

Layout of Bayesian networks

Kim Marriott

Peter Moulder

Lucas Hope

Charles Twardy

School of Comp. Sci. & Soft. Eng., Monash University, Australia.
{marriott,pmoulder,lhope,ctwardy}@mail.csse.monash.edu.au

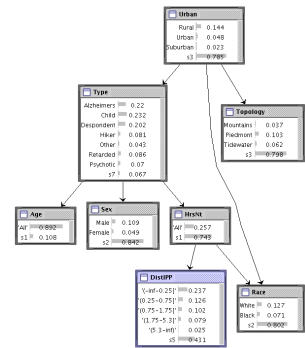


Figure 1: An example of a Bayesian network

Abstract

Bayesian networks are weighted directed acyclic graphs (DAGs). Layout of DAGs has been studied for many years, with variants of the horizontal layering algorithm of Sugiyama et al. (Sugiyama, Tagawa & Toda 1981) the preferred approach for medium size graphs. Following Gansner et al. (Gansner, Koutsofios, North & Vo 1993) we extend the classic horizontal layering algorithm with an additional vertex coordinate assignment phase which is designed to remove overlap between vertices with non-zero width and height. This is required because vertices in the Bayesian network are typically large and may vary greatly in height and width so a major component of good layout is to place vertices closely together without overlap. Our main contribution is to describe and evaluate a variety of novel techniques for vertex coordinate assignment.

1 Introduction

Bayesian networks (Pearl 1988, Neapolitan 1990) are an increasingly popular AI formalism for reasoning under uncertainty. Each node is a variable, and the network represents the joint probability distribution over the variables, in factored form. The arcs are usually given causal interpretations and so the network can also represent the effect of interventions, as well as observations.

A number of software tools have been built for creating and viewing Bayesian networks but generally speaking layout of Bayesian networks is not handled well in these tools. Essentially a Bayesian network is a weighted directed acyclic graph (DAG). The main

complication is that vertices in the network may be large and may vary greatly in height and width because they typically detail the probabilities associated with the attributes of that vertex. For example, Figure 1 shows a representative Bayesian network. Clearly, a major component of a good layout is to place vertices closely together without overlap.

Layout of DAGs has been studied for many years, with variants of the horizontal layering algorithm of Sugiyama et al. the preferred approach for medium size graphs (Sugiyama et al. 1981). This has three stages: (a) layer assignment in which vertices are assigned a vertical level; (b) crossing reduction in which vertices are iteratively reordered in each layer to reduce the number of edge crossings; and (c) coordinate assignment in which vertices and edges are assigned their coordinates in the drawing.

Coordinate assignment is the phase responsible for determining the compact placement of nodes and edges while ensuring no overlap. One of the first papers to investigate coordinate assignment with variable width vertices is that of Gansner et al. (Gansner et al. 1993). This describe a three part process. The first step is to assign x -coordinates to all vertices by solving a linear programming problem in which the total of the horizontal lengths of the edges is minimised with respect to the inequality constraints generated by adding “left-of” constraints between adjacent vertices on each layer to ensure that they do not overlap. The second step is to compute a minimal y -separation between adjacent layers. This is done by treating each layer as a box whose height is the maximum height of a vertex in the layer and placing these boxes the minimum distance apart. Vertices are then vertically centered in their layer’s box. The final step is to compute the edge placement. Gansner et al.’s approach gives good layout but has three disadvantages for our application:

- (1) It may lead to unnecessary asymmetry as parents are not necessarily centred with respect to their children since the objective function uses absolute value and so if a vertex has two children the objective function associates the same penalty with all points between the two children;
- (2) Solving the linear program can be relatively slow for large graphs;
- (3) The generated layout may not be as vertically compact as possible since it does not allow tall vertices to “push” into the layers above and below and it forces vertices on the same layer to have their centres horizontally aligned.

The main technical contribution of our paper are new techniques for coordinate assignment that overcome these problems. These are incorporated in Causal Reckoner, a Bayesian network visualisation tool (Korb, Hope, Nicholson & Axnick 2004), and the Bayesian network shown in Figure 1 has been drawn using the algorithms. Of course the algorithms are

any other formalisms which are based on DAGs and which have large vertices, for instance, class hierarchies and argument maps.

We give a new iterative algorithm for x -coordinate assignment which gives similar quality layout to Gansner et al.’s approach but is considerably faster and suitable for large graphs. It does not use linear programming but rather an iterative method similar to the edge crossing reduction step to assign x coordinates to each vertex. The algorithm iterates through the layers in the graph finding a placement for the vertices in each layer which preserves the ordering computed in the crossing reduction phase and which places the vertex as close as possible to the average of the parents’ and children’s x coordinates. The core of the approach is a linear time algorithm for solving the *horizontal placement problem*: given a sequence of vertices v_1, \dots, v_n each of which has a desired location, the problem is to find a placement for the vertices as close as possible to the desired location which preserves the order of the vertices while ensuring no vertices overlap.

Our second main contribution is to explore two new approaches to y -coordinate assignment which lead to compact layout even if vertices have different heights. One approach places each layer the minimum distance apart that ensures no vertices overlap in the vertical direction and that parents are above children while another approach solves a linear program which allows vertices on the same layer to move up/down relative to each other if this leads to more compact layout.

Finally, we give a systematic evaluation of the effectiveness and efficiency of these approaches for the case of Bayesian networks.

As suggested above, the most closely related research is that of Gansner et al. (Gansner et al. 1993) whose approach has the limitations detailed above. It is worth noting that they also describe an iterative technique for x -coordinate assignment which is based on iteratively solving a variant of the horizontal placement problem. However their technique is complex, has quadratic worst case time complexity, and is not proven to find an optimal solution to the horizontal placement problem. Several other techniques for x -coordinate assignment have been developed. For instance, see Brandes and Köpf (Brandes & Köpf 2001) and associated references.

Layout in Bayesian network visualisation tools is generally poor. Netica (Norsys Inc. n.d.) has no layout to speak of. BNJ (Kansas State Univ. Laboratory for Knowledge Discovery in Databases n.d.) and GeNIe (Decision Systems Laboratory, Univ. of Pittsburgh n.d.) use spring-based models which are not well-suited to full initial layout. GraphViz (a.k.a. Dot) (AT&T Labs n.d.) does not easily handle cell contents, and too readily bends nearby arcs. The high-end package BayesiaLab (Bayesia Ltd. n.d.) uses a general optimizer performing a directed search through layout space. It can generate nice layouts, but requires fiddling with relative goal weights, and possibly long searches.

2 Background

2.1 Problem Specification

We extend the terminology of (di Battista, Eades, Tamassia & Tollis 1999). The problem is to find a layout or drawing for a weighted directed acyclic graph, $G = \langle V, E \rangle$, with vertices V and edges E . Edge $e \in E$ goes from vertex $start(e)$ to vertex $end(e)$ and has weight $wt(e)$. Vertex $v \in V$ has height $ht(v)$ and

height and width are “padded” with the minimum vertical and horizontal separation between vertices.

A drawing Γ for G is an assignment of a coordinate $(\Gamma(v).x, \Gamma(v).y)$ to each vertex $v \in V$ and centre and an assignment of a poly-line $\Gamma(e)$ to each edge $e \in E$ such that $\Gamma(e)$ starts from $\Gamma(start(e))$ and finishes at $\Gamma(end(e))$.¹

We wish to find a drawing that satisfies the following mandatory aesthetic criteria:

- It is a downward drawing: for all $e \in E$, $\Gamma(start(e)).y \geq \Gamma(end(e)).y$. In fact we use a stronger requirement: that each segment in the poly-line assigned to an edge is downward.
- There are no overlapping vertices.
- Edges do not overlap vertices except at the endpoints.

In addition we prefer the layout to

- Have few overlapping edges
- Have edges with few bends
- Have short edges where the importance of this is proportional to the edge weight
- Be compact

2.2 The Basic Approach

Our algorithm has the same main stages as that of Gansner et al. (Gansner et al. 1993):

- (a) Layer assignment in which vertices are assigned a vertical level.
- (b) Edge crossing reduction in which vertices are iteratively reordered in each layer to reduce the number of crossings and length of edges.
- (c) Vertex placement in which vertices are assigned their coordinates.
- (d) Edge drawing.

Layer assignment assigns each vertex v a layer $lyr(v)$ where this is a positive integer. We require that if there is an edge from u to v then $lyr(u) < lyr(v)$. Layer assignment also inserts dummy vertices to break up “long” arcs into dummy edges to ensure that no edge crosses a layer. We use V_{lay} and E_{lay} to refer to the vertices and edges after layer assignment. Dummy nodes are given a small width of the order of the minimum separation and dummy edges are given the weight of the original edge. We experimented with two standard approaches to layer assignment (di Battista et al. 1999). We found that Coffman-Graham-Layering produced compact, narrow graphs, but long edges with many dummy vertices, which we felt obscured the structure of the original graph and which significantly increased the time taken to complete later layout stages. Consequently we explored an integer programming approach² due to Gansner et al. (Gansner et al. 1993) that minimizes the number of dummy vertices (hence layers) and so produces vertically compact graphs. The disadvantage is that the graphs can be arbitrarily wide but in most examples this was not too bad and preferable to the layer assignment obtained with Coffman-Graham-Layering.

We use the same iterative algorithm as Gansner et al. (Gansner et al. 1993) for edge crossing reduction: median method plus swapping. The initial order for vertices is also computed using their approach:

¹More exactly from the boundary of the rectangle drawn at each of these vertices.

²Although formally an integer programming problem this problem can be solved using linear programming.

On subsequent layers the order is computed as follows. Let the ordered sequence of vertices on layer j be v_1, \dots, v_K . Then all children of vertex v_1 are placed on layer $j+1$ followed by those children of the next vertex that are as yet unplaced and so on. The order of the children of a vertex are arbitrary.

One heuristic technique we use to improve compactness of the layout is to change the aspect ratio of vertices. We assume that we have a fixed number of allowable height/width configurations for each vertex. In our case these correspond to one column, two column etc. layout of the probability table. The one column layout is preferred but in order to better utilise horizontal space we iteratively add a column to the tallest vertex (and so shrink its height) until some maximal width of the layer is reached or all vertices have reached some minimal aspect ratio. More generally in the case of vertices containing plain text we could iterate through configurations containing different number of lines or we could also consider changing the orientation of graphics such as bar charts. We perform this step just before vertex placement.

The main novelty in our approach is in vertex placement. Gansner et al. assign x -coordinates to all vertices by solving a linear programming problem. More exactly, they find an assignment Ψ which maps each node to the x coordinate of its centre ($\Psi(v) = \Gamma(v).x$) which essentially minimizes

$$\sum_{e \in E_{lay}} wt(e) \times |\Psi(start(e)) - \Psi(end(e))|$$

subject to $\Psi(u) + x_sep(u, v) \leq \Psi(v)$ where u is the left neighbour of v on some layer and $x_sep(u, v) = \frac{1}{2}wd(u) + \frac{1}{2}wd(v)$.

This ensures that the total of the x -length components of the edges is minimised and that vertices in each layer do not overlap. After this the y -coordinate assignment to vertices is computed: Each layer is treated as a box whose height is the maximum height of a vertex in the layer and these boxes are placed the minimal y -separation apart. Vertices are vertically centred in their layer's box.

As noted in the introduction, this approach has disadvantages: it typically gives no preference to placing a node near the centre of its parents/children, solving a global linear program can be slow for large graphs, and it may waste vertical space. In the remainder of the paper we describe various methods which overcome these problems.

3 X-coordinate assignment: 3 approaches

3.1 Linear programming approach

The first disadvantage of Gansner et al.'s approach to x -coordinate assignment, namely of giving no specific preference for a node to be centred with respect to its parents/children, is relatively simple to overcome: we simply try to place vertices near the weighted average of their parents' and children's positions rather than trying to minimise the horizontal length of the edges.

Thus our first approach to x -coordinate assignment is a simple variant of Gansner et al.'s approach which we shall call the *linear programming* approach. It is based on using linear programming to find an assignment Ψ which maps each vertex to the x coordinate of its centre and which preserves the order found by edge crossing reduction for the vertices in each layer.

$$\Psi(v) = D_v + \frac{1}{wt(v)} \left(\frac{\sum_{e \in In_v} wt(e) \times \Psi(start(e)) + \sum_{e \in Out_v} wt(e) \times \Psi(end(e))}{2} \right)$$

where

$$Out_v = \{e \in E_{lay} \mid start(e) = v\}$$

$$In_v = \{e \in E_{lay} \mid end(e) = v\}$$

$$wt(v) = \sum_{e \in In_v} wt(e) + \sum_{e \in Out_v} wt(e).$$

The variable D_v measures the error of the constraint and its absolute value is minimised in the linear program.

Now consider the vertices in layer j , v_1, \dots, v_K previously sorted in the edge crossing reduction phase. We add the constraint

$$\Psi(v_i) + x_sep(v_i, v_{i+1}) \leq \Psi(v_{i+1})$$

for $i = 1, \dots, K-1$ to ensure that vertices on the same level do not overlap and the constraints

$$0 \leq \Psi(v_1) - \frac{1}{2}wd(v_1)$$

and

$$\Psi(v_K) + \frac{1}{2}wd(v_K) \leq W$$

where W is a new variable modelling the drawing width.

We minimise the objective function

$$\alpha \times W + \sum_{v \in V_{lay}} wt(v) \times |D_v|$$

where α is a scaling constant. This ensures that we try to satisfy the aesthetic criteria while minimising the width of the layout.

3.2 Iterative x -coordinate assignment

The disadvantage with the linear programming approach is that it may be slow for large graphs (See Table 2). We now give a fast and simple method for computing the x -coordinate assignment. We shall refer to this approach as the *iterative* approach as it works by repeatedly sweeping through the layers in the graph and computing a new placement for the vertices in layer j from the current placement of vertices in layers $j-1$ and $j+1$. It preserves the ordering of vertices in each layer computed in the edge crossing reduction step.

We call a placement of vertices on a layer j a *configuration* for j . We represent a configuration by an assignment Ψ which maps each vertex v to the x coordinate of its centre. Vertices are not allowed to overlap in a configuration.

Given configurations for the other layers in the graph, we define the *desired position*, $des(v)$, for vertex $v \in V_{lay}$ to be the weighted average of its parents' and children's positions: i.e. $des(v)$ is

$$\frac{1}{wt(v)} \left(\frac{\sum_{e \in In_v} wt(e) \times \Psi(start(e)) + \sum_{e \in Out_v} wt(e) \times \Psi(end(e))}{2} \right)$$

Of course we cannot simply place each vertex at its desired location since this may give a configuration which has overlapping vertices. In order to compute a valid configuration we need to solve the following constrained optimisation problem:

Horizontal placement problem: Given a sequence of vertices v_1, \dots, v_n each of which has a desired location $des(v_i)$ find a placement for the vertices as close as possible to the desired location which preserves the order of the vertices and in which no vertices overlap:

```

function optimal_layout([v1, ..., vn])
/* create a sentinel block */
Let v0 be a new vertex with des(v0) = -∞ and wd(v0) = 0
block[0] := block(v0, 0)
/* repeatedly merge blocks until there is no overlap */
for i := 1, ..., n do
  b := block(vi, i)
  block[i] := b
  while block(vb.first-1).posn +
    block(vb.first-1).width > b.posn do

    b := merge_blocks(block(vb.first-1), b)
    block[b.last] := b
    block[b.first] := b
  endwhile
endfor
/* now compute Ψ */
i := 1
repeat
  b := block[i]
  Ψ(vi) := b.posn + wd(vb.first)/2
  for i := i + 1, ..., b.last do
    Ψ(vi) := Ψ(vi-1) + x_sep(vi-1, vi)
  endfor
  i := b.last + 1
until i > n
return Ψ

function block(v, i)
let b be a new block s.t.
  b.first := i
  b.last := i
  b.width := wd(v)
  b.posn := des(v) - wd(v)/2
  b.wposn := wt(v) × b.posn
  b.weight := wt(v)
return b

function merge_blocks(b1, b2)
let b be a new block s.t.
  b.first := b1.first
  b.last := b2.last
  b.width := b1.width + b2.width
  b.wposn := b1.wposn + b2.wposn - b1.width × b2.weight
  b.weight := b1.weight + b2.weight
  b.posn := b.wposn/b.weight
return b

```

Figure 2: Algorithm to solve the horizontal placement problem for a sequence of vertices v_1, \dots, v_n

$$\begin{aligned}
& \text{minimize } \sum_{i=1}^n wt(v_i) \times (\Psi(v_i) - des(v_i))^2 \\
& \text{subject to } \Psi(v_i) + x_sep(v_i, v_{i+1}) \leq \Psi(v_{i+1}) \\
& \text{for all } i = 1, \dots, n-1.
\end{aligned}$$

One of the major technical contributions of this paper is an algorithm to solve this problem which requires only linear time. It is shown in Figure 2.

The main function *optimal_layout* works by merging vertices into larger and larger “blocks” of vertices where a block is a subsequence of vertices v_f, v_{f+1}, \dots, v_l which are constrained to be adjacent in the configuration, i.e. for $i = f, \dots, l-1$, we have that $\Psi(v_i) + x_sep(v_i, v_{i+1}) = \Psi(v_{i+1})$. We represent a block b using a record with six fields: *first* which is the index of the first vertex in the block; *last* which is the index of the last vertex in the block; *width* which is the total of the widths of the vertices in the block; *posn* which is the optimal placement for the front of the block; and the fields *wposn* and *weight* which we explain shortly. The algorithm also uses an array *block* which gives the block for the start and end vertex in a block.

We process the vertices left to right. At each stage the invariant is that we have found the optimal assignment to v_1, \dots, v_{i-1} . We process vertex v_i as follows. First we assign v_i to its own block created using the function *block* and placing it at *des*(v_i). The problem

is to check for this and if they overlap we merge them into a new block b using *merge_blocks* which also computes the optimal placement of the new block. We repeat this until the block no longer overlaps the preceding block, in which case we have computed the optimal assignment to v_1, \dots, v_i . We use a sentinel block containing a sentinel node v_0 to ensure that this backwards merging of blocks stops.

The optimal placement for a block of vertices is simply the weighted arithmetic mean of the desired locations in the block. To see this, consider x , the optimum placement for the front of a block consisting of the variables v_1, \dots, v_k . Each variable v_i must be placed at $x + (\sum_{j=1}^{i-1} wd(v_j)) + wd(v_i)/2$. Thus we have that x must minimize

$$\sum_{i=1}^k wt(v_i) \times \left(x - des(v_i) + \frac{wd(v_i)}{2} + \sum_{j=1}^{i-1} wd(v_j) \right)^2$$

Equating the derivative to zero we find that the optimal placement for x is at the weighted average of these:

$$\frac{\sum_{i=1}^k wt(v_i) \times (des(v_i) - \frac{wd(v_i)}{2} - \sum_{j=1}^{i-1} wd(v_j))}{\sum_{i=1}^k wt(v_i)}$$

In order to efficiently compute the weighted arithmetic mean when merging two blocks each block has two additional fields: *wposn*, the sum of the weighted desired locations of variables in the block and *weight* the sum of the weights of the variables in the block. When merging we simply add these after appropriately translating *wposn* of the second block.

As an example of *optimal_layout*’s operation consider the vertices:

- v_1 with $des(v_1) = 1$ and $wd(v_1) = 2$
 - v_2 with $des(v_2) = 3$ and $wd(v_2) = 2$
 - v_3 with $des(v_3) = 5$ and $wd(v_3) = 4$
- where all have equal weight 1.

The call *optimal_layout*([v_1, v_2, v_3]) is processed as follows. The sentinel vertex v_0 and block b_0 will be created with $block[0] = b_0$.

In the first iteration vertex v_1 is processed and placed in a new block b_1 where:

- $b_1.first = b_1.last = 1$
- $b_1.width = wd(v_1) = 2$
- $b_1.posn = des(v_1) - wd(v_1)/2 = 0$
- $b_1.wposn = 0 \times 1 = 0$
- $b_1.weight = weight(v_1) = 1$

Then b is set to be b_1 , $block[1] = b = b_1$ and $b = b_1$ is checked to see if it overlaps with the preceding block ($block[0] = b_0$) which it does not.

In the second iteration vertex v_2 is processed and placed in a new block b_2 where:

- $b_2.first = b_2.last = 2$
- $b_2.width = wd(v_2) = 2$
- $b_2.posn = des(v_2) - wd(v_2)/2 = 2$
- $b_2.wposn = 2 \times 1 = 2$
- $b_2.weight = weight(v_2) = 1$

Then b is set to be b_2 , $block[2] = b = b_2$ and $b = b_2$ is checked to see if it overlaps with the preceding block (b_1) which it does not.

In the third iteration vertex v_3 is processed and placed in a new block b_3 where:

- $b_3.first = b_3.last = 3$
- $b_3.width = wd(v_3) = 4$
- $b_3.posn = des(v_3) - wd(v_3)/2 = 3$
- $b_3.wposn = 3 \times 1 = 3$
- $b_3.weight = weight(v_3) = 1$

Then b is set to be b_3 , $block[3] = b_3$ and $b = b_3$ is

b_2 which it does since

$$b_2.posn + b_2.width = 4 > b_3.posn = 3.$$

Thus $merge_blocks(b_2, b_3)$ is called. This creates a new block b_{23} where:

- $b_{23}.first = b_2.first = 2$
 - $b_{23}.last = b_3.last = 3$
 - $b_{23}.width = b_2.width + b_3.width = 6$
 - $b_{23}.wposn = b_2.wposn + b_3.wposn - b_2.width \times b_3.weight = 2 + 3 - 2 \times 1 = 3$
 - $b_{23}.weight = b_2.weight + b_3.weight = 2$
 - $b_{23}.posn = b_{23}.wposn / b_{23}.weight = 3/2$
- and b is set to be b_{23} and $block[2] = block[3] = b_{23}$.

Now $b = b_{23}$ is checked to see if it overlaps with the preceding block b_1 which it does since $b_1.posn + b_1.width = 2 > b_{23}.posn = 1.5$. Thus $merge_blocks(b_1, b_{23})$ will be called. This creates a new block b_{123} where:

- $b_{123}.first = b_1.first = 1$
 - $b_{123}.last = b_{23}.last = 3$
 - $b_{123}.width = b_1.width + b_{23}.width = 8$
 - $b_{123}.wposn = b_1.wposn + b_{23}.wposn - b_1.width \times b_{23}.weight = 0 + 3 - 2 \times 2 = -1$
 - $b_{123}.weight = b_1.weight + b_{23}.weight = 3$
 - $b_{123}.posn = b_{123}.wposn / b_{123}.weight = -1/3$
- and b is set to be b_{123} and $block[1] = block[3] = b_{123}$.

Now $b = b_{123}$ is checked to see if it overlaps with the preceding block (b_0) which it does not so the first while and for loop of $optimal_layout$ are exited and Ψ is computed. This gives:

- $\Psi(v_1) = b_{123}.posn + wd(v_1)/2 = 2/3$
- $\Psi(v_2) = \Psi(v_1) + x_sep(v_1, v_2) = 2 \ 2/3$
- $\Psi(v_3) = \Psi(v_2) + x_sep(v_2, v_3) = 5 \ 2/3$

the optimal configuration.

One issue is how to compute the initial configuration for each layer since we need to have an initial x -coordinate position for all vertices so that we can compute the desired value for each vertex based on the position of both parents and children. We compute this configuration by choosing some arbitrary centre value for the graph and then placing the vertices on each layer as a single block centred on that centre value. This means that subsequent iterations will continue to have compact layout.

Lemma 1 *Evaluation of $optimal_layout([v_1, \dots, v_n])$ will call the functions $block$ and $merge_blocks$ at most $O(n)$ times.*

Theorem 1 *Execution of $optimal_layout([v_1, \dots, v_n])$ has $O(n)$ complexity given $des(v_i)$ is already computed.*

Theorem 2 *Let Ψ be the configuration computed by $optimal_layout([v_1, \dots, v_n])$. Then Ψ minimizes*

$$\sum_{i=1}^n wt(v_i) \times (\Psi(v_i) - des(v_i))^2$$

subject to $\Psi(v_i) + x_sep(v_i, v_{i+1}) \leq \Psi(v_{i+1})$ for $i = 1, \dots, n-1$.

For a proof of the above results the reader is referred to (Marriott, Moulder & Stuckey Forthcoming).

3.3 Merging iterative x -coordinate assignment with edge crossing reduction

As we have already noted the iterative approach to x -coordinate assignment is very similar to the standard iterative approach to edge crossing reduction,

step.

In more detail, we start from an initial ordering for each layer that is computed in the same way as for the edge crossing reduction but do not perform any edge crossing reduction steps instead moving directly to compute the initial (single centred block) for each layer and then iteratively compute configurations for each layer as we repeatedly sweep through the layers in the graph.

When computing a new configuration for the layer we first compute the order for the vertices by sorting the vertices on their desired value. Then this sequence is fed into $optimal_layout$ to determine the configuration for the level with no overlap. This is quite similar to the barycentre method (di Battista et al. 1999), a well-know heuristic for edge-crossing reduction. We call this approach the *merged x -coordinate assignment and edge crossing reduction*.

Unfortunately, the order based on the vertices' desired values may not lead to the minimal cost configuration because of interaction between variable weights and widths. For instance, consider v_1, v_2, v_3 where $wd(v_1) = 10, wt(v_1) = 1000, des(v_1) = 5, wd(v_2) = 1, wt(v_2) = 1, des(v_2) = 6, wd(v_3) = 1, wt(v_3) = 1000, des(v_3) = 6.1$ The minimal cost configuration is based on the order v_1, v_3, v_2 not v_1, v_2, v_3 as one would expect based solely on the desired values.

This suggests that we might also consider a variant of this approach in which we refine the initial ordering based on desired values by repeatedly allowing two vertices to swap if this will decrease the penalty of the layout. This simple idea is similar to the Adjacent-Exchange Algorithm for edge crossing reduction (di Battista et al. 1999). We will only perform a swap if this does not increase the number of edge crossings. The precise algorithm is given in Figure 3. It makes use of the function

$$penalty(v, x) = wt(v) \times (x - des(v))^2.$$

We call this approach the *merged x -coordinate assignment and edge crossing reduction with vertex swapping*.

The algorithm is guaranteed to terminate since at each iteration the total of the vertex penalties will strictly decrease. However, in principle it may swap vertices an exponential number of times so we also terminate the algorithm after a *timeout* is reached. Note that *benefit* is a lower bound on the improvement in the penalty if the vertices are swapped.

We could have improved efficiency by incrementally computing the new blocks in $optimal_layout$. The idea is that if we swap two vertices in block b then we break the block into single vertex blocks and run the main loop of $optimal_layout$ from the first of these blocks until all of the single vertex blocks have been processed and we have reached a block which does not overlap with its predecessor. Currently our implementation does not do this.

4 Y-coordinate assignment: 3 approaches

4.1 Simple compaction

Given the x -coordinates we can now compute the y -coordinates for the vertices.

The simplest approach is to compute each layer's height where the height of a layer is the maximum height of a vertex in the layer and the separation between each layer is half of the sum of the layer heights. We call this *simple compaction*. We believe this is the

```

vertex_ordering(L)
let vertices be the list of vertices in L sorted by des(v)
repeat
  call optimal_layout(vertices)
  let [v1, ..., vn] = vertices
  for j :=1, ..., n-1
    benefit(j) := penalty(vj, Ψ(vj) + wd(vj+1))
                  +penalty(vj+1, Ψ(vj+1) - wd(vj))
                  -penalty(vj, Ψ(vj))
                  -penalty(vj+1, Ψ(vj+1))
  endfor
  choose some j with positive maximal benefit(j) s.t.
  the swap will not increase the number of edge crossings
  vertices := [v1, ..., vj-1, vj+1, vj, vj+2, ..., vn]
until no swap was performed or timeout
return Ψ

```

Figure 3: Algorithm to compute the new configuration Ψ for a layer with vertex set L

usual approach in most implementations of the horizontal layering approach of Sugiyama et al. Unfortunately, this can lead to unnecessary separation between layers if vertex heights vary significantly which is the case for Bayesian networks.

4.2 Uniform compaction

A better approach is to place each layer the minimum distance apart to ensure that no vertices overlap in the vertical direction and that it is a downward drawing. We call this method *uniform vertical compaction*. It allows tall nodes to push into the layers above and below if there is space in these layers.

We do this by computing an assignment Φ which maps each vertex v to the y coordinate of its centre as follows. Let the vertices (including dummy vertices) on layer j be L_j .

We start with the top layer and for all vertices $v \in L_1$ we assign $\Phi(v) = c$ where c is some constant. Now we process the layers in turn. Consider layer $j + 1$. For each node $v \in L_{j+1}$ we compute $down_v$ the maximum height that v can be placed while still preserving downwardness of the drawing,

$$\min\{\Phi(start(e)) \mid e \in In_v \text{ and } start(e) \in L_j\},$$

and $nonoverlap_v$ the maximum height that v can be placed while still ensuring that it does not overlap any vertices on layer j ,

$$\min\{\Phi(u) - y_{sep}(u, v) \mid u \in L_j \text{ and } x_{ovrlp}(u, v)\},$$

where $x_{ovrlp}(u, v)$ holds iff the intervals

$$[\Psi(u) - \frac{1}{2}wd(u), \Psi(u) + \frac{1}{2}wd(u)]$$

$$[\Psi(v) - \frac{1}{2}wd(v), \Psi(v) + \frac{1}{2}wd(v)]$$

intersect and $y_{sep}(u, v) = \frac{1}{2}(ht(u) + ht(v))$. We then set $\Phi(v)$ for all $v \in L_{j+1}$ to

$$\min_{v' \in L_{j+1}} \{down_{v'}, nonoverlap_{v'}\}.$$

One subtlety is that uniform vertical compaction can lead to layouts which are downward but in which some edges cannot be drawn with a poly-line in which each line segment is downward. This is illustrated in Figure 4.

We overcome this by performing preliminary edge-routing then making sure that that if vertex u is on

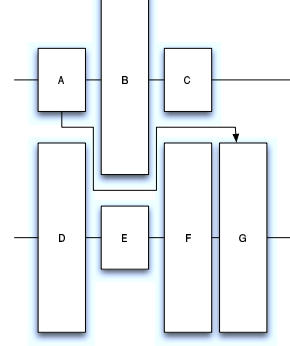


Figure 4: Edge with a non-downward line segment as a result of uniform vertical compaction

some layer j and v is on layer $j + 1$ and for some edge e , $start(e)$ is on layer j and $end(e)$ is on layer $j + 1$ and either

$$\Phi(start(e)) \leq \Phi(u) \leq \Phi(v) \leq \Phi(end(e))$$

or

$$\Phi(start(e)) \geq \Phi(u) \geq \Phi(v) \geq \Phi(end(e))$$

then we do not move vertex v above the bottom of vertex u , i.e. that we ensure that

$$\Phi(u) - a \geq \Phi(v) + b$$

where $a = 0$ if $u = start(e)$ and $a = ht(u)/2$ otherwise and $b = 0$ if $v = end(e)$ and $b = ht(v)/2$ otherwise. In our example the layer with vertices D to G would only move upwards until the top of vertices F is just below the bottom of B . Actually our implementation is slightly more complex than this—if there are multiple edges between the vertices then we need to add proportionally more vertical space.

4.3 Linear programming approach

Unfortunately, uniform vertical compaction can also lead to non-compact layout since as the layer is moved as a whole the vertices on each layer must remain horizontally aligned through their centres. Our third approach is to model y -coordinate assignment as a linear programming problem in which vertices on the same layer are free to move relative to each other.

One approach to ensuring that it is a downward drawing is to add for all $e \in E_{lay}$ the constraint

$$\Phi(start(e)) \geq \Phi(end(e))$$

However, as for uniform vertical compaction, we enforce the stronger requirement that each edge can be drawn with a poly-line in which each line segment is downward. Thus, instead we add constraints of the form

$$\Phi(u) - a \geq \Phi(v) + b$$

described in uniform vertical compaction.

We try to ensure that vertices on the same layer remain roughly horizontally aligned: For each layer j consisting of vertices v_1, \dots, v_K we introduce a new variable M_j and for each $i = 1, \dots, K$ add the constraint

$$\Phi(M_j) = \Phi(v_i) + M_v$$

where M_v is an error variable whose magnitude we try to minimise.

And we ensure that no vertices overlap: For each pair of vertices u and v on layers L_u and L_v respectively which overlap horizontally and for which there

which overlaps horizontally with both u and v we add the constraint

$$\Phi(u) - \Phi(v) \geq y_sep(u, v).$$

We now find the assignment which minimises:

$$\alpha \times \left(\sum_{v \in V_{lay}} |M_v| \right) + \sum_{e \in E_{lay}} wt(e) \times (\Phi(start(e)) - \Phi(end(e)))$$

where α is a scaling constant. This ensures that we try to satisfy the aesthetic criteria and that we minimise the total vertical length of the edges.

We call this the *linear programming approach* to y -coordinate assignment.

We have also investigated a variant in which we reduce the number of vertical non-overlap constraints by restricting the vertical movement of vertices. We add a variable Y_j for each layer j and constraint each node in layer j to be above Y_j . We also add the constraint that $Y_j \geq Y_{j+1}$. Then we constrain vertices in layer $j + 2$ to be below Y_j . This ensures that they do not overlap with any vertices in layers 1 to j and so we must only add explicit vertical non-overlap constraints for the vertices in the preceding layer $j + 1$. This is the variant used in the evaluation.

5 Evaluation

In this section we evaluate the effectiveness and efficiency of the algorithms described in the previous section. All experiments are on a 2010MHz AMD Athlon with 1GB RAM. We used the QOCA C++ Simplex-based linear constraint solver (Marriott & Chok 2002) to solve all linear programming models.

We evaluate the algorithms using 8 representative Bayesian networks. The networks are summarized in Table 1. Two of the models were made by human experts: *Syrotuck* captures the statistical summary of lost person behaviour given in (Syrotuck 1976, 1977, 2000), and *Goulburn* was derived from a formal consensus of experts and stakeholders modelling the Goulburn Broken Catchment, Victoria, Australia. (Woodberry 2003). The rest were generated from datasets by CaMML (Wallace & Korb 1999, Korb & Nicholson 2004), which infers causal models from data according to Minimum Message Length (Wallace & Boulton 1968) principles. *SAR* is a small network on lost-person behaviour learned from the Virginia dataset (Koester 2001). *Davidson* is a 54-node clique from a 235-node network learned to show which of the 40 rain gauges (on 6 previous days) predict the bacteria levels at Davidson Beach in Sydney Harbour. The remaining four were generated from datasets in the UCI machine-learning database repository (Blake & Merz 1998). The dataset *letter-recognition* was chosen because it has a very tall node, *ttt* because it had the highest arc density, *syncon* because it was reasonably large and *adult* because it had lead to bad layout with the Gansner *et al.* layout algorithm.

In our first experiment we evaluate the approaches to x -coordinate assignment. The results are given in Tables 2 and 3. We also show the layout resulting from the four approaches for the *adult* and *Goulburn* benchmarks as representative examples in Figures 5 and 6 after performing y -coordinate assignment using simple compaction and edge routing. Note that edge weights are proportional to the line thickness and arrow size and that we only show the bounding box of the vertices not the actual text.

layer assignment and the number of layers, the time for edge crossing reduction and the number of edge crossings, and the time, width of the layout and the total weighted edge length for x -coordinate assignment using the linear programming approach, and the iterative x -coordinate assignment. Table 3 evaluates merged x -assignment and edge crossing reduction approaches. The first approach does not use vertex swapping while the second does. We did not limit the number of swaps allowed, that is *timeout* is set to ∞ . For each method we give the time, number of edge crossings, width of the layout and the total weighted edge length. All times are in milliseconds and the edge lengths are computed by summing the lengths of the edges (multiplied by the edge weight) produced after performing y -coordinate assignment using simple compaction and edge routing.

Our results indicate that the linear programming and the iterative x -coordinate assignment approaches lead to similar quality layout but that the iterative approach is an order of magnitude faster. They also show that allowing vertex-swapping when combining iterative x -coordinate assignment with edge crossing reduction into a single phase reduces edge crossings and that this combined approach with vertex swapping gives rise to about the same number of edge crossings as using the iterative approach after a separate edge crossing reduction phase, but is somewhat slower.

In our second experiment we evaluate the approaches to y -coordinate assignment. The results are given in Table 4. For each benchmark we started from the layout obtained with iterative x -coordinate assignment and give the time (in milliseconds), the height of the drawing and the total weighted edge length for y -coordinate assignment using simple compaction, uniform compaction and the linear programming approach.

We also show the layout resulting from uniform compaction and the linear programming approaches for the *adult* and *Goulburn* benchmarks as representative examples in Figures 7 and 8 after performing x -coordinate assignment using the iterative method. The layout using simple compaction for these two benchmarks has already been shown in Figures 5 and 6.

We found, as expected, that the linear programming approach generally produces the most compact layout and shortest edges, followed by uniform compaction and lastly simple compaction. However, the differences are relatively small. The linear programming approach is considerably slower than the other two approaches (which are too fast to measure).

6 Conclusion

We have considered the layout of Bayesian networks. The main complication in their layout is that vertices in the network may be large and may vary greatly in height and width. Our approach is based on the horizontal layering algorithm of Sugiyama *et al.*, and in particular the variant of Gansner *et al.* devised for layout of vertices with non-zero width. This has three stages: layer assignment, crossing reduction, and coordinate assignment. The main contribution of this paper is to describe and evaluate techniques for coordinate assignment which better handle vertices with varying height and width.

We described and evaluated four new techniques for x -coordinate assignment: a linear programming approach which is similar to the Gansner *et al.* approach except that instead of minimising the x -component of the distance between connected nodes

	Nodes			Max			Avg			
adult	15	27	8	3.6	296	149	210	259	11	68
Davidson	54	54	6	2.0	118	118	118	9	9	9
Goulburn	35	64	21	3.6	216	125	164	85	9	18
letter-recognition	17	28	11	3.3	186	113	177	66	20	47
SAR	8	8	4	2.0	186	125	155	36	9	19
syncon	61	72	36	2.3	255	174	249	55	16	37
Syrotuck	8	8	5	2.0	240	136	178	46	10	20
ttt	10	23	8	4.6	158	110	115	12	10	10

Table 1: Characteristics of the benchmark Bayesian networks. Node area is kilopixels. Degenerate (zero-area) nodes dropped.

Name	Layer Assignment		Edge Crossing Rdn.		Linear Programming			Iterative		
	time	# layers	time	# crossings	time	width	edge length	time	width	edge length
adult	22	8	4	9	33	1026	11882	7	959	11807
Davidson	89	7	5	4	51	2512	5561	12	2488	5111
Goulburn	81	8	34	27	565	2948	29591	16	2856	26546
letter-recognition	26	10	5	7	61	1391	10645	13	1211	10172
SAR	8	4	1	0	17	594	990	1	572	992
syncon	142	8	32	13	72	9608	102354	16	9326	103648
Syrotuck	7	2	0	3	17	934	1253	0	934	1233
ttt	18	8	6	8	129	461	6531	12	409	6447

Table 2: Evaluation of linear programming and iterative approach to x -placement. Time in milliseconds.

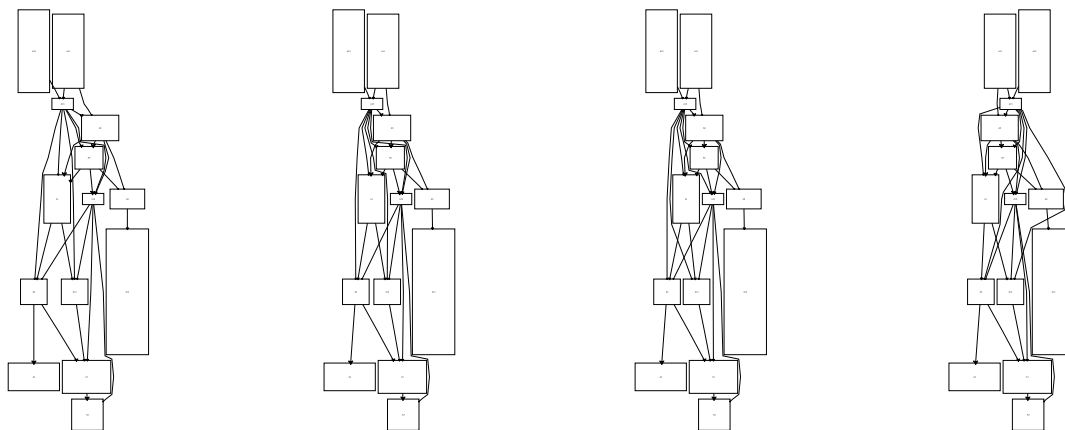


Figure 5: Layout resulting from the different x -coordinate assignment approaches (linear programming, simple iterative, merged without/with swapping) for the *adult* benchmark.

we try and place each node as closely as possible to the weighted average of its children and parents’ x -coordinates; a simple iterative approach which is based on linear time algorithm for solving the horizontal placement problem; and two variants of the iterative approach which combine edge crossing reduction and x -coordinate assignment into a single step. We found that the iterative approach after a separate edge crossing reduction phase was the preferred method.

We also described and evaluated three techniques for y -coordinate assignment: The first, simple compaction, is the standard approach, in which the height of each layer is computed by taking the maximum of the vertex heights in the layer and placing the layers this distance apart. The other two approaches are, we believe, new. Uniform compaction places each layer the minimum distance apart that ensures that no vertices overlap in the vertical direction and that the layout is a downward drawing. The third approach solves a linear program which is similar to uniform compaction but which allows vertices on the same layer to move up/down relative to each other if this leads to more compact layout. We found that both uniform compaction and the linear programming approach produce slightly more compact layouts with shorter edges but that the linear programming ap-

proach is slow and that both uniform compaction and the linear programming approach were difficult to implement because of the aesthetic requirement that edges can be drawn with downward arcs. Thus, disappointingly, based on the current implementations, simple compaction is probably the best choice.

Acknowledgements

We would like to thank Nathan Hurst for suggesting that QOCA be used for Bayesian network layout and that the four of us talk to each other.

References

- AT&T Labs (n.d.), ‘GraphViz’, Software. <http://www.research.att.com/sw/tools/graphviz/>.
- Bayesia Ltd. (n.d.), ‘BayesiaLab’, Software. v3.1, <http://www.bayesia.com>.
- Blake, C. & Merz, C. (1998), ‘UCI repository of machine learning databases’. <http://www.ics.uci.edu/~mlearn/MLRepository.html>.

	time	# crossings	width	edge length	time	# crossings	width	edge length
adult	14	7	963	11630	31	15	924	11971
Davidson	14	7	2477	4085	63	4	2487	3933
Goulburn	40	50	2864	30531	176	33	2856	30550
letter-recognition	20	7	986	10712	65	7	1148	10214
SAR	2	0	619	925	6	0	619	925
syncon	37	18	9313	96467	142	10	9597	97349
Syrotuck	1	3	975	1223	14	3	954	1146
ttt	10	19	453	6508	76	14	473	6520

Table 3: Evaluation of merged x -coordinate assignment and edge crossing reduction approach to x -placement.

Name	Simple			Uniform			Linear Programming		
	time	height	edge length	time	height	edge length	time	height	edge length
adult	0	2982	11807	0	2882	10916	153	2722	9172
Davidson	0	832	5111	0	832	5109	853	906	5202
Goulburn	0	1264	26546	1	1106	25701	1969	999	25314
letter-recognition	0	2720	10172	0	2676	9852	417	2672	9468
SAR	0	722	992	0	712	964	29	722	910
syncon	0	1670	103648	1	1610	103141	1791	1538	102205
Syrotuck	0	462	1233	0	462	1232	22	480	858
ttt	0	1090	6447	0	1090	6447	562	1090	6350

Table 4: Evaluation of simple, uniform, and linear programming approaches to y -placement

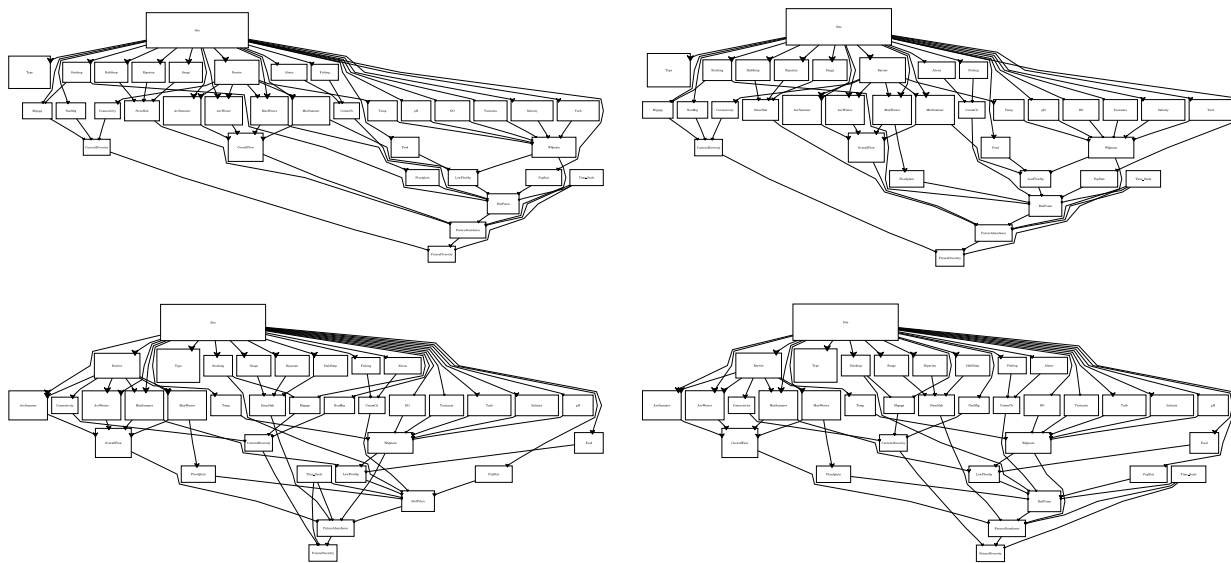


Figure 6: Layout resulting from the different x -coordinate assignment approaches (linear programming, simple iterative, merged without/with swapping) for the *Goulburn* benchmark.

Brandes, U. & Köpf, B. (2001), Fast and simple horizontal coordinate assignment, in 'Proc. 9th Int. Symp. on Graph Drawing', Springer-Verlag LNCS 2265, pp. 31–44.

Decision Systems Laboratory, Univ. of Pittsburgh (n.d.), 'GeNIe', Software. <http://www.sis.pitt.edu/~genie/>.

di Battista, T., Eades, P., Tamassia, R. & Tollis, I. (1999), *Graph Drawing: Algorithms for the visualization of graphs*, Prentice Hall.

Gansner, E. R., Koutsofios, E., North, S. C. & Vo, K.-P. (1993), 'A technique for drawing directed graphs', *IEEE Transactions on Software Engineering* **19**(3), 214–230.

Kansas State Univ. Laboratory for Knowledge Discovery in Databases (n.d.), 'BNJ:

Bayesian Networks in Java', Software. v3, <http://bndev.sourceforge.net>.

Koester, R. J. (2001), 'Virginia dataset on lost-person behavior', Excel file. Contact author at: <http://www.dbs-sar.com>.

Korb, K. B., Hope, L. R., Nicholson, A. E. & Axnick, K. (2004), Varieties of causal intervention, in C. Zhang, H. Guesgen & W. Yeap, eds, 'PRICAI 2004: Trends in Artificial Intelligence', Vol. 3157 of *LNAI*, Pacific Rim International Conference on Artificial Intelligence, Springer-Verlag, Berlin Heidelberg, Auckland, pp. 322–331.

Korb, K. B. & Nicholson, A. E. (2004), *Bayesian Artificial Intelligence*, Chapman & Hall/CRC, chapter 10.

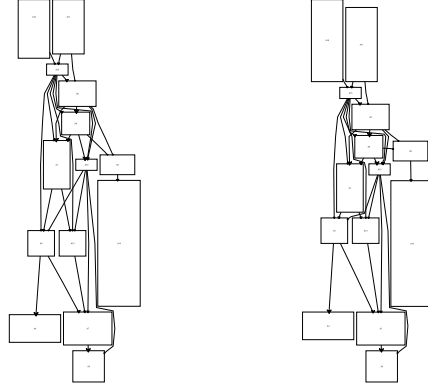


Figure 7: Layout resulting from the different y -coordinate assignment approaches (uniform and linear programming resp.) for the *adult* benchmark. (Figure 5 shows simple layered y -coordinate assignment.)

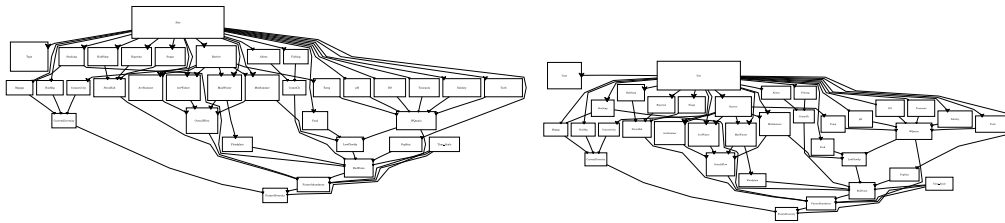


Figure 8: Layout resulting from the different y -coordinate assignment approaches (uniform and linear programming resp.) for the *Goulburn* benchmark. (Figure 6 shows simple layered y -coordinate assignment.)

Marriott, K. & Chok, S. (2002), ‘QOCA: A constraint solving toolkit for interactive graphical applications’, *Constraints* 7(3/4), 229–254.

Marriott, K., Moulder, P. & Stuckey, P. (Forthcoming), Solving separation constraints with desired values.

Neapolitan, R. E. (1990), *Probabilistic Reasoning in Expert Systems*, Wiley & Sons, Inc.

Norsys Inc. (n.d.), ‘Netica v2.17’, Software. <http://www.norsys.com>.

Pearl, J. (1988), *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, San Mateo, CA.

Sugiyama, K., Tagawa, S. & Toda, M. (1981), ‘Methods for visual understanding of hierarchical system structures’, *IEEE Transactions on Systems, Man, and Cybernetics* 11, 109–125.

Syrotuck, W. G. (1976, 1977, 2000), *Analysis of Lost Person Behavior*, Barkleigh Productions, Inc., Mechanicsburg, PA. NASAR version, 2000.

Wallace, C. & Boulton, D. (1968), ‘An information measure for classification’, *The Computer Journal* 11, 185–194.

Wallace, C. S. & Korb, K. B. (1999), Learning linear causal models by MML sampling, in A. Gamerman, ed., ‘Causal Models and Intelligent Data Management’, Springer-Verlag.

Woodberry, O. J. (2003), Knowledge engineering a Bayesian network for an ecological risk assessment, Honours thesis, Monash University, CSSE, Australia. <http://www.csse.monash.edu.au/hons/projects/2003/Owen.Woodberry/>.