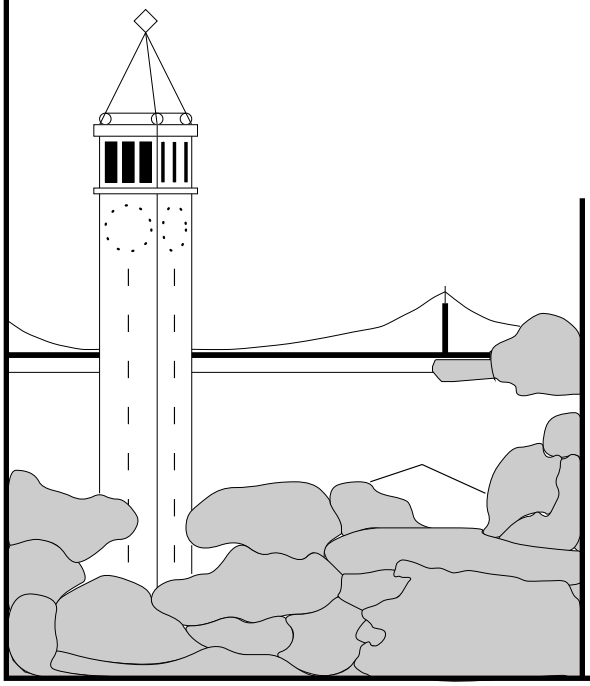
Junction tree algorithms for solving sparse linear systems

Mark A. Paskin
Gregory D. Lawrence



Report No. UCB/CSD-03-1271

September 8, 2003

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Junction tree algorithms for solving sparse linear systems*

Mark A. Paskin
Computer Science Division
University of California, Berkeley
mark@paskin.org

Gregory D. Lawrence
Computer Science Division
University of California, Berkeley
gregl@cs.berkeley.edu

September 8, 2003
(Revised November 7, 2003)

Abstract

In this technical report we present a self-contained introduction to algorithms that solve sparse linear systems by message passing on a junction tree (a.k.a. clique tree). When solving the system $Ax = b$ where A is a sparse $n \times n$ matrix, these methods require $O(n \cdot w^3)$ time and $O(n \cdot w^2)$ space, where w is the treewidth of A's sparsity graph. These algorithms are useful for parallel or distributed solutions to linear systems, or to efficiently solve sets of similar linear systems.

1 Introduction

Consider a linear system of the form

$$Ax = b \tag{1}$$

where A (an invertible $n \times n$ matrix) and b are given and x is to be computed. In this technical report, we present a self-contained introduction to junction tree (a.k.a. clique tree) algorithms for solving the linear system (1). These algorithms require $O(n \cdot w^3)$ time and $O(n \cdot w^2)$ space, where w is the treewidth of the sparsity graph of G.

The connection between junction trees and sparse matrix computation has been well studied [1], and the algorithms presented herein are directly related to earlier techniques, such as the "multifrontal" method [2], which can be formulated in terms of junction trees [3, 4], or inference in a Gaussian graphical model [5], where A (the inverse covariance matrix) is positive definite. The formulation in this technical report is more general than some

*This technical report is work in progress. Please contact the first author for the latest version.

previous treatments in that the matrix A is not assumed to be symmetric or positive definite (although if it is, the algorithm can leverage this structure). Our goal is a presentation that is concise, general, and self-contained. No background in scientific computation is required; only basic linear algebra and basic graph theory are assumed.

1.1 Partial matrices

We will be working extensively with blocks of the matrix A and the vectors x and b, so it will be useful to define some data structures. An *index vector* is an ordered subset of $\mathbf{i} \triangleq \{1, 2, \ldots, n\}$; we will use boldface lower-case letters (e.g., $\mathbf{u}$, $\mathbf{v}$, ...) to denote index vectors. A *partial matrix* is a matrix whose rows and columns are indexed by arbitrary index vectors. For example, here is a partial matrix whose rows are indexed by $\{1, 3, 6\}$ and whose columns are indexed by $\{2, 4, 6\}$:

$$\begin{array}{c} \\ 1 \\ 3 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 4 & 6 \end{array} \\ \left[\begin{array}{ccc} 2 & 9 & -1 \\ 4 & 4 & 2 \\ -3 & 1 & -8 \end{array}\right] \end{array} \tag{2}$$

If M is a partial matrix whose rows are indexed by $\mathbf{j}$ and whose columns are indexed by $\mathbf{k}$, we write $M \in \mathbb{R}_{\mathbf{j}\times\mathbf{k}}$.

Addition and multiplication of partial matrices are defined intuitively. Given two partial matrices $X, Y \in \mathbb{R}_{\mathbf{c}\times\mathbf{d}}$, their sum $S = X + Y$ is the partial matrix $S \in \mathbb{R}_{\mathbf{c}\times\mathbf{d}}$ where $S(i,j) = X(i,j) + Y(i,j)$. Given two partial matrices $X \in \mathbb{R}_{\mathbf{b}\times\mathbf{c}}$ and $Y \in \mathbb{R}_{\mathbf{c}\times\mathbf{d}}$, the product $P = XY$ is the partial matrix $P \in \mathbb{R}_{\mathbf{b}\times\mathbf{d}}$ where

$$P(i,j) = \sum_{k\in\mathbf{c}} X(i,k) \cdot Y(k,j) \tag{3}$$

If $M \in \mathbb{R}_{\mathbf{c}\times\mathbf{c}}$ is invertible then $M^{-1} \in \mathbb{R}_{\mathbf{c}\times\mathbf{c}}$ is a partial matrix with the same row and column indices such that $MM^{-1} = I_{\mathbf{c}\times\mathbf{c}}$ is the (partial) identity matrix.

The chief advantage of the partial matrix concept is that we can generalize the notion of addition. Let $X \in \mathbb{R}_{\mathbf{c}\times\mathbf{d}}$ and let $Y \in \mathbb{R}_{\mathbf{e}\times\mathbf{f}}$ be two partial matrices with different indexing. We define the sum $S = X + Y$ to be the partial matrix $S \in \mathbb{R}_{(\mathbf{c}\cup\mathbf{e})\times(\mathbf{d}\cup\mathbf{f})}$ where

$$S(i,j) = \begin{cases} X(i,j) + Y(i,j) & \text{if } (i,j) \in (\mathbf{c}\cap\mathbf{e}) \times (\mathbf{d}\cap\mathbf{f}) \\ X(i,j) & \text{if } (i,j) \in \mathbf{c}\times\mathbf{d} \wedge (i,j) \notin \mathbf{e}\times\mathbf{f} \\ Y(i,j) & \text{if } (i,j) \notin \mathbf{c}\times\mathbf{d} \wedge (i,j) \in \mathbf{e}\times\mathbf{f} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

More intuitively, we can think of padding X and Y with zero rows and columns so that their rows are indexed by $\mathbf{c}\cup\mathbf{e}$ and their columns are indexed by $\mathbf{d}\cup\mathbf{f}$, and then adding these partial matrices element-wise.

We can subscript a partial matrix by index vectors to extract subblocks. Let $M \in \mathbb{R}_{\mathbf{j}\times\mathbf{k}}$ be a partial matrix, and let $\mathbf{u} \subseteq \mathbf{j}$ and $\mathbf{v} \subseteq \mathbf{k}$. We write $M(\mathbf{u}, \mathbf{v})$ to mean the partial matrix $B \in \mathbb{R}_{\mathbf{u}\times\mathbf{v}}$ such that $B(i, j) = M(i, j)$.

Partial vectors are defined similarly. We write $v \in \mathbb{R}_{\mathbf{k}}$ for a partial vector v whose elements are indexed by $\mathbf{k}$.

2 Decomposable linear systems

We start with a standard result about inverting matrices, and consider its application to the solution of linear systems.

Lemma 1 (partitioned inverse formula). *Let*

$$A = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix} \tag{5}$$

be an invertible matrix. Then

$$A^{-1} = \begin{bmatrix} (A/Z)^{-1} & -(A/Z)^{-1}XZ^{-1} \\ -Z^{-1}Y(A/Z)^{-1} & Z^{-1} + Z^{-1}Y(A/Z)^{-1}XZ^{-1} \end{bmatrix} \tag{6}$$

where

$$A/Z = W - XZ^{-1}Y \tag{7}$$

is called the Schur complement of A with respect to Z.

Proof. Diagonalize A as

$$\underbrace{\begin{bmatrix} I & -XZ^{-1} \\ 0 & I \end{bmatrix}}_{E} \underbrace{\begin{bmatrix} W & X \\ Y & Z \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} I & 0 \\ -Z^{-1}Y & I \end{bmatrix}}_{F} = \underbrace{\begin{bmatrix} W - XZ^{-1}Y & 0 \\ 0 & Z \end{bmatrix}}_{G} \tag{8}$$

Taking the inverse of both sides, we get

$$\begin{aligned} (EAF)^{-1} &= G^{-1} && (9)\\ F^{-1}A^{-1}E^{-1} &= G^{-1} && (10)\\ A^{-1} &= FG^{-1}E && (11) \end{aligned}$$

G is block-diagonal, so we can invert it by inverting its blocks. Thus, we get

$$\begin{aligned} A^{-1} &= FG^{-1}E && (12)\\ &= \begin{bmatrix} I & 0 \\ -Z^{-1}Y & I \end{bmatrix} \begin{bmatrix} (A/Z)^{-1} & 0 \\ 0 & Z^{-1} \end{bmatrix} \begin{bmatrix} I & -XZ^{-1} \\ 0 & I \end{bmatrix} && (13) \end{aligned}$$

where we define $A/Z = W - XZ^{-1}Y$. Multiplying this out yields (6). □

Lemma 2. *The solution x to the block linear system*

$$\begin{bmatrix} W & X \\ Y & Z \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix} \tag{14}$$

satisfies the smaller linear system

$$(W - XZ^{-1}Y)\, x = (b - XZ^{-1}c) \tag{15}$$

Proof. Call the matrix A. From (14) we get

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix}^{-1} \begin{bmatrix} b \\ c \end{bmatrix} \tag{16}$$

Now applying the partitioned inverse formula, we get

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} (A/Z)^{-1} & -(A/Z)^{-1}XZ^{-1} \\ -Z^{-1}Y(A/Z)^{-1} & Z^{-1} + Z^{-1}Y(A/Z)^{-1}XZ^{-1} \end{bmatrix} \begin{bmatrix} b \\ c \end{bmatrix} \tag{17}$$

which implies

$$\begin{aligned} x &= (A/Z)^{-1}b - (A/Z)^{-1}XZ^{-1}c && (18) \\ &= (A/Z)^{-1}(b - XZ^{-1}c) && (19) \end{aligned}$$

Multiplying both sides by the Schur complement

$$A/Z = W - XZ^{-1}Y \tag{20}$$

yields the result (15). □

Definition 1 (decomposition). Let A be a square matrix with index set $\mathbf{i}$. Let $\mathbf{u}, \mathbf{s}, \mathbf{v}$ be a three-way partition of $\mathbf{i}$. *A decomposes on* $(\mathbf{u}, \mathbf{s}, \mathbf{v})$ if $A(\mathbf{u}, \mathbf{v}) = A(\mathbf{v}, \mathbf{u}) = 0$, i.e., if

$$A = \begin{bmatrix} A(\mathbf{u},\mathbf{u}) & A(\mathbf{u},\mathbf{s}) & 0 \\ A(\mathbf{s},\mathbf{u}) & A(\mathbf{s},\mathbf{s}) & A(\mathbf{s},\mathbf{v}) \\ 0 & A(\mathbf{v},\mathbf{s}) & A(\mathbf{v},\mathbf{v}) \end{bmatrix} \tag{21}$$

A linear system $Ax = b$ is *decomposable* if A decomposes on some triple $(\mathbf{u}, \mathbf{s}, \mathbf{v})$.

Lemma 3. *The solution x to the decomposable linear system*

$$\begin{bmatrix} A(\mathbf{u},\mathbf{u}) & A(\mathbf{u},\mathbf{s}) & 0 \\ A(\mathbf{s},\mathbf{u}) & A(\mathbf{s},\mathbf{s}) & A(\mathbf{s},\mathbf{v}) \\ 0 & A(\mathbf{v},\mathbf{s}) & A(\mathbf{v},\mathbf{v}) \end{bmatrix} \begin{bmatrix} x(\mathbf{u}) \\ x(\mathbf{s}) \\ x(\mathbf{v}) \end{bmatrix} = \begin{bmatrix} b(\mathbf{u}) \\ b(\mathbf{s}) \\ b(\mathbf{v}) \end{bmatrix} \tag{22}$$

satisfies the smaller linear system

$$\bar{A}_{\mathbf{us}} \begin{bmatrix} x(\mathbf{u}) \\ x(\mathbf{s}) \end{bmatrix} = \bar{b}_{\mathbf{us}} \tag{23}$$

where

$$\bar{A}_{\mathbf{us}} = \begin{bmatrix} A(\mathbf{u},\mathbf{u}) & A(\mathbf{u},\mathbf{s}) \\ A(\mathbf{s},\mathbf{u}) & A(\mathbf{s},\mathbf{s}) - A(\mathbf{s},\mathbf{v})A(\mathbf{v},\mathbf{v})^{-1}A(\mathbf{v},\mathbf{s}) \end{bmatrix} \tag{24}$$

$$\bar{b}_{\mathbf{us}} = \begin{bmatrix} b(\mathbf{u}) \\ b(\mathbf{s}) - A(\mathbf{s},\mathbf{v})A(\mathbf{v},\mathbf{v})^{-1}b(\mathbf{v}) \end{bmatrix} \tag{25}$$

Proof. If we apply Lemma 2 with the partition

$$A = \left[\begin{array}{cc|c} A(\mathbf{u},\mathbf{u}) & A(\mathbf{u},\mathbf{s}) & 0 \\ A(\mathbf{s},\mathbf{u}) & A(\mathbf{s},\mathbf{s}) & A(\mathbf{s},\mathbf{v}) \\ \hline 0 & A(\mathbf{v},\mathbf{s}) & A(\mathbf{v},\mathbf{v}) \end{array}\right] \qquad b = \left[\begin{array}{c} b(\mathbf{u}) \\ b(\mathbf{s}) \\ \hline b(\mathbf{v}) \end{array}\right] \tag{26}$$

we get that we can obtain part of the solution by solving the smaller system

$$\bar{A}_{\mathbf{us}} \begin{bmatrix} x(\mathbf{u}) \\ x(\mathbf{s}) \end{bmatrix} = \bar{b}_{\mathbf{us}} \tag{27}$$

where

$$\bar{A}_{\mathbf{us}} = A/A(\mathbf{v},\mathbf{v}) \tag{28}$$

$$= \begin{bmatrix} A(\mathbf{u},\mathbf{u}) & A(\mathbf{u},\mathbf{s}) \\ A(\mathbf{s},\mathbf{u}) & A(\mathbf{s},\mathbf{s}) \end{bmatrix} - \begin{bmatrix} 0 \\ A(\mathbf{s},\mathbf{v}) \end{bmatrix} A(\mathbf{v},\mathbf{v})^{-1} \begin{bmatrix} 0 & A(\mathbf{v},\mathbf{s}) \end{bmatrix} \tag{29}$$

$$= \begin{bmatrix} A(\mathbf{u},\mathbf{u}) & A(\mathbf{u},\mathbf{s}) \\ A(\mathbf{s},\mathbf{u}) & A(\mathbf{s},\mathbf{s}) - A(\mathbf{s},\mathbf{v})A(\mathbf{v},\mathbf{v})^{-1}A(\mathbf{v},\mathbf{s}) \end{bmatrix} \tag{30}$$

and

$$\bar{b}_{\mathbf{us}} = \begin{bmatrix} b(\mathbf{u}) \\ b(\mathbf{s}) \end{bmatrix} - \begin{bmatrix} 0 \\ A(\mathbf{s},\mathbf{v}) \end{bmatrix} A(\mathbf{v},\mathbf{v})^{-1}b(\mathbf{v}) \tag{31}$$

$$= \begin{bmatrix} b(\mathbf{u}) \\ b(\mathbf{s}) - A(\mathbf{s},\mathbf{v})A(\mathbf{v},\mathbf{v})^{-1}b(\mathbf{v}) \end{bmatrix} \tag{32}$$

□

Lemma 3 can be interpreted as a describing a "vectorized" form of the elimination operations performed in Gaussian elimination. Consider the decomposable linear system

$$\begin{array}{ccccccc} a_{11}x_1 & + & a_{12}x_2 & & & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & = & b_2 \\ & & a_{32}x_2 & + & a_{33}x_3 & = & b_3 \end{array} \tag{33}$$

which has the augmented matrix

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & 0 & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ 0 & a_{32} & a_{33} & b_3 \end{array}\right] \tag{34}$$

To eliminate x_3, we can subtract a scalar multiple of the bottom row from the middle row. Since

$$a_{23} - \frac{a_{23}}{a_{33}} a_{33} = 0 \tag{35}$$

that scalar multiple should be a_{23}/a_{33}. This gives the augmented matrix

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & 0 & b_1 \\ a_{21} & a_{22} - (a_{23}/a_{33})a_{32} & 0 & b_2 - (a_{23}/a_{33})b_3 \\ 0 & a_{32} & a_{33} & b_3 \end{array}\right] \tag{36}$$

The top two rows of this augmented matrix represent a linear system that does not mention the variable x_3. Its augmented matrix is

$$\left[\begin{array}{cc|c} a_{22} & a_{23} & b_2 \\ a_{32} & a_{33} - (a_{23}/a_{33})a_{32} & b_3 - (a_{23}/a_{33})b_3 \end{array}\right] \tag{37}$$

Breaking this into the corresponding matrix and vector, we see we have recovered (30) and (32) where $\mathbf{u} = \{1\}$, $\mathbf{s} = \{2\}$, and $\mathbf{v} = \{3\}$. Thus, Lemma 3 proves the correctness of a "vectorized" form of Gaussian elimination. In fact, Lemma 3 can also be proved by retracing these steps when $\mathbf{u}$, $\mathbf{s}$, and $\mathbf{v}$ are arbitrary (i.e., not necessarily singleton) sets. The proof above uses the partitioned inverse formula rather than appealing to the correctness of the row operations used by Gaussian elimination.

We now present a theorem that "repackages" the result of Lemma 3 so that the linear system can be represented in a decomposed form.

Theorem 1. *Let $A \in \mathbb{R}_{\mathbf{i}\times\mathbf{i}}$ be an invertible matrix that decomposes on $(\mathbf{u}, \mathbf{s}, \mathbf{v})$ and let $b \in \mathbb{R}_{\mathbf{i}}$ be a vector. Let $\mathbf{c} = \mathbf{u} \cup \mathbf{s}$ and let $\mathbf{d} = \mathbf{s} \cup \mathbf{v}$. Furthermore, let the partial matrices $A_{\mathbf{c}} \in \mathbb{R}_{\mathbf{c}\times\mathbf{c}}$ and $A_{\mathbf{d}} \in \mathbb{R}_{\mathbf{d}\times\mathbf{d}}$ and the partial vectors $b_{\mathbf{c}} \in \mathbb{R}_{\mathbf{c}}$ and $b_{\mathbf{d}} \in \mathbb{R}_{\mathbf{d}}$ satisfy*

$$A = A_{\mathbf{c}} + A_{\mathbf{d}} \tag{38}$$

$$b = b_{\mathbf{c}} + b_{\mathbf{d}} \tag{39}$$

Then the solution x to the decomposable linear system

$$\left[\begin{array}{ccc} A(\mathbf{u},\mathbf{u}) & A(\mathbf{u},\mathbf{s}) & 0 \\ A(\mathbf{s},\mathbf{u}) & A(\mathbf{s},\mathbf{s}) & A(\mathbf{s},\mathbf{v}) \\ 0 & A(\mathbf{v},\mathbf{s}) & A(\mathbf{v},\mathbf{v}) \end{array}\right] \left[\begin{array}{c} x(\mathbf{u}) \\ x(\mathbf{s}) \\ x(\mathbf{v}) \end{array}\right] = \left[\begin{array}{c} b(\mathbf{u}) \\ b(\mathbf{s}) \\ b(\mathbf{v}) \end{array}\right] \tag{40}$$

satisfies the linear systems

$$\bar{A}_{\mathbf{c}} \left[\begin{array}{c} x(\mathbf{u}) \\ x(\mathbf{s}) \end{array}\right] = \bar{b}_{\mathbf{c}} \tag{41}$$

$$\bar{A}_{\mathbf{d}} \left[\begin{array}{c} x(\mathbf{s}) \\ x(\mathbf{v}) \end{array}\right] = \bar{b}_{\mathbf{d}} \tag{42}$$

where

$$\bar{A}_{\mathbf{c}} = \begin{bmatrix} A_{\mathbf{c}}(\mathbf{u},\mathbf{u}) & A_{\mathbf{c}}(\mathbf{u},\mathbf{s}) \\ A_{\mathbf{c}}(\mathbf{s},\mathbf{u}) & A_{\mathbf{c}}(\mathbf{s},\mathbf{s}) + A_{\mathbf{d}}(\mathbf{s},\mathbf{s}) - A_{\mathbf{d}}(\mathbf{s},\mathbf{v})A_{\mathbf{d}}(\mathbf{v},\mathbf{v})^{-1}A_{\mathbf{d}}(\mathbf{v},\mathbf{s}) \end{bmatrix} \tag{43}$$

$$\bar{b}_{\mathbf{c}} = \begin{bmatrix} b_{\mathbf{c}}(\mathbf{u}) \\ b_{\mathbf{c}}(\mathbf{s}) + b_{\mathbf{d}}(\mathbf{s}) - A_{\mathbf{d}}(\mathbf{s},\mathbf{v})A_{\mathbf{d}}(\mathbf{v},\mathbf{v})^{-1}b_{\mathbf{d}}(\mathbf{v}) \end{bmatrix} \tag{44}$$

$$\bar{A}_{\mathbf{d}} = \begin{bmatrix} A_{\mathbf{d}}(\mathbf{s},\mathbf{s}) + A_{\mathbf{c}}(\mathbf{s},\mathbf{s}) - A_{\mathbf{c}}(\mathbf{s},\mathbf{u})A_{\mathbf{c}}(\mathbf{u},\mathbf{u})^{-1}A_{\mathbf{c}}(\mathbf{u},\mathbf{s}) & A_{\mathbf{d}}(\mathbf{s},\mathbf{v}) \\ A_{\mathbf{d}}(\mathbf{v},\mathbf{s}) & A_{\mathbf{d}}(\mathbf{v},\mathbf{v}) \end{bmatrix} \tag{45}$$

$$\bar{b}_{\mathbf{d}} = \begin{bmatrix} b_{\mathbf{d}}(\mathbf{s}) + b_{\mathbf{c}}(\mathbf{s}) - A_{\mathbf{c}}(\mathbf{s},\mathbf{u})A_{\mathbf{c}}(\mathbf{u},\mathbf{u})^{-1}b_{\mathbf{c}}(\mathbf{u}) \\ b_{\mathbf{d}}(\mathbf{v}) \end{bmatrix} \tag{46}$$

Proof. First note that the conditions (38) and (39) imply the following equalities:

$$A_{\mathbf{c}}(\mathbf{u},\mathbf{u}) = A(\mathbf{u},\mathbf{u}) \tag{47}$$
$$A_{\mathbf{c}}(\mathbf{u},\mathbf{s}) = A(\mathbf{u},\mathbf{s}) \tag{48}$$
$$A_{\mathbf{c}}(\mathbf{s},\mathbf{u}) = A(\mathbf{s},\mathbf{u}) \tag{49}$$
$$A_{\mathbf{c}}(\mathbf{s},\mathbf{s}) + A_{\mathbf{d}}(\mathbf{s},\mathbf{s}) = A(\mathbf{s},\mathbf{s}) \tag{50}$$
$$A_{\mathbf{d}}(\mathbf{s},\mathbf{v}) = A(\mathbf{s},\mathbf{v}) \tag{51}$$
$$A_{\mathbf{d}}(\mathbf{v},\mathbf{s}) = A(\mathbf{v},\mathbf{s}) \tag{52}$$
$$A_{\mathbf{d}}(\mathbf{v},\mathbf{v}) = A(\mathbf{v},\mathbf{v}) \tag{53}$$

and also

$$b_{\mathbf{c}}(\mathbf{u}) = b(\mathbf{u}) \tag{54}$$
$$b_{\mathbf{c}}(\mathbf{s}) + b_{\mathbf{d}}(\mathbf{s}) = b(\mathbf{s}) \tag{55}$$
$$b_{\mathbf{d}}(\mathbf{v}) = b(\mathbf{v}) \tag{56}$$

Plugging these into (24) and (25) from Lemma 3 gives (43) and (44). (45) and (46) are proved by interchanging $\mathbf{u}$ and $\mathbf{v}$. □

There is an important analogy underlying Theorem 1. Imagine we have two processors that are connected by a communication link. We would like to solve the decomposable linear system (40). We start by dividing the problem input (A and b) into two pieces $(A_{\mathbf{c}}, b_{\mathbf{c}})$ and $(A_{\mathbf{d}}, b_{\mathbf{d}})$ so that (38) and (39) hold, i.e., so that the pieces add up to the original problem. The notation of Theorem 1 is such that we can regard all terms subscripted by $\mathbf{c}$ as being stored on the first processor, and we can regard all terms subscripted by $\mathbf{d}$ as being stored on the second processor.

Using its portion of the problem, the second processor computes

$$A_{\mathbf{dc}} \stackrel{\triangle}{=} A_{\mathbf{d}}(\mathbf{s},\mathbf{s}) - A_{\mathbf{d}}(\mathbf{s},\mathbf{v})A_{\mathbf{d}}(\mathbf{v},\mathbf{v})^{-1}A_{\mathbf{d}}(\mathbf{v},\mathbf{s}) \tag{57}$$

(a partial matrix in $\mathbb{R}_{\mathbf{s}\times\mathbf{s}}$) and the vector

$$b_{\mathbf{dc}} \stackrel{\triangle}{=} b_{\mathbf{d}}(\mathbf{s}) - A_{\mathbf{d}}(\mathbf{s},\mathbf{v})A_{\mathbf{d}}(\mathbf{v},\mathbf{v})^{-1}b_{\mathbf{d}}(\mathbf{v}) \tag{58}$$

(a partial vector in $\mathbb{R}_{\mathbf{s}}$) and sends the matrix-vector pair $(A_{\mathbf{dc}}, b_{\mathbf{dc}})$ as a "message" to the first processor. Then, the first processor combines these quantities with its portion of the problem:

$$\bar{A}_{\mathbf{c}} = A_{\mathbf{c}} + A_{\mathbf{dc}} \tag{59}$$

$$\bar{b}_{\mathbf{c}} = b_{\mathbf{c}} + b_{\mathbf{dc}} \tag{60}$$

and then solves the linear system (41) using standard techniques.

The first processor also sends a message to the second. In this case the matrix is

$$A_{\mathbf{cd}} \triangleq A_{\mathbf{c}}(\mathbf{s},\mathbf{s}) - A_{\mathbf{c}}(\mathbf{s},\mathbf{u})A_{\mathbf{c}}(\mathbf{u},\mathbf{u})^{-1}A_{\mathbf{c}}(\mathbf{u},\mathbf{s}) \tag{61}$$

and the vector is

$$b_{\mathbf{cd}} \triangleq b_{\mathbf{c}}(\mathbf{s}) - A_{\mathbf{c}}(\mathbf{s},\mathbf{u})A_{\mathbf{c}}(\mathbf{u},\mathbf{u})^{-1}b_{\mathbf{c}}(\mathbf{u}) \tag{62}$$

The second processor then combines these quantities with its portion of the problem to get the linear system (42), which it can solve by standard techniques. Thus, by sending one message in each direction, each processor is able to compute the solution to their portion of the problem. By combining the partial solutions to (41) and (42), we obtain a solution to the complete problem.

Note that if there is special structure present in the original matrix A which makes solving the system easier, then this structure may also propagate to the smaller systems. For example, if A is positive symmetric definite (so that an efficient Cholesky solver can be used), then the smaller system will also be positive symmetric definite, so we can also use the Cholesky solver for the smaller problem.

3 Junction tree decompositions

We now generalize this to the case of multiple processors that are arranged in a tree.

Definition 2 (linear system). A *linear system indexed by* $\mathbf{i}$ is a pair $L = (A, b)$ where A is an invertible partial matrix and b is a partial vector, where both A and b are indexed by the same index vector $\mathbf{i}$.

Definition 3 (junction tree). Let $G = (V, E)$ be a (directed or undirected) graph. A *junction tree for* G is an undirected graph $\mathcal{T} = (\mathcal{C}, \mathcal{E})$ with the following properties.

- Each vertex $\mathbf{c} \in \mathcal{C}$ of the junction tree is a subset of V. These vertices are called *clusters.*
- The junction tree is singly connected; i.e., there is a unique path between any pair of clusters.
- For each edge $(i, j) \in E$ in the original graph G, there is some cluster that contains both i and j.

- If two clusters $\mathbf{c}, \mathbf{d} \in \mathcal{C}$ contain the same element $i \in V$, then all clusters on the unique path between $\mathbf{c}$ and $\mathbf{d}$ also contain i. This is called the *running intersection property.*

Given an edge $\{\mathbf{c}, \mathbf{d}\} \in \mathcal{E}$ of the junction tree, we call $\mathbf{c} \cap \mathbf{d}$ the *separator between* $\mathbf{c}$ *and* $\mathbf{d}$.

Definition 4 (sparsity graph). The sparsity graph for an $n \times n$ matrix A is the directed graph $G = (V, E)$ where $V = \{1, \ldots, n\}$ and

$$(i, j) \in E \iff A(i, j) \neq 0 \tag{63}$$

Definition 5 (junction tree decomposition). Let $L = (A, b)$ be a linear system indexed by $\mathbf{i}$. A *junction tree decomposition* of L is a tuple $(\mathcal{T}, \mathcal{L})$ where

- $\mathcal{T} = (\mathcal{C}, \mathcal{E})$ is a junction tree for the sparsity graph of A
- $\mathcal{L} = \{(A_{\mathbf{c}}, b_{\mathbf{c}}) : \mathbf{c} \in \mathcal{C}\}$ is a set of matrix-vector pairs, one per cluster of the junction tree, where each $A_{\mathbf{c}}$ is a partial square matrix indexed by $\mathbf{c}$ and each $b_{\mathbf{c}}$ is a partial vector indexed by $\mathbf{c}$. These matrix-vector pairs must satisfy

$$A = \sum_{\mathbf{c} \in \mathcal{C}} A_{\mathbf{c}} \tag{64}$$

$$b = \sum_{\mathbf{c} \in \mathcal{C}} b_{\mathbf{c}} \tag{65}$$

We say that $L = (A, b)$ is *the linear system represented by* $(\mathcal{T}, \mathcal{L})$.

A junction tree decomposition of a linear system $L = (A, b)$ defines a *set* of decompositions of L (in the sense of definition 1). In particular, each edge of the junction tree defines such a decomposition:

Lemma 4 (decomposition on an edge). *Let $\mathcal{D} = (\mathcal{T}, \mathcal{L})$ be a junction tree decomposition representing the linear system $L = (A, b)$. For each edge $\{\mathbf{c}, \mathbf{d}\} \in \mathcal{E}$ of the junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{E})$, we have the property that A decomposes on $(\mathbf{c} \setminus \mathbf{d}, \mathbf{c} \cap \mathbf{d}, \mathbf{d} \setminus \mathbf{c})$.*

Proof. Let $\mathbf{u} = \mathbf{c} \setminus \mathbf{d}$ be the indices in $\mathbf{c}$ that are not in $\mathbf{d}$, let $\mathbf{s} = \mathbf{c} \cap \mathbf{d}$ be the intersection of these clusters, and let $\mathbf{v} = \mathbf{d} \setminus \mathbf{c}$ be the indices in $\mathbf{d}$ that are not in $\mathbf{c}$. To prove that A decomposes on $(\mathbf{u}, \mathbf{s}, \mathbf{v})$, we must prove that $A(\mathbf{u}, \mathbf{v}) = A(\mathbf{u}, \mathbf{v}) = 0$.

Assume for a contradiction that $A(\mathbf{u}, \mathbf{v}) \neq 0$ or $A(\mathbf{u}, \mathbf{v}) \neq 0$. Then there must be two indices $i \in \mathbf{u}$ and $j \in \mathbf{v}$ such that $A(i, j) \neq 0$ or $A(j, i) \neq 0$, and therefore the sparsity graph of A has an edge $i \to j$ or $j \to i$. By the definition of a junction tree decomposition, then, there must be some cluster in the junction tree that contains both i and j: call it $\mathbf{b}$. Also, note that because $i \in \mathbf{u}$ and $j \in \mathbf{v}$, we have $i \in \mathbf{c}$, $i \notin \mathbf{d}$, $j \in \mathbf{d}$, and $j \notin \mathbf{c}$.

Because $\mathcal{T}$ is a tree, there must exist a unique path from $\mathbf{b}$ to $\mathbf{c}$, and also a unique path from $\mathbf{b}$ to $\mathbf{d}$. Because $\mathcal{T}$ is a tree and because $\mathbf{c}$ and $\mathbf{d}$ are adjacent in that tree, there are two cases to consider:

1. $\mathbf{c}$ *is on the path from* $\mathbf{b}$ *to* $\mathbf{d}$. In this case the running intersection property does not hold because $\mathbf{b}$ and $\mathbf{d}$ both contain j but $\mathbf{c}$ does not.

2. $\mathbf{d}$ *is on the path from* $\mathbf{b}$ *to* $\mathbf{c}$. In this case too the running intersection property does not hold because $\mathbf{b}$ and $\mathbf{c}$ both contain i but $\mathbf{d}$ does not.

In both cases the premise that $\mathcal{T}$ was a junction tree is contradicted, and so the lemma is proved. □

Definition 6 (passing flows). Let $\mathcal{D}$ be a junction tree decomposition and let $\mathbf{c}$ and $\mathbf{d}$ be two adjacent clusters of its junction tree. Let $\mathbf{s} = \mathbf{c} \cap \mathbf{d}$ be the intersection of these clusters, and let $\mathbf{u} = \mathbf{c} \setminus \mathbf{d}$ be the indices in $\mathbf{c}$ that are not in $\mathbf{d}$. Define

$$A_{\mathbf{cd}} \triangleq A_{\mathbf{c}}(\mathbf{s},\mathbf{s}) - A_{\mathbf{c}}(\mathbf{s},\mathbf{u})A_{\mathbf{c}}(\mathbf{u},\mathbf{u})^{-1}A_{\mathbf{c}}(\mathbf{u},\mathbf{s}) \tag{66}$$

$$b_{\mathbf{cd}} \triangleq b_{\mathbf{c}}(\mathbf{s}) - A_{\mathbf{c}}(\mathbf{s},\mathbf{u})A_{\mathbf{c}}(\mathbf{u},\mathbf{u})^{-1}b_{\mathbf{c}}(\mathbf{u}) \tag{67}$$

The matrix-vector pair $(A_{\mathbf{cd}}, b_{\mathbf{cd}})$ (which is indexed by $\mathbf{s}$) is called *the flow from* $\mathbf{c}$ *to* $\mathbf{d}$.

Let $\mathcal{D}'$ be another junction tree decomposition which is identical to $\mathcal{D}$ except that the matrix-vector pair for cluster $\mathbf{d}$ is $(A'_{\mathbf{d}}, b'_{\mathbf{d}})$ where

$$A'_{\mathbf{d}} = A_{\mathbf{d}} + A_{\mathbf{cd}} \tag{68}$$

$$b'_{\mathbf{d}} = b_{\mathbf{d}} + b_{\mathbf{cd}} \tag{69}$$

The process of updating $\mathcal{D}$ to $\mathcal{D}'$ is called *passing a flow* from $\mathbf{c}$ to $\mathbf{d}$; $\mathbf{c}$ is said to have *passed a flow to* $\mathbf{d}$ and $\mathbf{d}$ is said to have *absorbed a flow from* $\mathbf{c}$.

Definition 7 (leaf elimination). Let $\mathcal{D} = (\mathcal{T}, \mathcal{L})$ be a junction tree decomposition, let $\mathbf{c}$ be a leaf cluster of the junction tree $\mathcal{T}$ and let $\mathbf{d}$ be the cluster adjacent to $\mathbf{c}$. To **eliminate** the leaf $\mathbf{c}$ from the decomposition, we (1) pass a flow from $\mathbf{c}$ to $\mathbf{d}$, and then (2) remove $\mathbf{c}$ from $\mathcal{T}$ and remove its corresponding system $(A_{\mathbf{c}}, b_{\mathbf{c}})$ from $\mathcal{L}$.

Definition 8 (partial solution). Let $L_1 = (A_1, b_1)$ be a linear system with index set $\mathbf{i}_1$ and let $L_2 = (A_2, b_2)$ be a linear system with index set $\mathbf{i}_2 \subseteq \mathbf{i}_1$. The solution $x_2 = A_2^{-1}b_2$ of L_2 is a *partial solution* to L_1 if $x_2 = x_1(\mathbf{i}_2)$ where $x_1 = A_1^{-1}b_1$.

Theorem 2. *Let $\mathcal{D}$ be a junction tree decomposition, and let $\mathcal{D}''$ be the junction tree decomposition obtained by eliminating a leaf from $\mathcal{D}$. Then the solution to the linear system represented by $\mathcal{D}''$ is a partial solution to the linear system represented by $\mathcal{D}$.*

Proof. Let (A, b) be the linear system represented by the junction tree decomposition $\mathcal{D} = (\mathcal{T}, \mathcal{L})$. Let $\mathbf{c}$ be the leaf of the junction tree $\mathcal{T}$ to be eliminated. Furthermore, let $\mathbf{d}$ be the neighbor cluster of $\mathbf{c}$ and let $\mathbf{s} = \mathbf{c} \cap \mathbf{d}$ be their intersection.

Let $\mathbf{u} = \mathbf{c} \setminus \mathbf{s}$ and let $\mathbf{w} = \mathbf{d} \setminus \mathbf{s}$. Because $\mathbf{c}$ is a leaf of $\mathcal{T}$, we can use Lemma 4 to permute A to have the form

$$A = \begin{bmatrix} A(\mathbf{u},\mathbf{u}) & A(\mathbf{u},\mathbf{s}) & 0 & 0 \\ A(\mathbf{s},\mathbf{u}) & A(\mathbf{s},\mathbf{s}) & A(\mathbf{s},\mathbf{w}) & A(\mathbf{s},\mathbf{x}) \\ 0 & A(\mathbf{w},\mathbf{s}) & A(\mathbf{w},\mathbf{w}) & A(\mathbf{w},\mathbf{x}) \\ 0 & A(\mathbf{x},\mathbf{s}) & A(\mathbf{x},\mathbf{w}) & A(\mathbf{x},\mathbf{x}) \end{bmatrix} \tag{70}$$

where $\mathbf{x}$ is the set of all indices that are not in $\mathbf{u}$, $\mathbf{s}$, or $\mathbf{w}$. When we eliminate $\mathbf{c}$, we first pass a flow from $\mathbf{c}$ to $\mathbf{d}$. This consists of the updates (68) and (69). Using (64) and (65), we see that this creates a junction tree decomposition $\mathcal{D}'$ representing (A', b') where

$$A' = \begin{bmatrix} A(\mathbf{u},\mathbf{u}) & A(\mathbf{u},\mathbf{s}) & 0 & 0 \\ A(\mathbf{s},\mathbf{u}) & A(\mathbf{s},\mathbf{s}) - A(\mathbf{s},\mathbf{u})A^{-1}(\mathbf{u},\mathbf{u})A(\mathbf{u},\mathbf{s}) & A(\mathbf{s},\mathbf{w}) & A(\mathbf{s},\mathbf{x}) \\ 0 & A(\mathbf{w},\mathbf{s}) & A(\mathbf{w},\mathbf{w}) & A(\mathbf{w},\mathbf{x}) \\ 0 & A(\mathbf{x},\mathbf{s}) & A(\mathbf{x},\mathbf{w}) & A(\mathbf{x},\mathbf{x}) \end{bmatrix} \tag{71}$$

$$b' = \begin{bmatrix} b(\mathbf{u}) \\ b(\mathbf{s}) - A(\mathbf{s},\mathbf{u})A^{-1}(\mathbf{u},\mathbf{u})b(\mathbf{u}) \\ b(\mathbf{w}) \\ b(\mathbf{x}) \end{bmatrix} \tag{72}$$

Then we remove the leaf $\mathbf{c}$ from the junction tree decomposition, which creates a decomposition $\mathcal{D}''$ representing (A'', b'') where

$$A' = \begin{bmatrix} A(\mathbf{s},\mathbf{s}) - A(\mathbf{s},\mathbf{u})A^{-1}(\mathbf{u},\mathbf{u})A(\mathbf{u},\mathbf{s}) & A(\mathbf{s},\mathbf{w}) & A(\mathbf{s},\mathbf{x}) \\ A(\mathbf{w},\mathbf{s}) & A(\mathbf{w},\mathbf{w}) & A(\mathbf{w},\mathbf{x}) \\ A(\mathbf{x},\mathbf{s}) & A(\mathbf{x},\mathbf{w}) & A(\mathbf{x},\mathbf{x}) \end{bmatrix} \tag{73}$$

$$b' = \begin{bmatrix} b(\mathbf{s}) - A(\mathbf{s},\mathbf{u})A^{-1}(\mathbf{u},\mathbf{u})b(\mathbf{u}) \\ b(\mathbf{w}) \\ b(\mathbf{x}) \end{bmatrix} \tag{74}$$

If we let $\mathbf{v} = \mathbf{w} \cup \mathbf{x}$, we see that we have obtained the system given by (42), and so by Theorem 1, the solution to the system represented by $\mathcal{D}''$ is a partial solution to the system represented by $\mathcal{D}$. □

This theorem hints at a simple *cluster elimination* algorithm for solving decomposable linear systems. We choose a root cluster $\mathbf{r}$ and then repeatedly eliminate leaf clusters from the junction tree decomposition until only $\mathbf{r}$ remains. We then solve the linear system at $\mathbf{r}$ to obtain part of the solution to the problem. We then restore the original junction tree and repeat the process with a different root cluster $\mathbf{r}'$ to obtain another part of the solution. This procedure is repeated with each cluster as root, and then the solution to the entire problem is obtained by concatenating the partial solutions.

4 Junction tree algorithms

The cluster elimination algorithm is inefficient because it recomputes each flow many times; for example, the computations performed when $\mathbf{r}$ is chosen as the root are almost identical to the computations performed when the root is a neighbor of $\mathbf{r}$. In this section, we show how dynamic programming can be used to design algorithms in which each flow is computed once.

In this section we present three efficient junction tree algorithms. The first is based upon a message-passing architecture, and its correctness is proved by demonstrating that

it computes the same results as cluster elimination. The second algorithm returns to a flow-based representation, and its correctness is proved by relating it to the message-passing algorithm. The third algorithm combines flows with backsubstitution to obtain an algorithm that is equivalent to Gaussian elimination.

All three algorithms have the same asymptotic time and space complexity, but the flow-based algorithm has a better time constant than the message-based algorithm, and the backsubstitution algorithm has an even better time constant. The message-based and the flow-based algorithms have an important property that the backsubstitution algorithm lacks: they can efficiently solve "similar" sets of linear systems; we describe how this can be exploited to efficiently invert a matrix.

4.1 A message-passing junction tree algorithm

Recall that when we eliminate a leaf cluster $\mathbf{c}$, we pass a flow to its neighbor $\mathbf{d}$ and then remove it from the decomposition. Consider an algorithm in which we instead compute the flow from $\mathbf{c}$ to $\mathbf{d}$ and send it as a "message" from $\mathbf{c}$ to $\mathbf{d}$—the destination cluster $\mathbf{d}$ stores the flow instead of absorbing it. The source cluster $\mathbf{c}$ is not removed from the junction tree decomposition, and its matrix-vector pair $(A_{\mathbf{c}}, b_{\mathbf{c}})$ does not change.

In the cluster elimination algorithm, we pass a flow from $\mathbf{c}$ to $\mathbf{d}$ when $\mathbf{c}$ is a leaf, which implies that $\mathbf{c}$ has previously absorbed flows from all of its neighbors except $\mathbf{d}$. In the message-passing algorithm, $\mathbf{c}$ receives these flows as messages and stores them. When all of the messages have arrived, $\mathbf{c}$ computes

$$A_{\mathbf{c}}^{\mathbf{d}} = A_{\mathbf{c}} + \sum_{\substack{(\mathbf{b},\mathbf{c})\in\mathcal{E} \\ \mathbf{b}\neq\mathbf{d}}} A_{\mathbf{bc}} \tag{75}$$

$$b_{\mathbf{c}}^{\mathbf{d}} = b_{\mathbf{c}} + \sum_{\substack{(\mathbf{b},\mathbf{c})\in\mathcal{E} \\ \mathbf{b}\neq\mathbf{d}}} b_{\mathbf{bc}} \tag{76}$$

This is the matrix-vector pair that would have been local to $\mathbf{c}$ if we were running the elimination algorithm and $\mathbf{c}$ were a leaf. Then $\mathbf{c}$ computes the flow to $\mathbf{d}$ using this matrix-vector pair; if $\mathbf{s} = \mathbf{c} \cap \mathbf{d}$ and $\mathbf{t} = \mathbf{c} \setminus \mathbf{d}$, the flow is

$$A_{\mathbf{cd}} = A_{\mathbf{c}}^{\mathbf{d}}(\mathbf{s},\mathbf{s}) - A_{\mathbf{c}}^{\mathbf{d}}(\mathbf{s},\mathbf{t})\, A_{\mathbf{c}}^{\mathbf{d}}(\mathbf{t},\mathbf{t})^{-1}\, A_{\mathbf{c}}^{\mathbf{d}}(\mathbf{t},\mathbf{s}) \tag{77}$$

$$b_{\mathbf{cd}} = b_{\mathbf{c}}^{\mathbf{d}}(\mathbf{s}) - A_{\mathbf{c}}^{\mathbf{d}}(\mathbf{s},\mathbf{t})\, A_{\mathbf{c}}^{\mathbf{d}}(\mathbf{t},\mathbf{t})^{-1}\, b_{\mathbf{c}}^{\mathbf{d}}(\mathbf{t}) \tag{78}$$

After this point we discard $(A_{\mathbf{c}}^{\mathbf{d}}, b_{\mathbf{c}}^{\mathbf{d}})$.

To compute the message from $\mathbf{c}$ to $\mathbf{d}$, we must have previously computed the messages to $\mathbf{c}$ from neighbors other than $\mathbf{d}$. This constraint is called the *message-passing protocol:*

$$\text{Before } \mathbf{c} \text{ can pass a message to its neighbor } \mathbf{d}\text{, it must receive messages from all of its other neighbors.} \tag{79}$$

Because the clusters are arranged in a tree, there exist schedules for computing all of the messages that respect this protocol without computing any message more than once. One

simple method of obtaining a schedule is to choose some cluster $\mathbf{r}$ to be the root of the junction tree, so that its edges become directed; then we pass messages from the leaves upwards to the root, and then downward from the root back to the leaves.

Now, assume that we have all of the messages directed to a particular cluster $\mathbf{c}$. Using these messages and the local system $(A_\mathbf{c}, b_\mathbf{c})$ we can construct a new system $(\bar{A}_\mathbf{c}, \bar{b}_\mathbf{c})$ where

$$\bar{A}_\mathbf{c} = A_\mathbf{c} + \sum_{(\mathbf{b},\mathbf{c})\in\mathcal{E}} A_\mathbf{bc} \tag{80}$$

$$\bar{b}_\mathbf{c} = b_\mathbf{c} + \sum_{(\mathbf{b},\mathbf{c})\in\mathcal{E}} b_\mathbf{bc} \tag{81}$$

This is the same system that would arise from running the elimination algorithm with $\mathbf{c}$ as the root: we are simply absorbing all of the flows to $\mathbf{c}$ at the same time. Thus, once we have computed the messages directed to $\mathbf{c}$, we can form a linear system whose solution is a partial solution to the original problem. By doing this at each cluster, we can reconstruct a complete solution to the original problem.

4.2 A flow-based junction tree algorithm

The message-based algorithm is a significant improvement over the cluster elimination algorithm; each flow is computed once, rather than many times. However, there is further redundancy in the computation that can be avoided. Consider the computation of the messages sent by a cluster $\mathbf{c}$ to each of its neighbors. For each neighbor $\mathbf{d}$, $\mathbf{c}$ constructs the system $(A_\mathbf{c}^\mathbf{d}, b_\mathbf{c}^\mathbf{d})$ using (75) and (76). These equations sum together $\mathbf{c}$'s local matrix-vector pair $(A_\mathbf{c}, b_\mathbf{c})$ with all but one of the messages received by $\mathbf{c}$. Thus, if $\mathbf{b}$ and $\mathbf{d}$ are two neighbors of $\mathbf{c}$, then the messages $(A_\mathbf{c}^\mathbf{b}, b_\mathbf{c}^\mathbf{b})$ and $(A_\mathbf{c}^\mathbf{d}, b_\mathbf{c}^\mathbf{d})$ are very similar; we could compute one from another by adding in one message and subtracting out another.

This suggests a scheme where each cluster $\mathbf{c}$ stores the messages it has received as well as a cached sum of these messages and its local linear system. In fact, we can use the storage for $\mathbf{c}$'s local matrix-vector pair $(A_\mathbf{c}, b_\mathbf{c})$ as this cache. It is initialized with $\mathbf{c}$'s portion of the linear system, and each time a message $(A_\mathbf{bc}, b_\mathbf{bc})$ is received, it is summed into the cache $(A_\mathbf{c}, b_\mathbf{c})$ and also stored. Thus, we have returned to a flow-based formulation where the local systems at each cluster are updated with information from adjacent clusters. The difference now is that in addition, the flows are also stored.

When $\mathbf{c}$ sends a message to a neighbor $\mathbf{d}$, it subtracts out the message it most recently received from $\mathbf{d}$ (which it has stored) to obtain the sum of its linear system and the messages from all neighbors but $\mathbf{d}$. Then it computes its message as usual.

When all of the messages have been passed according to the message passing protocol (79), the linear system at each cluster is the original linear system plus all of the messages it has received; therefore, the local system at each cluster $\mathbf{c}$ has now become $(\bar{A}_\mathbf{c}, \bar{b}_\mathbf{c})$, and so by solving it we obtain a partial solution to the original problem.

It will be useful for us to extend this formulation so that each cluster can absorb flows from the same neighbor many times. To do this, each cluster $\mathbf{c}$ is initialized as if it received

a $(0,0)$ flow from each of its neighbors. Then, when $\mathbf{c}$ receives a new flow from a neighbor $\mathbf{b}$, the previous flow is first subtracted from the local system $(A_\mathbf{c}, b_\mathbf{c})$, and then the new flow is added in and stored. This makes passing a flow an idempotent operation: passing the same flow twice has the same effect as passing it once.

4.3 Junction tree algorithm using backsubstitution

Each of the two junction tree algorithms we have described so far is equivalent to the cluster elimination algorithm. Given our interpretation of passing a flow as a vectorized form of Gaussian elimination (see the remarks following Lemma 3), it makes sense to define a junction tree algorithm that is formally equivalent to the complete Gaussian elimination algorithm, including the backsubstitution step.

Definition 9 (backsubstitution). Let (A, b) be a linear system indexed by $\mathbf{i}$, let $\mathbf{s} \subseteq \mathbf{i}$, $\mathbf{u} = \mathbf{i} \setminus \mathbf{s}$, and let $x_\mathbf{s} \in \mathbb{R}_\mathbf{s}$ be a partial solution to (A, b). The *backsubstitution of* $x_\mathbf{s}$ *into* (A, b) is the solution $x_\mathbf{u} \in \mathbb{R}_\mathbf{u}$ to the linear system

$$A(\mathbf{u}, \mathbf{u})\, x_\mathbf{u} \;=\; b(\mathbf{u}) - A(\mathbf{u}, \mathbf{s})\, x_\mathbf{s} \tag{82}$$

Lemma 5. *Let $\mathcal{D}$ be a junction tree decomposition of the linear system (A, b), let $\mathbf{c}$ be a leaf cluster of $\mathcal{D}$, let $\mathbf{d}$ be its neighbor, and let $\mathbf{s} = \mathbf{c} \cap \mathbf{d}$. If $x_\mathbf{s} \in \mathbb{R}_\mathbf{s}$ is a partial solution to (A, b), then the backsubstitution of $x_\mathbf{s}$ into $(A_\mathbf{c}, b_\mathbf{c})$ is also a partial solution of (A, b).*

Proof. Because $\mathbf{c}$ is a leaf of the junction tree, we can use Lemma 4 to permute the linear system to have the form

$$\begin{bmatrix} A_\mathbf{c}(\mathbf{u}, \mathbf{u}) & A_\mathbf{c}(\mathbf{u}, \mathbf{s}) & 0 \\ A_\mathbf{c}(\mathbf{s}, \mathbf{u}) & A(\mathbf{s}, \mathbf{s}) & A(\mathbf{s}, \mathbf{w}) \\ 0 & A(\mathbf{w}, \mathbf{s}) & A(\mathbf{w}, \mathbf{w}) \end{bmatrix} \begin{bmatrix} x(\mathbf{u}) \\ x(\mathbf{s}) \\ x(\mathbf{w}) \end{bmatrix} = \begin{bmatrix} b(\mathbf{u}) \\ b(\mathbf{s}) \\ b(\mathbf{w}) \end{bmatrix} \tag{83}$$

where $\mathbf{w}$ is the set of all indices that are not in $\mathbf{c}$ or $\mathbf{s}$. The solution to this equation satisfies the linear system obtained from the first row of A which can written as

$$\begin{bmatrix} A_\mathbf{c}(\mathbf{u}, \mathbf{u}) & A_\mathbf{c}(\mathbf{u}, \mathbf{s}) \end{bmatrix} \begin{bmatrix} x(\mathbf{u}) \\ x(\mathbf{s}) \end{bmatrix} \;=\; b(\mathbf{u}) \tag{84}$$

or

$$A_\mathbf{c}(\mathbf{u}, \mathbf{u})\, x(\mathbf{u}) \;=\; b(\mathbf{u}) - A_\mathbf{c}(\mathbf{u}, \mathbf{s})\, x(\mathbf{s}) \tag{85}$$

Note that $x(\mathbf{u})$ and $x(\mathbf{s})$ are both partial solutions to (A, b). Thus, if we define

$$A_\mathbf{c}(\mathbf{u}, \mathbf{u})\, x_\mathbf{u} \;=\; b(\mathbf{u}) - A_\mathbf{c}(\mathbf{u}, \mathbf{s})\, x_\mathbf{s} \tag{86}$$

then $x_\mathbf{u}$, which is the backsubstitution of $x_\mathbf{s}$ into $(A_\mathbf{c}, b_\mathbf{c})$, is a partial solution whenever $x_\mathbf{s}$ is a partial solution. □

This lemma allows us to define a junction tree algorithm which is structurally equivalent to Gaussian elimination. We designate one cluster of the junction tree as the root, which induces a directionality on the edges. Then we pass flows from the leaves of this directed junction tree inward towards the root. Once the root $\mathbf{r}$ has absorbed all of its flows, it solves its local linear system to obtain a partial solution $x_{\mathbf{r}}$ to the linear system. Then, instead of passing flows outwards, it sends portions of the solution vector $x_{\mathbf{r}}$.

Let $\mathbf{t}$ be a neighbor of the root, and let $\mathbf{s} = \mathbf{r} \cap \mathbf{t}$. The root sends to $\mathbf{t}$ the subvector $x_{\mathbf{r}}(\mathbf{s})$ of its partial solution $x_{\mathbf{r}}$. When $\mathbf{t}$ receives this partial solution, it computes the back-substitution of $x_{\mathbf{r}}(\mathbf{s})$ into $(A_{\mathbf{t}}, b_{\mathbf{t}})$, the matrix-vector pair at $\mathbf{t}$. Because $\mathbf{t}$ has absorbed flows from all neighbors but the root $\mathbf{r}$, we can view it as a leaf of a junction tree decomposition obtained by eliminating its descendant clusters. Now applying Lemma 5, we get that the backsubstitution computed by $\mathbf{t}$ is also a partial solution to the problem. So $\mathbf{t}$ can continue the backsubstitution process by sending this solution to its children in the junction tree. The process terminates when each leaf has received a partial solution from its parent and has performed its local backsubstitution.

4.4 Efficient local updates

One of the advantages of the (message-based and flow-based) junction tree algorithms is that they provide an efficient means of solving similar linear systems. Assume that we have used a junction tree decomposition $\mathcal{D}$ to compute the solution to a linear system (A, b), and that we now want to solve a new system $(A + \tilde{A}_{\mathbf{c}}, b + \tilde{b}_{\mathbf{c}})$, where $\tilde{A}_{\mathbf{c}}$ is a partial matrix indexed by a cluster $\mathbf{c}$ and $\tilde{b}_{\mathbf{c}}$ is a partial vector indexed by $\mathbf{c}$.

In this case, we can update the junction tree decomposition by adding the local update into $(A_{\mathbf{c}}, b_{\mathbf{c}})$, the local matrix-vector pair at $\mathbf{c}$:

$$A'_{\mathbf{c}} = A_{\mathbf{c}} + \tilde{A}_{\mathbf{c}} \tag{87}$$

$$b'_{\mathbf{c}} = b_{\mathbf{c}} + \tilde{b}_{\mathbf{c}} \tag{88}$$

This creates a decomposition $\mathcal{D}'$ that represents the new system $(A + \tilde{A}_{\mathbf{c}}, b + \tilde{b}_{\mathbf{c}})$. To solve the new system, we could pass all of the messages in $\mathcal{D}'$, but this would not make use of the similarity between the new and old systems.

The following lemma proves that when we update the local system of a single cluster, only the flows directed away from that cluster must be recomputed.

Lemma 6 (single cluster update). *Let $\mathcal{D} = (\mathcal{T}, \mathcal{L})$ be a junction tree decomposition, and let $\mathcal{D}' = (\mathcal{T}, \mathcal{L}')$ be a decomposition obtained by changing the local system $(A_{\mathbf{c}}, b_{\mathbf{c}})$ for some cluster $\mathbf{c}$. Then the flows directed towards $\mathbf{c}$ are the same in $\mathcal{D}$ and $\mathcal{D}'$.*

Proof. Induce a directionality on the junction tree by choosing $\mathbf{c}$ as root. Consider the message passing schedule in which we first send messages from the leaves of this tree toward the root node and then send messages outward from the root to the leaves. Any messages sent during the first phase will remain the same because $\mathcal{L}'$ differs from $\mathcal{L}$ only at the root node $\mathbf{c}$. Thus, all flows directed towards $\mathbf{c}$ are the same in both $\mathcal{D}$ and $\mathcal{D}'$. □

Thus, when the linear system changes within a single cluster $\mathbf{c}$, only half of the flows must be recomputed. In the case where $\mathcal{L}'$ differs from $\mathcal{L}$ in multiple clusters, we can view these changes as occurring in sequence where each step involves making a change to a single cluster. Applying Lemma 6 we find that the set of messages that change is the union of the messages that change in each single cluster update.

Note that we can obtain additional speed-up by instead using the backsubstitution-based algorithm: we can compute the partial solution at the updated cluster $\mathbf{c}$ and propagate partial solutions outward to the leaf clusters. However, in contrast to the message-based or flow-based techniques, this scheme does not leave the junction tree decomposition in a state that permits further efficient local updates in clusters other than $\mathbf{c}$.

One important application of this optimization is when we wish to compute the inverse of A. This problem is typically posed as solving the matrix equation

$$AX = I \tag{89}$$

Solving this equation reduces to solving a set of linear systems, one for each column of X (and I). In this case we have a sequence of linear systems to solve, where each is of the form

$$Ax = e_i \tag{90}$$

where e_i is the vector whose elements are zero, except for $e_i(i) = 1$. We can efficiently solve these equations by first solving the system $Ax = 0$ using one of the junction tree algorithms that send messages in both directions. Even though this equation has a trivial solution, it is important that we actually compute the messages so that we can later take advantage of the single cluster updates. Because each e_i differs from 0 in one element, we can solve for e_i by resending messages out from the cluster that contains that element. This allows us to send half of the messages that would normally be required.

In this particular application, further optimization can be obtained because A remains constant between problems. This means that when we pass a flow, we do not need to perform the update (66). It also means that we can cache the $A_{\mathbf{u}}^{\mathbf{w}}(\mathbf{s}, \mathbf{t})\, A_{\mathbf{u}}^{\mathbf{w}}(\mathbf{t}, \mathbf{t})^{-1}$ term that appears in (67).

References

[1] Jean R. S. Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In Alan George, John R. Gilbert, and Joseph W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, volume 56 of *The IMA Volumes in Mathematics and its Applications*, pages 1–30. Springer–Verlag, 1993.

[2] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9(3):302–325, 1983.

[3] Barry Peyton. *Some applications of clique trees to the solution of sparse linear systems.* PhD thesis, Department of Mathematical Sciences, Clemson University, 1986.

[4] Alex Pothen and Chunguang Sun. Distributed multifrontal factorization using clique trees. In Jack Dongarra, Ken Kennedy, Paul Messina, Danny C. Sorensen, and Robert G. Voigt, editors, *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing, Houston, Texas, USA, March 25-27, 1991*, pages 34–40. SIAM, 1992.

[5] R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, New York, NY, 1999.

When solving the system Ax = b where A is a sparse nA—n matrix, these methods require O(n Ä· w3) time and O(n Ä· w2) space, where w is the treewidth of A's sparsity graph. These algorithms are useful for parallel or distributed solutions to linear systems, or to efficiently solve sets of similar linear systems. Discover the world's research. 20+ million members.Â Junction-tree [56,57] is a parallel direct solving approach. Junction-tree can be considered as both an ordering algorithm and a solving algorithm: Refer to [41,56,57] for further details. The junction-tree ordering can also be used in the "factorise & backsolve" manner described in this chapter [57]. ... Contribute to Sobhym/Sparse-Linear-Systems-Solvers development by creating an account on GitHub.Â Implementation of different iterative methods to solve large sparse linear systems of equations written in matrix form, i.e., Ax=b. Sequential Monte Carlo solvers. Iterative methods that uses Monte Carlo to estimate the inner product of the solution with a weighting vector. Monte Carlo Almost optimal (MAO) algorithm introduced by Dimov. Reference: Dimov, I., & Karaivanova, A. (1997). Iterative Monte Carlo algorithms for linear algebra problems. Numerical Analysis and Its Applications, 150-160. A new faster algorithm I developed during my research. Basic iterative solvers. Basic iterative methods studied in 'Advanced Matrix Algorithims' graduate course. If your system is truly sparse, a sparse solver (that is clever about avoiding fill-ins) will almost always be more efficient than a dense solver.Â Jacobiâ€™s Iterative method and Gauss - Seidel method. For solving a system of linear equations, there are direct methods and iterative methods. The iterative methods are further classified as Jacobiâ€™s iteration method and Gauss - Seidel iterative method. Iterative algorithms are of much use in Finite Element softwares. These algorithms are useful for parallel or distributed solutions to linear systems, or to efficiently solve sets of similar linear systems. (Originally published on September 8, 2003; this is a revised version.) BibTeX citationÂ %0 Report %A Paskin, Mark A. %A Lawrence, Gregory D. %T Junction Tree Algorithms for Solving Sparse Linear Systems %I EECS Department, University of California, Berkeley %D 2003 %@ UCB/CSD-03-1271 %U http://www2.eecs.berkeley.edu/Pubs/TechRpts/2003/5808.html %F Paskin:CSD-03-1271. In Eigen, there are several methods available to solve linear systems when the coefficient matrix is sparse. Because of the special representation of this class of matrices, special care should be taken in order to get a good performance. See Sparse matrix manipulations for a detailed introduction about sparse matrices in Eigen. This page lists the sparse solvers available in Eigen. The main steps that are common to all these linear solvers are introduced as well. Depending on the properties of the matrix, the desired accuracy, the end-user is able to tune those steps in order to improve the performance of its code.