A practical algorithm for finding optimal triangulations

Kirill Shoikhet and Dan Geiger
Computer Science Department
Technion
Haifa 32000, ISRAEL
kirill@cs.technion.ac.il, dang@cs.technion.ac.il

Abstract

An algorithm called QUICKTREE is developed for finding a triangulation T of a given undirected graph G such that the size of T's maximal clique is minimum and such that no other triangulation of G is a subgraph of T. We have tested QUICKTREE on graphs of up to 100 nodes for which the maximum clique in an optimal triangulation is of size 11. This is the first algorithm that can optimally triangulate graphs of such size in a reasonable time frame. This algorithm is useful for constraint satisfaction problems and for Bayesian inference through the clique tree inference algorithm.

Introduction

An undirected graph is triangulated (chordal) if for every cycle C of length greater than 3 the graph contains a chord, that is, an edge which connects two non adjacent vertices of C. Given an undirected graph G, a supergraph of G which is triangulated is called a *triangulation* of G. The problem we address is to find a triangulation T of G such that the size of its maximal clique is minimum and such that T is edge-minimal, i.e., no other triangulation of G is a subgraph of T. Such a triangulation is called an *optimal triangulation* and the size minus 1 of its largest clique is called the *treewidth* of G.

The AI application which prompted our attention to this problem is the updating problem which is to compute the posterior probability of a random variable in a Bayesian network given specific values to a set of other random variables (Pe86; LS88; Pe88; JLO90). This application is also treated in (BG96). Dechter demonstrated that a variety of problems in AI can be solved efficiently if a good triangulation is made available (De96). These problems are: satisfiability, most probable explanations, maximum aposteriori hypothesis, and maximum expected utility.

The problem of finding optimal triangulations is important in other areas of computer science because many NP-complete problems on graphs can be solved polynomially if the input graph has a triangulation with sufficiently small cliques and if such a triangulation can be found efficiently (Ar85; ALS91). Some of these problems are: independent set, dominating set, graph K-colorability and Hamiltonian circuit.

Finding an optimal triangulation is NP-complete (ACP87). However, for a graph with n vertices and a fixed treewidth k there exits an $O(n^{k+2})$ algorithm that finds an optimal triangulation based on (ACP87). This algorithm is not practical for moderate sizes of k and n (say, $k = 10$ and $n = 100$). Another algorithm for finding an optimal triangulation has a complexity of $O(f(k)n)$ where $f(k)$ is a super-exponential function of k (Bo93). This algorithm is practical for treewidth of size $k = 4$ at most. For larger values of k there has been no algorithm to date that could find an optimal triangulation sufficiently fast.

In this paper, an algorithm called QUICKTREE is developed for finding optimal triangulations. We have tested QUICKTREE on graphs of up to 100 nodes for which the treewidth is 10. This is the first algorithm that can optimally triangulate graphs of such size in a reasonable time frame. Our method is similar to Arnborg et al's algorithm in the idea of using dynamic programming to build up triangulations of larger parts from triangulations of smaller parts. It differs in the number of small parts that are created (far less) and in the way they are combined.

The paper is organized as follows. In the next section, we provide facts about triangulations and describe their close relationship to clique trees and to decomposability of graphs. Then we describe the algorithm for finding an optimal triangulation, followed by implementation details and experimental validation. We conclude with possible extensions of our results.

Definitions and basic facts

A *graph* G is a pair (V, E), where V is a finite set of *vertices* called the *vertex set*, and E is a set of pairs of vertices (*edges*) called the *edge set*. Given a graph G, $V(G)$ denotes the vertex set of G. Two vertices u and v of G are *adjacent* if there is an edge (u, v) in the edge set of G. The set of vertices of G that are adjacent to a vertex v is called the *neighborhood* of v in G. A sequence of distinct vertices $[v_0, v_1, \ldots, v_l]$ of $G = (V, E)$ is called a *path* from v_0 to v_l, provided

that $(v_{i-1}, v_i) \in E$ for $i = 1, \ldots, l$. A *cycle* is a closed path, i.e. path from v to v, where v is a vertex of G. A *chordless cycle* is a cycle $[v_0, \ldots, v_l, v_0]$, such that $v_i \neq v_j$ for $i \neq j$ and $(v_i, v_j) \notin E$, if i and j differ by more than 1 mod $l + 1$. A connected graph without cycles is called a *tree*.

The *union* of n graphs $G_i = (V_i, E_i)$, $i = 1, \ldots, n$ is the graph $\bigcup_{i=1}^{n} G_i = (\bigcup_{i=1}^{n} V_i, \bigcup_{i=1}^{n} E_i)$. The graph $G' = (V', E')$ is called a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A subgraph G' of G is a *proper subgraph* if $G' \neq G$. The graph $G[S]$, $S \subseteq V(G)$, is the *subgraph of G induced by S*, namely its vertex set is S and its edge set contains only edges of G with both ends in S. The subgraph of G induced by some subset of $V(G)$ is called an *induced subgraph* of G. A *supergraph of* $G = (V, E)$ is a graph $G' = (V, E')$, such that $E \subset E'$. A *complete graph* or *clique* is a graph with edge set consisting of all possible edges between the vertices of the graph. We denote by $K[S]$ the clique built on vertices from S. A *maximal clique* in a graph G is a clique $G[S]$ such that there exists no vertex set $S' \supset S$ for which $G[S']$ is a clique. The set of maximal cliques of G is denoted by $\mathcal{K}_G$.

A graph G is *connected* if for each pair u, v of its vertices there exists a path from u to v. If G is not connected, then a connected subgraph C of G is called a *(maximal) connected component* if there exists no connected subgraph C' of G, such that C is a proper subgraph of C'. Given a connected graph G with vertex set $V = V(G)$, a subset $S \subset V$ is called a *vertex separator* for nonadjacent vertices a and b in $V \setminus S$ if a and b are in different connected components of $G[V \setminus S]$. If S is a vertex separator for a and b but no proper subset of S is a vertex separator of a and b, then S is called a minimal vertex separator for a and b or a *minimal a, b-separator*. A subset $S \subset V$ is called a *minimal vertex separator* if there exists a pair of nonadjacent vertices for which S is a minimal vertex separator. Throughout, we abbreviate the term vertex separator with the term separator. According to the definition of a minimal separator, a minimal separator S of G can be properly contained in another minimal separator of G. In this case S is a *nested separator in G*. We say that a separator S of G *crosses* another separator S' if there are vertices u and v in S' which are separated by S in G. It is easy to see, that if S crosses S', then S' separates some vertices of S, and therefore S' crosses S.

A graph is called *triangulated* or *chordal* if it does not contain a chordless cycle of length greater than 3. An induced subgraph of a triangulated graph is triangulated. A *simplicial vertex of* a graph $G = (\{v_1, \ldots, v_n\}, E)$ is a vertex, such that its neighborhood induces a clique in G. Every triangulated graph G has a simplicial vertex and if G is not a clique, then it has two nonadjacent simplicial vertices (Go80). An ordering of the vertices $\sigma = [v_1, \ldots, v_n]$ is called a *perfect elimination sequence* for G if for every $1 \leq i \leq n$, v_i is a simplicial vertex in $G[\{v_i, \ldots, v_n\}]$. Given a connected graph G, a tree T whose vertex set is the set of all maximal cliques of G (i.e. $\mathcal{K}_G$), is called a *clique-tree* of G if it satisfies the *clique-intersection property*: for every pair of distinct nodes of T, $K, K' \in \mathcal{K}_G$, the set $K \cap K'$ is contained in every node on the path connecting K and K' in T.

The following theorem lists several well known characterizations of triangulated graphs (Go80).

Theorem 1 *Let G be a connected graph. Then the following conditions are equivalent.*

1. *G is triangulated*
2. *G has a perfect elimination sequence*
3. *There exists a clique-tree of G*
4. *Every minimal separator of G induces a complete subgraph of G*

A *triangulation* of a graph G is a triangulated graph which is a supergraph of G. A triangulation T of G is *minimal* if there exists no proper subgraph of T, which is a triangulation of G. The *width* of a triangulated graph T is $\max_{K \in \mathcal{K}_T}(|K| - 1)$, where $\mathcal{K}_T$ is the set of maximal cliques of T. The *treewidth* of a graph G is the minimal width over all triangulations of G. An *optimal triangulation* of G is a minimal triangulation T of width k where k is the treewidth of G. The problem addressed herein is to construct an optimal triangulation of a given graph G.

If S is a separator of $G = (V, E)$, the graph $G[V \setminus S]$ contains several connected components having distinct vertex sets $V_1, \ldots, V_l$ where $l \geq 2$. The graphs $F_i = G[V_i \cup S] \cup K[S]$ are called the *fragments of G obtained by S* or *S-fragments of G*. An S-fragment F_i is obtained from $G[V_i \cup S]$ by adding all possible edges between vertices of S such that $F_i[S]$ is a clique. Let S be a minimal separator of G of size at most k and $F_1, \ldots, F_l$ be the fragments of G obtained by S. The graph $\bigcup_{i=1}^{l} T(F_i)$, where $T(F_i)$ is an optimal triangulation of F_i, is called the *S-composite graph* of G. We refer to the operation of obtaining an S-composite graph of G as *simple composition*. Arnborg et al. (ACP87) use simple composition in order to construct a triangulation T of a given graph G such that all maximal cliques of T are of size $k + 1$. Such triangulations are called *k-trees*.

The triangulation algorithm

In order to construct an optimal triangulation of G by dynamic programming, we produce fragments of G obtained by the minimal separators of size at most k and we generate some special cliques of G. Then we repeatedly combine optimal triangulations of smaller parts to obtain optimal triangulations of larger ones. We now define the combination operation which we call simultaneous composition.

Let $\mathcal{S} = \{S_1, \ldots, S_l\}$ be a set of minimal separators of a graph G. The *kernel* of $\mathcal{S}$, denoted by $K_{\mathcal{S}}$, is

the graph induced from G by $\bigcup_i S_i$ with additional edges that make each S_i a clique, i.e., it is the graph $G[\bigcup_i S_i] \cup \bigcup_i K[S_i]$. Let $\mathcal{F}_i$ be the set of S_i-fragments of G whose vertex sets do not contain any S_j for $j \neq i$. The union of $\mathcal{F}_i$ over all i is called the *fragment set* of $\mathcal{S}$ and is denoted by $\mathcal{F}_\mathcal{S}$.

A set $\mathcal{S} = \{S_1, \ldots, S_l\}$ of minimal separators of size at most k of G is called a *k-regular set of separators* if the following four conditions are satisfied:

(i) The kernel $K_\mathcal{S}$ is a clique.

(ii) For each i at least one S_i-fragment of G is included in the fragment set $\mathcal{F}_\mathcal{S}$.

(iii) Every vertex of G is either a vertex in the maximal clique containing $K_\mathcal{S}$ in the graph $G \cup K_\mathcal{S}$ or a vertex of a fragment in $\mathcal{F}_\mathcal{S}$.

(iv) If a vertex v of G is contained in fragments $F_1, \ldots, F_h$, $h > 1$, where F_i is an S_i-fragment in $\mathcal{F}_\mathcal{S}$, and v is not in $\bigcup_i S_i$, then there exist a minimal separator S of G and an S-fragment F, such that $S \subseteq \bigcap_{i=1}^{h} S_i$ and $V(F) \setminus S = V(F_i) \setminus S_i$ for all $1 \leq i \leq h$.

The maximal clique containing $K_\mathcal{S}$ in the graph $G \cup K_\mathcal{S}$ is called the *base clique* of $\mathcal{S}$ and is denoted by $B_\mathcal{S}$. Let $\mathcal{T}_\mathcal{S}$ denote a set of optimal triangulations of the fragments in $\mathcal{F}_\mathcal{S}$ such that $T_i \in \mathcal{T}_\mathcal{S}$ is an optimal triangulation of $F_i \in \mathcal{F}_\mathcal{S}$. The set $\mathcal{T}_\mathcal{S}$ is called *legal* if for every pair of fragments F_i and F_j in $\mathcal{F}_\mathcal{S}$ for which $V(F_i) \setminus S_i = V(F_j) \setminus S_j = W$, the corresponding triangulations T_i and T_j induce the same graph on W (i.e., $T_i[W] = T_j[W]$). The graph $B_\mathcal{S} \cup \left(\bigcup_{T \in \mathcal{T}_\mathcal{S}} T\right)$ is called an *$\mathcal{S}$-composite graph* of G, whenever $\mathcal{T}_\mathcal{S}$ is legal. We refer to the operation of obtaining an $\mathcal{S}$-composite graph of G as *simultaneous composition*. For $l = 1$ simple and simultaneous composition coincide and the base clique is just $K[S]$. Note that a k-regular set $\mathcal{S}$ of separators of G does not contain a separator S_i which is a subset of another separator S_j in $\mathcal{S}$ lest every S_j-fragment of G would contain S_i and excluded from $\mathcal{F}_\mathcal{S}$ (contradicting condition **(ii)**).

Lemma 2 *Let $G = (V, E)$ be a connected graph of treewidth k and let $\mathcal{S} = \{S_1, \ldots, S_l\}$ be a k-regular set of separators of G. If the base clique of $\mathcal{S}$ is of size at most $k + 1$, then every $\mathcal{S}$-composite graph of G is an optimal triangulation of G.*

Proof: Let $B_\mathcal{S}$ be the base clique of $\mathcal{S} = \{S_1, \ldots, S_l\}$, $|V(B_\mathcal{S})| \leq k + 1$, and $\mathcal{F}_\mathcal{S}$ be the fragment set of $\mathcal{S}$. Let H be an $\mathcal{S}$-composite graph of G, that is, $H = B_\mathcal{S} \cup \left(\bigcup_{T \in \mathcal{T}_\mathcal{S}} T\right)$, where $\mathcal{T}_\mathcal{S}$ is legal. The graph H is a supergraph of G because every vertex of G is contained either in the base of $\mathcal{S}$ or in a fragment in $\mathcal{F}_\mathcal{S}$ and the edge set of H is a superset of the edge set of G. We now prove that H is triangulated and then that H is an optimal triangulation of G.

Let C be a cycle of H of length greater than 3. Clearly, if C is completely in the base clique $B_\mathcal{S}$ then it has a chord. If it is included in $T(F)$ for some fragment F in $\mathcal{F}_\mathcal{S}$, then it also has a chord because $T(F)$ is triangulated. If it is partially included in a triangulation $T(F)$ of an S_i-fragment F, then it must contain two vertices of S_i which are not adjacent in C. These vertices are connected because S_i is a clique in H. Hence C has a chord and H is triangulated.

Finally we show that H is an optimal triangulation of G. Consider an edge in H that is not an edge in $B_\mathcal{S}$. Clearly, no such edge can be removed from any $T(F)$ because these graphs are all minimal and either disjoint or completely equal except on $B_\mathcal{S}$. Now consider an edge e in $B_\mathcal{S}$. If e is an edge in G, it clearly cannot be removed. If e is a fill-in edge, then it connects two vertices of some separator S_i in $\mathcal{S}$. Since for each i at least one S_i-fragment is in $\mathcal{F}_\mathcal{S}$, S_i is a separator in H (which separates a vertex in an S_i-fragment in $\mathcal{F}_\mathcal{S}$ from a vertex outside this fragment). Each S_i in $\mathcal{S}$ is actually a minimal separator of H because it is a minimal separator of G and a separator of H. Consequently, e cannot be removed from H because if e is removed, S_i remains a minimal separator, which contradicts the property that each minimal separator in a triangulated graph induces a complete graph. Thus H is a minimal triangulation of G. It is an optimal triangulation because the size of $B_\mathcal{S}$ is at most $k + 1$ and each $T(F)$ for $F \in \mathcal{F}_\mathcal{S}$ is an optimal triangulation. □

The triangulation algorithm shown in Figure 1 applies simultaneous composition on a set of fragments until an optimal triangulation is obtained or the composition operation cannot be applied in which case the algorithm outputs a valid statement that the treewidth is larger than k. The algorithm consists of several phases. The first phase generates the set of minimal vertex separators of size at most k of the given graph. The second phase constructs the set of first and second level fragments of the graph. The third phase (initialization) marks each triangulated fragment F with $T(F) \neq \emptyset$. The final phase (dynamic programming) combines in increasing order of size optimal triangulations of smaller fragments to obtain optimal triangulations of larger ones.

The algorithm defines the following concepts and notations. Given a connected graph G and a set Σ of minimal separators of G, the fragments of G obtained by separators in Σ, are called *first-level fragments* of G by Σ. An S-fragment F' of a first-level fragment F of G is called a *second-level fragment* of G if S is a minimal separator of G (and hence of F) and if F' is not a first-level fragment of G. The set of minimal separators of G of size at most k is denoted by $\mathcal{S}_G$. Let $\mathcal{F}_G$ denote the union of the following two sets. The first set is the set of first-level fragments of G obtained by separators in $\mathcal{S}_G$. The second set is the set of second-level fragments of G obtained by separators in $\mathcal{S}_G$ such that each fragment in this set is a clique.

The next two theorems show that whenever the algorithm outputs a graph, it is an optimal triangulation of

G and whenever the algorithm fails to output a graph ($T(G) = \emptyset$), there exists no triangulation of width k of G.

Theorem 3 *Let G be a connected graph of treewidth k and let F be a fragment in $\mathcal{F}_G$. If $T(F) \neq \emptyset$, where $T(F)$ is defined by the algorithm, then $T(F)$ is an optimal triangulation of F.*

Proof: Let $T(F)$ be defined in the initialization phase of the algorithm. If $F \in \mathcal{F}_G$ is a clique, then $T(F) = F$ is an optimal triangulation of F. Lemma 2 shows that compositions of optimal triangulations by separators in $\mathcal{S}_G$, as performed by the dynamic programming phase, produce optimal triangulations. Since the only change in $T(F)$ is a result of composition operations, $T(F)$ is an optimal triangulation of F unless $T(F) = \emptyset$. □

Theorem 4 *If G is a connected graph of treewidth k, then, when the algorithm terminates, $T(G)$ is an optimal triangulation of G. If the treewidth is larger than k, then the algorithm outputs this fact.*

Implementation

In this section we describe how we have implemented phase one and how we find k-regular sets of separators in the dynamic programming phase.

Kloks and Kratsch present an algorithm that generates the minimal a, b-separators of an undirected graph G. Finding all minimal separators of G is then done by examining all possible pairs of vertices $\{a, b\}$ in $O(Rn^5)$ steps where R is the number of minimal separators of G (KK94; Kl95). We use this algorithm to generate the minimal separators of G and then we extract all separators of size at most k. We introduced two heuristic changes that improve the usual running time. First, if we find a minimal a, b-separator S of G such that $G[S]$ is a clique, then we ignore all pairs of vertices that are separated by S because their separators will be found by examining other pairs. Second, we order the pairs of vertices by decreasing mutual distance in G. This heuristic produces more minimal separators which induce cliques in early stages of the algorithm.

The way we currently find k-regular sets of F is by examining the maximal cliques of a graph which we call the *validity graph*. The validity graph H for an S_0-fragment F of G is an undirected graph (Σ, E). Each vertex in H is some minimal separator of F. A separator S is in Σ if and only if $|S \cup S_0| \leq k + 1$. For every minimal separator S of F we define the *cover set* of S to be the set of all vertices of already triangulated S-fragments of F, except the vertices of S itself. The graph H contains an edge between S_i and S_j if and only if the following three conditions hold: (1) S_j does not cross S_i, (2) $|S_i \cup S_j \cup S_0| \leq k + 1$, and (3) the intersection of the cover sets of S_i and of S_j is either empty or equal to the cover set of a separator S' which is nested in both S_i and S_j. Each k-regular set of F is a maximal clique in the validity graph of F.

Algorithm QUICKTREE

Input: A connected graph $G = (V, E)$ and an integer $k > 1$.
Output: A minimal triangulation of G or a valid statement that G has a treewidth $> k$.

{*Phase 1: Build a set $\mathcal{S}_G$ of minimal separators of G which are sufficient for finding a minimal triangulation of width at most k if such exists*}

$\mathcal{S}_G \leftarrow \{S |$ *S is a minimal separator of G of size* $\leq k\}$

{*Phase 2: Build a set $\mathcal{F}_G$ of fragments*}
$\mathcal{F}_G \leftarrow \emptyset$
for each S in $\mathcal{S}_G$ **add** the fragments of G obtained by S into $\mathcal{F}_G$
{*These are called first-level fragments of G*}
for each fragment F in $\mathcal{F}_G$ **do**
  for each S in $\mathcal{S}_G$
    compute the fragments of F obtained by S
    if an S-fragment F' of F is a clique and F' is not in $\mathcal{F}_G$ **then add** F' to $\mathcal{F}_G$
{*These are called second-level fragments of G*}

{*Initialization of $T(F)$ - an optimal triangulation of a fragment F unless $\emptyset$*}

for each fragment F in $\mathcal{F}_G$ **do**
  if F is a clique of size $> k + 1$ **then**
    return "G has a treewidth $> k$"
  if F is a clique **then** $T(F) \leftarrow F$
  else $T(F) \leftarrow \emptyset$

{*The dynamic programming phase: Composing fragments from smallest to largest to create an optimal triangulation*}

sort the fragments in $\mathcal{F}_G$ in increasing order of size: $\{F_1, \ldots, F_m\}$
for each F_i in $\mathcal{F}_G, 1 \leq i \leq m$ **do**
  {*Exit condition*}
  if there exists a separator S in $\mathcal{S}_G$ **such that** for each fragment F of G obtained by S, $T(F) \neq \emptyset$
    then return $\bigcup_{F \in \mathcal{F}} T(F)$, where $\mathcal{F}$ is the set of fragments of G obtained by S.
{*An optimal triangulation of G has been found via simple composition*}
  if F_i is triangulated **then set** $T(F_i) \leftarrow F_i$ **and continue** with F_{i+1}
  FIND a k-regular set $\Sigma = \{S_1, \ldots, S_l\} \subseteq \mathcal{S}_G$ of separators of F_i **such that** $T(F) \neq \emptyset$ for each F in the fragment set $\mathcal{F}_\Sigma$ of Σ
  if such Σ is found
    then $T(F_i) \leftarrow F_i \cup [\bigcup_{F \in \mathcal{F}_\Sigma} T(F)]$
{*An optimal triangulation of F_i has been found via simultaneous composition*}
return "G has a treewidth greater than k"

Figure 1: The triangulation algorithm

Experimental validation

We tested QUICKTREE on graphs with 50, 75 and 100 nodes for which the treewidth is at most 10 (i.e., the maximum clique size in an optimal triangulation is at most 11). The graphs were generated by creating a random clique tree in which every maximal clique has the same size and then randomly dropping 30%, 40% or 50% of the edges. This way we created graphs for which the maximal clique size in an optimal triangulation is bounded. We applied QUICKTREE on the resulting graphs.

Table 1 shows the run time in seconds needed to triangulate a graph when 30% of the edges are dropped. The entry for a 75/7 graph is the time to process a graph with 75 nodes and treewidth 7. The time is divided into two parts: T_1–the time to generate all minimal separators and T_2–the time to find a triangulation. Each entry in every table is based on an average of 3 graphs. The program is written in C^{++} including the Standard Template Library and was compiled by g++ version 2.7.2. We run our experiments of tables 1 and 2 on a 50MHz Sun Sparc20 server with 122M RAM. Tables 3 and 4 were produced on a 100MHz HP/725 workstation with 114M RAM.

	50		75		100	
30%	T_1	T_2	T_1	T_2	T_1	T_2
5	85.8	1.96	99	5.9	1500	13.6
7	235	2.22	841	6.2	2694	12
10	317	2.65	11887	10.4		

Table 1

Table 1 shows that the bottleneck of the algorithm is the first phase. This phase of the algorithm is polynomial in the number of minimal separators; it requires $O(Rn^5)$ steps where R is the number of minimal separators and n is the number of nodes. Unfortunately, these terms are quite big in large graphs. The remaining algorithm runs very fast which could be surprising because it uses a procedure FIND which in the worst case is exponential in the size of the largest validity graph. The value of T_1 in the 75/10 graphs is the average of 4998, 5780 and 24,883 seconds. The number of minimal separators found in these graphs, denoted by R, has been 1412,1377,4167, respectively, which partially explains the running time differences. The number of minimal separators smaller than k as needed for the dynamic programming phase for these graphs, denoted by R_k, has merely been 92,86 and 91, respectively; less than 7% of the separators generated.

Graphs of size 100/10 with 30% dropout of edges take over 10 hours due to the large number of minimal separators generated by Phase 1. This class of graphs is the current practical limit of our implementation using 100MHz machines with 120M RAM. Graphs of size 100/10 with a 20% dropout of edges yield $T_1 = 7188$ and $T_2 = 7$.

Table 2 shows that the total number of maximal cliques FIND encounters (denoted by Cliq) in all the validity graphs of a single run is low and that the number of nodes in the largest validity graph (denoted by Nod) is low as well.

	50		75		100	
30%	Nod	Cliq	Nod	Cliq	Nod	Cliq
5	19.7	10	42	24	43	28.3
7	23.3	6.3	40	16.3	34	6
10	28.3	16.3	44.3	24		

Table 2

It is clear that QUICKTREE runs fast on graphs with small number of minimal separators. When we remove more edges from the initial k-tree we produce graphs with higher number of minimal separators. Table 3 shows how the running time deteriorates when more edges are dropped starting with a graph of 75 nodes and treewidth of 7.

75/7	R	R_k	Frag	T_1	T_2
20%	152	79	255	274	3.16
30%	245	79	282	569	3.54
40%	296	82	290	832	3.69
50%	443	79	275	1562	3.86

Table 3

The entry Frag in Table 3 measures the number of fragments produced for the dynamic programming phase. The entries R and R_k measure, respectively, the number of minimal separators and the number of minimal separators of size less than k where $k = 7$. Note again that R_k is smaller than R; Many minimal separators are generated in the first phase of QUICKTREE but are not needed for the dynamic programming phase. A second reason for the high values of T_1 is that the algorithm runs over all pairs of vertices and for each pair $\{a, b\}$ produces all minimal a, b-separators. However, after a few pairs, the algorithm usually finds most of the minimal separators and the remaining run time is just used to verify that indeed all minimal separators have been generated. Table 4 shows the number of Good-Pairs—pairs that generated at least one new minimal separator. All-Pairs denote the number of pairs we used which guarantee that all minimal separators have been generated. For Table 4, 30% of the edges were dropped.

75 nodes, 30%	5	7	10
All-Pairs	1158	2113	2264
Good-Pairs	50	59	75

Table 4

Table 4 suggests that if there is no needed guarantee of optimal triangulation, then Phase 1 can be run on a fraction of the possible pairs of vertices and then the dynamic programming phase can be applied. For example, on 3 graphs with 75 nodes and treewidth 7, we selected the top 20% of pairs that had a maximal mutual distance and got close to optimal triangulations (twice the optimal treewidth 7 and once 9 instead of 7). The average running time was reduced from 675 to 373 seconds.

Discussion

In many applications the treewidth of a triangulation is just an approximation to the real optimization problem. For example, for the updating problem in Bayesian networks, one needs to find a triangulation that minimizes the sum $\sum_i 2^{w(C_i)}$ where w is a positive additive weight function on the vertices of G (Kj90; BG96). This sum takes into account the size of the state space of every variable in the given Bayesian network. The triangulation algorithm presented herein can be modified to accommodate such variants. Currently, the algorithm stops whenever an optimal triangulation is generated. However, if we let the algorithm run until all fragments have been examined, then all optimal triangulations will be generated. Among these optimal triangulations one can select a triangulation with the smallest required sum. Since the dominant term in the sum is minimized, this method should often yield triangulations with close to optimal sums.

Our experiments show that the bottleneck of QUICKTREE is phase 1 which generates all minimal separators of size at most k. There are several ways to improve. First, our work leaves open the problem of generating all minimal separators of size at most k of G in time $O(R_k \text{poly}(n))$ where R_k is the number such minimal separators. Such an algorithm could improve QUICKTREE because R and R_k differ considerably as shown by our experiments. Second, one can possibly find all minimal separators without running over as many pairs as we use. Finally, since the dynamic programming phase is very fast, it is possible to generate a set of minimal separators, apply dynamic programming and continue this cycle until a satisfactory triangulation is at hand.

QUICKTREE finds a k-regular set by examining all maximal cliques of the validity graph. We believe that with minor changes to the definition of a validity graph, k-regular sets will be in one-to-one correspondence with maximal cliques and so could be found more easily in time polynomial in the number of minimal separators. A similar relationship between maximal cliques of a (larger) graph of separators and minimal triangulations is established in (PS95) where such a polynomial algorithm is derived for d-trapezoid graphs. This possible amendment will resolve a theoretical problem but will hardly affect the running time of our algorithm.

QUICKTREE produces as an output a triangulated graph and a perfect elimination sequence. Consequently, it can be added as a preprocessor to many existing inference routines with minor changes. Our code is still under development; It will be made available along with a more detailed paper that describes the implementation at: http://www.cs.technion.ac.il/~dang/.

Acknowledgment

We thank Hans Bodlænder, Petra Scheffler, and Seffi Naor for pointers they provided us about treewidth.

References

Arnborg S., Efficient algorithms for combinatorial problems on graphs with bounded decomposability, *BIT* 25, pp. 2–23, 1985.

Arnborg S., Corneil D.G., and Proskurowski A. Complexity of finding embeddings in a k-tree, *SIAM J. Alg. Disc. Meth.* 8, pp. 277–284, 1987.

Arnborg S., Lagergren J., and Seese D., Easy problems for tree-decomposable graphs, *J. of Alg.* 12, pp. 308–340, 1991.

Arnborg S. and Proskurowski A. Characterization and recognition of partial 3-trees, *SIAM J. Alg. Disc. Meth.* 7, pp. 305–314, 1986.

Becker A. and Geiger D., A sufficiently fast algorithm for finding close to optimal junction trees, *Artificial Intelligence*, to appear. An earlier version in *Proceedings* of the Twelfth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, pp. 81–89, 1996.

Blair J.R.S. and Peyton B., An Introduction to chordal graphs and clique trees, *in Graph theory and sparse matrix computation*, Eds. George A., Gilbert J.R., Liu J., Springer-Verlag, NY, 1–29, 1993.

Bodlænder H.L., A linear time algorithm for finding tree-decompositions of small treewidth, Proceedings of the 25th ACM STOC, pp. 226–234.

Dechter R., Bucket elimination: A unifying framework for probabilistic inference, In *Proceedings* of the Twelfth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, pp. 211–219, 1996.

Golombic M., Algorithmic graph theory and perfect graphs, *Academic Press*, New York 1980.

Jensen F. V., Lauritzen S.L., and Olesen K.G., Bayesian updating in causal probabilistic networks by local computations, *Computational Statistics Quarterly* 4 (1990), 269–282.

Kjærulff U., Triangulation of graph-algorithms giving small total state space. Technical Report R 90-09, Department of Mathematics and Computer Science, Aalborg university, Denmark, March 1990.

Kloks T., *Treewidth*, Springer-Verlag, Lecture notes in Computer Science #842, 1995.

Kloks T. and Kratsch D., Finding all minimal separators of a graph, *Proceedings* 11th STACS, Lecture notes in Computer Science #775, Springer-Verlag, 759–768, 1994.

Lauritzen, S.L. and Spiegelhalter, D.J. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal Royal Statistical Society*, B, 1988, 50(2):157-224.

Parra A. and Scheffler P. How to use the minimal separators of a graph for its chordal triangulations. in *Proceedings* of ICALP95; Lecture Notes in Computer Science, Springer Verlag, #944, pp. 123-134, 1995.

Pearl, J., *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann, San Mateo, California, 1988.

Pearl, J., *Fusion, propagation and structuring in belief networks, Artificial Intelligence*, 29:3 (1986), 241–288.

Rose D., Triangulated graphs and the elimination process, *J. Math. anal. appl.*, 32 (1974), 597–609.