, the connected components $C'_1, \cdots, C'_r$ of $C_i \cup B_i \setminus B'$ are computed in time $O(m)$. Let S'_h, $h \leq r$, be the subset of the vertices of S_i that are adjacent to some vertex in C'_h, and let Q_h be the smallest subpath of Q dominating S'_h. Computing the sets $S'_1, \cdots, S'_r$ only requires a time $O(m)$ since we have only to check the edges in B'. For any $h \leq r$, add a bag $B'_h = Q_h \cup S'_h$ adjacent to B'.

It is easy to check that this algorithm follows the above proof and that it eventually computes the desired tree-decomposition or returns a large cycle.

To conclude, it is easy to check that the set of edges alive after one step is contained in the set of edges alive before this step, and that, at each step at least one edge (the one(s) from w to S_i) become dead. Therefore, at each step, the number of alive edges strictly decreases and the algorithm terminates when there are no more. Since each step takes time $O(m)$ and there are at most m steps, the result follows. □

From the above theorem, it is easy to get the following corollaries.

Theorem 4. *Let G be a graph that admits a k-good tree-decomposition. Then $tw(G) \leq (k-1)(\Delta-1)+2$ where Δ is its maximum degree, and $tl(G) \leq k$.*

Proof. It directly follows the fact that, in a k-good tree-decomposition, each bag has a dominating path with $< k$ vertices. □

Theorem 5. *Any graph G that admits a k-good tree-decomposition has hyperbolicity at most $\lfloor \frac{3}{2}k \rfloor$.*

Proof. Let $G = (V, E)$ be a graph that admits a k-good tree-decomposition $(\{X_i | i \in I\}, T = (I, M))$. Let T be rooted at bag X_0, $0 \in I$. For any $u, v \in V$, let us denote the distance between u and v by $d(u, v)$ in G. By definition of a k-good decomposition, for any $i \in I$ and for any $u, v \in X_i$, $d(u, v) \leq k$.

Let $x, y, z \in V$ and let P_1, P_2, P_3 be any three shortest paths in G between x and y, y and z, x and z respectively. Let $u \in V(P_1)$. To prove the Theorem, we show that there is $v \in V(P_2) \cup V(P_3)$ such that $d(u, v) \leq \lfloor \frac{3}{2}k \rfloor$.

First, let us assume that there is $i \in I$ such that $u \in X_i$ and there is $v \in (V(P_2) \cup V(P_3)) \cap X_i \neq \emptyset$. In that case, $d(u, v) \leq k$ and the result holds.

Otherwise, let T_u be the subtree of T induced by $\{i \in I \ : \ u \in X_i\}$. Similarly, let T_x be the subtree of T induced by $\{i \in I \ : \ x \in X_i\}$ and T_y be the subtree of T induced by $\{i \in I \ : \ y \in X_i\}$. Let P be the path in T between T_x and T_y. Note that P may be empty if $V(T_x) \cap V(T_y) \neq \emptyset$. Let $j \in V(T_x) \cup V(T_y) \cup V(P)$ that is closest to T_u in T. Note that $j \notin V(T_u)$ because otherwise we would be in the first case. Note also that either X_j is a separator between x and u or $x \in X_j$, and either X_j is a separator between y and u or $y \in X_j$. Let P_{xu} and P_{uy} be the subpaths of P_1 from x to u and from u to y respectively. By remark above, there exist vertices $w \in V(P_{xu}) \cap X_j$ and $t \in V(P_{uy}) \cap X_j$. Possibly, $w = t$. Then $d(w, u) + d(u, t) = d(w, t)$ because P_1 is a shortest path, therefore, $d(w, u) + d(u, t) = d(w, t) \leq k$. So there is $\ell \in X_j$ with $d(u, \ell) \leq \lfloor \frac{k}{2} \rfloor$.

Finally, let us show that there is $h \in V(P_2 \cup P_3) \cap X_j$. If $x \in X_j$ or $y \in X_j$, it is obvious. Otherwise, this is because X_j separates x and y in G, and therefore, z cannot be in both the component of $G \setminus X_j$ containing x and of the one containing y, therefore, one of the paths P_2 or P_3 should path trough X_j.

To conclude, $d(u,h) \leq d(u,\ell) + d(\ell,h) \leq \lfloor \frac{k}{2} \rfloor + k \leq \lfloor \frac{3}{2}k \rfloor$. □

Corollary 1. *Any k-chordal graph G with maximum degree Δ has treewidth at most $(k-1)(\Delta-1)+2$, tree-length at most k and hyperbolicity at most $\lfloor \frac{3}{2}k \rfloor$.*

Proof. By definition of k-chordal graph and Theorem 3, any k-chordal graph admits a k-good tree-decomposition. The result follows Theorems 4 and 5. □

Corollary 2. *There is an algorithm that, given a m-edge graph G and $k \geq 3$, states that either G has chordality at least $k+1$ or G has hyperbolicity at most $\lfloor \frac{3}{2}k \rfloor$, in time $O(m^2)$.*

4 Application of k-good tree-decompositions for routing

In this section, we propose a compact routing scheme for any n-node graph G that admit a k-good tree-decomposition (this includes k-chordal graphs). Δ denotes the maximum degree of G and, for any $v \in V(G)$, d_v is its degree.

4.1 Model and performance of the routing scheme

We propose a *labelled* routing scheme which means that we are allowed to give one identifier, $name(v)$, of $O(\log n)$ bits to any vertex v of G. Moreover, following [**?**], we consider the *designer-port* model, which allows us to give a label of $O(\log d_v)$ bits to any edge of G incident to $v \in V(G)$. More precisely, in our case, for any $e \in E(G)$ and any $v \in V(G)$ incident to e, e receives one port-number $q_{e,v} \in \{1, \cdots, d_v\}$ plus one bit. Finally, to any node $v \in V(G)$, we assign a routing table, denoted by $Table(v)$, where are stored local information of size $O(\max\{k \cdot \log\Delta, \log n\})$ bits. Any message has a *header* that contains the address $name(t)$ of the destination t, three modifiable integers $pos \in \{1, \cdots, k\}, cnt_d, cnt'_d \in \{-1, \cdots, \Delta+1\}$, one bit *start* and some memory *path* of size $O(k \cdot \log\Delta)$ bits.

Following our routing scheme, a message at some node v uses its header, $name(v)$, $Table(v)$ and the port-numbers of the edges incident to v to compute its new header and to choose an edge $e = \{v,u\}$ along which is goes. Arriving at u, the message also knows by which edge it arrived. The length of the path followed by a message from a source $s \in V(G)$ to a destination $t \in V(G)$, using the routing scheme, is denoted by $|P(s,t)|$, and the *stretch* of the scheme is $\max_{s,t \in V(G)} |P(s,t)| - d(s,t)$ where $d(s,t)$ is the distance between s and t in G.

To design our routing schem, we combine the compact routing scheme in trees of [**?**] together with the k-good tree-decomposition. Roughly, the scheme consists in following the paths in a BFS-tree F of G, using the scheme in [**?**], and uses one bag of the tree-decomposition as a short-cut between two branches of F. Intuitively, if the source s and the destination d are "far appart", then there is a bag X of the tree-decomposition that separates s and d in G. The message follows the path in F to the root of F until it reaches X, then an exhaustive search is done in X until the message finds an ancestor y of d, and finally it follows the path from y to d in F using the scheme of [**?**]. The next theorem summarize the performances of our routing scheme.

Theorem 6. *For any n-node m-edge graph G with maximum degree Δ and admitting a k-good tree-decomposition, there is a labelled routing scheme $\mathcal{R}$ with the following properties. $\mathcal{R}$ uses addresses of size $O(\log n)$ bits, port-numbers of size $O(\log\Delta)$ bits and routing tables of size $O(\max\{k \cdot \log\Delta, \log n\})$ bits. The routing tables, addresses and port-numbers can be computed in time $O(m^2)$. Except the address of the destination (not modifiable), the header of a message contains $O(k \cdot \log\Delta)$ modifiable bits. The header and next hop is computed in time $O(1)$ at each step of the routing. Finally, the additive stretch is $\leq 2k(\lceil \log\Delta \rceil + \frac{5}{2}) - 3$.*

The remaining part of this Section is devoted to the proof of Theorem 6.

4.2 Data structures

4.2.1 Routing in trees [?].

Since, we use the shortest path routing scheme proposed in [**?**] for trees, we start by recalling some of the data structures they use. Let T be a tree rooted in $r \in V(T)$. For any $v \in V(T)$, let T_v be the subtree of T rooted in v and let $w_T(v) = |V(T_v)|$ be the *weight* of v. Consider a Depth-First-Search (DFS) traversal of T, starting from r, and guided by the weight of the vertices, i.e., at each vertex, the DFS numbering visits first the largest subtree, then the second largest subtree, and so on. For any $v \in V(T)$, let $Id_T(v) \in \{1, \cdots, n\}$ be the rank of v in the DFS. It is important to note that, for any $u, v \in V(T)$, $v \in V(T_u)$ if and only if $Id_T(u) \leq Id_T(v) \leq Id_T(u) + w_T(u) - 1$.

For any $v \in V(T)$ and any e incident to v, the edge e receives a *port-number* $p_T(e,v)$ at v as follows. $p_T(e,v) = 0$ if $v \neq r$ and e leads to the parent of v in T, i.e., e is the first edge of the path from v to r. Otherwise, let $u_1, \cdots, u_d$ be the children of v ($d = d_v$ if $v = r$ and $d = d_v - 1$ otherwise) ordered by the size of their weight, i.e., such that $w_T(u_1) \geq \cdots \geq w_T(u_d)$. Then, let $p_T(\{u_i, v\}, v) = i$, for any $i \leq d$.

Finally, each vertex $v \in V(T)$ is assigned a routing table $RT_T(v)$ and an address $\ell_T(v)$ of size $O(\log n)$ bits allowing a shortest path routing in trees (see details in [**?**]).

4.2.2 Our data structures.

Let G be a graph with the k-good tree-decomposition $(T = (I, M), \{X_i | i \in I\})$. Let $r \in V(G)$. Let F be a Breadth-First-Search(BFS) tree of G rooted at r. Let T be rooted in $b \in I$ such that $r \in X_b$.

We use (some of) the data structures of [**?**] for both trees F and T. More precisely, for any $v \in V(G)$, let $Id_F(v), w_F(v), \ell_F(v)$ and $RT_F(v)$ defined as above for the BFS-tree F. Moreover, for any edge $e \in E(G)$ incident to v, let $p_{e,v} = -1$ if $e \notin E(F)$ and $p_{e,v} = p_F(e,v)$ is defined as above if $e \in E(F)$. These structures will be used to route in F.

For any $i \in I$, let $Id_T(i)$ and $w_T(i)$ be defined as above for the tree T. For any $v \in V(G)$, let $B_v \in I$ be the bag of T containing v, i.e., $v \in X_{B_v}$, and that is closest to the root b of T. To simplify the notations, we set $Id_T(v) = Id_T(B_v)$ and $w_T(v) = w_T(B_v)$. These structures will be used to decide "where" we are in the tree-decomposition when the message reaches $v \in V(G)$.

For any $v \in V(G)$, let $\{u_1, \cdots, u_{d_v}\} = N(v)$ be its neighborhood ordered such that $Id_F(u_1) < Id_F(u_2) < \cdots < Id_F(u_{d_v})$. We give a second port-number to any edge. More precisely, for any $e = \{v, u_i\}$, $i \leq d_v$, let $q_{e,v} = i$. This ordering will allow to decide quickly, in time $O(\log \Delta)$ by binary search, whether a vertex has some particular vertex in its neighbors.

Finally, for any $i \in I$, let $P_i = (v_1, \cdots, v_\ell)$ be the backbone of B_i with $\ell \leq k-1$ (recall we consider a k-good tree decomposition). Let $\{e_1, \cdots, e_{\ell-1}\}$ be the set of edges of P_i in order. We set $Backbone_i = \{q_{e_1,v_1}, q_{e_1,v_2}, q_{e_2,v_2}, \cdots, , q_{e_{\ell-1},v_{\ell-1}}, q_{e_{\ell-1},v_\ell}\}$. For any v such that $Id_T(v) = i \in I$, if $v = v_j \in P_i$, then $back(v) = (\emptyset, j)$ and if $v \notin P_i$, let $back(v) = (q_{e,v}, j)$ where $e = \{v, v_j\}$ and v_j ($j \leq \ell$) is the neighbor of v in P_i with j minimum. This information will be used to cross a bag (using its backbone) of the tree-decomposition.

We are now ready to define the address $name(v)$ and the routing table $Table(v)$ of any $v \in V(G)$.

Now for every $v \in V(G)$, we define the address $name(v) = \langle \ell_F(v), Id_T(v) \rangle$. Note that, in particular, $\ell_F(v)$ contains $Id_F(v)$. We also define the routing table of v as $Table(v) = \langle RT_F(v), w_T(v), Backbone(v), back(v) \rangle$.

Next table summarizes all these data structures.

	storage	description
address	$\ell_F(v)$	the address of v in tree F [**?**]
$name(v)$	$id_T(v)$	the identifier of the highest bag B_v containing v in T
routing table	$RT_F(v)$	the routing table used of v for routing in F [**?**]
	$w_T(v)$	the weight of the subtree of T rooted in B_v
$Table(v)$	$Backbone(v)$	information to navigate in the backbone of B_v
	$back(v)$	information to reach the backbone of B_v from v

Clearly, $name(v)$ has size $O(\log n)$ bits and $Table(v)$ has size $O(\max\{k\cdot\log\Delta,\log n\})$ bits.

Moreover, any edge e incident to v receives two port-numbers $p_{e,v}$ and $q_{e,v}$ of size $O(\log\Delta)$ bits. Actually, we can replace $p_{e,v}$ by one bit $b_{e,v}$ that equals 1 if $e\in E(F)$ and $b_{e,v}=0$ otherwise. Indeed, if $u_1,\cdots,u_{d_v}$ are the neighbors of v ordered by increasing Id_F, $(p_{e'_1,v},\cdots,p_{e'_{d'},v})$ is the subsequence of $(q_{e_1,v},\cdots,q_{e_{d_v},v})$ (where $e_i=\{v,u_i\}$, $i\le d_v$) obtained by keeping only the edges e' with $b_{e',v}=1$. Therefore, it is possible to compute $p_{e,v}$ using the port-numbers q and the bits b. In the sequels, we use $p_{e,v}$ and $q_{e,v}$ for an easier description.

4.3 Routing algorithm in k-good tree-decomposable graphs

Let us consider a message that must be sent to some destination $t\in V(G)$. Initially, the header of the message contains $name(t)$, the three counters pos, cnt_d, $cnt'_d=-1$, the bit $start=0$ and the memory $path=\emptyset$.

Let $v\in V(G)$ be the current node where the message stands. First, using $Id_F(t)$ in $\ell_F(t)\in name(t)$, $Id_F(v)$ in $name(v)$ and $w_F(v)$ in $RT_F(v)\in Table(v)$, it is possible to decide in constant time if v is an ancestor of t in F (which is the case iff $Id_F(v)\le Id_F(t)\le Id_F(v)+w_F(v)-1$). Similarly, using $Id_T(t)$ in $name(t)$, $Id_T(v)$ in $name(v)$ and $w_T(v)$ in $Table(v)$, it is possible to decide if the highest bag B_v containing v is an ancestor of B_t in T.

There are several cases to be considered.

- If v is the ancestor of t in F then, using $RT_F(v)$ and the scheme in trees of [**?**], the message follows the edge leading to the child w of v in F on the shortest path in F toward t.

  Since w is still an ancestor of t (or $w=t$) and F is a BFS-tree, the message will go on in that manner until it reaches t via a shortest path from v to t in G.

- Else, if $path=\emptyset$, then

  - if neither B_v is an ancestor of B_t in T nor $B_t=B_v$, then the message follows the edge leading to the parent of v in F, i.e., the edge with port-number $p_{e,v}=0$.

    Note that, going on that way, the message will eventually reach a node w such that either w is an ancestor of t in F or B_w is an ancestor of B_t in T since, by this rule, the message goes toward the root r of F (via a shortest path) and because B_r is the ancestor of any bag in T.

  - Else, $Backbone(v)$ is first copied in $path$ in the header of the message.

    The goal of it is to explore the bag B_v using its backbone $P=\{v_1,\cdots,v_\ell\}$ ($\ell<k$), until the message finds an ancestor of t. Clearly, an ancestor of t belongs to B_v since either $B_v=B_t$, or B_v is an ancestor of B_t and therefore, B_v is a separator between r and t, or $r\in B_v$.

    The first phase of the exploration of B_v is to reach its end v_1. To do so, the message uses $back(v)=(q,j)\in Table(v)$. First, pos is set to j. Then, if $v\notin P$, i.e., $q\neq\emptyset$, the message uses the port-number q. Doing so, it reaches the node $v_j\in P$. Finally, if $j=1$, $start$ is set to 1, $cnt_d=0$ and $cnt'_d=d_{v_1}+1$. This last operation means that the message can start the exploration of B_v from v_1. Note that, from now, $path\neq\emptyset$ and $pos\neq-1$.

- Else, if $start=0$, then the message is at $v=v_j\in P$ where $P=\{v_1,\cdots,v_\ell\}$ is the backbone of some bag of the tree-decomposition. Moreover, in the field $path$ of the header, there are the port-numbers allowing

to follow P. The message follows the corresponding port-number to reach v_{j-1}. $pos = j-1$. If $j-1=1$, start is set to 1, $cnt_d = 0$ and $cnt'_d = d_{v_1}+1$.

- Else, if $start = 1$, then the exploration of a bag containing an ancestor of t has begun. The key point is that any ancestor w of t in F is such that $Id_F(w) \leq Id_F(t) \leq Id_F(w)+w_F(w)-1$. Using this property, in each vertex v_j of the backbone $P = \{v_1, \cdots, v_\ell\}$, the message explores the neighbors of v_j by binary search.
  - If $cnt_d = cnt'_d - 1$, the neighborhood of the current node $v = v_j$, where $j = pos$, has already been explored and no ancestor of t has been found. In that case, using $path$, the message goes to v_{j+1} the next vertex in the backbone. $pos = j+1$.
  - Otherwise, let $r = \lfloor \frac{cnt'_d + cnt_d}{2} \rfloor$. The message takes port-number r in v (considering the port-numbers q, i.e., considering all neighbors of v) toward vertex w. If w is an ancestor of t, we go to the first case of the algorithm. Otherwise, the message goes back to $v = v_j$. This is possible because our model allows a message to remember the port-number by which it arrived to a node. Moreover, if $Id_F(w) < Id_F(t)+w_F(w)-1$, then cnt_d is set to r and cnt'_d is set to r otherwise.

The fact that the message eventually reaches its destination follows the above description. Moreover, the computation of the next hop and the modification of the header clearly take time $O(1)$.

4.4 Performance of our routing scheme

In this subsection, we give an upper bound on the stretch of the routing scheme described in previous section.

Lemma 1. *Our routing scheme has stretch* $\leq 2k(\lceil \log \Delta \rceil + \frac{5}{2}) - 5$.

Proof. Let s be the source and t be the destination. Recall the main idea of the algorithm: We route along the path from s to r in tree F until we arrive a vertex x, whose bag B_x is an ancestor of t's bag B_t in tree T. Then applying binary search algorithm, we search in the bag B_x for a vertex y, which is an ancestor of t in tree F. In the end, we route from y to t in tree F.

Because F is a BFS tree and x is an ancestor of s in F, the length of the path followed by the message from s to x is $d(s,x)$, the distance between s and x in G. Similarly, because y is an ancestor of t in F, the length of the path followed by the message from y to t is $d(y,t)$. Let $track(x,y)$ be the length of the path followed by the message in the B_x from x to y. Therefore, the length of the path followed by the message from s to t is $d(s,x)+track(x,y)+d(y,d)$.

From the binary search algorithm, for any node of the backbone, the message visits at most $\lceil \log \Delta \rceil$ neighbors and this causes a path of length $2\lceil \log \Delta \rceil$. There are at most $k-1$ nodes on the backbone of the bag B_x, but the worst case occurs when x and y are neighbors of the last end of the backbone. After arriving at x, the message goes to the first end of the backbone, then it visits $\lceil \log \Delta \rceil$ neighbors of all the $\leq k-1$ nodes of the backbone and y is the last vertex visited. Therefore, $track(u,a) \leq 2k(\lceil \log \Delta \rceil + 1) - 5$. Then it is sufficient to prove $d(s,x)+d(y,t) \leq d(s,t)+3k$.

If B_s is an ancestor of B_t, then $x = s$ and $d(s,x) = 0$. Moreover, if $B_t = B_x$, $d(y,t) = 0$. Otherwise, let B be the nearest common ancestor of B_s an B_t in the tree-decomposition T. Let Q be a shortest path between s and t. Because the set of vertices in B separates s from t in G, let x' be the first vertex of Q in B and let y' the last vertex of Q in B. Let $Q = Q_1 \cup Q_2 \cup Q_3$ where Q_1 is the subpath of Q from s to x', Q_2 is the subpath of Q from x' to y' and Q_3 is the subpath of Q from y' to t. Note that because each bag has diameter at most k, $d(x',y') \leq k$.

We first show that $x \in B$. If $B_x = B$, it is trivially the case. Let P_x be the path followed from s to x. Since B_x is an ancestor of B, B separates s from x. Therefore, $V(P_x) \cap B \neq \emptyset$. Let h be the first vertex of P_x in B. Since $h \in B$, the highest bag containing h is an common ancestor of B_s and B_t. Therefore, when arriving at h, the message must explore B_h. Hence, $h = x \in B$.

Finally, since $x \in B$, $d(x,x') \leq k$. Moreover, $y \in B_x$ therefore $d(y,x) \leq k$. Thus, $d(y,y') \leq d(y,x) + d(x,x') + |Q_2| \leq 2k + |Q_2|$. Finally, $d(s,x) \leq d(s,x') + d(x',x) \leq k + |Q_1|$ and $d(y,t) \leq d(y,y') + d(y',t) \leq 2k + |Q_2| + |Q_3|$. Therefore, $d(s,x) + d(y,t) \leq |Q_1| + |Q_2| + |Q_3| + 3k \leq |Q| + 3k = d(s,t) + 3k$. □

5 Conclusion and Further Work

It would be interesting to reduce the $O(k \cdot \Delta)$ stretch due to the dichotomic search phase of our routing scheme. Another interesting topic concerns the computation of tree-decompositions not trying to minimize the size of the bag but imposing some specific algorithmically useful structure.

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399