Finding Tree–2–Spanners

S.O. Krumke J. Valenta

Report No. 131 December 1995

Department of Computer Science, University of Würzburg
Am Hubland, D–97074 Würzburg, Germany
Tel.: +49 931 888 5057, Fax: +49 931 888 4600
Email: {krumke,valenta}@informatik.uni-wuerzburg.de

Abstract

A t–spanner of a finite undirected graph $G = (V, E)$ is a subgraph $G' = (V, E')$ such that for each pair u, v of nodes $d_{G'}(u, v) \leq t \cdot d_G(u, v)$, where $d_{G'}(u, v)$ and $d_G(u, v)$ denote the shortest distances between u and v in G' and G respectively. The problem of finding a sparsest t–spanner, i.e. a t–spanner with the least number of edges, is well known to be $\mathcal{NP}$–hard even for the case that $t = 2$.

In this paper we present a polynomial time algorithm for deciding whether a graph G has a 2–spanner that is also a tree. Our algorithm is of constructive nature, i.e. it either outputs the edges of a tree–2–spanner or provides the information that no tree-2–spanner is contained in the graph. We also investigate the structure of tree–2–spanners.

Keywords: Algorithms, Spanner, Tree, Spanning–Tree

Finding Tree–2–Spanners

S. O. Krumke J. Valenta
Department of Computer Science
University of Würzburg
Am Hubland, 97074 Würzburg, Germany
Email: {krumke,valenta}@informatik.uni-wuerzburg.de

11th April 1996

Abstract

A t–spanner of a finite undirected graph $G = (V, E)$ is a subgraph $G' = (V, E')$ such that for each pair u, v of nodes $d_{G'}(u, v) \leq t \cdot d_G(u, v)$, where $d_{G'}(u, v)$ and $d_G(u, v)$ denote the shortest distances between u and v in G' and G respectively. The problem of finding a sparsest t–spanner, i.e. a t–spanner with the least number of edges, is well known to be $\mathcal{NP}$–hard even for the case that $t = 2$.

In this paper we present a polynomial time algorithm for deciding whether a graph G has a 2–spanner that is also a tree. Our algorithm is of constructive nature, i.e. it either outputs the edges of a tree–2–spanner or provides the information that no tree-2–spanner is contained in the graph. We also investigate the structure of tree–2–spanners.

Keywords: Algorithms, Spanner, Tree, Spanning–Tree

1 Preliminaries

Throughout the presentation we will assume that $G = (V, E)$ is an undirected finite and connected[1] graph. If $v, w \in V$ are vertices of G, we denote by $d_G(v, w)$ the length of a shortest path between v and w. It should be noted here that in this paper we consider *unweighted* graphs, i.e. the length of a path equals the number of edges in the path.

Definition 1.1 Let $G = (V, E)$ be an undirected graph. A subgraph $G' = (V, E')$ of G having the same node set as G is called a t–*spanner* of G, if for each pair v, w of nodes we have $d_{G'}(v, w) \leq t \cdot d_G(v, w)$.

In this paper we focus on 2–spanners of graphs and more precisely on so called *tree–2–spanners:*

Definition 1.2 A 2–spanner that is a tree is called a tree–2–spanner or T2S for short. The *Tree–2–Spanner–Problem* is to decide, whether a graph G contains a T2S.

[1] If G is not connected, then the problem of finding a 2–spanner of G reduces to finding 2–spanners of G in each connected component.

Figure 1 shows a graph G and a corresponding T2S, which is indicated by the thick edges.

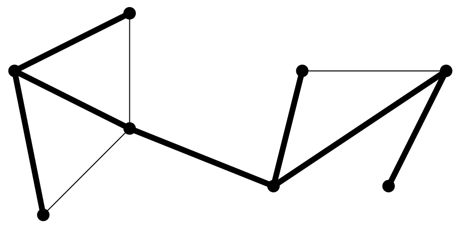

Figure 1: A graph having a T2S.

The problem of finding a 2–spanner of minimum sparsity, i.e. a 2–spanner with the least number of edges, is known to be $\mathcal{NP}$–hard (cf. [PS89]). An approximation algorithm for this problem with a performance guarantee of $O(\log \frac{|E|}{|V|})$ is presented in [KP94].

In this paper we show that the Tree–2–Spanner–Problem can be solved in polynomial time. The algorithm presented here is in fact a constructive algorithm, namely if G contains a T2S, the algorithm will output a set $E_T \subseteq E$ of edges that form a T2S. Observe that for those graphs that have a T2S, the question how sparse a 2–spanner could possibly be, can thus be answered in polynomial time, since a spanning tree is clearly the sparsest spanner.

Given a subset $E' \subseteq E$ of the edges of G, we say that an edge $e = (u, v) \notin E'$ is *covered by the edges in* E', if $(z, u), (z, v) \in E'$ for some node $z \in V$. It is easy to see that a subgraph $G' = (V, E')$ is a 2–spanner of G if and only if each edge $e \in E$ either belongs to E' or is covered by the edges in E'.

The rest of the notations used in this paper is fairly standard. Given a subset $V' \subseteq V$ of the vertices of $G = (V, E)$, we use $G[V']$ to denote the subgraph induced by V'. A vertex $v \in V$ is a *neighbor* of $w \in V$ if $(v, w) \in E$. We denote $N(v, G)$ to denote the set of neighbors of v in G. Given a subset E' of the edges of G, we use $N(v, E')$ to denote the set of of vertices adjacent to v in the subgraph $G' = (V, E')$, that is $N(v, E') := \{w \in V : (v, w) \in E'\}$. A subgraph $H = (V_H, E_H)$ of G is called a *claw*, if $E_H = \{(c, v) : v \in V_H\}$ for some $c \in V_H$. The vertex c is then called the *center* of the claw.

A subset $C \subseteq V$ is a *clique*, if the induced subgraph $G[C]$ is complete, i.e. if $(v, w) \in E$ for all $v, w \in C$. A clique C such that there is no other clique C' in G strictly containing C is said to be a *maximal clique*. Notice that maximal cliques can be found easily in polynomial time, by starting with the endpoints of an edge and then adding nodes until any further addition of nodes would destroy the clique property. This is in contrast to the problem of finding a *maximum* clique, i.e. a clique of maximum cardinality, which is well known to be $\mathcal{NP}$–hard and, moreover also known to be very hard to approximate (cf. [BS94]).

The paper is organized as follows: In Section 2 we will prove some basic results about tree–2–spanners and derive some structural properties for them. In Section 3 we will then present the polynomial time algorithm for the tree–2–spanner problem and prove its correctness.

2 Properties of T2Ss

In this section we derive some basic properties of T2Ss of a graph G. Also, we will establish necessary conditions for a graph G such that G can contain a T2S.

What are conditions that prohibit the

existence of a T2S? First it is a trivial observation that any tree contains a T2S, namely itself. Now consider the graph G shown in Figure 2. It is easy to see that G can have no T2S, because one a possible T2S T can not include all the edges of the cycle (a, b, c, d, e). But then in T the distance between two of the nodes from $\{a, b, c, d, e\}$ will be at least three.

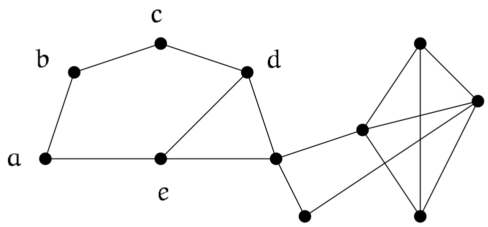

Figure 2: A graph that contains no T2S.

It seems that cycles play a crucial role for the existence of a T2S. In the last example (a, b, c, d, e) was a chordless cycle. At first glance one might conjecture that the *chordality* of a graph G is a necessary condition for G to contain a T2S. Recall that a graph G is *chordal* if it contains no chordless cycle of length greater than three. But it turns out that chordality and the existence of a T2S are not related, as Figure 3 illustrates.

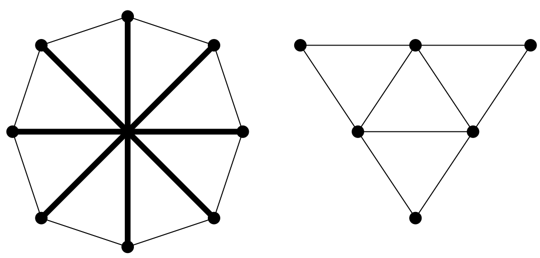

Figure 3: A non–chordal graph with a T2S and a chordal graph having no T2S.

In fact it is not the chordality of G that is necessary but rather a somewhat related property, which is stated in the following lemma:

Lemma 2.1 Let T be a T2S of $G = (V, E)$ and let $(v_1, \ldots, v_r)$ be a chordless cycle in G of length $r \geq 4$. Then there is a node $z \in V$ such that $(z, v_i) \in T$ for $i = 1, \ldots, r$.

Proof: T can not contain all the edges of the cycle, since T is a tree. Assume w.l.o.g. that $(v_1, v_2) \notin T$. Then (v_1, v_2) must be covered by two edges $(z, v_1), (z, v_2) \in T$ (see Figure 4).

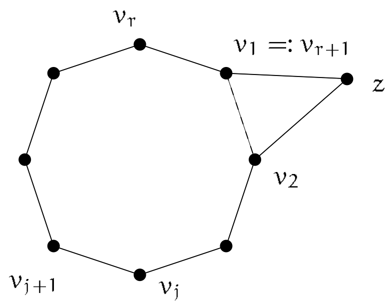

Figure 4: Proof of Lemma 2.1

Observe that z can not be contained in the cycle (and thus Figure 2.1 does not give a false impression), since otherwise the cycle would have a chord.

If $(z, v_i) \in T$ for $i = 1, \ldots, r$, then we have proved the claim. Otherwise, there is an edge in the cycle which is not covered by edges incident with z. Since T can not contain all of those edges, there is an edge (v_j, v_{j+1}) that is covered by $(z', v_j), (z', v_{j+1})$, where $z' \neq z$ (see Figure 5). It is now easy to see that T must contain a cycle, because each of the edges (v_i, v_{i+1}) with $2 \leq i \leq j-1$ and $j+1 \leq i \leq r$ is either in T or covered by a pair of edges $(z'', v_i), (z'', v_{i+1})$, where z'' is either different from z or from z'. $\rightarrow\leftarrow$ $\square$

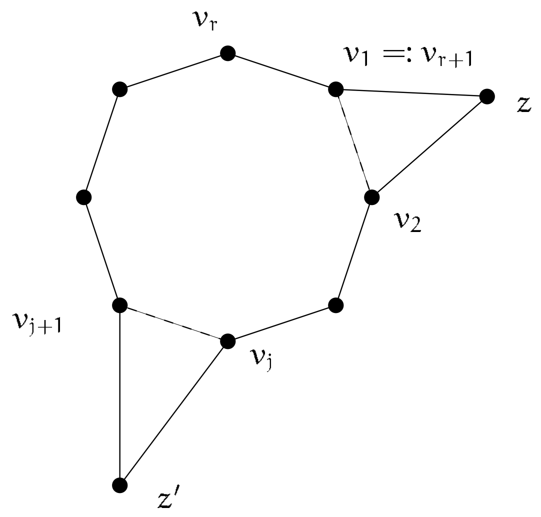

Figure 5: Proof of Lemma 2.1

We will now show that claws play an important role for a T2S:

Lemma 2.2 Let G be a graph and assume that T is a T2S of G. Then for each maximal clique C of G the T[C] is a claw.

Proof: Let C be a maximal clique. Since G is connected, $|C| \geq 2$. If $|C| = 2$, say $C = \{v, w\}$, then it follows that $e = (v, w)$ must be contained in T. If $|C| \geq 3$, then there are at least two vertices $v, w \in C$ such that $(v, w) \notin T$, because otherwise T would have cycles. Since (v, w) is not in T, there must be a vertex $c \in V$ dominating both v and w in T. We claim that c must be adjacent to ***all*** the vertices in C. This will prove that $c \in C$ (since C is maximal) and establish the claim.

Let $u \in C - \{v, w, c\}$ be arbitrary. Since c dominates both v and w and T is a tree, one of the edges (u, v) and (u, w) can not be contained in T. W.l.o.g. assume that $(u, v) \notin T$. Then again, (u, v) must be covered by a pair of edges $(u, z), (z, v)$. Clearly $z \neq w$.

If $z = c$, then we have shown that c is adjacent to u. Otherwise we see that also $(w, u) \notin T$ and that $d_T(w, u) \geq 3$ as a contradiction to the assumption that T is a T2S. □

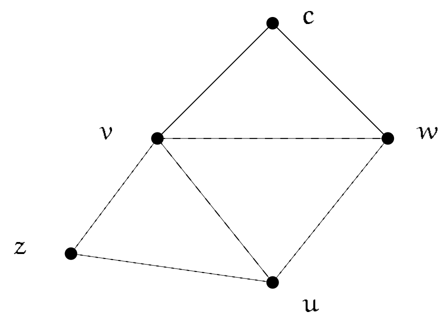

Figure 6: Proof of Lemma 2.2

Corollary 2.3 If C and C′ are maximal cliques in the graph G with $|C \cap C'| \geq 2$ and T is a T2S of G, then the center of the claw T[C] (the center of the claw T[C′]) is contained in $C \cap C'$. If $|C \cap C'| \geq 3$, then the centers of T[C] and T[C′] coincide.

Proof: By Lemma 2.2 T[C] and T[C′] are claws with centers c and c' respectively. Assume that $c \notin C \cap C'$. Let $x, y \in C \cap C'$. We know that $(c, x), (c, y) \in T$. If $x = c'$, then $(c, y) \in T$ and T contains a cycle (see Figure 7).

The same argumentation shows that y can not be the center of T[C′]. But then both x and y are adjacent to c' (see Figure 8) and no matter, whether $c' \in C \cap C'$ or not, T contains the cycle (c, x, c', y).

We have now seen that c and c' are both contained in $C \cap C'$. Assume now that $|C \cap C'| \geq 3$ and choose a node $v \in (C \cap C') - \{c, c'\}$. Assume that $c \neq c'$. Since c is the center of T[C] and $c', v \in C$, we can conclude that $(c', v), (c, c') \in T$.

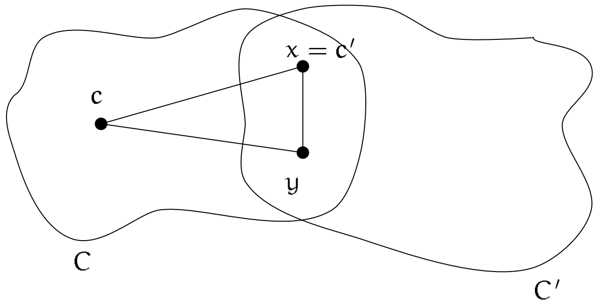

Figure 7: First case in the proof of Lemma 2.3.

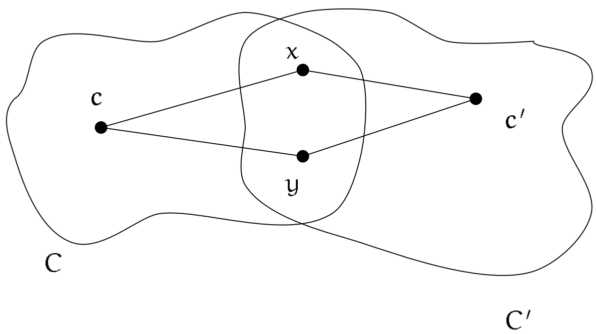

Figure 8: Second case in Lemma 2.3.

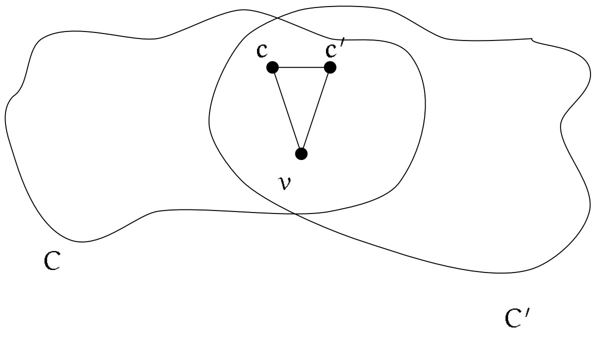

Figure 9: If $|C \cap C'| \geq 3$, then the centers of $T[C]$ and $T[C']$ coincide.

On the other hand c' is the center of $T[C']$ and $v \in C \cap C' \subset C'$. Thus $(c', v) \in T$. But then again, T contains a cycle, namely (c, c', v). □

The following lemma establishes basically the reverse of Lemma 2.2:

Lemma 2.4 Let $T = (V, E_T)$ be a cycle free subgraph of $G = (V, E)$ with the property that $T[C]$ is a claw for every maximal clique C of G. Then T is a T2S of G.

Proof: It suffices to show that T is a 2-spanner of G. To this end, let $(u, v) \in G$ be any edge not contained in T. Then $\{u, v\}$ is a clique and contained in a maximal clique C. By the assumption $T[C]$ is a claw and by the assumption contains the edges (c, u) and (c, v) for some $c \in C$. Thus (u, v) is covered by the edges in T. □

Although the following lemma will not be needed for the analysis of our algorithm in Section 3, it states yet another interesting propertiy of a T2S.

Lemma 2.5 Let C be a maximal clique in G and $c \in C$ be the center of $T[C]$. If there is a vertex $c' \in C$ that is adjacent to all vertices w with $(c, w) \in T$, then the graph T' obtained from T by deleting the edges (c, w) and inserting the edges (c', w) is a T2S of G.

Proof: It is easy to see that T' can contain no cycles. Thus it only remains to show that T' is a 2–spanner of G. Assume that (x, y) is an edge in G that is not covered by the edges of T'. Since T is a T2S of G, either $(x, y) \in T$ or it is covered by two other edges $(x, z), (y, z) \in T$.

In the first case, since we have deleted only edges adjacent to c, the edge is of the form (c, y). But then (c, y) is covered by the edges (c', c) and (c', y) from T'.

In the second case, again by the fact that we have only removed edges adjacent to c, we see that (x, y) must have been covered by (c, x) and (c, y) in T. Then both x and y are neighbors of c in T and T' contains the edges (c', x), (c', y), which together cover (x, y). $\square$

3 The Algorithm

The main algorithm is shown in Figure 10. It uses the procedure Claw–Spanner from Figure 11. We now establish a number of lemmas in order to prove the correctness of our algorithm. The first lemma is an easy consequence of Corollary 2.3:

Lemma 3.1 Let C be a maximal clique and consider a call to Claw–Spanner with argument C. Then as an invariant throughout the **while**–loop in steps 3 to 12, the center of $T[C]$ is contained in $Z(C)$.

Lemma 3.2 Let C' be a maximal clique that is added to $\mathcal{M}$ in the **while**–loop in steps 3 to 12 of Claw–Spanner. Then if $|C' \cap Z(C)| \geq 2$, the center of $T[C']$ is contained in $Z(C) \cap C'$. Moreover, in this case for each $\tilde{C} \in \mathcal{M}$ the center of $T[\tilde{C}]$ is contained in $Z(C) \cap C'$.

If $|C' \cap Z(C)| = 1$, then the center of $T[C]$ is the unique element in the intersection. If C' and $Z(C)$ have empty intersection, G does not have a T2S.

Proof: It is easy to see that $Z(C) = \bigcap_{\tilde{C} \in \mathcal{M}} \tilde{C}$. By induction we may assume that for each clique already in the collection $\mathcal{M}$ the center is contained in $Z(C)$.

If $|C' \cap Z(C)| \geq 2$, the maximal clique C' has at least two vertices in common with each clique in the collection $\mathcal{M}$. The claim now follows by the results of Corollary 2.3.

If $Z(C) \cap C' = \{c\}$ or $Z(C) \cap C' = \emptyset$, then Lemma 3.1 shows that c is the unique center of $T[C]$ or that G can not have a T2S respectively. $\square$

By the fact that $Z(C) = \bigcap_{\tilde{C} \in \mathcal{M}} \tilde{C}$, we note the following

```
procedure Tree–2–Spanner(G = (V, E))
  for all e ∈ E do
    m(e) := uncovered
  endfor
  while there is e = (u, v) ∈ E with m(e) = uncovered do
    Find a maximal clique C containing both endpoints u, v of e
    if |C| = 2 then E_T := E_T ∪ {e}; m(e) := covered
               else Claw–Spanner(C)
    endif
  endwhile
  if E_T contains cycles then output "G contains no T2S"
                         else output E_T
  endif
  end.
```

Figure 10: Main procedure for the tree-2-spanner algorithm.

Observation 3.3 Throughout the iteration of the **while**–loop in steps 3 to 12 of Claw–Spanner each node $c \in Z(C)$ is adjacent to all nodes from H in G.

Lemma 3.4 If in step 11 or step 15 the procedure Claw–Spanner outputs the information that G has no T2S, then in fact the graph G can not contain a T2S.

Proof: If the algorithm stops in step 11, the results of Lemma 3.2 show that the output information is correct.

Let w be a vertex with the property that $N(w) \cap C = \{v\}$, $N(w) \cap (H - C) \neq \emptyset$ and $N(w) \cap Z(C) = \emptyset$. Choose $y \in N(w) \cap (H - C)$. Then $y \in C'$ for some maximal clique in $\mathcal{M}$. By Lemma 3.2 the centers c and c' of C and C' respectively are both contained in $Z(C)$. No matter whether $c = c'$ or $c \neq c'$, a T2S T can not contain both edges (w, v) and (w, y), because in this case T would contain a cycle. But then, since w is not adjacent to any node from $Z(C)$ and in particular not to c or c' either $d_T(w, v) \geq 3$ or $d_T(w, y) \geq 3$, which contradicts the assumption that T is a T2S. □

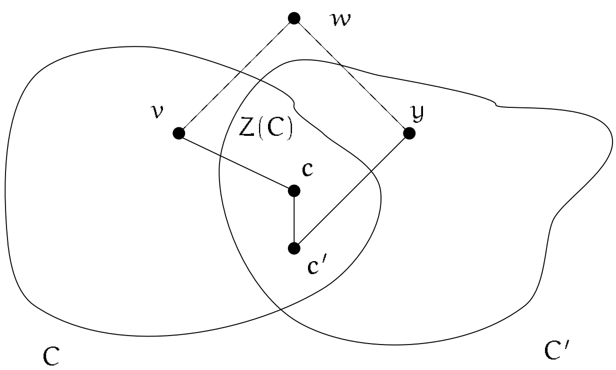

Figure 12: Proof of Lemma 3.4.

Lemma 3.5 If there is a vertex w satisfying the conditions of step 14 in Claw–Spanner, then for each $C' \in \mathcal{M}$ with $|N(w) \cap (C' - C)| \geq 1$ the node c satisfying $C \cap N(w) = \{c\}$ must be the center of $T[C']$.

Proof: By Lemma 3.2 we know that the center of $T[C']$ is contained in $Z(C)$. Let

procedure Claw–Spanner(C)
1 $Z(C) := C$; $\mathcal{M} := \{C\}$; $H := C$
2 *Comment:* Try to find the center of $T[C]$
3 **while** there is a node $w \in V - H$ s.t. $|N(w,G) \cap C| \geq 2$ **do**
4 Find a maximal Clique C' containing w and $N(w,G) \cap C$
5 $\mathcal{M} := \mathcal{M} \cup \{C'\}$; $H := H \cup C'$; $Z(C) := Z(C) \cap C'$
6 **if** $Z(C) = \{c\}$ for some $c \in C$ **then**
7 *Comment:* The center c has been found
8 $E_T := E_T \cup \{(c,v) : v \in C\}$
9 **goto** step 36
10 **endif**
11 **if** $Z(C) = \emptyset$ **then output** "G has no T2S"; **endif**
12 **endwhile**
13 *Comment:* Try to spot the centers of the cliques from $\mathcal{M}$
14 **while** there is a vertex w s.t. $|N(w,G) \cap C| = 1$ and $|N(w,G) \cap (H - C)| \geq 1$ **do**
15 **if** $N(w,G) \cap Z(C) = \emptyset$ **then output** "G has no T2S"; **endif**
16 **for all** $C' \in \mathcal{M}$ such that $|N(w,G) \cap (C' - C)| \geq 1$ **do**
17 Assuming that $N(w,G) \cap C = \{c\}$, set $E_T := E_T \cup \{(c,v) : v \in C'\}$
18 *Comment:* If $|Z(C)| \geq 3$ the center of C' is also the center of C
19 **if** $|Z(C)| \geq 3$ **then** $E_T := E_T \cup \{(c,v) : v \in H\}$; **goto** step 36; **endif**
20 **endwhile**
21 *Comment:* No further restrictions possible
22 **if** $|Z(C)| \geq 3$ **then**
23 Choose a node $c \in Z(C)$
24 Set $E_T := E_T \cup \{(c,v) : v \in H\}$
25 **else**
26 *Comment:* $|Z(C)| = 2$, since the case $|Z(C)| = 1$ is already handled in step 6
27 Let $Z(C) = \{c, c'\}$
28 $E_T := E_T \cup \{(c,c')\}$
29 **for each** $v \in H$ **do**
30 **if** some $y \in N(x,G) \cap (H - Z(C))$ of x or x itself is already adjacent to c in E_T
31 **then** $E_T := E_T \cup \{(c,x)\}$
32 **else** $E_T := E_T \cup \{(c',x)\}$
33 **endif**
34 **endfor**
35 **endif**
36 *Comment:* Mark newly covered and tree edges
37 Set $m(e) :=$ tree for all $e \in E_T$
38 Set $m(e) :=$ covered for all $(u,v) \in E - E_T$ such that $(x,u),(x,v) \in E_T$ for some $x \in V$
39 **return** E_T

Figure 11: Procedure handling the maximal cliques with the help of claws.

w be a vertex as described in step 14, $\{c\} = C \cap N(w)$. Also, let $C' \in \mathcal{M}$ contain a neighbor v of w which is not included in C. By Lemma 3.4 we know that $c \in Z(C)$. The three vertex set $\{c, w, v\}$ is a clique, which is contained in some maximal clique $\tilde{C}$ satisfying $\tilde{C} \cap Z(C) = \{c\}$ and $|\tilde{C} \cap C'| \geq 2$. By Corollary 2.3 and Lemma 3.2 the claim follows.□

Lemma 3.6 If G has a T2S, then as an invariant throughout the algorithm T2S(G) there is a T2S of G containing the edges in E_T.

Proof: It suffices to show that after each call to Claw–Spanner the claim of the lemma continues to hold. Let T be a T2S of G containing all the edges in E_T and consider a call to Claw–Spanner(C). By the results from Lemma 3.2, the edges added to E_T in step 8 must already belong to T. Also, Lemma 3.5 shows that the edges added to E_T in steps 14 to 20 must be included in *any* T2S of G. Hence we need only to worry about the edges that are added in steps 22 to 35.

To this end we distinguish between the two cases $|Z(C)| \geq 3$ and $|Z(C)| = 2$ as in the algorithm.

Case 1: $|Z(C)| = 3$

By the results of Corollary 2.3 and the fact that $Z(C) = \bigcap_{\tilde{C} \in \mathcal{M}} \tilde{C}$ we know that the T2S T of G must have a vertex $c' \in Z(C)$ such that $(c', v) \in T$ for all $v \in H$. If our algorithm in step 23 chooses just this node, then there is nothing to show. Thus we may assume that our algorithm chooses a node $c \neq c'$. Consider the graph T' obtained from T by replacing the edges (c', x) by the edges (c, x) $(x \in H)$. Clearly T' contains all the edges from T. We will prove that T' is a T2S of G. It is easy to see that T' contains no cycles, since $(c, c') \in T$.

For the sake of a contradiction assume that $(x, y) \in E$ is an edge that is not covered by the edges of T'. Since T was a T2S of G, either $(x, y) \in T$ or (x, y) was covered by a pair of edges from T.

If $(x, y) \in T$ then $(x, y) = (c', y)$ for some $y \in H$. But then the edges $(c, c') \in T'$ and $(c', y) \in T'$ cover (c', y). $\rightarrow\leftarrow$.

In the second case (x, y) was covered by $(c', x) \in T$ and $(c', y) \in T$. W.l.o.g. we can assume that (c', x) was removed from T because we added (c, x) to E_T $c \in Z(C)$. Again we have that (c, c') is contained in T'.

We distinguish again two cases: $y \in H$ and $y \notin H$. If $y \in H$, then (c, y) is added to E_T (and to T') and the edge (x, y) is covered as a contradiction.

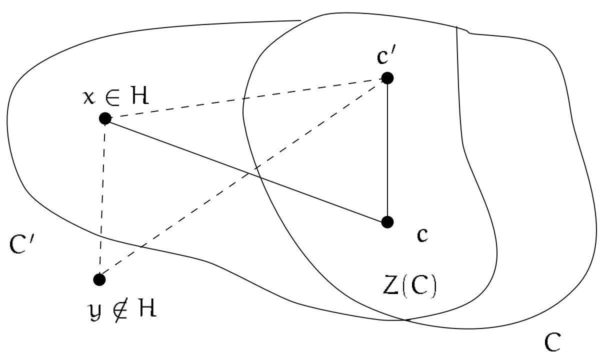

Figure 13: Case 1 and $y \notin H$ in Lemma 3.6.

In the second case, $N(y) \cap C = \{c'\}$, because otherwise y would have been added to H (see Figure 13). Then y is a node satisfying the conditions of the **while**–loop in step 14. But then c' is detected to be the center of a maximal clique in $\mathcal{M}$ and, since $|Z(C)| \geq 3$ also as the unique center of *all* the maximal cliques in $\mathcal{M}$. $\rightarrow\leftarrow$

Case 2: $|Z(C)| = 2$, say $Z(C) = \{c, c'\}$ We claim that, if G has a T2S, then *before* the execution of the **for**–loop in lines 29 to 34 there can be no two (in G) adjacent nodes $x, y \in H - Z(C)$ such that $(x, c) \in E_T$ and $(y, c') \in E_T$.

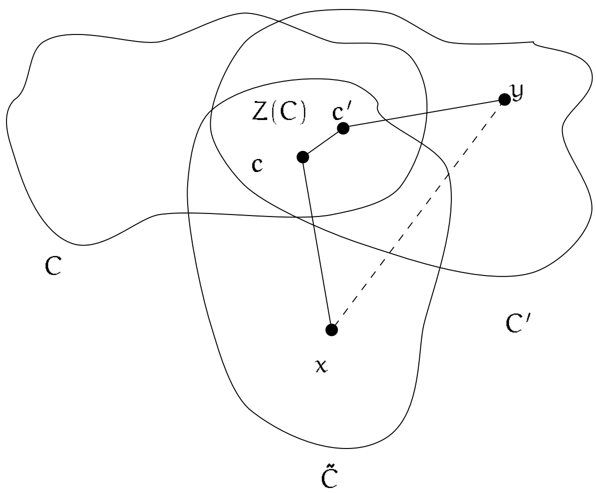

Figure 14: The neighbors x and y can not have different neighbors in Z(C).

In fact, by our assumption that all the edges of E_T are contained in the T2S T, before we execute the **for**–loop, in this case T would have to contain the edges $(c, c'), (c, x)$ and (c, y). Then $d_T(x, y) = 3$. $\rightarrow\leftarrow$.

Thus as a byproduct of this observation we obtain the following: If the algorithm makes the choice to add (c, x) to E_T in step 30, this choice is *independent* of the neighbor y.

Consider the addition of the edge (c, x) with $c \in Z(C)$ and $x \in C' \in \mathcal{M}$. We know that the center c' of $T[C']$ is contained in $Z(C)$. If $c \neq c'$, we remove (c', x) from T and add the edge (c, x). Let T' be the graph obtained from T by the replacement of all these edges. Again T' can contain no cycles and contains all the necessary edges. We need only show that T' is a 2–spanner of G.

Since for each node $x \in H - \{c, c'\}$ either the edge (c, x) or (c', x) is added to E_T, all the edges from T of the form (c, x) or (c', x) are either included in T or covered by the edges in T'. Since we only remove edges of just this form form T, all the removed edges are covered.

Assume that the edges of T' do not cover the edge (x, y) that been covered by $(c', y) \in T$ and $(c', x) \in T$. W.l.o.g. we can assume that (c', x) was removed from T because we added (c, x). Then $x \in H$.

If $y \notin H$, then as in the first case y is a node satisfying the conditions of the **while**–loop in step 14. Again c' is detected to be the center of a maximal clique in $\mathcal{M}$ that contains x, and then (c', x) is added to E_T before we enter the **for**–loop. Then (c', x) will also be in E_T (and T') after that loop contradicting the assumption that the algorithm added (c, x). $\rightarrow\leftarrow$.

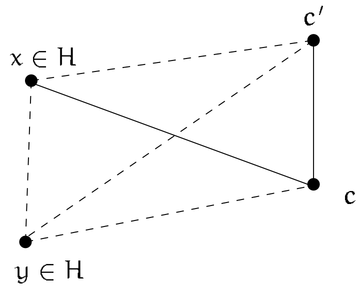

Figure 15: Case 2 and $y \in H$ in Lemma 3.6.

If $y \in H$ then by Observation 3.3: $(c, y) \in E$ (see Figure 15). If (c, y) was added to E_T in steps 14 to 20 and before the execution of the **for**–loop, then the edge (x, y) is covered by $(c, y) \in T'$ and $(c, y) \in T'$ as a contradiction to the assumption that the edge is *not* covered.

It thus follows that the algorithm either first considers x or y in the **for**–loop and that before those vertices have been considered in the loop, none of them is adjacent to any node from $Z(C)$ via the edges of E_T. But then according to the conditions of step 30 both vertices x and y will be adjacent to the same node in $Z(C)$. Thus (as we assume that $(c, x) \in E_T$), we can conclude that $(c, y) \in T'$. Again (x, y) is covered. $\rightarrow\leftarrow$ $\Box$

Now we can prove the main theorem of this paper:

Theorem 3.7 The algorithm Tree–2–Spanner is a polynomial time algorithm, which correctly decides whether a given undirected finite and connected graph G has a T2S.

Proof: By Lemma 3.6, the set E_T of edges that is produced during the run of the algorithm is always a subset of the edges of a T2S of G, if G has a T2S. Since the algorithm marks all the e that are covered by the edges in E_T with $m(e) :=$ covered, it follows that the set E_T at the end of the algorithm is the edge set of a 2-spanner of G. Thus, if G has a T2S, E_T can not contain cycles. Vice versa, if E_T does not have cycles, then we have found a cycle free 2–spanner of G, which then must be a tree.

It remains to show that the algorithm runs in polynomial time. The **while**–loop in steps 4 to 9 of Tree–2–Spanner is executed at most $|E|$ times, since in each iteration the uncovered edge e will be marked correctly as covered or as in the tree after that iteration.

We now consider the procedure Claw–Spanner. The first **while**-loop is executed at most $|V|$ times. A maximal clique C' containing w and the nodes from $N(w) \cap C$ can be found in $O(|V|^3)$ time.

It is now easy to see that all the remaining steps of Claw–Spanner can be carried out in polynomial time, that in fact the total running time of Claw–Spanner is $O(|V|^4)$. This results in an overall running time of $O(|E| \cdot |V|^4)$. $\Box$

4 Conclusion and Open Problems

We have presented a polynomial time algorithm to find a tree–2–spanner of a graph G, if there exists one. It should be noted here that the running time analysis in the proof of the last theorem is a very crude one. It should be possible to give better estimates and, maybe by the use of appropriate data structures, to cut down the total running time substantially.

It is clear that if a graph G contains the chordal graph from Figure 3 as an induced subgraph, then G can not contain a T2S. A direction of further research would be to find and classify the forbidden structures that prohibit the existence of a T2S. Also the existence of tree–spanners with a stretch factor of $t > 2$ might be investigated, although it seems very unlikely that the methods used in this paper carry over directly to this case.

References

[ADD+93] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete and Computational Geometry*, 9:81–100, 1993.

[BS94] Mihir Bellare and Madhu Sudan. Improved Non–Approximability Results. In *Proceedings of the 26th annual ACM Symposium on the Theory of Computing*, 5 1994.

[KP94] Kortsarz and Peleg. Generating Low-Degree 2-Spanners. In *ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1994.

[PS89] D. Peleg and A. A. Schäffer. Graph Spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

Preprint-Reihe
Institut für Informatik
Universität Würzburg

Verantwortlich: Die Vorstände des Institutes für Informatik.

[75] F. Hübner. *Output Process Analysis of the Peak Cell Rate Monitor Algorithm.* Januar 1994.

[76] K. Cronauer. *A Criterion to Separate Complexity Classes by Oracles.* Januar 1994.

[77] M. Ritter. *Analysis of the Generic Cell Rate Algorithm Monitoring ON/OFF-Traffic.* Januar 1994.

[78] K. Poeck, D. Fensel, D. Landes und J. Angele. *Combining KARL and Configurable Role Limiting Methods for Configuring Elevator Systems.* Januar 1994.

[79] O. Rose. *Approximate Analysis of an ATM Multiplexer with MPEG Video Input.* Januar 1994.

[80] A. Schömig. *Using Kanban in a Semiconductor Fabrication Environment — a Simulation Study.* März 1994.

[81] M. Ritter, S. Kornprobst und F. Hübner. *Performance Analysis of Source Policing Architectures in ATM Systems.* April 1994.

[82] U. Hertrampf, H. Vollmer und K.W. Wagner. *On Balanced vs. Unbalanced Computation Trees.* Mai 1994.

[83] M. Mittler und A. Schömig. *Entwicklung von „Due–Date"–Warteschlangendisziplinen zur Optimierung von Produktionssystemen.* Mai 1994.

[84] U. Hertrampf. *Complexity Classes Defined via* k*-valued Functions.* Juli 1994.

[85] U. Hertrampf. *Locally Definable Acceptance: Closure Properties, Associativity, Finiteness.* Juli 1994.

[86] O. Rose und M.R. Frater. *Delivery of MPEG Video Services over ATM.* August 1994.

[87] B. Reinhardt. *Kritik von Symptomerkennung in einem Hypertext-Dokument.* August 1994.

[88] U. Rothaug, E. Yanenko und K. Leibnitz. *Artificial Neural Networks Used for Way Optimization in Multi-Head Systems in Application to Electrical Flying Probe Testers.* September 1994.

[89] U. Hertrampf. *Finite Acceptance Type Classes.* Oktober 1994.

[90] U. Hertrampf. *On Simple Closure Properties of #P.* Oktober 1994.

[91] H. Vollmer und K.W. Wagner. *Recursion Theoretic Characterizations of Complexity Classes of Counting Functions.* November 1994.

[92] U. Hinsberger und R. Kolla. *Optimal Technology Mapping for Single Output Cells.* November 1994.

[93] W. Nöth und R. Kolla. *Optimal Synthesis of Fanoutfree Functions.* November 1994.

[94] M. Mittler und R. Müller. *Sojourn Time Distribution of the Asymmetric* M/M/1//N *– System with LCFS-PR Service.* November 1994.

[95] M. Ritter. *Performance Analysis of the Dual Cell Spacer in ATM Systems.* November 1994.

[96] M. Beaudry. *Recognition of Nonregular Languages by Finite Groupoids.* Dezember 1994.

[97] O. Rose und M. Ritter. *A New Approach for the Dimensioning of Policing Functions for MPEG-Video Sources in ATM-Systems.* Januar 1995.

[98] T. Dabs und J. Schoof. *A Graphical User Interface For Genetic Algorithms.* Februar 1995.

[99] M.R. Frater und O. Rose. *Cell Loss Analysis of Broadband Switching Systems Carrying VBR Video.* Februar 1995.

[100] U. Hertrampf, H. Vollmer und K. W. Wagner. *On the Power of Number-Theoretic Operations with Respect to Counting.* Januar 1995.

[101] O. Rose. *Statistical Properties of MPEG Video Traffic and their Impact on Traffic Modeling in ATM Systems.* Februar 1995.

[102] M. Mittler und R. Müller. *Moment Approximation in Product Form Queueing Networks.* Februar 1995.

[103] D. Rooß und K. W. Wagner. *On the Power of Bio-Computers.* Februar 1995.

[104] N. Gerlich und M. Tangemann. *Towards a Channel Allocation Scheme for SDMA-based Mobile Communication Systems.* Februar 1995.

[105] A. Schömig und M. Kahnt. *Vergleich zweier Analysemethoden zur Leistungsbewertung von Kanban Systemen.* Februar 1995.

[106] M. Mittler, M. Purm und O. Gihr. *Set Management: Synchronization of Prefabricated Parts before Assembly.* März 1995.

[107] A. Schömig und M. Mittler. *Autocorrelation of Cycle Times in Semiconductor Manufacturing Systems.* März 1995.

[108] A. Schömig und M. Kahnt. *Performance Modelling of Pull Manufacturing Systems with Batch Servers and Assembly-like Structure.* März 1995.

[109] M. Mittler, N. Gerlich und A. Schömig. *Reducing the Variance of Cycle Times in Semiconductor Manufacturing Systems.* April 1995.

[110] A. Schömig und M. Kahnt. *A note on the Application of Marie's Method for Queueing Networks with Batch Servers.* April 1995.

[111] F. Puppe, M. Daniel und G. Seidel. *Qualifizierende Arbeitsgestaltung mit tutoriellen Expertensystemen für technische Diagnoseaufgaben.* April 1995.

[112] G. Buntrock, und G. Niemann. *Weak Growing Context-Sensitive Grammars.* Mai 1995.

[113] J. García and M. Ritter. *Determination of Traffic Parameters for VPs Carrying Delay-Sensitive Traffic.* Mai 1995.

[114] M. Ritter. *Steady-State Analysis of the Rate-Based Congestion Control Mechanism for ABR Services in ATM Networks.* Mai 1995.

[115] H. Graefe. *Konzepte für ein zuverlässiges Message-Passing-System auf der Basis von UDP.* Mai 1995.

[116] A. Schömig und H. Rau. *A Petri Net Approach for the Performance Analysis of Business Processes.* Mai 1995.

[117] K. Verbarg. *Approximate Center Points in Dense Point Sets.* Mai 1995.

[118] K. Tutschku. *Recurrent Multilayer Perceptrons for Identification and Control: The Road to Applications.* Juni 1995.

[119] U. Rhein-Desel. *Eine „Übersicht“ über medizinische Informationssysteme: Krankenhausinformationssysteme, Patientenaktensysteme und Kritiksysteme.* Juli 1995.

[120] O. Rose. *Simple and Efficient Models for Variable Bit Rate MPEG Video Traffic.* Juli 1995.

[121] A. Schömig. *On Transfer Blocking and Minimal Blocking in Serial Manufacturing Systems — The Impact of Buffer Allocation.* Juli 1995.

[122] Th. Fritsch, K. Tutschku und K. Leibnitz. *Field Strength Prediction by Ray-Tracing for Adaptive Base Station Positioning in Mobile Communication Networks.* August 1995.

[123] R. V. Book, H. Vollmer und K. W. Wagner. *On Type-2 Probabilistic Quantifiers.* August 1995.

[124] M. Mittler, N. Gerlich, A. Schömig. *On Cycle Times and Interdeparture Times in Semiconductor Manufacturing.* September 1995.

[125] J. Wolff von Gudenberg. *Hardware Support for Interval Arithmetic - Extended Version.* Oktober 1995.

[126] M. Mittler, T. Ono-Tesfaye, A. Schömig. *On the Approximation of Higher Moments in Open and Closed Fork/Join Primitives with Limited Buffers.* November 1995.

[127] M. Mittler, C. Kern. *Discrete-Time Approximation of the Machine Repairman Model with Generally Distributed Failure, Repair, and Walking Times.* November 1995.

[128] N. Gerlich. *A Toolkit of Octave Functions for Discrete-Time Analysis of Queuing Systems.* Dezember 1995.

[129] M. Ritter. *Network Buffer Requirements of the Rate-Based Control Mechanism for ABR Services.* Dezember 1995.

[130] M. Wolfrath. *Results on Fat Objects with a Low Intersection Proportion.* Dezember 1995.

[131] S. O. Krumke and J. Valenta. *Finding Tree–2–Spanners.* Dezember 1995.

[132] U. Hafner. *Asymmetric Coding in (m)-WFA Image Compression.* Dezember 1995.