AALBORG UNIVERSITY

Triangulation of Graphs – Algorithms Giving Small Total State Space

by

Uffe Kjærulff

R 90-09 ISSN 0908-1216 March 1990

Institute for Electronic Systems
Department of Mathematics and Computer Science
Fredrik Bajers Vej 7E — DK 9220 Aalborg Ø — Denmark
Tel.: +45 98 15 85 22 — TELEX 69 790 aub dk

Triangulation of Graphs – Algorithms Giving Small Total State Space

Uffe Kjærulff
Judex Datasystemer A/S
DK-9000 Aalborg, Denmark

March 1, 1990

Abstract

The problem of achieving small total state space for triangulated belief graphs (networks) is considered. It is an $\mathcal{NP}$-complete problem to find a triangulation with minimum state space.

Our interest in this topic originates from the field of knowledge engineering where the applied knowledge representation scheme is provided by the notion of causal probabilistic networks (belief networks); CPNs for short. The application of a generalised evidence propagation scheme in CPNs requires triangularity (chordality) of the actual network.

The paper includes a survey and evaluation of existing triangulation algorithms most of which are found to be highly ineffective w.r.t. the applied efficiency measure. Simple heuristic methods are presented and found to produce high-quality triangulations. Moreover, we introduce a method by which any non-minimal triangulation may be turned into a minimal one.

Furthermore, we present a stochastic algorithm based on a technique known as simulated annealing by which the optimal solution to $\mathcal{NP}$-complete problems may be found with probability arbitrarily close to 1.

Key words. graph triangulation, graph algorithms, causal probabilistic networks, belief networks, $\mathcal{NP}$-complete problems, simulated annealing.

Contents

Chapter 1

Introduction

1.1 Background

The notion of triangulated (chordal, decomposable, ...) graphs applies to problem solving within as widely different areas as solution of sparse symmetric systems of linear equations, pedigree analysis, and evidence propagation in belief graphs. Our interest in graph triangulation originates from the latter of these areas. The actual evidence propagation technique was discovered by Lauritzen & Spiegelhalter (1988). This technique is general in the sense that if the belief graph is triangulated no additional dependency relations will ever be needed in order to ensure correctness during propagation of new evidence. The present paper deals with the problem of achieving approximate optimal triangulations of (undirected) graphs. In the context of belief graphs the notion of optimality relates to the sum total of the state space sizes of the cliques of the triangulated graph. This optimality task has been proved to be $\mathcal{NP}$-complete (Arnborg, Corneil & Proskurowski 1987).

The basic technique applied to triangulate a graph G is to add the extra edges T produced by eliminating the vertices of G one by one. (A vertex v is eliminated by adding edges such that the vertices adjacent to v are pairwise adjacent and by subsequent deletion of v and its incident edges.) This technique is not guaranteed to produce minimal triangulations neither in terms of number of edges nor in terms of the size of the state space when the vertices are selected at random. Furthermore, in average these quantities are typically far above those of a minimum triangulation.

Triangulating graphs using the elimination technique is essentially a problem of establishing a sequential ordering of the vertices specifying the order in which they should be eliminated. Hence the term *elimination ordering* (or *sequence*) refers to such an ordering. Several algorithms have been suggested for establishing elimination orderings which have certain desired properties. As the task of establishing an elimination ordering may be perceived as a sorting problem such algorithms are quite often referred to as *sorting algorithms*. Since most of the algorithms apply some sort of search strategy the term *search algorithms* is also frequently used.

In Chapter 2 we consider and evaluate some well-known search algorithms. We also develop a method by which a set $R \subset T$ of redundant edges in a triangulation T may be pointed out such that $T \setminus R$ is a minimal triangulation. This method is then applied to the triangulations produced by one of the conventional search algorithms which does not guarantee minimality. However, simple heuristic algorithms turn out to be clearly superior to all of the more sophisticated algorithms (at least for the employed test graphs), but no guarantees can be made as to the optimality of these heuristics.

In Chapter 3 we approach the sorting problem from the point of view of stochastic optimisation. That is, the set of all possible elimination orderings is regarded as a search space within which we perform a controlled random walk to find an ordering of minimum cost. By applying this technique we obtain slightly better triangulations than those obtained by the heuristic algorithms.

1.2 Terminology

A graph G consisting of a finite set of *vertices* V and a set of *edges* E (vertex pairs from V) is denoted $G = (V, E)$. Throughout this paper we shall use $n = |V|$ to denote the number of vertices and $e = |E|$ to denote the number of edges. Let A be a set of vertices and B be a set of edges. Then $G(A)$ and $G(B)$ denote the subgraphs of G induced by A and B, respectively. $V(B)$ ($E(A)$) denotes the subset of vertices (edges) induced by B (A). An *ordering* of V is a bijection $\# : V \leftrightarrow \{1, 2, \dots, n\}$. $G_\# = (V, E)$ is an *ordered graph.*[1] A graph G is called a *belief graph*, if a finite set of elements, called *states*, is attached to each vertex in G. The base 2 logarithm of the number of states (state size) of a vertex v is denoted $w(v)$ (the *weight* of v).

Two vertices $v, w \in V$ are *adjacent* (or *neighbours*) if $\{v, w\} \in E$. The set

$$\mathrm{adj}(v, G) = \{w \in V \mid \{v, w\} \in E\}$$

is the vertices adjacent to v in G. The set of vertices *monotonely adjacent* to v in G is defined by

$$\mathrm{madj}(v, G) = \mathrm{adj}(v, G) \cap \{w \in V \mid \#(v) < \#(w)\}$$

A *complete subgraph* of G is a set of vertices all of which are pairwise adjacent. A *clique* is a maximal complete subgraph of G. A vertex $v \in V$ is said to be *simplicial* if $\mathrm{adj}(v, G)$ induce a complete subgraph of G (which then together with v form a clique in G).

A vertex $v \in V$ is said to be *eliminated* if the adjacency set of v is made a complete subgraph of $G = (V, E)$ and v and its incident edges are removed from G. Let $G_i = (V_i, E_i)$, where $i = 1, \dots, n$, denote the ordered graph obtained by eliminating in order

[1] When it appears clearly from the context that a graph is ordered we shall often drop the #-index.

the vertices $\#^{-1}(1), \ldots, \#^{-1}(i-1) \in V$ from the ordered graph $G = (V, E)$, where

$$\begin{aligned} V_i &= \{v \in V \mid \#(v) > i\} \\ E_i &= \{\{v, w\} \in E_{i-1} \mid v, w \in V_i\} \cup \\ &\quad \{\{v, w\} \subseteq V_i \mid v, w \in \mathrm{madj}(u), \text{ where } \#(u) = i\} \\ E_0 &= E \end{aligned}$$

By $C_i(G)$ we denote the clique, if any, produced by eliminating vertex $\#^{-1}(i)$ in G_{i-1}. If $C_i(G)$ is a clique it consists of the set $\{\#^{-1}(i)\} \cup \{\mathrm{adj}(\#^{-1}(i), G_{i-1})\}$.

A *chain* in $G = (V, E)$ is a sequence of distinct vertices $<v_1, \ldots, v_k>$ such that $\{v_i, v_{i+1}\} \in E$ for $i = 1, \ldots, k-1$. A *cycle* is a chain $<v_1, \ldots, v_k>$ with the exception that $v_1 = v_k$ and such that $k > 3$. A *chord* in a cycle is an edge $\{v_i, v_j\}$ such that $1 < |i - j| < k - 2$ (i.e. $\{v_i, v_j\}$ connects the two non-consecutive vertices v_i and v_j in the cycle). G is *triangulated* (or *chordal*) if any cycle of length greater than 3 has a chord.

Let $G_\# = (V, E)$ be an ordered graph. Then G^t denotes the triangulated graph obtained by eliminating the vertices of G in the order defined by $\#$. G^t has vertex set V and edge set $E \cup T$, where the set T of *fill edges* is a *triangulation* of G. A fill edge $f \in T$ is *redundant* if $T \setminus \{f\}$ is also a triangulation of G. T is *minimal* if for any $T' \subset T$ (proper containment) $G' = (V, E \cup T')$ is not triangulated. T is *minimum* if for any triangulation T' of G, $|T| \leq |T'|$. $T_i(G)$ denotes the fill edges produced by eliminating vertex $\#^{-1}(i)$ in G_{i-1} and

$$T(G) = \bigcup_{i=1}^{n} T_i(G)$$

Note that $G^t = (V, E \cup T(G))$.

Let $C = \{v_1, \ldots, v_k\}$ represent a clique of a triangulated graph G. The *size* of C is $s(C) = k$ and the *weight* (base 2 logarithm of the state size) of C is

$$w(C) = \sum_{i=1}^{k} w(v_i)$$

The size of G, $s(G)$, is the sum of the sizes of all of its cliques, and the weight of G is

$$w(G) = \log_2 \sum_{C} 2^{w(C)}$$

For $C \subseteq V$,

$$\mathrm{adj}(C, G) = \{v \mid v \in V \setminus C, v \in \bigcup_{w \in C} \mathrm{adj}(w, G)\}$$

denotes the set of vertices separating G into components C and $V \setminus (C \cup \mathrm{adj}(C, G))$.

Throughout this paper we assume that the graphs considered are *connected* (i.e. for each pair of distinct vertices v, w there is a chain $<v, \ldots, w>$). Note that connectivity is invariant under elimination.

Chapter 2

Strategic Search

2.1 Well-known Triangulation Methods

Rose, Tarjan & Lueker (1976) provided the following useful characterisation of the fill edges produced by the elimination technique.

Theorem 1 (Rose et al. (1976)) *Let* $G = (V, E)$ *be an ordered graph. Then* $\{v, w\}$ *is an element in* $T(G)$ *if and only if there is a chain* $<v, v_1, \ldots, v_k, w>$ *in* G *such that* $\#^{-1}(v_i) < min(\#^{-1}(v), \#^{-1}(w))$ *for all* $1 \leq i \leq k$.

Note that for $k = 0$ (i.e. $\{v, w\} \in E$) the condition is regarded as satisfied. From that theorem we may formulate the following corollaries.

Corollary 1 *Let* $G = (V, E)$ *be an ordered graph and* $<v_1, \ldots, v_k, w>$ *be a chain in* G*, where* $\#^{-1}(v_i) < \#^{-1}(w)$ *for all* $i \leq k$*. Then* $N \setminus \{v_1, \ldots, v_k\} \subseteq adj(w, G^t)$*, where* $N = \bigcup_{v \in \{v_1, \ldots, v_k\}} adj(v, G)$.

Proof: Follows directly from Theorem 1. □

Corollary 2 *Let* $G_{\#_a} = (V, E)$ *and* $G'_{\#_b} = (V, E)$ *be ordered graphs, where* $\#_a^{-1}(i+1) = \#_b^{-1}(i+1)$*. Then* $G_i = G'_i$ *for all* $k \leq i \leq n$*, where* $0 \leq k \leq n$.

Proof: Clearly G_i and G'_i consist of the same vertex set, and from Theorem 1 it is equally clear that they also have identical edge sets. □

It appears from Theorem 1 that in order to profit by it in establishing an elimination ordering (e.g. to keep the size of T as small as possible) the ordering should be constructed backwards. The two most well-known ordering algorithms (lexicographic search by Rose et al. (1976) and maximum cardinality search by Tarjan & Yannakakis (1984)) are based on this principle.

The basic idea in the backwards approach is to separate the vertices already ordered in a way such that there is no chains between them of the type described above. The maximum cardinality search does that by continuously selecting a vertex (to be ordered next) with the highest number of ordered neighbours. The lexicographic search does something similar, but in addition it is guaranteed to produce minimal triangulations (in $O(ne)$ time), whereas the maximum cardinality search is not. Maximum cardinality search has the property of not adding edges to an already triangulated graph and hence may be used as an efficient test for graph chordality as it runs in $O(n + e)$ time.

Fujisawa & Orino (1974) have described an extended elimination technique which does not necessarily add fill edges between all non-adjacent vertices in the adjacency set of a vertex being eliminated. This extended elimination strategy produces minimal triangulations no matter which ordering of vertices is employed.

By comparing the ordinary elimination technique with theirs it appears that there are two distinct cases in which the ordinary technique fails to produce minimal triangulations. The two cases are described below.

Let $G = (V, E)$ be a graph and let $v \in V$ be the vertex to be eliminated next.

Case I. $\{v\} \cup \text{adj}(v, G) = V$. If T is a triangulation of $G' = G \setminus \{v\}$ then it is also a triangulation of G. Hence we need not complete $\text{adj}(v, G)$ when v is eliminated, it suffices to just remove v from G.

Case II. (See Figure 2.1) $\text{adj}(v, G)$ separates G into components $\{v\}, C_1, \ldots, C_m$, $m > 1$. If

$$\begin{aligned} G_0 &= (\text{adj}(v, G), E_0), \\ G_1 &= (C_1 \cup \text{adj}(C_1, G), E_1), \\ &\vdots \\ G_m &= (C_m \cup \text{adj}(C_m, G), E_m) \end{aligned}$$

are triangulated so is $G'' = (V, E \cup T)$, where $T = \bigcup_{i=0}^{m} E_i \setminus E$ is a triangulation of $G' = \bigcup_{i=0}^{m} G_i$. Again we need not complete $\text{adj}(v, G)$, it suffices to complete each $\text{adj}(C_i, G)$.

Fujisawa & Orino (1974) have proved that by extending ordinary vertex elimination by the results from these two observations any elimination ordering produces a minimal triangulation. More precisely they have formulated and proved the following theorems.

Theorem 2 (Fujisawa & Orino (1974)) *Let v be a vertex of a connected graph $G = (V, E)$ such that $\{v\} \cup adj(v, G) = V$, and let T be a minimal triangulation of the section graph $G \setminus \{v\}$. Then T is a minimal triangulation of G.*

Theorem 3 (Fujisawa & Orino (1974)) *Let v be a vertex of a connected graph $G = (V, E)$ such that $\{v\} \cup adj(v, G) \subset V$, and let $adj(v, G)$ be a separator with components $\{v\}, C_1, \ldots, C_m$, $m \geq 1$. Furthermore, let F be the set of edges needed to make each $adj(C_i, G)$ a complete subgraph and let $G \setminus \{v\}$. If T is a minimal triangulation of G' then $T \cup F$ is a minimal triangulation of G.*

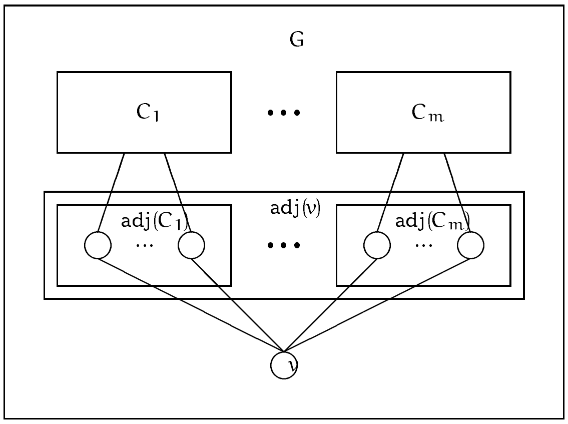

Figure 2.1: By eliminating v we need not complete $\mathrm{adj}(v)$, it suffices to complete each $\mathrm{adj}(C_i)$.

Based on these theorems they have formulated an $O(n(e+f))$ time algorithm for finding a minimal triangulation T of a graph $G = (V, E)$, where $f = |T|$.

Both the lexicographic search and the extended elimination algorithms produce minimal orderings, but such orderings are not necessarily close to a *minimum fill*[1] ordering. On the other hand a minimum fill ordering is not necessarily a minimum weight ordering. If, for example, we let the solid circles in Figure 2.2 represent vertices with large relative weights then only (a) is a minimum weight triangulation, whereas both (a) and (b) are minimum fill triangulations. However, we claim that a triangulation T_1 is very unlikely to have weight less than a triangulation T_2 if $|T_1|$ is significantly greater than $|T_2|$. As will be demonstrated in Section 2.3 this claim seems empirically reasonable.

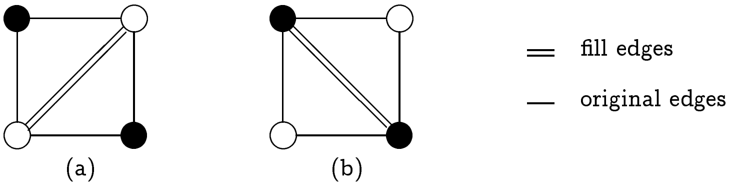

Figure 2.2: Both of the triangulations (a) and (b) are minimum fill, but only (a) is minimum weight (solid circles represent vertices with large relative weights).

Depending on the problem to be solved by graph triangulation, optimality may be defined differently. Three typical candidates are

- minimum fill

[1] We shall use the terms minimum fill, minimum weight, and minimum size triangulations as short for triangulations consisting of a minimum number of fill edges, triangulations producing triangulated graphs with minimum weight, and triangulations producing triangulated graphs for which the sum of the clique sizes is minimum, respectively.

- minimum size
- minimum weight

In order to meet these optimality criterions using the elimination technique the most immediate strategy would be to successively choose the next vertex to be eliminated to be one which produces as few fill edges as possible, the smallest clique, or the clique with the least weight, respectively. Henceforth we will call the algorithms based upon these three heuristic rules the *minimum fill*, the *minimum size*, and the *minimum weight* heuristic, respectively. Rose (1973) described the first two of these heuristics under the names *minimum deficiency* and *minimum degree* algorithms and referred to other papers (e.g. Sato & Tinney (1963) and Tinney & Walker (1967)) in which it is assumed that these algorithms produce near optimal orderings.

As noted by Rose (1973) the minimum size heuristic is fast (can be implemented to run in $O(n + e')$ time, where $e' = e + |T|$), but has some disadvantages:

- does not, in general, produce a perfect ordering[2] if the graph is triangulated
- does not, in general, produce minimal triangulations
- there exist examples for which the triangulation produced is arbitrarily greater than a minimum (fill) triangulation

However, this last feature does not really count as a disadvantage in the context of belief networks as we do not care about the number of fill edges.

The minimum fill heuristic has the advantage of producing a perfect ordering when the graph is triangulated, but has the following disadvantages:

- slightly slower than the minimum size heuristic, because the adjacency set of each vertex must be edge tested
- does not, in general, produce minimal triangulations

The minimum weight heuristic has exactly the same advantages and disadvantages as the minimum size heuristic. Note that if all vertices have equal weights these two heuristics are identical.

As the complexity of finding a minimum triangulation grows like $n!$ we are not able to run a straightforward exhaustive search, except for very small n. However, by constructing an elimination ordering successively and cutting off when the sum of the weights of the cliques produced so far exceeds the current least weight of a complete ordering we might be able to run an exhaustive search even for graphs of moderate sizes. Usually this cutting and branching technique is called *branch-and-bound*. As we are dealing with an $\mathcal{NP}$-complete problem we should not, in general, expect a branch-and-bound algorithm

[2]If $G_\# = (V, E)$ is an ordered graph then $\#$ is a perfect ordering if $T(G_\#) = \emptyset$. Any triangulated graph $G'_\# = (V, E \cup T(G_\#))$ has a perfect ordering, since $\#$ is a perfect ordering of this graph.

to find an optimal ordering within certain acceptable time bounds. That is, the algorithm must terminate prematurely when the number of vertices exceeds a certain limit or when the number of permutations cut off grows too slowly. Of course, the initial ordering heavily influences the success of this algorithm. So a branch-and-bound algorithm should preferably be cascaded (as the last element) with other faster ordering algorithms to set-up a "good" initial ordering for it in order to avoid checking too many useless orderings and in order to minimise the distance to some minimum ordering (assuming that a low cost ordering is closer to a minimum one than a high cost ordering is).

2.2 Minimal Triangulations by Recursive Thinning

When an elimination ordering has been established and the corresponding triangulation, T, of a graph, G, has been determined it might turn out that there is a subset, $T' \subset T$, such that T' is also a triangulation of G. For the minimum fill criterion T' is, of course, strictly better than T. For the minimum weight criterion T' is no worse than T, and most often T' is significantly better than T. But for the minimum size criterion T is normally better than T'. The fact that T' is most often better than T for the minimum weight criterion will become evident in the following.

In order to develop an algorithm to remove redundant fill edges we first characterise a minimal triangulation in the following way.

Theorem 4 (Rose et al. (1976)) *Let* $G = (V, E)$ *be a graph and* $G' = (V, E \cup T)$ *be triangulated. Then* T *is minimal if and only if each edge in* T *is a unique chord of a 4-cycle in* G'.

An equivalent formulation of Theorem 4 is provided by the following corollary.

Corollary 3 *Let* $G = (V, E)$ *be a graph and* $G' = (V, E \cup T)$ *be triangulated. Then* T *is minimal if and only if for each edge* $\{v, w\} \in T$ *there is a pair of distinct vertices* $\{x, y\} \subseteq adj(v, G') \cap adj(w, G')$ *such that* $\{x, y\} \notin E \cup T$.

Proof: As $\{v, w\}$ is a chord of a 4-cycle $<v, x, w, y, v>$ in G', x and y will both be adjacent to both v and w in G'. By the uniqueness of the $\{v, w\}$ chord it follows that $\{x, y\} \notin E \cup T$. □

By formulating Theorem 4 that way we easily get the next corollary.

Corollary 4 *Let* T *be a triangulation of a graph* $G = (V, E)$, *let* $G' = (V, E \cup T)$ *be the corresponding triangulated graph, and let* $\{v, w\} \in T$. *Then* $T \setminus \{\{v, w\}\}$ *is a triangulation of* G *if and only if the set of vertices* $N = adj(v, G') \cap adj(w, G')$ *induces a complete subgraph* $G'(N)$ *of* G'.

Proof: Let us first prove that if $T \setminus \{\{v, w\}\}$ is a triangulation of G then $G'(N)$ is complete. By assuming $G'(N)$ is not complete (e.g. $x, y \in N$ are non-adjacent) we get the unchorded 4-cycle $<v, x, w, y, v>$ which contradicts the definition of triangulated graphs.

Secondly we shall prove that completeness of $G'(N)$ implies that $G'(E \cup T \setminus \{\{v, w\}\})$ is triangulated. Assuming $G'(E \cup T \setminus \{\{v, w\}\})$ is not triangulated there must be an unchorded cycle of length 4 in it because if it had length > 4 there would be an unchorded cycle of length > 3 before $\{v, w\}$ was removed. As G' is triangulated this cycle must include both v and w and as the cycle has length 4 it must be a $<v, x, w, y, v>$ cycle, where $x, y \in N$. But as $G'(N)$ is complete the cycle has a chord. □

Notice that Theorem 4 and Corollary 3 characterises the fill edges of a minimal triangulation $T' \subseteq T$, whereas Corollary 4 characterises the fill edges in $T \setminus T'$. Thus an alternative proof of Corollary 4 could be based on the results of Corollary 3.

The above propositions tell us that the redundant edges of a triangulation are characterised by not being unique chords of 4-cycles. Note that this set of edges is identical to the set $\{e_i\}$ for which all e_i is a proper subset of a unique clique in the graph. To be precise we formalise this observation as:

Lemma 1 *Let $G = (V, E)$ be a triangulated graph, let $E_{\overline{u}} = \{e \in E \mid e$ is not a unique chord of a 4-cycle in $G\}$, and let $\mathcal{C} = \{C_i\}$ be the set of cliques in G. Then for all $e \in E_{\overline{u}}$ there is a unique $C \in \mathcal{C}$ such that $e \subset C$.*

Proof: Assume $e = \{v, w\} \in E_{\overline{u}}$ is a subset of more than one clique. Then there is an unchorded cycle $<v, \ldots, w, \ldots, v>$ of length greater than 3. Thus the graph is no longer triangulated which contradicts the fact that e is redundant. □

As a redundant (fill) edge, $e = \{v, w\}$, can only be subset of a single clique, C, it is now clear that this clique splits into two new cliques $C_1 = C \setminus \{v\}$ and $C_2 = C \setminus \{w\}$ the sum of weights of which is typically significantly less than the weight of C and in any case no worse than that.

A non-minimal triangulation T may become minimal by simply dropping the edges in T that do not fulfil the conditions of Corollary 3. However, it does not suffice in general to sweep through the edges just once, which is illustrated by the following example. In Figure 2.3 the vertices b and c have a common pair of non-adjacent neighbours (i.e. $\{b, c\}$ is a unique chord of the 4-cycle $<a, b, c, d, a>$) and hence we are not allowed to remove edge $\{b, c\}$. The vertices c and d do not have such a pair in common and we may remove edge $\{c, d\}$. Examining the reduced graph we now find that $\{b, c\}$ may be removed.

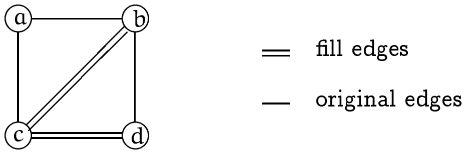

Figure 2.3: A non-minimal triangulation produced by the elimination ordering a, b, c, d.

Clearly for a fill edge which in a previous sweep was found to be a unique chord of a 4-cycle to become redundant, according to Corollary 4, one of the four edges in that 4-cycle must have been removed. So, assuming it takes k sweeps to minimise a triangulation, for $i = 2, \ldots, k$ it suffices in sweep i to test the fill edges which intersect (i.e. share an end-point (vertex)) with an edge removed in sweep $i - 1$. Depending on the number of sweeps required this observation may substantially improve the minimisation process.

Based on Corollary 4 and the above discussion we may formulate the following $O(c^2|T|^2)$ time[3] algorithm which removes redundant fill edges from a non-minimal triangulation T.

MINT $(T, G = (V, E \cup T), R)$ (initially $R = T$)
  $R' = \{e_1 \in T \mid \exists e_2 \in R : e_1 \cap e_2 \neq \emptyset\}$
  $T' = \{\{v, w\} \in R' \mid G(\mathrm{adj}(v, G) \cap \mathrm{adj}(w, G))$ is complete$\}$
  if $T' \neq \emptyset$ **then**
    return (MINT $(T \setminus T', G = (V, E \cup T \setminus T'), T')$)
  else
    return (T)

Obviously the number of sweeps required heavily depends on the order in which we arrange the fill edges and will typically be much less than $|T|$. The clearest example appears in the generalisation of the graph in Figure 2.3 (a *string graph*[4] triangulated by successively eliminating vertices in the middle of the string), where only one fill edge may be removed in each sweep if the edges are ordered according to the order in which they have been generated, whereas all of them may be removed in just one sweep if they are investigated in the reverse order. This gives us the clue of a considerably more efficient minimisation algorithm. The existence of such an algorithm is verified by the proof of Theorem 5. However, before presenting this theorem let us have a closer look at the rationale leading to it.

An immediate idea emerging from the above discussion is: Is one sweep sufficient in general provided that the fill edges are ordered appropriately? There are two obvious ways of ordering the fill edges of a triangulated graph. The first being the order of creation (i.e. corresponding to the elimination ordering) and the second being the reversed of the first. As became clear from the string graph example above the first ordering strategy is not applicable, whereas the second could be. Notice that these orderings are not uniquely defined as the fill edges are created in groups (one group, possibly empty, for each vertex elimination). There seems not to be any obvious way of ordering fill edges belonging to the same group. So in order for reversed orderings to be minimised in one sweep it is a requirement that each group of edges can be minimised in one sweep no matter how they are ordered. The example in Figure 2.4 shows that this requirement cannot be satisfied. The elimination of vertex a produces three fill edges (none of which are necessary as the graph is already triangulated). In order to minimise this group (which is assumed to be the last group of a series) we first apply MINT on edge $\{d, c\}$ and find

[3]The constant c expresses the density of the graph and equals the cardinality of the intersection of the adjacency sets of the vertices which constitute a fill edge.

[4]The term *string graph* denotes a graph which topologically resembles a string of pearls.

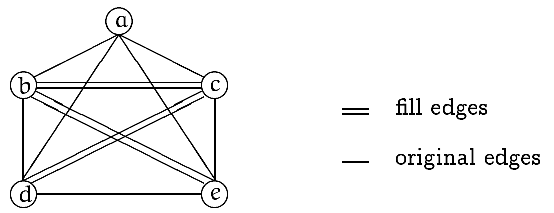

Figure 2.4: The group of fill edges generated by the elimination of a single vertex cannot be minimised in one sweep.

that it is redundant and we remove it. Then we apply MINT on $\{b, e\}$ and find that it cannot be removed. Finally $\{b, c\}$ is found to be redundant and it is removed. Now $\{b, e\}$ becomes redundant as well. This shows us that in order to (locally) minimise a group of fill edges MINT must keep its recursive mode of operation. Furthermore, the example in Figure 2.3 demonstrates that the groups have to be ordered according to the reverse ordering scheme if a single sweep through the sequence of groups should be enough to minimise a triangulation. These were the considerations leading to Theorem 5.

Theorem 5 *Let* $G = (V, E)$ *be an ordered graph,* $T = T(G)$*, and* $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ *where* $F_i = \text{MINT}(T_i(G), G')$*,* $G' = (V, E \cup F)$*, and* $F = \bigcup_{j=1}^{i} T_j(G) \cup \bigcup_{k=i+1}^{n} F_k$*. Then* $T' = \bigcup_i F_i$ *is a minimal subset of* T*.*

Proof: Let $G' = (V, E \cup T)$ be the initial triangulated graph. First we observe that since G' is triangulated the removal of a fill edge that according to Corollary 4 is redundant does not introduce unchorded 4-cycles (the resulting graph will still be triangulated).

Now, assume we have just finished the recursive thinning of $T_i(G)$, and that $\{u, v\} \in F_i$. Note that $\{u, v\}$ was once produced by the elimination of a vertex x and hence is a unique chord of the 4-cycle $<u, w, v, x, u>$. We will prove that $\{u, v\}$ is not redundant in the final triangulated graph $G'' = (V, E \cup T')$ either. Thus T' will be minimal. Let us therefore assume that $\{u, v\}$ is in fact redundant in G'' (which should lead to a contradiction). As we do not add new edges during the thinning process the only way in which $\{u, v\}$ could have become redundant is that one or more of the edges in the 4-cycle $<u, w, v, x, u>$ have been removed. Let us consider the case where just one edge, $\{v, x\}$ say, has been removed. According to Theorem 1 there is a chain $<v, v_1, \ldots, v_k, x>$ from v to x in G (and hence also in G'') such that all vertices $v_1, \ldots, v_k$ are eliminated before both v and x. Next, Corollary 1 tells us that $\{u, v, w\} \cap \bigcup_{y \in \{v_1, \ldots, v_k\}} \text{adj}(y, G) = \emptyset$. Otherwise, there would be a fill edge $\{w, x\}$ (which would make $\{u, v\}$ redundant) and/or a fill edge $\{u, v\}$ which contradicts the fact that the elimination of x produced $\{u, v\}$. Thus, there is an unchorded cycle $<u, w, v, v_1, \ldots, v_k, x, u>$ the length of which is clearly greater than 3. Repeating this argument in the cases where two, three or all four edges have been removed result in similar conclusions. This completes the proof. □

Based on Theorem 5 we may now formulate a significantly more efficient minimisation algorithm, FMINT.

FMINT ($\mathcal{F}, G = (V, E \cup T)$)
  for $i = n$ **downto** 1 **do**
    $R = \mathcal{F}[i] \setminus$ MINT ($\mathcal{F}[i], G = (V, E \cup T)$)
    $T = T \setminus R$
  return (T)

$\mathcal{F}$ is an array of length R of sets of fill edges ordered according to the order of creation when applying the elimination procedure.

The magnitude of the improved performance achieved by FMINT compared to MINT depends on the distribution of the fill edges over the series of the n groups. The more equal they are distributed the better performance is achieved. In the worst case (i.e. all edges in the same group) no improvement is obtained. If the edges are equally distributed ($\frac{|T|}{|V|}$ edges in each group) FMINT runs in $O((c\frac{|T|}{|V|})^2)$ time. However, it is the size of the greatest group that determines the performance of FMINT. If the size of the greatest group is $\frac{|T|}{m}$, $m \geq 1$, FMINT runs in $O((c\frac{|T|}{m})^2)$ time. So for example even if one fourth of all fill edges belongs to the same group, FMINT will run 16 times as fast as MINT.

Letting $G = (V, E)$ be an ordered non-triangulated graph it appears from the above discussion that by approaching the graph triangulation problem by means of vertex elimination the fill edges $T'' = \bigcup_{i=k}^{n} T_i(G)$ seem to prevent a proper thinning of the fill edges $T' = \bigcup_{i=1}^{k-1} T_i(G)$ before T'' has been thinned out. An idea emerging naturally from this observation is: Is it possible, in general, to obtain minimal triangulations by applying MINT on each group $T_i(G)$ before proceeding to the elimination of vertex $\#^{-1}(i+1)$? In case of an affirmative answer we have found that the global search necessary to implement a triangulation algorithm based on the results by Fujisawa and Orino can be replaced by a local search. This seems intuitively very unlikely to be the case. The following example clearly shows that the question unfortunately must be returned a negative answer. Consider the example in Figure 2.5 and assume that vertex v is to be eliminated (cf. Case II above) which will produce a fill edge $\{u, w\}$. Theorem 3 states that $\{u, w\}$ is necessary if u and w are connected by a path not including v and that it is redundant otherwise. Since the problem of determining if such a path exists cannot be solved locally our previous results which apply to triangulated graphs are insufficient when we deal with non-triangulated graphs.

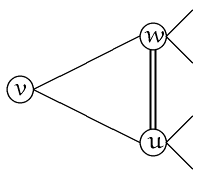

Figure 2.5: Fill edge $\{u, w\}$ is only necessary if u and w are connected by a path not including v.

However, note that in the special case where u and w are eliminated last we may remove $\{u, w\}$ irrespective of the topology of the rest of the graph as $\{u, w\}$ will be reproduced if u and w *are* connected by a path not including v (cf. Theorem 1).

2.3 Evaluating the Algorithms

In order to assess the virtues and vices of the various triangulation algorithms we want to perform a series of tests. These tests will be based on two "real world" graphs and on two artificial ones.

The two "real world" graphs, called Medianus-I[5] and Medianus-II originate from the development of the MUNIN[6] system (Andreassen, Jensen, Andersen, Falck, Kjærulff, Woldbye, Sørensen, Rosenfalck & Jensen 1989, Olesen, Kjærulff, Jensen, Jensen, Falck, Andreassen & Andersen 1989). Medianus-II is a modified version of Medianus-I (see Figures 2.6-2.7). Both of these graphs have a layered structure, which means that the number of fill edges required should be moderate.

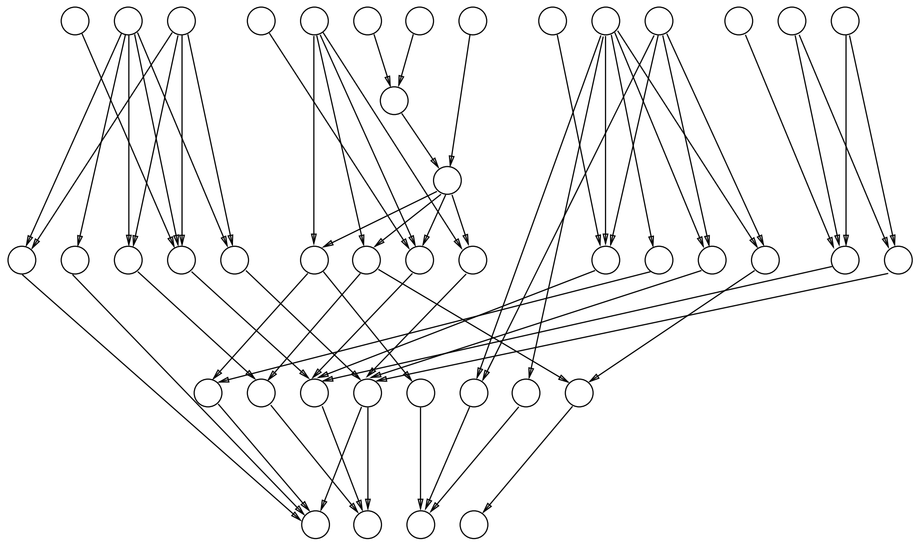

Figure 2.6: The Medianus-I graph.

For a graph $G = (V, E)$ to be connected a minimum of $n - 1$ edges are required. The maximum number of edges in G is

$$\binom{n}{2} = \frac{n^2 - n}{2}$$

[5]Bayesian inference network describing relations between disorders, pathophysiological features and measurements for the human median nerve.

[6]Knowledge based system for diagnosing diseases and malfunctions in the human neuromuscular system.

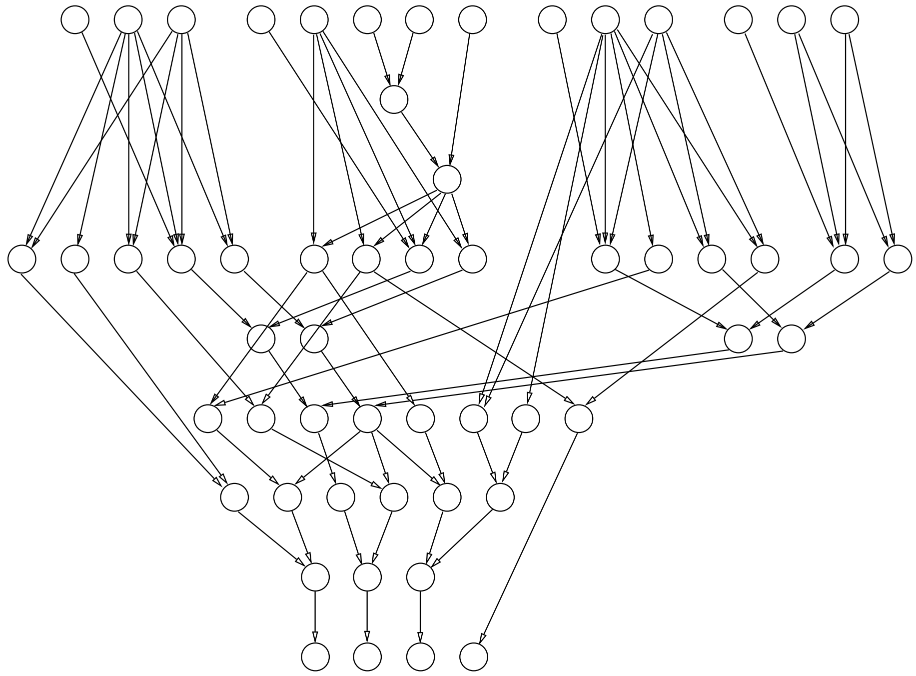

Figure 2.7: The Medianus-II graph.

in which case G is complete.

The artificial graphs consist of a sparse and a dense graph. The notions of sparsity and density are based on the following definition.

Definition 1 Let $G = (V, E)$ be a (connected) graph. G is sparse if $n - 1 \leq e \leq \frac{1}{10}\left(\frac{n^2-n}{2}\right)$ (i.e. the number of edges is less than or equals one tenth of the maximum number of edges). G is dense if $e \geq \frac{1}{4}\left(\frac{n^2-n}{2}\right)$. □

From this definition follows that sparse graphs are only defined for $n \geq 20$, whereas dense graphs are defined for all $n \geq 0$.

We let a computer generate the graphs by first making a string graph (to assure that the graphs will be connected) and then adding edges at random by visiting the vertices one by one to assure a fairly equal distribution.

In the implementation of the algorithms for which the next vertex to be enumerated is not selected at random, but according to some ranking strategy, ties are solved by picking one of the equally ranked vertices using a random number generator.

For the Medianus graphs the algorithms were ran 400 times, whereas for the artificial graphs (let us call them Sparse and Dense) the number of runs were reduced to 100, but as will become quite clear below the conclusion is not influenced by that.

The algorithms tested are simple random elimination, lexicographic search, maximum cardinality search, extended random elimination, the FMINT algorithm applied to the triangulations produced by simple random elimination and by maximum cardinality search, and finally the three heuristic algorithms minimum fill, minimum size, and minimum weight. The minimum weight heuristic is the only one that takes the weight of the vertices into account when selecting the next vertex to be eliminated.

The algorithms were first ran on the graphs with the vertices having different weights (for the Medianus graphs the state sizes lies in the range from 3 to 21 with an average of approximately 6, and for the artificial graphs the state sizes are uniformly distributed between 2 and 5). Subsequently all orderings first obtained, except those produced by the minimum weight heuristic, were applied to the graphs with all vertices being of equal weight (state size 6 for the Medianus graphs and 2 for the artificial ones).

For each algorithm we record the following two key figures for a triangulated graph $G = (V, E \cup T)$: Number of fill edges ($|T|$) and weight of G ($w(G)$). Tables 2.1-2.4 show the minimum, average/median and maximum of these two key figures for the four graphs mentioned above. Note that the weight figures in the middle column represent the median weights (i.e. the base 2 logarithm of the average size of the state space).

The algorithms are ordered in Tables 2.1-2.4 according to the median weight of the triangulated graphs in the case where the vertices are of different weight (the reason being that this is the key figure from the point of view of belief graphs).

The most immediate result of the test runs is the overwhelming superiority of the heuristic algorithms no matter which kind of measure is employed (i.e. $|T|$ or $w(G)$). This superiority becomes even more distinct when the results of the test runs are displayed

	Minimum			Average/median			Maximum		
Algorithm[1]	|T|	$w(G)^2$	$w(G)^3$	|T|	$w(G)^2$	$w(G)^3$	|T|	$w(G)^2$	$w(G)^3$
Random elimination	100	27.75	30.28	156	41.66	48.36	244	47.25	54.61
Ext. random elim.	63	22.84	24.65	100	30.94	36.26	175	36.19	41.44
Lexicographic search	68	23.44	25.05	100	30.29	35.30	177	36.50	41.47
Thinning (random)	62	22.86	24.66	98	30.14	35.51	162	35.41	41.69
Max. cardinality	68	23.63	25.27	100	29.00	31.97	136	32.63	36.49
Thinning (max card)	63	22.86	24.58	82	26.42	28.56	111	28.30	31.47
Min. size heuristic	57	22.71	24.55	58	23.27	24.88	60	24.17	26.15
Min. fill heuristic	57	22.71	24.55	58	23.14	24.80	61	24.79	26.31
Min. weight heuristic	57	22.94	− − −	57	22.99	− − −	57	23.08	− − −

1: 400 runs
2: Different vertex state sizes (between 3 and 21)
3: Equal vertex state sizes (= 6)

Table 2.1: Key figures for the triangulation of the Medianus-I graph.

	Minimum			Average/median			Maximum		
Algorithm[1]	|T|	$w(G)^2$	$w(G)^3$	|T|	$w(G)^2$	$w(G)^3$	|T|	$w(G)^2$	$w(G)^3$
Random elimination	134	29.28	32.08	213	46.35	50.32	345	53.86	56.94
Lexicographic search	98	22.28	23.61	145	30.59	33.71	253	35.21	39.57
Ext. random elim.	89	21.67	23.27	130	30.07	33.36	198	36.01	39.77
Thinning (random)	83	21.09	22.99	129	29.97	32.53	209	35.74	37.94
Max. cardinality	101	22.38	23.96	133	29.17	31.88	188	35.28	38.82
Thinning (max card)	87	21.61	23.17	116	26.93	28.87	153	31.68	34.85
Min. size heuristic	74	20.50	22.11	79	22.19	24.27	86	24.88	26.28
Min. fill heuristic	74	20.57	22.15	78	21.78	23.47	84	23.49	24.89
Min. weight heuristic	76	20.65	− − −	76	20.65	− − −	76	20.65	− − −

1: 400 runs
2: Different vertex state sizes (between 3 and 21)
3: Equal vertex state sizes (= 6)

Table 2.2: Key figures for the triangulation of the Medianus-II graph.

	Minimum			Average/median			Maximum		
Algorithm[1]	|T|	$w(G)^2$	$w(G)^3$	|T|	$w(G)^2$	$w(G)^3$	|T|	$w(G)^2$	$w(G)^3$
Random elimination	283	32.86	19.72	390	48.31	26.19	565	54.77	31.46
Max. cardinality	222	27.24	16.51	305	36.61	20.57	406	41.46	23.42
Lexicographic search	217	26.41	16.49	297	33.77	19.84	405	38.25	22.82
Thinning (random)	184	25.01	15.59	254	32.95	18.83	339	37.04	22.20
Ext. random elim.	204	24.72	15.88	260	32.86	18.91	343	37.20	21.09
Thinning (max card)	182	24.62	15.42	224	31.36	17.53	300	35.97	20.53
Min. size heuristic	149	23.05	14.36	157	25.93	15.11	169	28.67	16.37
Min. weight heuristic	156	25.06	− − −	158	25.34	− − −	160	25.92	− − −
Min. fill heuristic	149	24.82	14.57	151	24.97	14.60	153	25.61	14.74

1: 100 runs
2: Different vertex state sizes (between 2 and 5)
3: Equal vertex state sizes (= 2)

Table 2.3: Key figures for the triangulation of the Sparse graph.

	Minimum			Average/median			Maximum		
Algorithm[1]	|T|	w(G)[2]	w(G)[3]	|T|	w(G)[2]	w(G)[3]	|T|	w(G)[2]	w(G)[3]
Random elimination	538	62.84	37.33	677	73.57	41.99	795	77.40	44.83
Max. cardinality	499	59.14	34.82	567	66.41	38.21	615	67.92	39.21
Lexicographic search	432	54.39	32.59	538	65.41	36.73	644	70.84	40.80
Ext. random elim.	439	53.72	31.70	525	63.92	36.29	633	69.94	41.00
Thinning (random)	412	52.88	31.74	528	62.73	36.25	638	66.87	39.66
Thinning (max card)	438	56.42	32.92	481	59.57	34.26	545	62.53	35.72
Min. size heuristic	384	55.01	31.69	387	56.37	32.06	394	57.09	32.22
Min. fill heuristic	374	54.40	31.82	390	56.35	32.31	398	57.81	32.73
Min. weight heuristic	388	53.48	– – –	388	53.48	– – –	388	53.48	– – –

1: 100 runs
2: Different vertex state sizes (between 2 and 5)
3: Equal vertex state sizes (= 2)

Table 2.4: Key figures for the triangulation of the Dense graph.

graphically as in Figure 2.8 which shows the graph weights obtained for the Medianus-I graph with different vertex weights (a similar representation of graph weights employing equal vertex weights yields essentially the same result). The 400 figures are ordered in 50 intervals and displayed in a bar chart with the height of each bar being proportional to the number of figures in the corresponding interval. (However, if the interval width in terms of pixels[7] is less than 1, given the width of the display area provided, the number of intervals for the particular set of figures is reduced such that 1 pixel represents the width of one interval.)

The tiny marks in Figure 2.8 indicate the overall minimum weight (in this case found by the minimum size heuristic) and the median weights for each algorithm.

A similar graphical representation of the number of fill edges for the Medianus-I triangulations looks very much like Figure 2.8. The main difference is that for the heuristic algorithms the dispersion is even less than in Figure 2.8.

Representing the results obtained from the Medianus-II, the Sparse, and the Dense graphs in the same fashion provide bar charts of very similar appearance.

Figure 2.9 shows enlarged representations of the graph weight results for the four graphs tested obtained by the heuristic methods. The most striking result being that the range and dispersion of results for the minimum weight heuristic is very small, except maybe for the Sparse graph. This seems, however, to be a mixed blessing as for the Medianus and the Sparse graphs it is not able to find triangulations producing graph weights that are as small as those produced by some of the triangulations found by the minimum size heuristic which on the other hand produces very dispersed results.

Instead of breaking ties at random it might be possible to achieve less dispersed results by applying some sort of strategic tie breaking. To find a good strategy we should be able to identify some characteristic features of the choices made during the construction

[7] A pixel is the atomic element in computer graphics

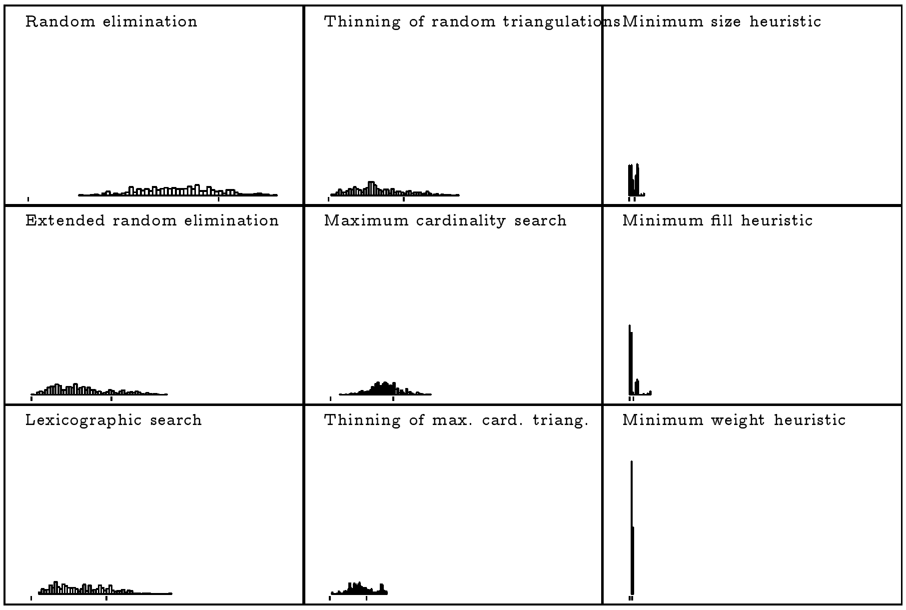

Figure 2.8: Distribution of graph weights obtained by 400 runs on the Medianus-I graph with different vertex weights.

of the best orderings. However, by inspecting the elimination process pertaining to these orderings it seems impossible to find any pattern, at least from a local point of view, in the way ties are broken.

A way of improving the performance of the heuristic algorithms would be to try out all orderings that might possibly be produced by the algorithms. That is, for every tie to record the candidate vertices and then to perform a depth-first search in the tree of orderings successively constructed. However, by counting the number of candidate vertices for every tie, it was experimentally found that for the Medianus-I graph we may expect the number of possible orderings which can be achieved by the minimum size and the minimum fill heuristics to be as high as $2 \cdot 10^7$. This means that an exhaustive search in the tree of candidate orderings is absolutely infeasible for these two algorithms. The tree size for the minimum weight heuristic amounts to only 24 orderings for the Medianus-I graph. On the other hand, not much can be gained by an exhaustive search in this case as the dispersion is rather modest.

A total of 1000 triangulations were produced by the heuristic algorithms and, surprisingly, it appeared that all of these were minimal. However, as mentioned in Section 2.1 none of the heuristic algorithms are, in general, guaranteed to produce minimal triangulations; although in practice there seems to be no reason to run the FMINT algorithm to ensure minimality.

Although the heuristic algorithms and especially the minimum weight seem to pro-

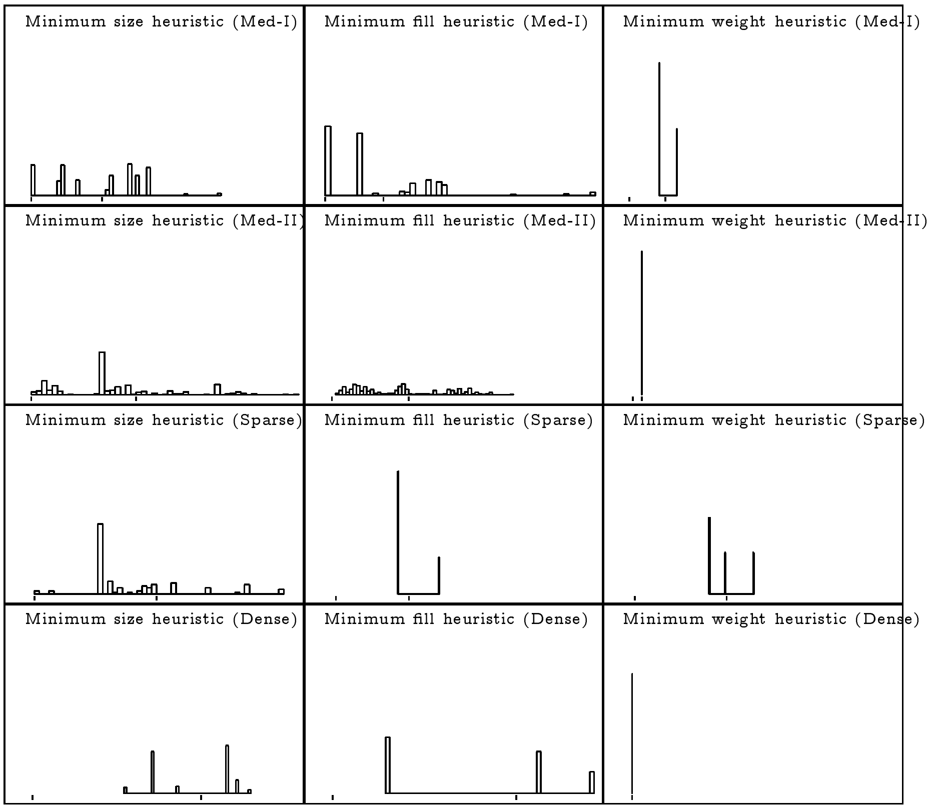

Figure 2.9: Distribution of graph weights obtained by running the heuristic algorithms 400 times on the Medianus graphs and 100 times on the artificial graphs. Different vertex weights.

duce fairly satisfactory triangulations we still face a major question: How far are we from a minimum weight triangulation?

Assuming the distance[8] between the elimination orderings produced by the heuristic algorithms and a minimum weight ordering, say #, is modest the branch-and-bound algorithm (cf. Section 2.1) might provide an answer to our question. This algorithm has the essential property of searching in a very systematic way as all permutations with identical prefix $(v_1, \ldots, v_k)$ are tested before permutations with prefix $(v_1, \ldots, v_i)$, where $i < k$. This has the important implication that the number of iterations needed by the branch-and-bound algorithm to reach # might be orders of magnitude greater than the minimum distance to #. This might, however, not necessarily mean that the algorithm does not have any practical significance. If the cardinality of the set, S, of orderings

[8]Here we define the distance between two orderings in terms of the minimum number of interchanges between neighbouring (i.e. adjacent in the ordering) vertices in one of the orderings required to reach the other (cf. Chapter 3).

the weights of which differs insignificantly from the minimum weight, is large enough and the distances between the members of S are generally relatively large there might be a chance that the branch-and-bound algorithm is a practical solution to refinement of orderings found by heuristic methods. On the other hand, if the cardinality of S is small we should, in general, expect an algorithm which chooses the next permutation in a non-deterministic manner to perform considerably better.

The best orderings known yet for the Medianus-I graph were found by the minimum size heuristic. About one seventh (56 out of 400) of the orderings found by this heuristic produced triangulated graphs with practically identical weights. A sample of 10 of these orderings were tested two by two to measure the distances between them (i.e. a total of 45 measure were calculated). It turned out that no two of these orderings were identical and that the distances between them were equally distributed in the interval from 20 to 58. The result of this little test indicates that we might expect a lot of orderings to be minimum and the distances between them to be quite significant.

Still, in order to get answers to the above questions we need to know, at least by high probability, what the minimum weight triangulation of a graph is. In the next section we consider a stochastic algorithm which with probability arbitrarily close to 1 finds a minimum triangulation, and hence we postpone the discussion of the utility of the branch-and-bound algorithm until the *simulated annealing* algorithm has been evaluated.

Chapter 3

Stochastic Search

3.1 Simulated Annealing

Simulated annealing is a stochastic algorithm which is used for finding global minimum cost configurations for $\mathcal{NP}$-complete problems with cost functions having many local minima. The trick applied to avoid getting stuck in local minima is to accept cost function increases with positive probability.

For a given optimisation problem to be suitable for the annealing algorithm it should be possible to describe it in terms of a *solution space* S the elements of which all relate to some cost function value $c(s)$, $s \in S$, and which are connected by some appropriate neighbouring structure. That is, S has a certain topology which has the essential feature that for every pair of configurations, $s_i, s_j \in S$, s_j should be reachable from s_i, where reachability is defined as a series of *transitions*, $s_k \rightarrow s_l$, each of which consist of a *perturbation* imposed on configuration s_k such that it changes to a neighbouring configuration s_l. The *distance* between two configurations $s_i, s_j \in S$ is defined as the least number of transitions required to reach s_j from s_i. The *radius* of a space S is defined as the maximum distance between two configurations in S. Of course the radius depends on the perturbation scheme employed.

In our case S is the set of all possible orderings of the vertices V of a graph $G = (V, E)$ (i.e. $|S| = n!$, where $n = |V|$) and a transition $s_i \rightarrow s_j$ is simply defined as the interchange of two vertices in ordering s_i resulting in ordering s_j. As the aim is to minimise the weight of a triangulated graph $G^t = (V, E \cup T(G_\#))$ the cost function is defined as

$$c(\#) = w(G^t)$$

where $G_\# = (V, E)$. So we want to find an optimal ordering $\#^*$ such that $c(\#^*) \leq c(\#)$ for all $\# \in S$.

The annealing algorithm starts at some, possibly random, configuration $\#_i$ $(i = 0)$, selects two vertices v and w (according to a perturbation scheme as discussed below), and computes the cost $c(\#_p)$ of the potential new configuration $\#_p$ obtained by interchanging

ν and w. Now, $\#_p$ is accepted if

$$c(\#_p) \leq c(\#_i) - t_i \log u$$

where $u \in]0;1]$ is a uniform random number and t_i is a control parameter called the *temperature* which is gradually decreased as described below. This means that an uphill step of λt_i will be allowed with probability $e^{-\lambda}$. In other words the new configuration $\#_{i+1}$ is set to $\#_p$ with probability

$$p = \min\left\{1, e^{-\frac{\Delta}{t_i}}\right\}$$

and to $\#_i$ with probability $1-p$, where $\Delta = c(\#_p) - c(\#_i)$. Large values of t_i allow large cost function increases, and small values allow only small increases.

The initial temperature t_0 should be set to some suitable large value such that all perturbations are accepted (this ensures that all configurations can be reached). That is, if $\#_i$ and $\#_j$ are adjacent configurations in S and if we define

$$U = \max_{i,j} |c(\#_i) - c(\#_j)|$$

to be the maximum cost function increase then an uphill step of U should be accepted with probability close to 1. Thus if $U = \alpha t_0$ we get the following rule

$$e^{-\alpha} \approx 1$$

meaning that t_0 should be large compared with U or $0 < \alpha \ll 1$.

(Lundy & Mees 1986) concluded in their study on convergence properties of the annealing algorithm that the temperature should be lowered exponentially as

$$t_{i+1} = \frac{t_i}{1 + \beta t_i} \tag{3.1}$$

where $\beta \ll \frac{1}{U}$, and that the final temperature, t_f, should (essentially) obey the following inequality

$$t_f \leq \frac{\epsilon}{\log |S|} \tag{3.2}$$

where ϵ is the acceptable error (i.e. $\epsilon = c(\#_f) - c(\#^*)$). An equivalent formulation of Equation 3.1 is

$$t_{i+1} = \frac{t_0}{1 + i\frac{\gamma}{\alpha}} \tag{3.3}$$

where $\alpha = \frac{U}{t_0}$, $0 < \gamma \ll 1$ and $\frac{\gamma}{\alpha} = \beta t_0$. By solving Equation 3.3 for i it appears that the number of iterations required to reach a temperature t_{i+1} is less than $\frac{U}{\gamma t_{i+1}}$ and hence to reach the final temperature t_f takes I iterations, where

$$I < \frac{U \log |S|}{\gamma \epsilon}$$

However, even if we set $\alpha = 0.1$ (a reasonable value would rather be 0.01) and $\gamma = 0.1$ (which is a very large value knowing that ideally $\frac{\gamma}{\alpha} \ll 1$), we find by accepting an error $\epsilon = U \cdot 10^{-3}$ that $I \approx n(\log n - 1) \cdot 10^4$ which for $n = 50$ is approximately $1.5 \cdot 10^6$. So in order to get reasonable response times from the annealing algorithm we have to adopt a more pragmatic approach. There are three parameters which determine the number of iterations required: The initial temperature t_0, the cooling rate, and the final temperature t_f. To allow essentially free motion in the space S in the initial phase (to avoid getting trapped in local minima at an early stage) the probability that any uphill step is accepted should be close to 1 as discussed above. Thus the rule $t_0 \gg U$ must not be violated. A further increase of the cooling rate seems not to be a fair solution to the problem as again the probability of getting trapped in high level basins becomes too large. Specifying a reasonable acceptable error ϵ often leads to extremely small values of both t_f and the changes in t_i when t_i is close to t_f. This means that the algorithm may have reached configurations $\#$ for which $c(\#) - c(\#^*) \leq \epsilon$, ($\#^*$ still being the optimal configuration) long before t_i has reduced to t_f. Furthermore, by constantly keeping in mind the least-cost configuration visited, t_f can definitely be set at a considerably higher value than recommended by Equation 3.2. That is, we may specify a considerably higher "acceptable error" than really wanted.

In the initial phase it seems intuitively reasonable to keep the radius of the search space S as small as possible to keep the distance between any two configurations as small as possible. Later, when the algorithm has found an attractive subspace (basin) containing low-cost configurations, it seems to be more convenient to have a large radius (i.e. narrowing down the focus of attention by reducing the number of neighbours for each configuration). Initially the only constraint we put on the perturbation scheme is that we define a perturbation to be the interchange of only two vertices v and w. With that constraint in mind the scheme providing the least possible radius chooses v and w completely arbitrarily from a configuration $\#$, and the scheme providing the largest possible radius chooses an arbitrary pair v, w for which $|\#(v) - \#(w)| = 1$ (i.e. v and w are adjacent in $\#$). For both schemes the pair of configurations $\#_i, \#_j \in S$ between which the distance equals the radius of S are characterised by the one being the reverse of the other (i.e. $\#_i(v) \neq \#_j(v)$ for all $v \in V$ except for at most one vertex). Thus for the two perturbation schemes the radii r_{min} and r_{max} of S are given by

$$r_{min} = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even} \\ \frac{n-1}{2} & \text{if } n \text{ is odd} \end{cases}$$

$$r_{max} = \sum_{i=1}^{n-1} n - i = \frac{n^2 - n}{2}$$

As we see $r_{max} = (n-1)r_{min}$ if n is even and $r_{max} = n \cdot r_{min}$ if n is odd, and hence there is a substantial difference between r_{min} and r_{max} even for a moderate number of vertices. The two perturbation schemes may be replaced by the general scheme for which the interchange of v and w is restricted by the condition $|\#(v) - \#(w)| < \omega$, where

$2 \leq \omega \leq n$. That is, we define a "window" of width ω within which the perturbations take place. Initially we set $\omega = n$ and decrease it gradually until it reaches the lower bound 2. A reasonable way of narrowing the window width (i.e. increasing the radius of the search space) could be to follow the reduction scheme applied to the temperature.

Given the cost $c(\#_i)$ of configuration $\#_i$ and given the candidate next configuration $\#_p$ it should be possible based on $c(\#_i)$ to locally compute $c(\#_p)$. Otherwise we have to find all cliques of the graph $G^t = (V, E \cup T(G_{\#_p}))$, where $G_{\#_p} = (V, E)$, to compute its weight $w(G) = c(\#_p)$. Theorem 6 states that $c(\#_p)$ can be derived from $c(\#_i)$.

Theorem 6 *Let* $G_{\#_a} = (V, E)$ *and* $H_{\#_b} = (V, E)$ *be ordered graphs such that* $\#_a = \#_b$ *except that* $\#_b(u) = j$ *and* $\#_b(v) = i$*, where* $\#_a(u) = i < \#_a(v) = j$ *and* $u, v \in V$*. Then* $w(H^t) = w(G^t) + \delta$*, where*

$$\delta = \sum_{k=i}^{j} w(C_k(H)) - \sum_{k=i}^{j} w(C_k(G))$$

Proof: The proof consists of two parts: First we shall prove that all cliques $C_k(G)$ for $0 < k < i$ and $j < k \leq n$, remain untouched by the interchange of u and v in the elimination ordering. Knowing that the triangulation of a graph determines how the cliques of the resulting triangulated graph are composed the problem is equivalent to the problem of proving that the fill edges $T_k(G) = T_k(H)$ for all $0 < k < i$ and for all $j < k \leq n$. For $0 < k < i$ the problem is trivial as the set of fill edges produced by eliminating a vertex solely depends on the original edges E and on the fill edges already produced. The validity of the equality for $j < k \leq n$ follows immediately from Theorem 1.

Secondly we shall prove that for every new clique $C_k(H)$, $i \leq k \leq j$, C_k cannot be a superset of a clique $C_l(G)$, $0 < l < i$ and $j < l \leq n$. For $0 < l < i$ the problem is trivial as $\#_a^{-1}(l) = \#_b^{-1}(l) \in C_l$ and hence cannot be a member of C_k. To prove it for the case where $j < l \leq n$ we assume there is a clique C_l such that $C_l \subset C_k$. For this to be true the vertex $\#_b^{-1}(k)$ (the elimination of which created C_k) must have been adjacent to all vertices in C_l in H_{k-1}. By Corollary 2 we see that there would also have been such a vertex in G_{k-1} and hence C_l would not have been a clique. □

3.2 Annealing in Practice

From the above discussion it appears that the total number of attempted configuration transitions before reaching the final temperature depends on the size of the search space and on the maximum cost difference between any two configurations. In relation to the problem of low weight (cost) graph triangulations this means that in practice there is a limit in terms of number of vertices and dispersion of vertex weights (loosely speaking) above which the annealing algorithm is inapplicable. This is, of course, a rather unsatisfactory situation which can be avoided by allowing more "coarse-grained" solutions for large graphs. That is, if we adopt such a rather pragmatic approach the stopping criterion must be defined such that it becomes independent of the parameters mentioned above. From Equation 3.3 we get

$$\begin{aligned} I &= \frac{\alpha}{\gamma}\left(\frac{t_0}{t_f} - 1\right) \\ &= \frac{1}{\gamma}\left(\frac{U}{t_f} - \alpha\right) \end{aligned}$$

from which we see that if the final temperature is defined in terms of U the number of iterations required depends solely on the cooling rate. One way of defining t_f is to say that it is the temperature at which an uphill step of cU, where $0 < c < 1$, is accepted with probability p, where $0 < p < 1$. Recalling that an uphill step of $\lambda t_f = cU$ is accepted with probability $e^{-\lambda} = p$ at temperature t_f we get

$$I = -\frac{1}{\gamma}\left(\frac{\log p}{c} + \alpha\right), \; p < e^{-\alpha c} \tag{3.4}$$

as desired. By fixing α and c we observe that the cooling rate may be given either explicitly via γ or implicitly via I and p. Similarly I may be given explicitly or implicitly via γ and p.

The annealing algorithm performs a search in the graph of configurations (the solution space) which is normally controlled solely by the temperature parameter. However, by allowing a gradual increase of the radius of the graph during the search process we have introduced an additional control parameter. Ideally the final temperature, t_f, should be low (relative to the acceptable error, ϵ) and the final radius should be large (i.e. ω should be small). However, to determine the effect of the two control parameters separately a series of test runs in which one of them is kept at a fixed value which can be small, medium or large must be performed.

The three temperature levels are defined in the same way as t_f above with $c = 0.1$ and probabilities $p_s = 0.05$, $p_m = 0.5$, and $p_l = 0.95$. Thus

$$\begin{aligned} t_s &= \frac{U}{10 \log 20} \\ t_m &= \frac{U}{10 \log 2} \\ t_l &= \frac{U}{-10 \log 0.95} \end{aligned}$$

The radius levels are defined as

$$\begin{aligned} r_s &= r_{min} \\ r_m &= \frac{r_{min} + r_{max}}{2} \\ r_l &= r_{max} \end{aligned}$$

As the radius of the search graph is determined by ω, r is not an explicit control parameter. So we need an algebraic relationship between r and ω. This is, however, rather hard to find if we insist on employing the perturbation scheme described above. If we accept a slightly different scheme, namely a "move-and-push" scheme, ω can be expressed in terms of r and n as

$$\omega = \left\lceil \frac{(n-1)^2}{2r-n+1} + 0.5 \right\rceil$$

The interchange and the "move-and-push" schemes are identical for $\omega = 2$. For $\omega = n$ "move-and-push" yields $r_{min} = n - 1$. Thus

$$r_m = \frac{r_{min} + r_{max}}{2} = \frac{n^2+n-2}{4}$$

$$\omega_m = \left\lceil \frac{2(n-1)^2}{n(n-1)} + 0.5 \right\rceil = \left\lceil 2(1-\frac{1}{n}) + 0.5 \right\rceil$$

The non-fixed control parameter may be subjected to a slow, medium or fast reduction (t) / increase (r) scheme.

The rate by which the temperature parameter is decreased is determined by the ratio $\frac{\gamma}{\alpha}$. Recalling that $e^{-\alpha}$ expresses the probability of accepting uphill steps of U we select $\alpha = 0.1$ (i.e. with probability 0.9 an uphill step of U is accepted when $t = t_0$). The fact that the rule of essentially free motion in the initial phase is not violated is clearly demonstrated by the fact that at temperature t_i uphill steps of U is accepted with probability $e^{-\frac{U}{t_i}}$ (i.e. even with the temperature lowered to one tenth of t_0 more than one third of the potential uphill steps of size U is accepted). So, with α fixed the cooling rate is determined by γ. A slow cooling rate is defined such that the temperature equals t_l when only 10% of the iterations are left, for a medium rate the final temperature is reached at the last iteration, and for a fast cooling rate the temperature equals t_s when only 10% of the iterations are done. In the case of a medium cooling rate we set $t_f = t_s$. Thus

$$\begin{aligned}
\gamma_s &= -\frac{1}{0.9I}\left(\frac{\log p_l}{c} + \alpha\right) = \frac{-10\log 0.95 - \alpha}{0.9I} \\
\gamma_m &= -\frac{1}{I}\left(\frac{\log p_s}{c} + \alpha\right) = \frac{10\log 20 - \alpha}{I} \qquad (3.5) \\
\gamma_f &= -\frac{1}{0.1I}\left(\frac{\log p_s}{c} + \alpha\right) = \frac{10\log 20 - \alpha}{0.1I}
\end{aligned}$$

Even though we argued above that ϵ can be set to a value which is considerably higher than the actual acceptable error provided the least-cost configuration visited is kept in mind it seems hard to justify $t_f = t_s$ as ϵ then becomes comparable with U. A further discussion of whether $t_f = t_s$ is a reasonable choice will be postponed till after the results of the test runs are known.

Now we should assign values to either I or γ (note that values of p have already been determined). As I is a derived parameter which is defined in terms of the cooling rate and the final temperature we naturally choose the latter. Knowing that $\frac{\gamma}{\alpha} \ll 1$ or $\gamma \ll \alpha$ a value of $\gamma = 10^{-3}$ seems reasonably and we find from Equation 3.5

$$I = \frac{10 \log 20 - \alpha}{\gamma_m} \approx 10^4 \log 20 \approx 3 \cdot 10^4$$

The window width, ω, is decreased exponentially as

$$\omega_{i+1} = \frac{\omega_0}{1 + i\tau}$$

The rates by which r is increased is thus determined by

$$\tau_s = \frac{\frac{\omega_0}{\omega_m} - 1}{I}$$

$$\tau_m = \frac{\frac{\omega_0}{\omega_s} - 1}{0.8I}$$

$$\tau_f = \frac{\frac{\omega_0}{\omega_s} - 1}{0.2I}$$

Note that ω_m corresponds to r_m, $\omega_0 = n$, and ω_s corresponds to r_l. That is, a slow rate is defined such that the radius of the search space is medium when the annealing terminates, for a medium rate the radius is maximum when the annealing is 80% done, and for a fast rate it is maximum when 20% done.

Now, the only undefined parameter is the maximum cost function increase, U, which, depending on the problem at hand, can be found either analytically or experimentally. For the graph triangulation problem it seems very hard to find U the analytic way as it depends on a number of parameters such as the size and density of the graph, the homogeneity of the graph w.r.t the distribution of edges and vertex weights, and the window width ω. So we will stick to the experimental approach.

In order to find the optimal controlling scheme a selected set of schemes is first tested on a particular graph and secondly a subset of these schemes which appeared to produce good results is then applied to the three other graphs. Then, hopefully, we should be able to point out which controlling scheme(s) are generally preferable.

In the first test cycle we select the control schemes for which at least one of the control parameters is varying, giving a total of 27 distinct combinations. As the target graph we use the Medianus-I for which we find $U \approx 7$. Because of either the extremely large acceptable error (high final temperature) in the cases of slow and medium cooling or high basin trapping in the case of fast cooling we cannot be sure that the annealing algorithm will end up finding the same least-cost configuration every time given a particular control scheme. Therefore we have to run the algorithm several times for each scheme. Table 3.1 shows the minimum, median, and maximum least costs found for the 27 different control schemes with 10 runs for each instance. The schemes have been ordered according to the median least cost value (average state size).

A varying control parameter is indicated by a γ for the temperature or a τ for the radius and the rate is indicated by index s, m, or f (i.e. slow, medium, or fast). A fixed

	Min.	Med.	Max.		Min.	Med.	Max.
1. t_s, τ_m	22.63	22.71	22.92	15. γ_m, τ_f	22.71	23.41	24.40
2. γ_m, r_s	22.68	22.73	22.81	16. γ_m, r_l	22.69	23.46	24.23
3. t_s, τ_s	22.66	22.78	23.14	17. t_m, τ_m	23.17	23.63	24.37
4. t_s, τ_f	22.65	22.95	23.87	18. t_m, τ_f	22.98	23.81	24.43
5. γ_f, r_s	22.63	22.96	23.44	19. γ_s, r_s	23.99	24.95	25.83
6. γ_m, τ_m	22.70	22.99	23.47	20. t_l, τ_s	24.07	25.16	26.03
7. γ_f, τ_s	22.63	23.00	23.44	21. γ_s, τ_s	24.35	25.23	26.18
8. γ_f, τ_m	22.70	23.03	24.06	22. t_l, τ_m	24.12	25.34	26.33
9. γ_m, τ_s	22.74	23.07	23.63	23. γ_s, τ_m	24.07	25.66	27.05
10. γ_f, r_m	22.63	23.13	24.06	24. γ_s, τ_f	25.36	26.43	27.28
11. γ_f, τ_f	22.63	23.19	23.78	25. t_l, τ_f	25.06	26.67	28.41
12. γ_m, r_m	22.68	23.25	24.19	26. γ_s, r_m	25.64	27.18	30.46
13. γ_f, r_l	22.63	23.35	24.06	27. γ_s, r_l	25.07	27.44	29.12
14. t_m, τ_s	22.79	23.36	23.92				

Table 3.1: Minimum, median, and maximum least costs found for 27 different control schemes applied 10 times to the Medianus-I graph.

control parameter is indicated by a t for the temperature or an r for the radius and the level is indicated by index s, m, or l (i.e. small, medium, or large).

Considering the results provided by the top third control schemes in Table 3.1 we observe that for all of these the final temperature is low (i.e. $t_f \leq t_s$) and that the initial radius is small (i.e. $r_0 = r_s$). The results obtained by applying these schemes to the three other test graphs are displayed in Table 3.2. In Table 3.1 the most striking observation is that the three schemes with a constant low temperature belong to the top five schemes. Therefore an interesting experiment would be to test a strictly descent version of the annealing algorithm (i.e. $t = 0$). However, before doing that we will add a few more comments on Tables 3.1-3.2. The last third of the schemes are characterised by a high final temperature. For the middle third the final temperature is either low or medium. Note that the overall average state size (median weight) is more than doubled from the 18th best to the 19th best scheme (i.e. the boundary between middle and last third). By calculating the average relative deviation of the median values from the least median for each control scheme in Table 3.2 we find that the ranking of the schemes are 3, 2, 1, 6, 8, 7, 4, 9, 5 (i.e. scheme number 3 is the best), where the differences between scheme 3 and scheme 2 is insignificant.

To test the utility of a descent algorithm we perform four additional tests: i) Constant small radius ii) slow increasing radius iii) medium increasing radius and iv) fast increasing radius. Table 3.3 shows the resultant minimum, average, and maximum least costs obtained by these four descent schemes applied 10 times to the four test graphs. Not surprisingly, constant small radius is found to be superior. More surprising is the fact that the results produced by this scheme turn out to be comparable with the results of the top four schemes (i.e. 3, 2, 1, and 6) above.

The conclusion to be drawn from the above analysis is the following: If the temperature is kept at a constant low level the radius should either be constantly small or

	Medianus-I			Medianus-II		
	Min.	Med.	Max.	Min.	Med.	Max.
1. t_s, τ_m	22.63	22.71	22.92	20.49	20.76	22.44
2. γ_m, r_s	22.68	22.73	22.81	20.49	20.58	20.79
3. t_s, τ_s	22.66	22.78	23.14	20.49	20.54	20.64
4. t_s, τ_f	22.65	22.95	23.87	20.52	20.86	22.44
5. γ_f, r_s	22.63	22.96	23.44	20.45	20.57	20.77
6. γ_m, τ_m	22.70	22.99	23.47	20.67	21.16	22.09
7. γ_f, τ_s	22.63	23.00	23.44	20.46	20.96	22.43
8. γ_f, τ_m	22.70	23.03	24.06	20.54	20.73	21.20
9. γ_m, τ_s	22.74	23.07	23.63	20.46	20.61	20.79

	Sparse			Dense		
	Min.	Med.	Max.	Min.	Med.	Max.
1. t_s, τ_m	22.71	22.80	22.87	50.88	51.42	52.70
2. γ_m, r_s	22.75	22.93	23.15	51.05	51.34	52.02
3. t_s, τ_s	22.72	22.82	22.97	50.96	51.15	51.61
4. t_s, τ_f	22.73	23.08	24.35	51.02	52.29	53.32
5. γ_f, r_s	22.61	22.64	22.66	50.88	51.96	53.49
6. γ_m, τ_m	22.81	23.37	23.85	51.14	53.04	53.49
7. γ_f, τ_s	22.61	22.76	22.98	50.88	51.57	52.58
8. γ_f, τ_m	22.64	22.77	23.27	50.88	51.82	53.49
9. γ_m, τ_s	22.94	23.19	23.51	51.09	52.31	53.29

Table 3.2: Minimum, median, and maximum least costs found for 9 different control schemes applied 10 times to the four test graphs.

increased slowly (by a medium or fast rate of increase we may be trapped in a high level basin — see schemes 1 and 4 in Table 3.2). If the temperature is lowered at a medium rate (schemes 2, 5, and 9) we observe that the radius must be constantly small (to compensate for the relatively large number of uphill steps). But as small radii are computationally more expensive than large radii we reject these schemes. Finally, if a strict descent algorithm is employed the radius must be constantly small (to compensate for the lack of ability to climb hills). Thus, based on the performed analysis we recommend a combination of control parameters given by a positive low temperature and a radius which initially should be small and then slowly increased.

	Medianus-I			Medianus-II		
	Min.	Med.	Max.	Min.	Med.	Max.
1. $t = 0, \tau_s$	22.63	22.89	23.12	20.45	20.96	22.43
2. $t = 0, \tau_m$	22.63	22.97	23.44	20.45	20.96	21.77
3. $t = 0, r_s$	22.63	23.18	24.25	20.45	20.60	21.08
4. $t = 0, \tau_f$	22.63	23.23	24.25	20.45	20.76	22.79

	Sparse			Dense		
	Min.	Med.	Max.	Min.	Med.	Max.
1. $t = 0, \tau_s$	22.61	23.13	24.33	50.88	51.30	52.70
2. $t = 0, \tau_m$	22.61	22.87	23.35	50.88	51.78	52.71
3. $t = 0, r_s$	22.61	22.77	23.28	50.88	51.26	52.70
4. $t = 0, \tau_f$	22.84	23.74	26.86	50.88	52.74	53.37

Table 3.3: Minimum, median, and maximum least costs found for 4 different descent control schemes applied 10 times to the four test graphs.

Chapter 4

Conclusion

With the practical constraints given in Section 3.2 it appeared that simulated annealing performed best with a constant low temperature[1] and a slowly increasing radius of the search space. In the following discussion it is implicitly assumed that the term *annealing* means simulated annealing applied with that control scheme. Furthermore, the results by the heuristic algorithms referred to below is implicitly assumed to be those which for the actual graph are produced by the algorithm which provided the least median weight.

By loosely comparing the figures of Table 2.1-2.4 and Table 3.2 we observe that the best elimination orderings found by annealing for the Medianus graphs are only slightly better than the best found by the heuristic algorithms, whereas those found for the Sparse and Dense graphs by annealing are clearly superior.

A detailed comparison tells us that for Medianus-I annealing may in the worst case produce triangulated graphs the state size of which are about 10–15% greater than that of the graph produced by minimum weight. In the average case, however, annealing performs 20% better than minimum weight. For the Medianus-II annealing is in no cases worse than minimum weight; in the average case it is about 10% better. For the Sparse and Dense graphs, however, annealing is considerably (300–800%) better than the best heuristic method.

The lesson to be learnt from this comparison is: If time is not a crucial parameter it might be recommendable to use annealing. The clearest example appears with the Sparse graph where annealing at best performs almost an order of magnitude better than any of the heuristics. On the other hand, if triangulation have to be fast we can by no means recommend the annealing algorithm which is orders of magnitude slower than the heuristic algorithms.

Since annealing performs best with a constant low temperature (with the time constraints given in Section 3.2) there is no reason to start with an arbitrary vertex ordering wasting a lot of time testing useless orderings. Instead it is highly recommended to apply one of the heuristic ordering algorithms first and then let the annealing process start

[1] It is not really quite correct to call it simulated annealing then, but anyway we will stick to that term.

with that ordering.

In Chapter 2 we mentioned the branch-and-bound algorithm which in a way is the deterministic equivalent of the annealing algorithm. Comparisons with the Medianus-I as the target graph showed that branch-and-bound could not at all compete with annealing. A test was conducted by first applying minimum weight 10 times and the corresponding orderings were stored. Subsequently the annealing and the branch-and-bound algorithms were ran with these orderings as starting points and a fixed amount of time were allocated for each run. In 9 out of the 10 cases annealing improved the initial ordering down to a level very close to the lowest level known (which by very high probability is the minimum weight level). Branch-and-bound, on the other hand, made modest improvements in just 2 out of the 10 cases.

As mentioned in Section 2.3 none of the triangulations produced by the heuristic algorithms turned out to be non-minimal and hence there does not seem to be a plausible reason to apply the FMINT algorithm to ensure minimality. However, as none of the heuristic algorithms are, in general, guaranteed to produce minimal triangulations there might be cases where it would be beneficial to apply FMINT.

Another feature of the minimum size and the minimum weight heuristics is that they do not, in general, produce perfect orderings if the graph is triangulated. This means that a simplicial vertex, v, is not guaranteed to be eliminated before a non-simplicial vertex even though the (complete) set $C = \{v\} \cup \mathrm{adj}(v)$ eventually will turn out to be either a proper subset of a clique or a clique itself. That is, there is no point in postponing the elimination of v, and at worst there will be a clique which is a proper superset of C. The minimum fill heuristic takes simplicial vertices first, but at the expense of increased time complexity compared to the minimum size and weight heuristics. However, by expanding minimum size and weight with test for simpliciality no effects were observed concerning the weights of the triangulated graphs. So, if time complexity is an important parameter we cannot recommend such an expansion to be carried out. On the other hand if time complexity is of little importance this expansion should preferably be included.

Acknowledgments

I wish to thank my supervisors Finn V. Jensen and Steffen L. Lauritzen for their valuable comments and suggestions during the process of producing this paper. I am also grateful to Frank Jensen who gave me his source code of the branch-and-bound algorithm. In my implementation of the simulated annealing algorithm I found benefit in reuseing some of the ideas from this code.

The research was financially supported by Judex Datasystemer A/S and The Danish Academy of Technical Sciences.

Bibliography

Andreassen, S., Jensen, F., Andersen, S., Falck, B., Kjærulff, U., Woldbye, M., Sørensen, A., Rosenfalck, A. & Jensen, F. (1989). MUNIN — an expert EMG assistant, *in* J. Desmedt (ed.), *Computer-Aided Electromyography and Expert Systems*, Elsevier Science Publishers, Amsterdam, chapter 21.

Arnborg, S., Corneil, D. & Proskurowski, A. (1987). Complexity of finding embeddings in a k-tree, *SIAM Journal on Algebraic and Discrete Methods* **8**: 277–284.

Fujisawa, T. & Orino, H. (1974). An efficient algorithm of finding a minimal triangulation of a graph, *IEEE International Symposium on Circuits and Systems*, San Francisco, pp. 172–175.

Lauritzen, S. & Spiegelhalter, D. (1988). Local computations with probabilities on graphical structures and their application to expert systems, *Journal of the Royal Statistical Society, Series B* **50**(2): 157–224.

Lundy, M. & Mees, A. (1986). Convergence of an annealing algorithm, *Mathematical Programming* **34**: 111–124.

Olesen, K., Kjærulff, U., Jensen, F., Jensen, F., Falck, B., Andreassen, S. & Andersen, S. (1989). A MUNIN network for the median nerve — a case study on loops, *Applied Artificial Intelligence*. Special issue: Towards Causal AI Models in Practice.

Rose, D. (1973). A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations, *in* R. C. Read (ed.), *Graph Theory and Computing*, Academic Press, New York, pp. 183–217.

Rose, D., Tarjan, R. & Lueker, G. (1976). Algorithmic aspects of vertex elimination on graphs, *SIAM Journal on Computing* **5**: 266–283.

Sato, N. & Tinney, W. (1963). Techniques for exploiting the sparsity of the network admittance matrix, *IEEE, PAS* pp. 944–950.

Tarjan, R. & Yannakakis, M. (1984). Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM Journal on Computing* **13**(3): 566–579.

Tinney, W. & Walker, J. (1967). Direct solutions of sparse network equations by optimally ordered triangular factorization, *Proc. IEEE* **55**: 1801–1809.