Branch–Induced Sparsity in Rigid–Body Dynamics

Roy Featherstone
Dept. Information Engineering, RSISE
The Australian National University
Branch–Induced Sparsity

What is it?

a pattern of zeros appearing in the joint–space inertia matrix (and some other matrices) as a direct consequence of branches in a kinematic tree

Why is it interesting?

exploiting this sparsity greatly improves the efficiency of $O(n^3)$ dynamics algorithms

What is the main application?

efficient dynamics calculations
Branch–Induced Sparsity

What is it?  
a pattern of zeros appearing in the joint–space *inertia matrix* (and some other matrices) as a direct consequence of branches in a *kinematic tree*

Why is it interesting?  
exploiting this sparsity greatly improves the efficiency of $O(n^3)$ dynamics algorithms

What is the main application?  
efficient dynamics calculations
Kinematic Trees

A rigid–body system can be represented by a connectivity graph in which:

- one node represents a fixed base, or fixed reference frame
- this special node is the root node of the graph
- all other nodes represent bodies
- arcs represent joints

If the connectivity graph is a tree, then the system it represents is a kinematic tree.
Example

- Body 1
- Body 2
- Body 3
- Joint 1
- Joint 2
- Base

Connectivity Graph:
- Root Node
- Joint 1
- Body 1
- Body 2
- Body 3
Numbering Scheme

- the root node is numbered 0
- the other nodes are numbered 1 to $N$ in any order such that each node has a higher number than its parent
- arcs are numbered such that arc $i$ connects node $i$ to its parent
- bodies and joints have the same numbers as their nodes and arcs

Examples
Floating Bases

A mobile robot, or other mobile device, is connected to a fixed base via a \textit{6DoF joint} — a joint that does not impose any motion constraints.

The body that is connected directly to the fixed base is called a \textit{floating base}.
Describing Connectivity

$\kappa(i)$ — all the joints between node $i$ and the root
$\lambda(i)$ — the parent of node $i$
$\mu(i)$ — the children of node $i$
$\nu(i)$ — all the bodies beyond joint $i$
Describing Connectivity

\[ \lambda(1) = 0 \quad \mu(0) = \{1\} \]
\[ \lambda(2) = 1 \quad \mu(1) = \{2,4\} \]
\[ \lambda(3) = 2 \quad \mu(2) = \{3\} \]
\[ \lambda(4) = 1 \quad \mu(3) = \{\} \]

\[ \kappa(1) = \{1\} \quad \nu(1) = \{1,2,3,4,5,6\} \]
\[ \kappa(2) = \{1,2\} \quad \nu(2) = \{2,3\} \]
\[ \kappa(3) = \{1,2,3\} \quad \nu(3) = \{3\} \]
\[ \kappa(4) = \{1,4\} \quad \nu(4) = \{4,5,6\} \]
Describing Connectivity

The parent array, $\lambda$, defines both the connectivity and the numbering scheme.

$$\lambda = [ \lambda(1), \lambda(2), \ldots, \lambda(N) ]$$

\begin{align*}
\lambda &= [0, 1, 2, 1, 4, 4] \\
\lambda &= [0, 1, 1, 2, 3, 2] \\
\lambda &= [0, 1, 2, 0, 1, 2, 5, 5, 2]
\end{align*}
Describing Connectivity

- $\lambda$ provides a complete description of the connectivity; so the sets $\mu(i)$, $\nu(i)$ and $\kappa(i)$ can all be calculated from $\lambda$.

- Most dynamics algorithms only need $\lambda$.

Many algorithms rely on the property $0 \leq \lambda(i) < i$. 
Joint–Space Inertia Matrix

The equation of motion of a kinematic tree can be expressed in the following canonical form:

\[ \tau = H \ddot{q} + C \]

where

\( \tau \) is a vector of joint force variables

\( \ddot{q} \) is a vector of joint acceleration variables

\( H \) is the joint–space inertia matrix

\( C \) is a vector containing Coriolis, centrifugal and gravity terms
Joint–Space Inertia Matrix

The joint–space inertia matrix of a kinematic tree is given by the equation

\[
H_{ij} = \begin{cases} 
S_i^T I_i^c S_j & \text{if } i \in \nu(j) \\
S_i^T I_j^c S_j & \text{if } j \in \nu(i) \\
0 & \text{otherwise}
\end{cases}
\]

The third case in this equation applies whenever \(i\) and \(j\) lie on different branches of the tree. This is the case that gives rise to \textit{branch–induced sparsity}.

\[
H_{ij} = 0 \text{ if } i \text{ and } j \text{ are on different branches}
\]
Joint–Space Inertia Matrix

The joint–space inertia matrix of a kinematic tree is given by the equation

\[
H_{ij} = \begin{cases} 
S_i^T I_i^c S_j & \text{if } i \in \nu(j) \\
S_i^T I_j^c S_j & \text{if } j \in \nu(i) \\
0 & \text{otherwise}
\end{cases}
\]

in general, this is a submatrix

The third case in this equation applies whenever \( i \) and \( j \) lie on different branches of the tree. This is the case that gives rise to *branch–induced sparsity*.

\[ H_{ij} = 0 \text{ if } i \text{ and } j \text{ are on different branches} \]
Sparsity Patterns

- Nonzero submatrix or element
How can we exploit the sparsity?

1. If we factorize $H$ into either $L^T L$ or $L^T D L$, rather than the usual $L L^T$ (Cholesky) or $L D L^T$, then the sparsity pattern in $H$ is preserved in the factors.

2. Algorithms that perform matrix multiplication and back-substitution can be modified to iterate over only the nonzero elements.

3. The more sparsity there is in $H$, the faster it can be calculated and factorized.
Maximizing Sparsity

Choose a floating base near the middle.

\[ H = \]

\[ H = \]
Maximizing Sparsity

Choose a branchy spanning tree.

closed-loop system

spanning tree

$H = \begin{bmatrix}
    1 & 1 & 0 \\
    1 & 1 & 0 \\
    0 & 0 & 1
\end{bmatrix}$

$H = \begin{bmatrix}
    1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
    0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
    0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
    0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{bmatrix}$
$L^T L$ Versus $LL^T$

$H = LL^T$ (Cholesky)

$H = L^T L$
Innovations Factorization

The $L^TDL$ factorization is numerically almost identical to the innovations factorization of the joint–space inertia matrix that was discovered by Rodriguez, Jain, et al. at NASA JPL.

\[
M = (1 + H\phi K) D (1 + H\phi K)^* \\
H \quad L^T \quad D \quad L
\]

\[
M^{-1} = (1 - H\psi K)^* D^{-1} (1 - H\psi K) \\
H^{-1} \quad L^{-1} \quad D^{-1} \quad L^{-T}
\]
Sparse Factorization Algorithms

\[
\begin{align*}
LTL( H, \lambda_e ) & \rightarrow L \\
LTDL( H, \lambda_e ) & \rightarrow L, D
\end{align*}
\]

**Inputs**

- \( H \) — the matrix to be factorized
- \( \lambda_e \) — the *expanded parent array*

**Outputs**

- \( L, D \) — factors returned in \( H \)

**Applicability**

\( H \) can be *any* symmetric positive-definite matrix with the sparsity pattern described by \( \lambda_e \). It does not have to be an inertia matrix.
Expanded Parent Array

\( \lambda \) is an \( N \)-element array, where \( N \) is the number of joints.

\( \lambda_e \) is an \( n \)-element array, where \( n \) is the number of joint variables.

\( H \) is an \( N \times N \) block matrix

\( H \) is an \( n \times n \) matrix

\( \lambda \) describes the sparsity pattern in the submatrices of \( H \).

\( \lambda_e \) describes the sparsity pattern in the elements of \( H \).

\( \lambda_e \) is obtained from \( \lambda \) by formally replacing each multi-DoF joint with an equivalent chain of single-DoF joints and renumbering the nodes and arcs.
Expanded Parent Array

original graph

expanded graph

3 DoF joint

$\lambda = [0,1,1,2,2,3]$

$\lambda_e = [0,1,2,3,1,4,4,5]$
function \texttt{LTDL}( \mathbf{H}, \lambda_e )
for \( k = n \) to 1 do
\( i = \lambda_e(k) \)
while \( i \neq 0 \) do
\( a = \frac{H_{ki}}{H_{kk}} \)
\( j = i \)
while \( j \neq 0 \) do
\( H_{ij} = H_{ij} - H_{kj} \ a \)
\( j = \lambda_e(j) \)
end
\( H_{ki} = a \)
\( i = \lambda_e(i) \)
end
end
How the algorithm works

By iterating only over the ancestors of $k$, the algorithm performs the least possible amount of work, e.g. by updating only 5 elements at $k = 7$ instead of 27.
By iterating only over the ancestors of $k$, the algorithm performs the least possible amount of work, e.g. by updating only 5 elements at $k = 7$ instead of 27.
How the algorithm works

By iterating only over the ancestors of \( k \), the algorithm performs the least possible amount of work, e.g. by updating only 5 elements at \( k = 7 \) instead of 27.
Computational Cost Formulae

\[ L^T D L \] factorization \[ D_1 d + D_2 (m + a) \]
back-substitution \[ nd + 2D_1 (m + a) \]

where

\[ D_1 = \sum_{i=1}^{n} (d_i - 1) \quad \text{and} \quad D_2 = \sum_{i=1}^{n} \frac{d_i (d_i - 1)}{2} \]

\[ d_i = 1 + d_{\lambda_e(i)} \quad (d_0 = 1) \]

\( d_i \) is the depth of node \( i \) in the expanded connectivity graph; and \( d, m \) and \( a \) are the costs of floating-point divide, multiply and add/subtract operations.
Computational Complexity

$D_1$ and $D_2$ are bounded by

$$D_1 \leq n(d - 1) \quad \text{and} \quad D_2 \leq nd(d - 1)/2$$

where $d = \max_i d_i$ is the depth of the expanded connectivity graph.

The complexity of factorization is therefore $O(nd^2)$
Dynamics Calculation Efficiency

- **$O(n)$** algorithms
  - branches have little effect on these algorithms.

- **$O(n^3)$** algorithms
  - branches substantially improve the efficiency of these algorithms, and reduce their complexity from $O(n^3)$ to $O(nd^2)$. 
Dynamics Calculation Efficiency

A typical $O(n^3)$ algorithm performs three steps:

1. calculate $C$ \hspace{2cm} $O(n)$
2. calculate $H$ \hspace{2cm} $O(n^2) \rightarrow O(nd)$
3. solve $H\ddot{q} = \tau - C$ \hspace{2cm} $O(n^3) \rightarrow O(nd^2)$

Branches accelerate steps 2 and 3, and reduce their computational complexity.
Calculating $H$

The *composite–rigid–body algorithm* (CRBA) is the best available algorithm for calculating $H$.

Branch–induced sparsity *improves the efficiency* of this algorithm, and *reduces its complexity* to $O(nd)$, because

1. the CRBA implicitly exploits branch–induced sparsity by calculating only the nonzero elements of $H$, and

2. there are only $n + 2D_1$ nonzero elements in $H$, which is $O(nd)$. 
A Numerical Example

Let us compare the computational cost of forward dynamics for a 30–DoF unbranched chain and the 30–DoF humanoid (or quadruped) shown below.
A Numerical Example

\[ H = \]

\begin{align*}
\text{each} \ \square \ \text{is a} \\
\text{6x6 matrix}
\end{align*}

\( H \) contains:
- 468 nonzero elements
- 432 zero elements

\( H \) is therefore 48% zeros
Cost Figures for Unbranched Chain

\[ O(n^3) \]

<table>
<thead>
<tr>
<th>O(n^3)</th>
<th>RNEA</th>
<th>CRBA</th>
<th>Factor &amp; Solve</th>
</tr>
</thead>
</table>

\[ O(n) \]

ABA

(total arithmetic operations)

5000 10,000 15,000 20,000 25,000

RNEA: Recursive Newton–Euler Algorithm
CRBA: Composite Rigid Body Algorithm
ABA: Articulated–Body Algorithm
Cost Figures for Humanoid/Quadruped

CRBA exploits sparsity

new factorization algorithm
Summary

- branches in a kinematic tree cause sparsity in the joint-space inertia matrix
- exploiting this sparsity, using the new factorization algorithms presented here, greatly improves the efficiency and computational complexity of $O(n^3)$ dynamics algorithms

Further Reading