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

$$\tau = \boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C}$$

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

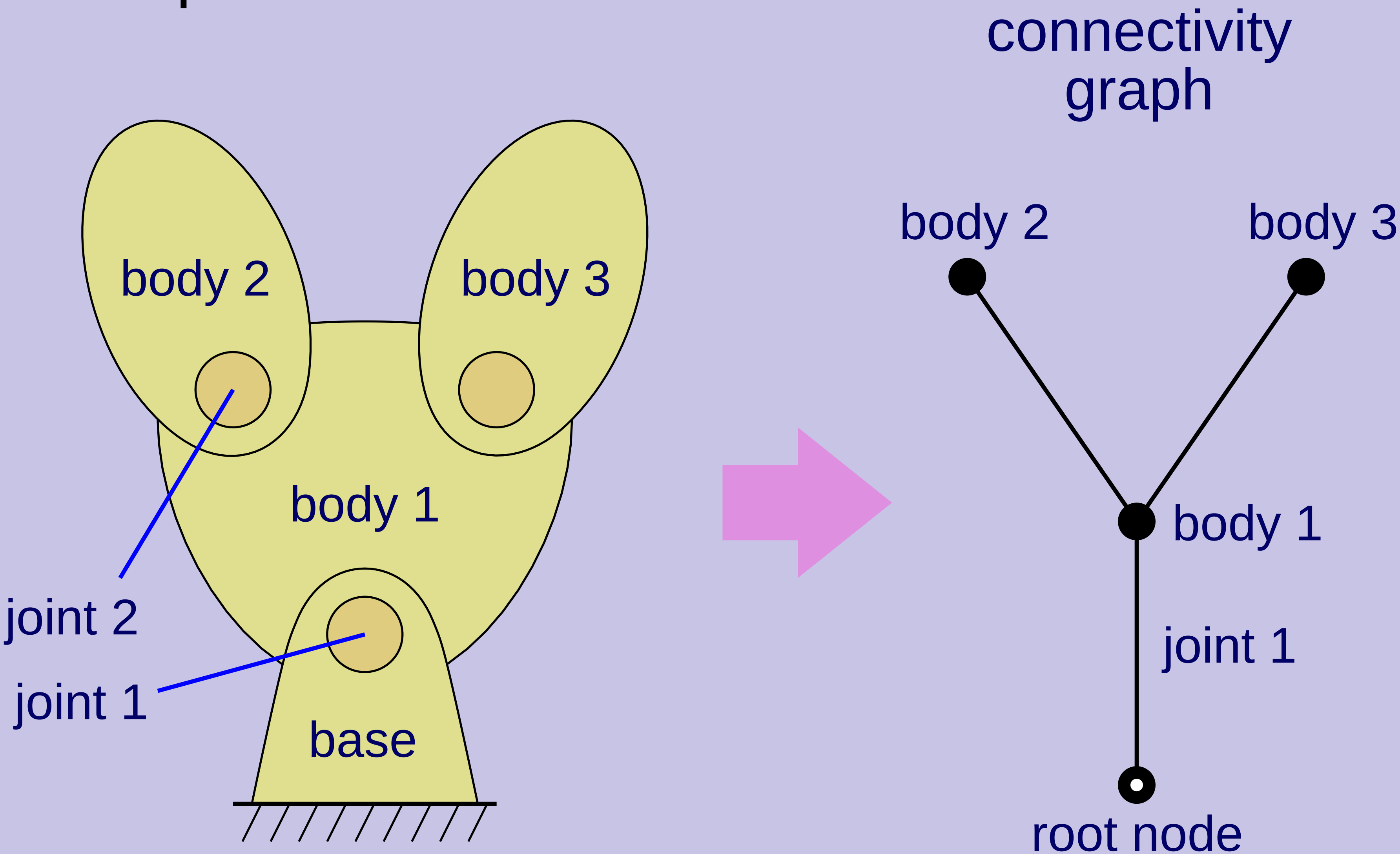
Kinematic Trees

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*.

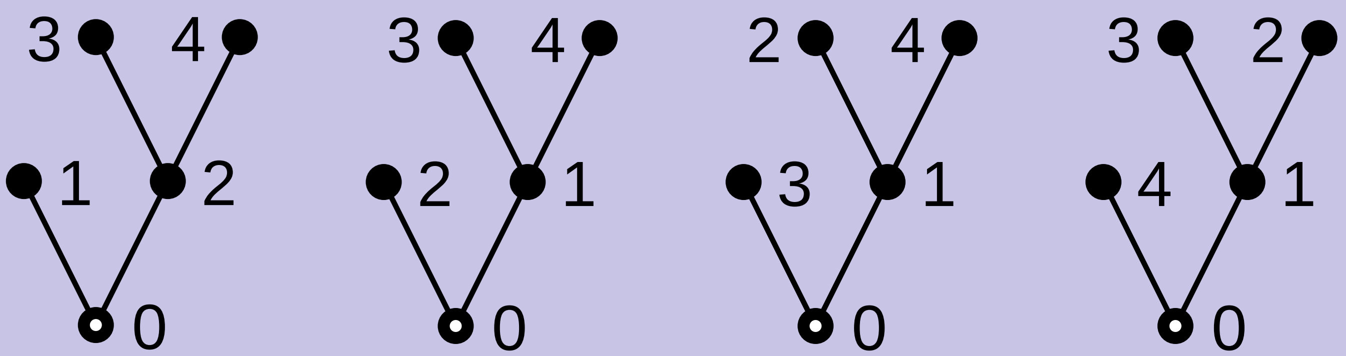
Example


connectivity
graph
body 2
body 3
body 1
joint 2
joint 1
base
body 2
body 3
body 1
joint 1
root node

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

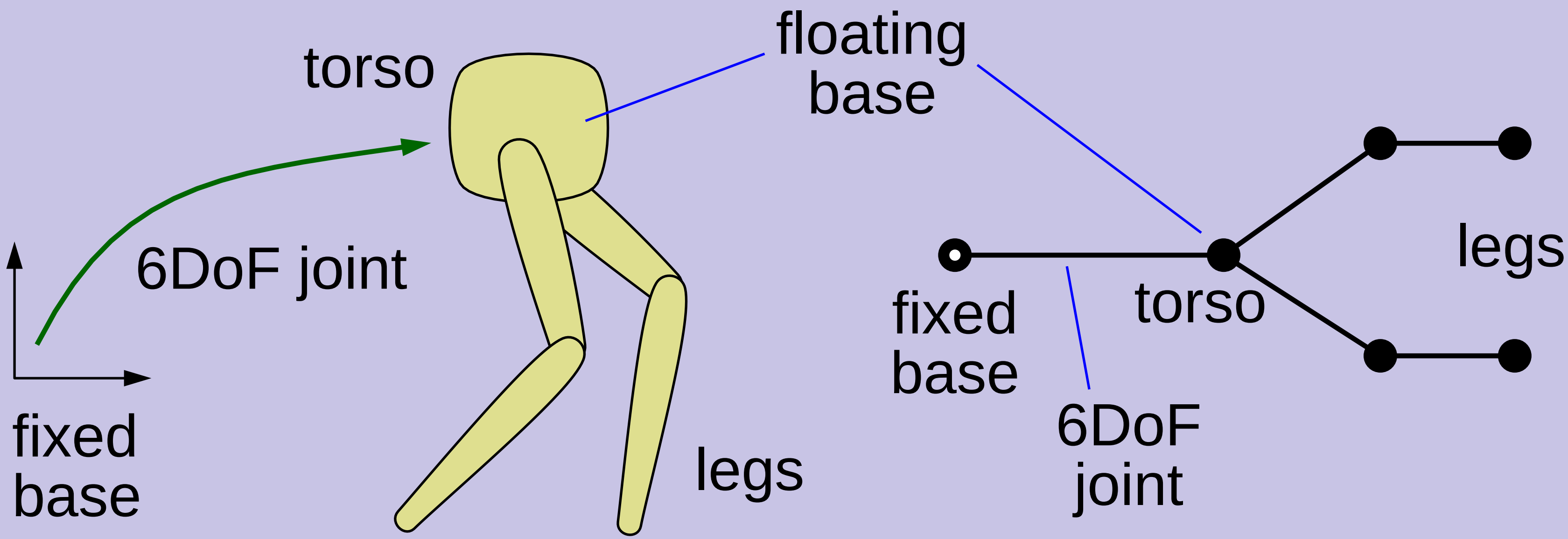
Examples



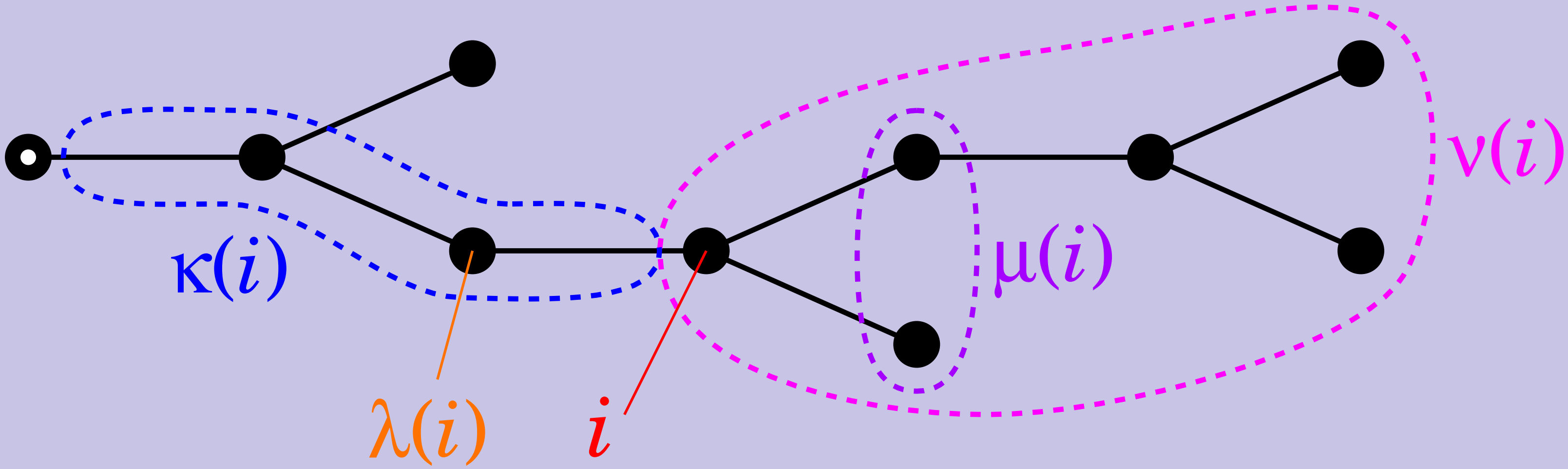
Floating Bases

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

The body that is connected directly to the fixed base is called a *floating base*.



Describing Connectivity



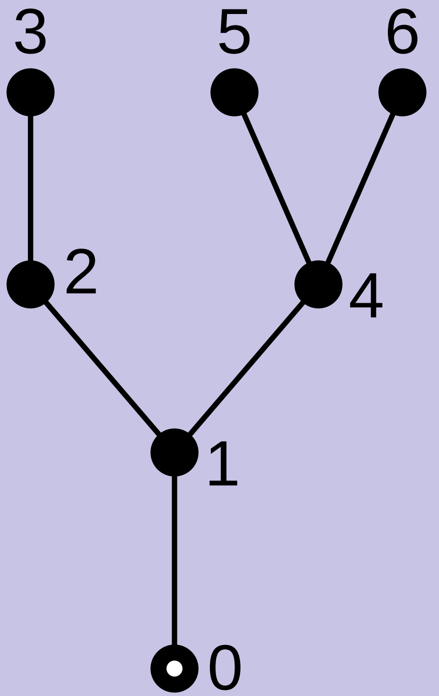
$\kappa(i)$ — all the joints between node i and the root

$\lambda(i)$ — the parent of node i

$\mu(i)$ — the children of node i

$\nu(i)$ — all the bodies beyond joint i

Describing Connectivity



$\lambda(1) = 0$ $\quad$ $\mu(0) = \{1\}$

$\lambda(2) = 1$ $\quad$ $\mu(1) = \{2,4\}$

$\lambda(3) = 2$ $\quad$ $\mu(2) = \{3\}$

$\lambda(4) = 1$ $\quad$ $\mu(3) = \{\}$

$\kappa(1) = \{1\}$ $\quad$ $\nu(1) = \{1,2,3,4,5,6\}$

$\kappa(2) = \{1,2\}$ $\quad$ $\nu(2) = \{2,3\}$

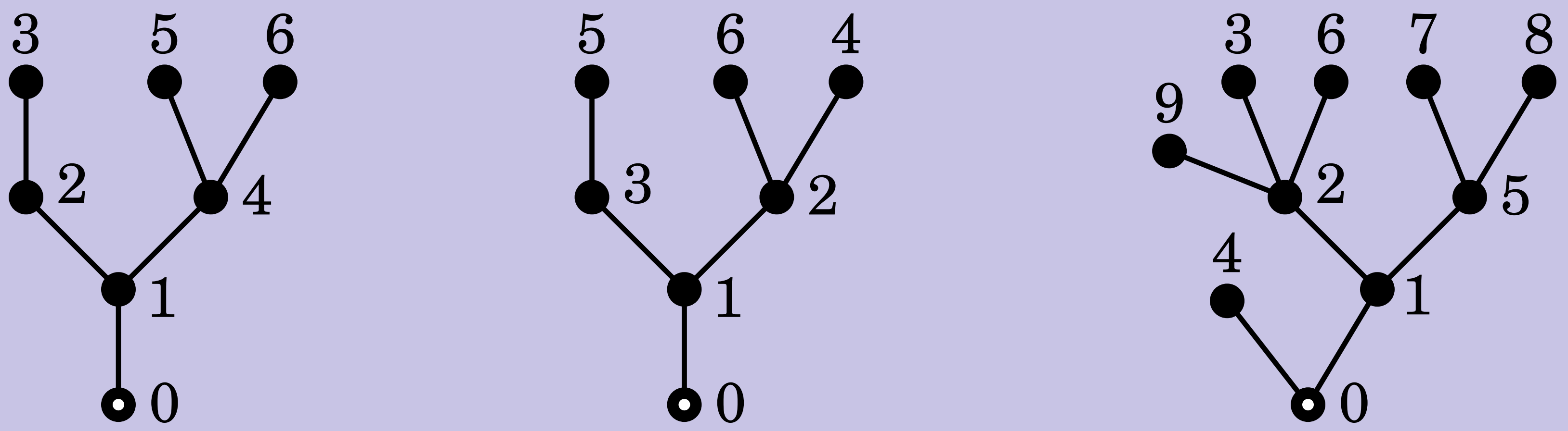
$\kappa(3) = \{1,2,3\}$ $\quad$ $\nu(3) = \{3\}$

$\kappa(4) = \{1,4\}$ $\quad$ $\nu(4) = \{4,5,6\}$

Describing Connectivity

The parent array, λ, defines both the connectivity and the numbering scheme.

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



$\lambda = [0,1,2,1,4,4]$ $\lambda=[0,1,1,2,3,2]$ $\lambda=[0,1,2,0,1,2,5,5,2]$

Describing Connectivity

- λ provides a complete description of the connectivity; so the sets $\mu(i)$, $\nu(i)$ and $\kappa(i)$ can all be calculated from λ.
- Most dynamics algorithms only need λ.

Many algorithms rely on the property $0 \le \lambda(i) < i$.

Joint−Space Inertia Matrix

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

$$\tau = \boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C}$$

where

τ is a vector of joint force variables

$\ddot{\boldsymbol{q}}$ is a vector of joint acceleration variables

$\boldsymbol{H}$ is the *joint−space inertia matrix*

$\boldsymbol{C}$ 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

$$\boldsymbol{H}_{ij} = \begin{cases} \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_i^{\mathrm{c}} \boldsymbol{S}_j & \text{if } \; i \in \nu(j) \\ \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_j^{\mathrm{c}} \boldsymbol{S}_j & \text{if } \; j \in \nu(i) \\ \boldsymbol{0} & \text{otherwise} \end{cases}$$

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 *branch−induced sparsity*.

$\boldsymbol{H}_{ij} = \boldsymbol{0}$ if i and j are on different branches

Joint−Space Inertia Matrix

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

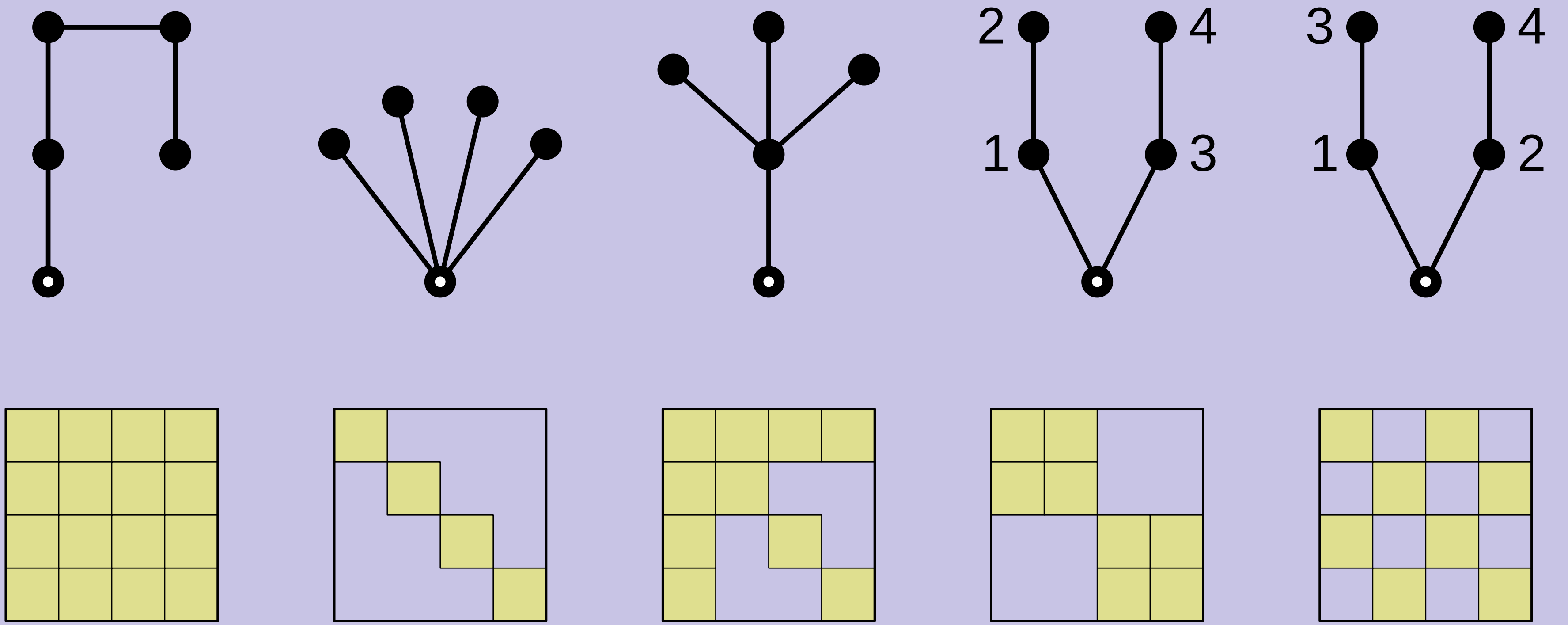
$$\boldsymbol{H}_{ij} = \begin{cases} \boldsymbol{S}_i^{\mathrm{T}}\, \boldsymbol{I}_i^{\mathrm{c}}\, \boldsymbol{S}_j & \text{if} \quad i \in \nu(j) \\ \boldsymbol{S}_i^{\mathrm{T}}\, \boldsymbol{I}_j^{\mathrm{c}}\, \boldsymbol{S}_j & \text{if} \quad j \in \nu(i) \\ \mathbf{0} & \text{otherwise} \end{cases}$$

in general, this is a submatrix

The … n 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*.

$\boldsymbol{H}_{ij} = \mathbf{0}$ if i and j are on different branches

Sparsity Patterns



= nonzero submatrix or element

How can we exploit the sparsity?

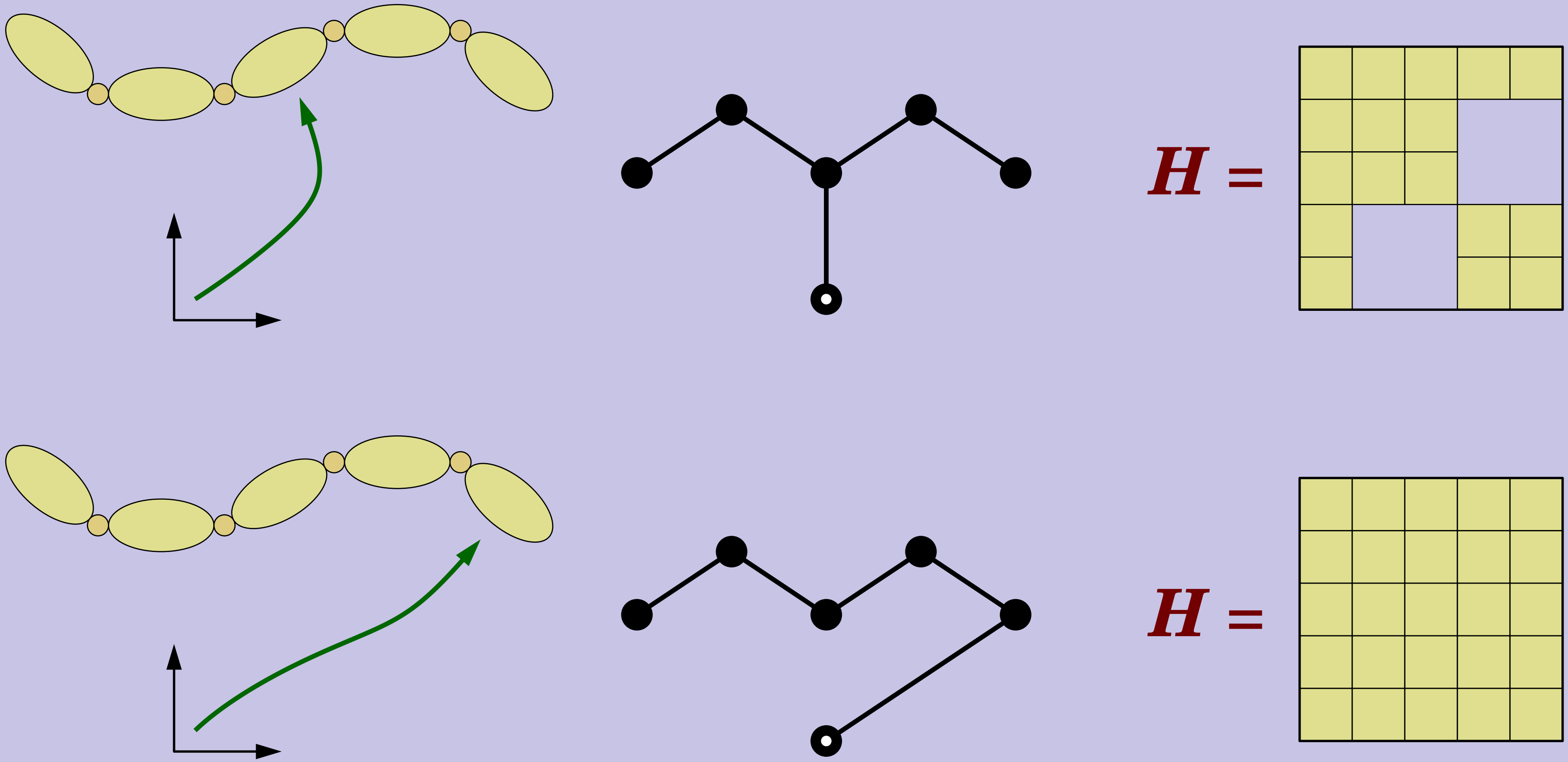
1. If we factorize $\boldsymbol{H}$ into either $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ or $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}$, rather than the usual $\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$ (Cholesky) or $\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\mathrm{T}}$, then the sparsity pattern in $\boldsymbol{H}$ 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 $\boldsymbol{H}$, the faster it can be calculated and factorized.

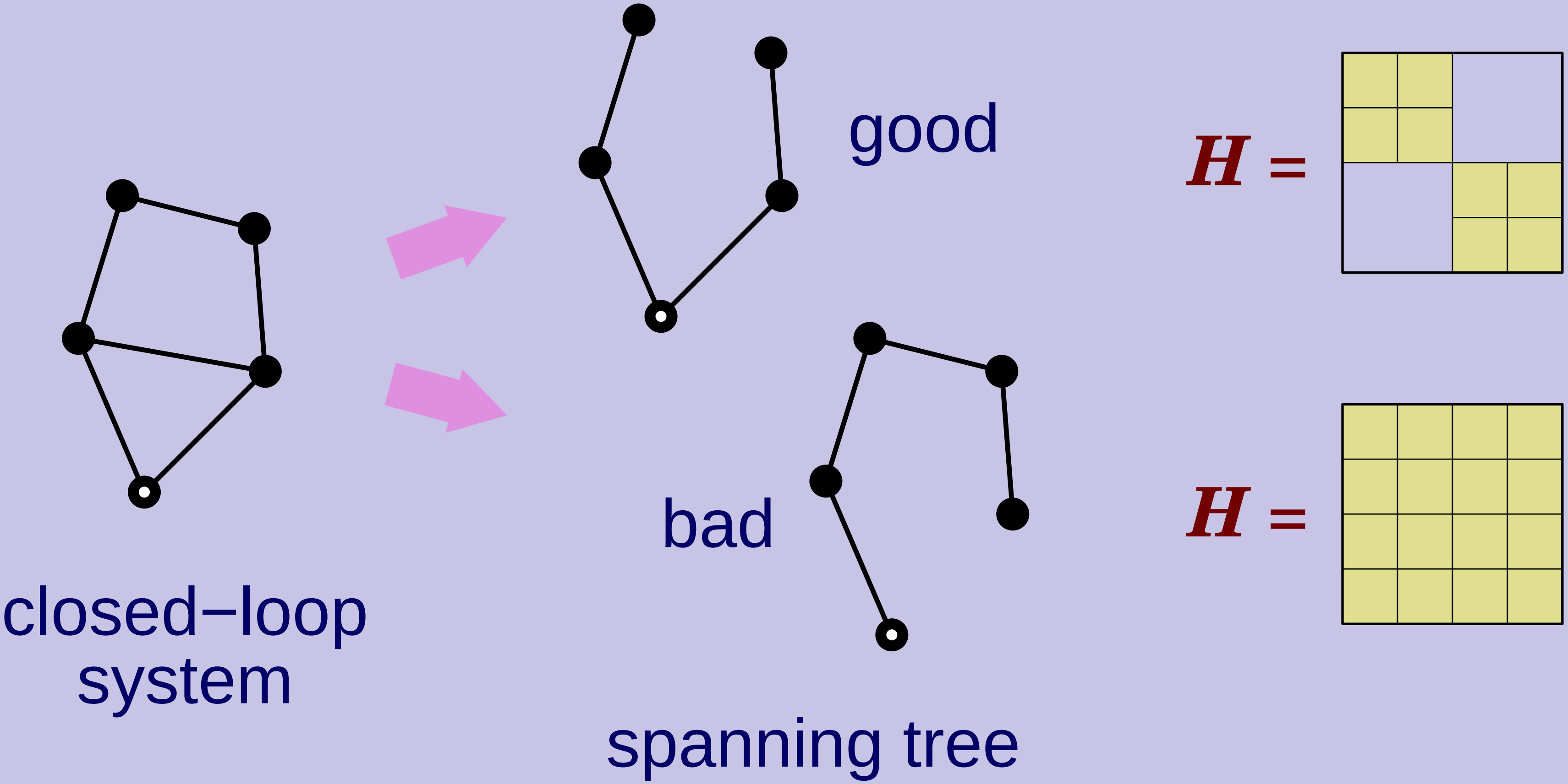
Maximizing Sparsity

Choose a floating base near the middle.

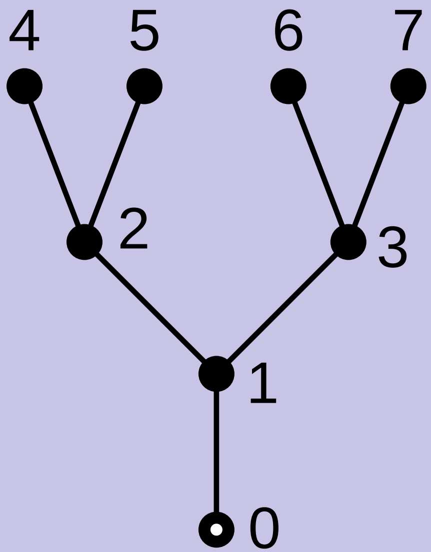


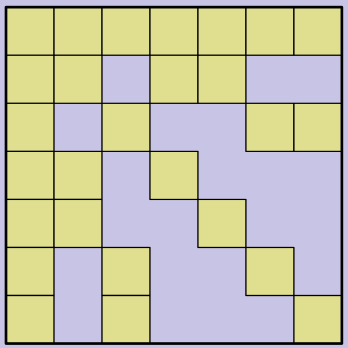
Maximizing Sparsity

Choose a branchy spanning tree.

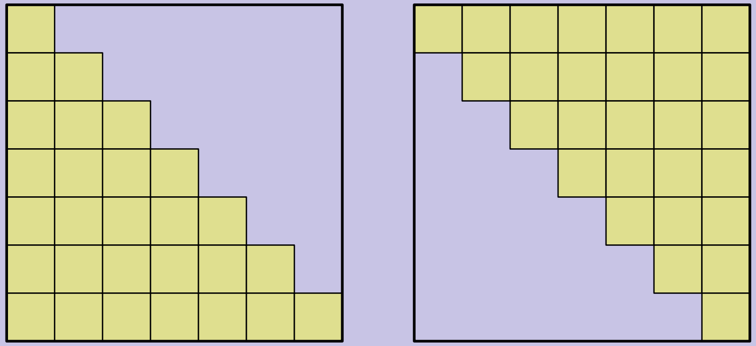


$L^{\mathrm{T}}L$ Versus LL^{T}

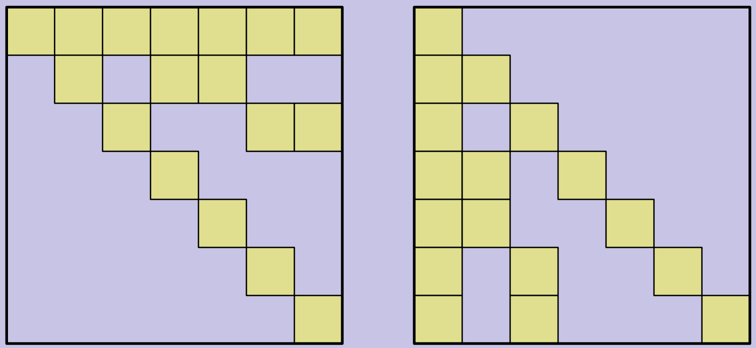


$H =$ 

$H = LL^{\mathrm{T}}$ (Cholesky)

$H =$ 

$H = L^{\mathrm{T}}L$

$H =$ 

Innovations Factorization

The $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}$ 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.

$$\underbrace{\boldsymbol{M}}_{\boldsymbol{H}} = \underbrace{(1+\boldsymbol{H}\phi\boldsymbol{K})}_{\boldsymbol{L}^{\mathrm{T}}}\ \underbrace{\boldsymbol{D}}_{\boldsymbol{D}}\ \underbrace{(1+\boldsymbol{H}\phi\boldsymbol{K})^{*}}_{\boldsymbol{L}}$$

$$\underbrace{\boldsymbol{M}^{-1}}_{\boldsymbol{H}^{-1}} = \underbrace{(1-\boldsymbol{H}\psi\boldsymbol{K})^{*}}_{\boldsymbol{L}^{-1}}\ \underbrace{\boldsymbol{D}^{-1}}_{\boldsymbol{D}^{-1}}\ \underbrace{(1-\boldsymbol{H}\psi\boldsymbol{K})}_{\boldsymbol{L}^{-\mathrm{T}}}$$

Sparse Factorization Algorithms

$$\mathrm{LTL}(\boldsymbol{H}, \lambda_e) \longrightarrow \boldsymbol{L}$$
$$\mathrm{LTDL}(\boldsymbol{H}, \lambda_e) \longrightarrow \boldsymbol{L}, \boldsymbol{D}$$

Inputs	$\boldsymbol{H}$	—	the matrix to be factorized
	λ_e	—	the *expanded parent array*
Outputs	$\boldsymbol{L}, \boldsymbol{D}$	—	factors returned in $\boldsymbol{H}$

Applicability

$\boldsymbol{H}$ can be *any* symmetric positive−definite matrix with the sparsity pattern described by λ_e. It does not have to be an inertia matrix.

Expanded Parent Array

λ is an N−element array, where N is the number of joints.

$\boldsymbol{H}$ is an $N \times N$ block matrix

λ describes the sparsity pattern in the submatrices of $\boldsymbol{H}$.

λ_{e} is an n−element array, where n is the number of joint variables.

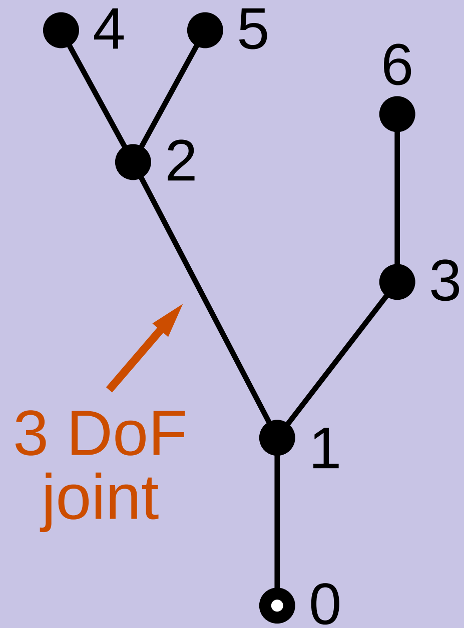
$\boldsymbol{H}$ is an $n \times n$ matrix

λ_{e} describes the sparsity pattern in the elements of $\boldsymbol{H}$.

λ_{e} is obtained from λ by formally replacing each multi−DoF joint with an equivalent chain of single−DoF joints and renumbering the nodes and arcs.

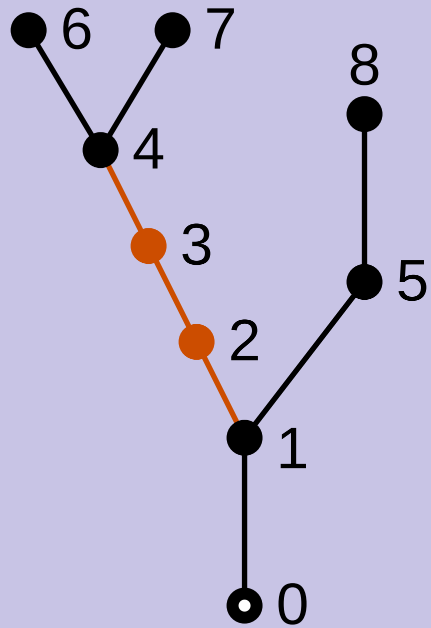
Expanded Parent Array

original graph



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

expanded graph



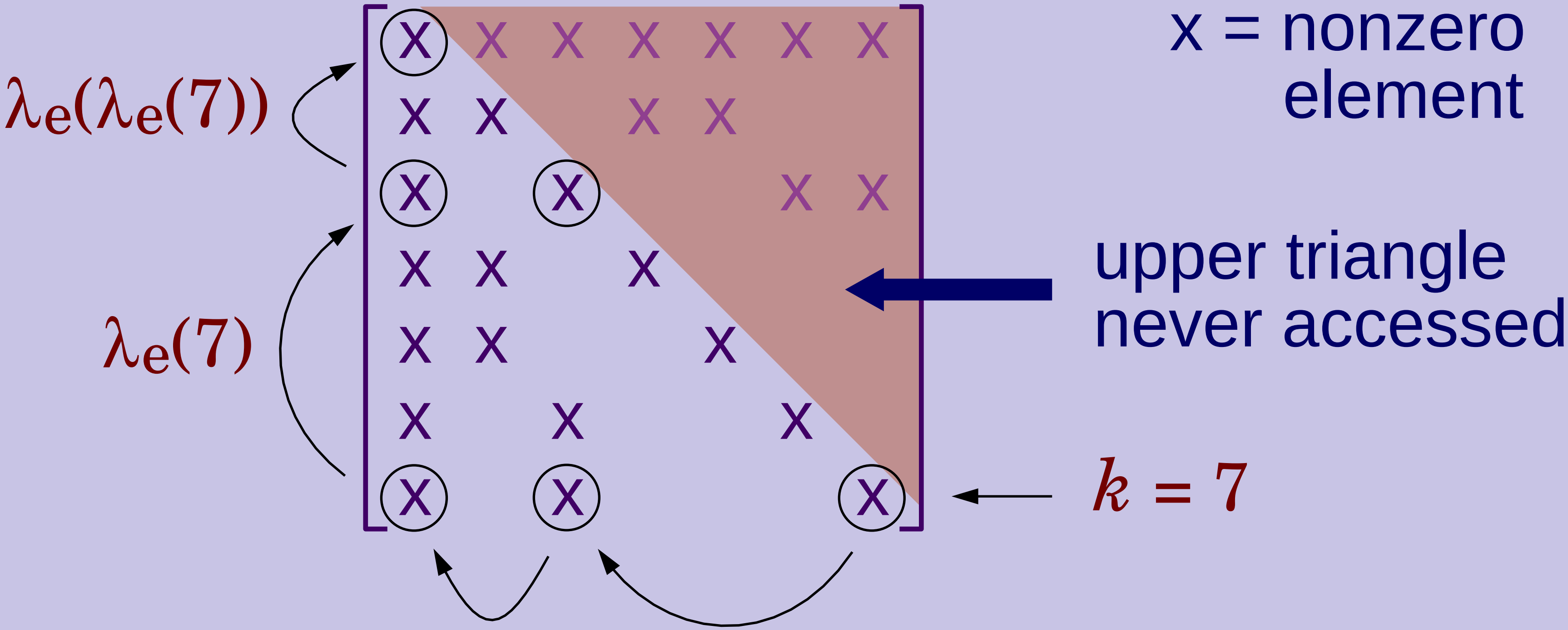
$$\lambda_e = [0,1,2,3,1,4,4,5]$$

```
function LTDL( H, λe )
    for k = n to 1 do
        i = λe(k)
        while i ≠ 0 do
            a = H_ki / H_kk
            j = i
            while j ≠ 0 do
                H_ij = H_ij − H_kj a
                j = λe(j)
            end
            H_ki = a
            i = λe(i)
        end
    end
```

loop runs backwards

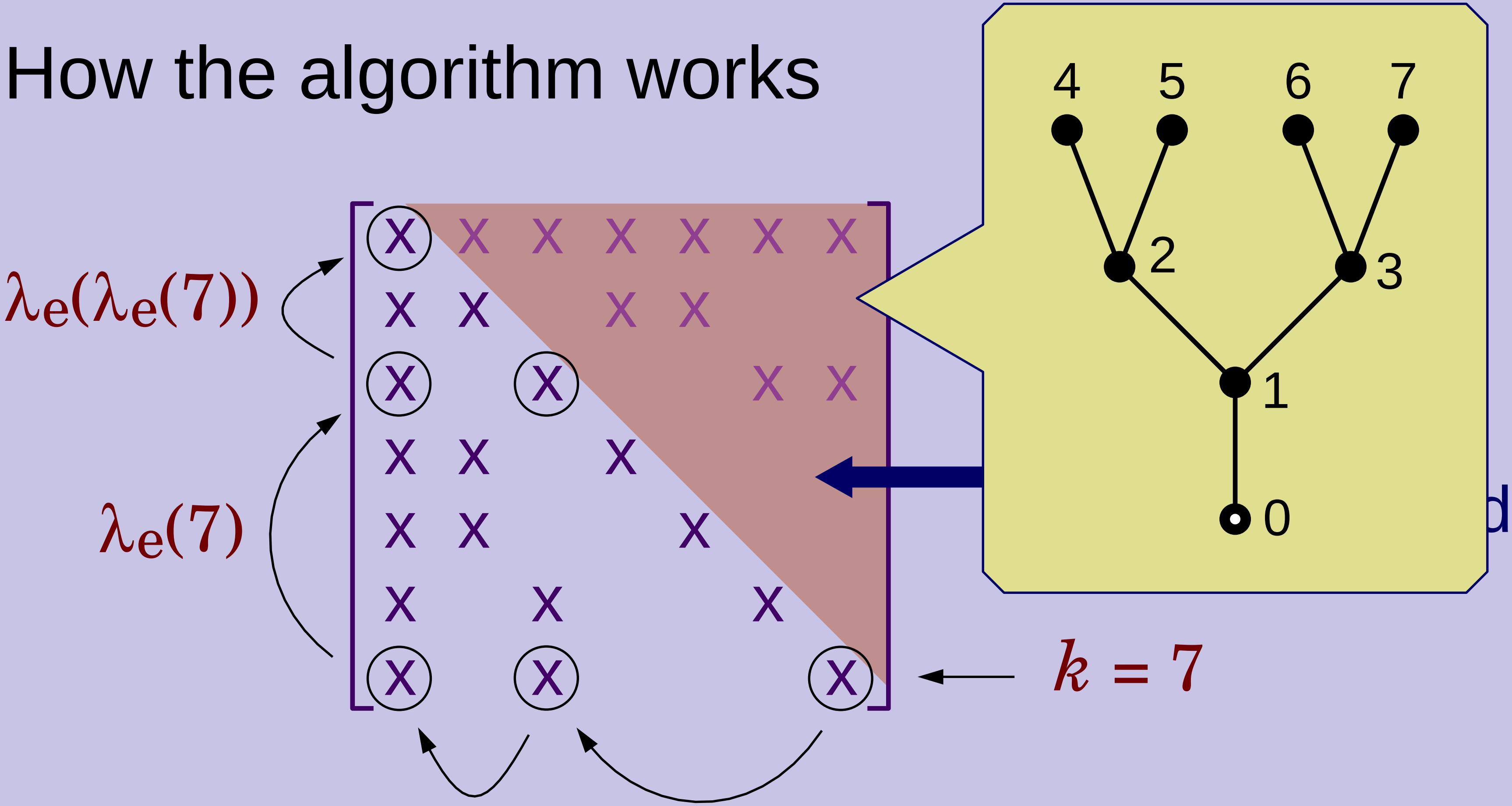
loops iterate over ancestors of k

How the algorithm works



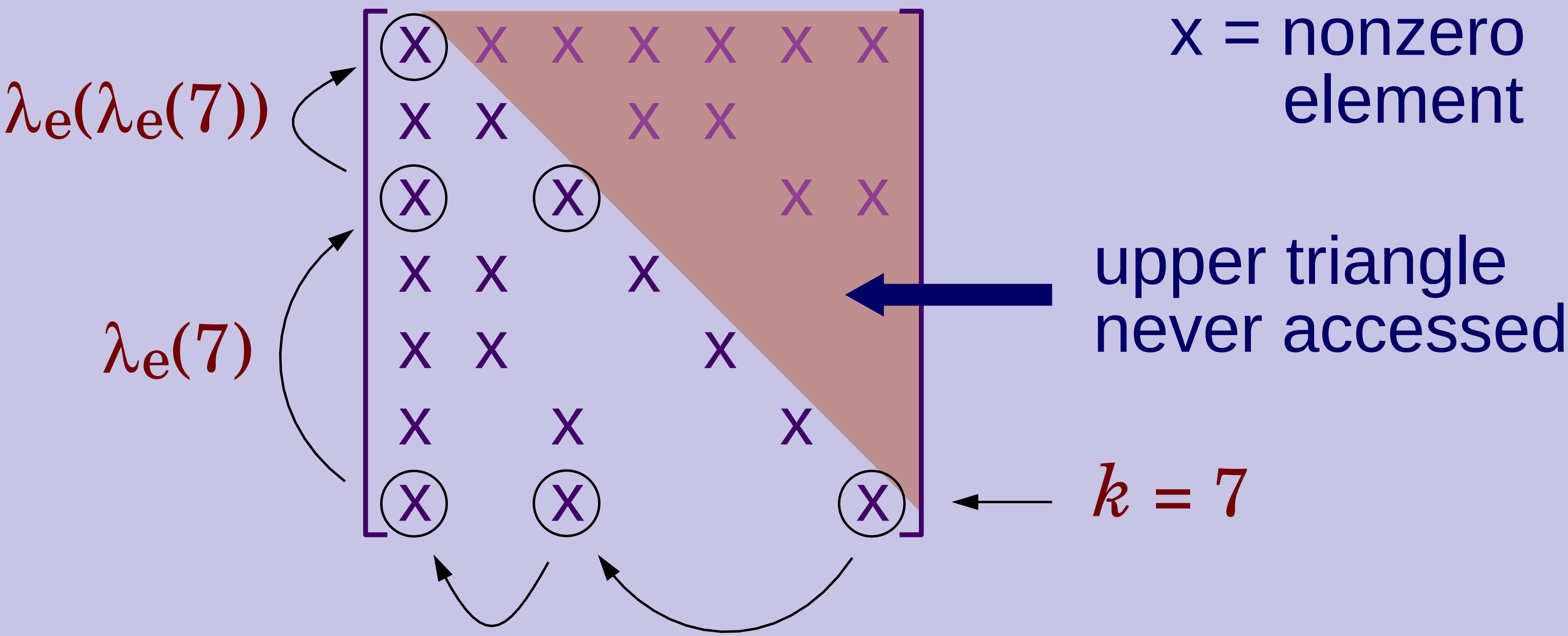
By iterating only over the ancestors of k, the algorithm performs the least possible amount of work, e.g. by updating only 5 elements at $k = 7$ instead of 27.

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.

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

$\boldsymbol{L}^{\mathrm{T}}\boldsymbol{DL}$ factorization	$D_1\mathrm{d} + D_2(\mathrm{m} + \mathrm{a})$
back−substitution	$n\mathrm{d} + 2D_1(\mathrm{m} + \mathrm{a})$

where

$$D_1 = \sum_{i=1}^{n} (d_i - 1) \qquad D_2 = \sum_{i=1}^{n} \frac{d_i(d_i-1)}{2}$$

and

$$d_i = 1 + d_{\lambda_e(i)} \qquad (d_0 = 1)$$

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

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

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

The complexity of factorization is therefore $\boxed{O(nd^2)}$

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 $\boldsymbol{C}$ $\qquad O(n)$
2. calculate $\boldsymbol{H}$ $\qquad O(n^2) \longrightarrow O(nd)$
3. solve $\boldsymbol{H}\ddot{\boldsymbol{q}} = \tau - \boldsymbol{C}$ $\qquad O(n^3) \longrightarrow O(nd^2)$

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

Calculating $\boldsymbol{H}$

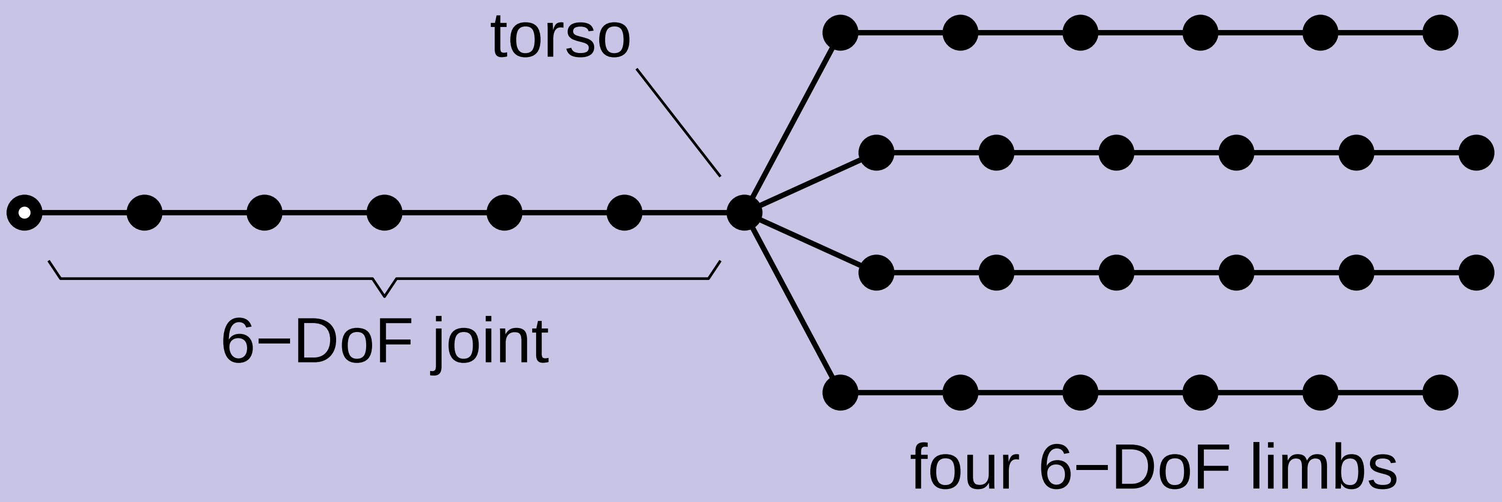
The *composite−rigid−body algorithm* (CRBA) is the best available algorithm for calculating $\boldsymbol{H}$.

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

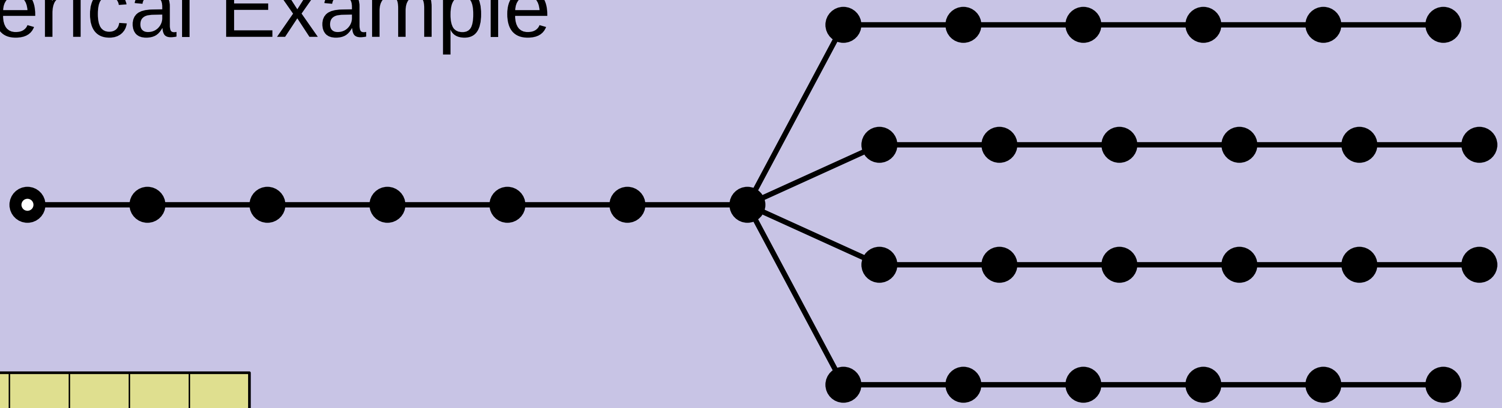
1. the CRBA implicitly exploits branch−induced sparsity by calculating only the nonzero elements of $\boldsymbol{H}$, and
2. there are only $n + 2D_1$ nonzero elements in $\boldsymbol{H}$, which is $O(nd)$.

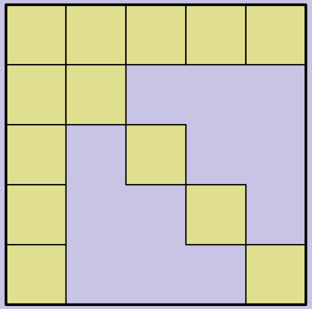
A Numerical Example

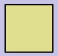
Let us compare the computational cost of forward dynamics for a 30−DoF unbranched chain and the 30−DoF humanoid (or quadruped) shown below.



A Numerical Example



$\boldsymbol{H}$ = 

each  is a 6x6 matrix

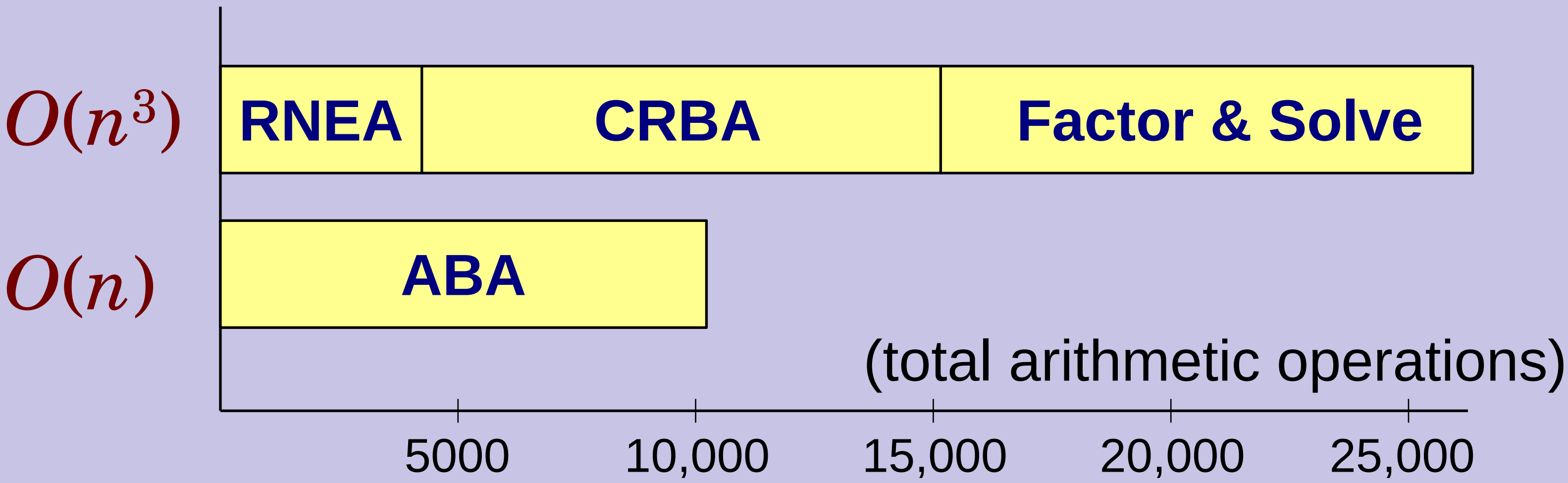
$\boldsymbol{H}$ contains:

468 nonzero elements

432 zero elements

$\boldsymbol{H}$ is therefore 48% zeros

Cost Figures for Unbranched Chain

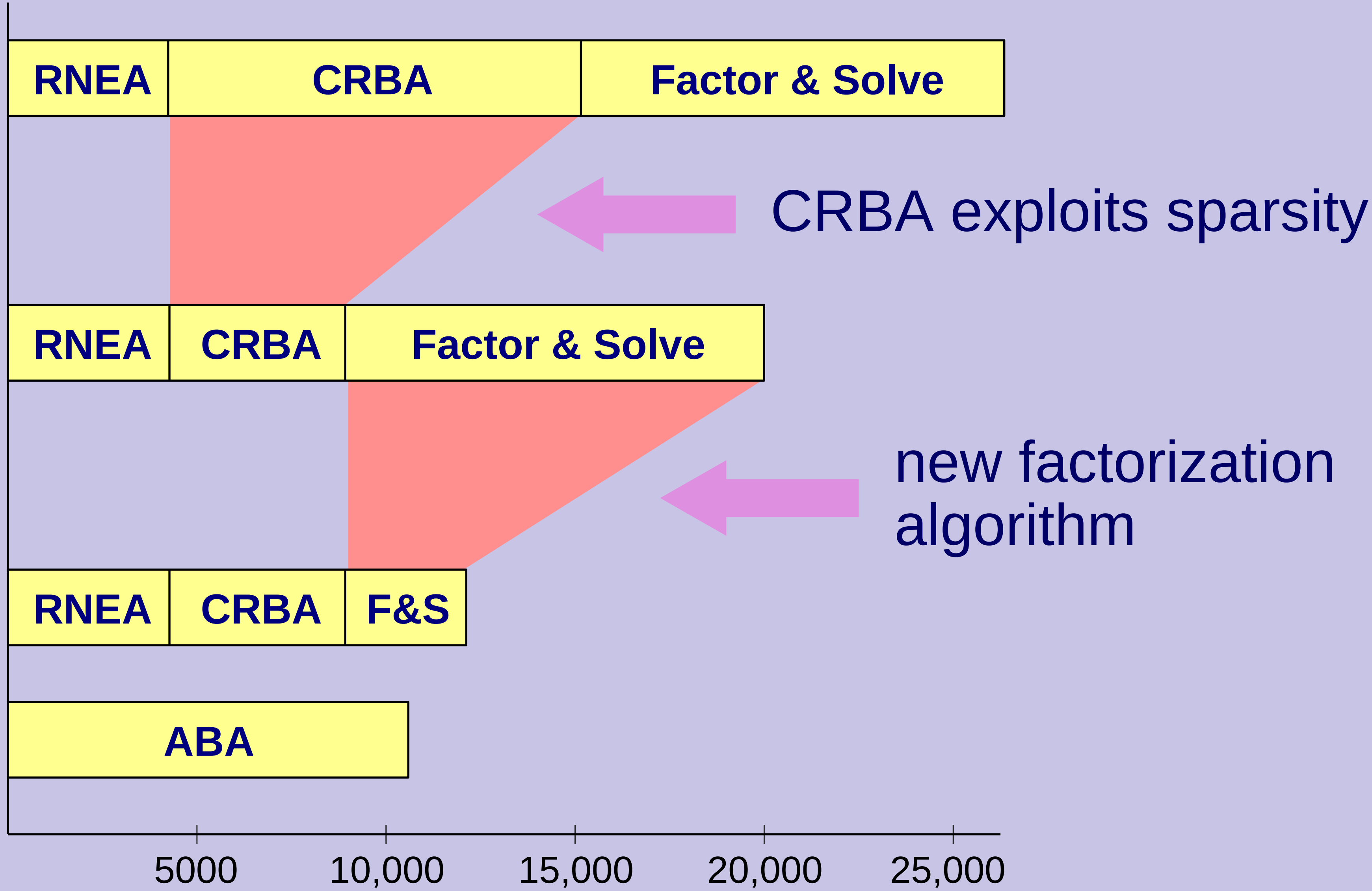


RNEA: Recursive Newton−Euler Algorithm

CRBA: Composite Rigid Body Algorithm

ABA: Articulated−Body Algorithm

Cost Figures for Humanoid/Quadruped



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

- R. Featherstone, 2005. Efficient Factorization of the Joint Space Inertia Matrix for Branched Kinematic Trees. *Int. J. Robotics Research*, 24(6):487−500.
- R. Featherstone, 2008. *Rigid Body Dynamics Algorithms*. New York: Springer.