Science & Technology Facilities Council
Rutherford Appleton Laboratory

Exploiting sparsity in the solution phase for large sparse equations

Iain S. Duff

Joint work with: Patrick Amestoy, Mila Slavova, and Bora Uçar

iain.duff@stfc.ac.uk

STFC Rutherford Appleton Laboratory

Oxfordshire, UK.

and

CERFACS, Toulouse, France

Outline

Task is to solve

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

where $\mathbf{A}$ is large and sparse and $\mathbf{b}$ is also sparse.

Outline

- Negative result
- Sparsity in right-hand side
- Entries of the inverse
- Sparse null space bases

Negative result

Duff, Erisman, Gear, and Reid (1988) showed that for any irreducible sparse matrix, its inverse is dense.

This effectively means that the solution to $Ax = b$ will be dense, however sparse b is.

So the question is: how can we exploit sparsity in the right-hand side?

Sparsity in right-hand side

One of the proofs for showing an inverse is dense is to look at the factorization

$$A = LU$$

and the subsequent solution of $Ax = b$ using these triangular factors by solving

$$Ly = b$$

followed by

$$Ux = y.$$

Assuming irreducibility, the solution to $Ly = b$ will have a nonzero entry in the last entry of y so that the solution of $Ux = y$ will be dense.

Thus the negative result depends on irreducibility and on the coupled solution using forward elimination and back substitution.

It is by avoiding these that we can exploit the right-hand side sparsity.

Reducible systems

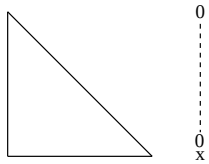
The prior theorem only holds for irreducible matrices, and Hall and McKinnon (2005) exploit this in the context of the simplex method in linear programming.

Linear programming matrices are often very reducible and they use this to obtain sparse solutions calling this hyper-sparsity.

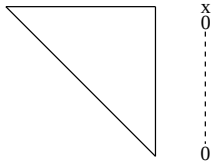
We will not discuss this application in the present talk but note that, even for an irreducible matrix, the factors are not normally dense and solving for just L or U can preserve sparsity in the right-hand sides.

Preservation of sparsity in solution

We note that the sparse right-hand side whose only entry is in the last position will yield a sparse solution of the same pattern for solution by any sparse lower triangular matrix.



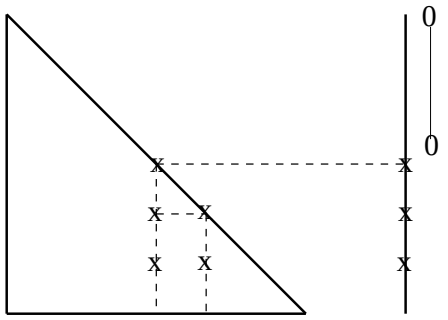
Similarly, a sparse right-hand side whose only entry is in the first position will yield a sparse solution of the same pattern for solution by any sparse upper triangular matrix.



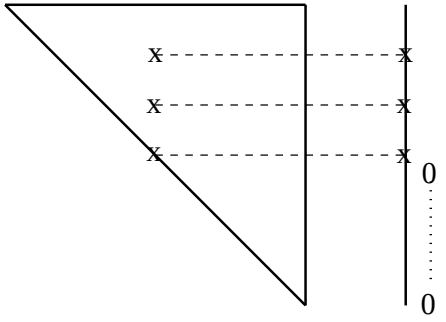
We now characterize the relationship between a sparse triangular factor and a right-hand side with a single entry knowing that the pattern for a more general sparse right-hand side can be obtained from superposition.

Preservation of sparsity in solution

For forward elimination, fill-in in the right-hand side only occurs after the first nonzero and then only within the pattern of the factor.



Similarly, for back substitution, fill-in in the right-hand side only occurs before the last nonzero and then only within the pattern of the factor.



Preservation of sparsity in solution

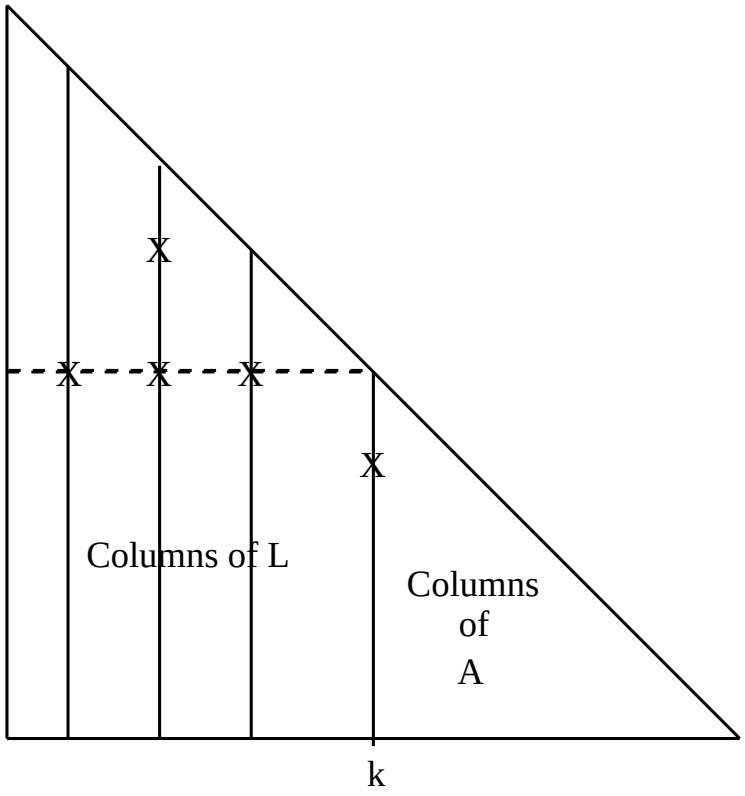
We will characterize the exploitation of right-hand side sparsity using the elimination tree which is fundamentally associated with the symbolic factorization of a sparse symmetric matrix.

It is a spanning tree for the graph of the matrix factors.

We will use this tree in defining the operations and data that we require for our sparse solutions.

Symbolic Factorization

$$A \rightarrow LL^T$$



Generate columns of L one at a time in sequence.

At stage k the structure of column k of L is the union of the structure of column k of A with previous columns of L whose FIRST nonzero is in row k.

Elimination Tree

Generation of elimination tree

Each node corresponds to a column of the matrix/factor. We generate the structure of L ($A = LDL^T$) at the same time as the elimination tree.

for k from 1 to n do .....

Generate structure of column k of L from symbolic sum of columns from 1 to $k-1$ which have their **first** nonzero entry in row k.

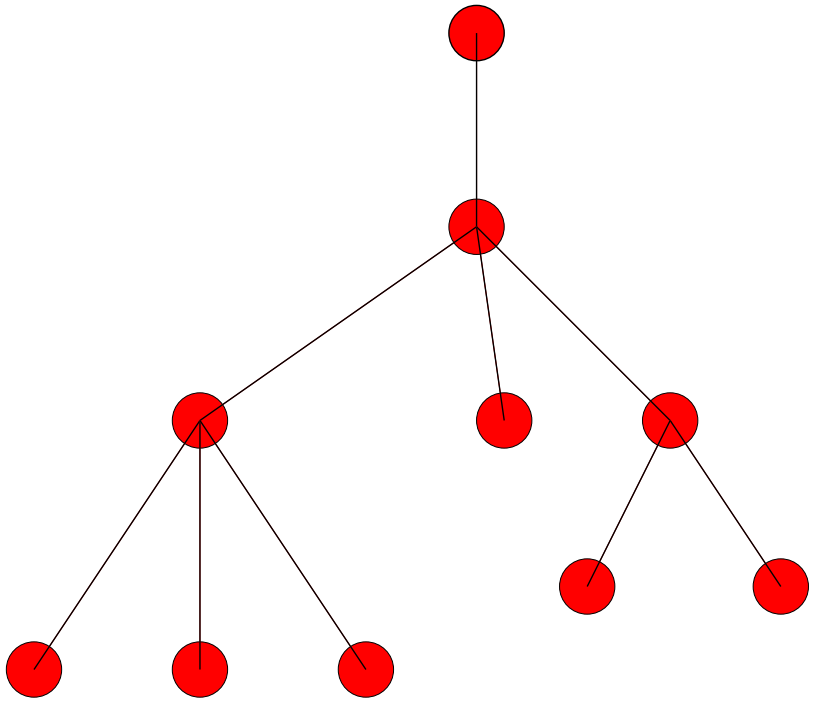
Draw edges in graph between node k and the nodes corresponding to columns just used above.

Note that if there are no entries before the diagonal in row k, k will be a leaf node.

Note also that the complexity of this algorithm is

$$\mathcal{O}(n) + \mathcal{O}(\tau)$$

Assembly Tree



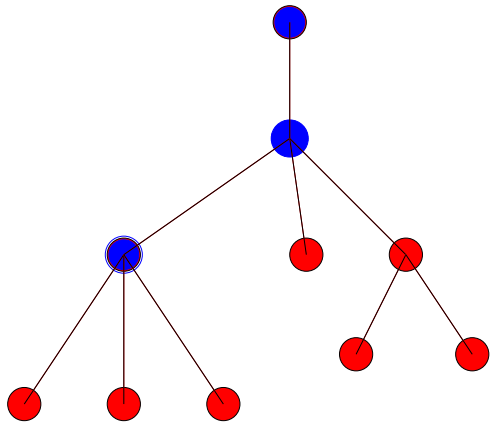
ASSEMBLY TREE

In practice, we use an assembly tree rather than the elimination tree. In the elimination tree there is only one elimination at each node, whereas there can be many in the assembly tree, corresponding to a block of pivots.

Single entry right-hand side

For forward elimination (that is solving with L) we traverse the tree from the node corresponding to the nonzero entry to the root. At other nodes, the solution is zero. We show this in the figure by colouring nodes on this path blue.

We note that this is all of the tree that need be used when computing the solution vector.

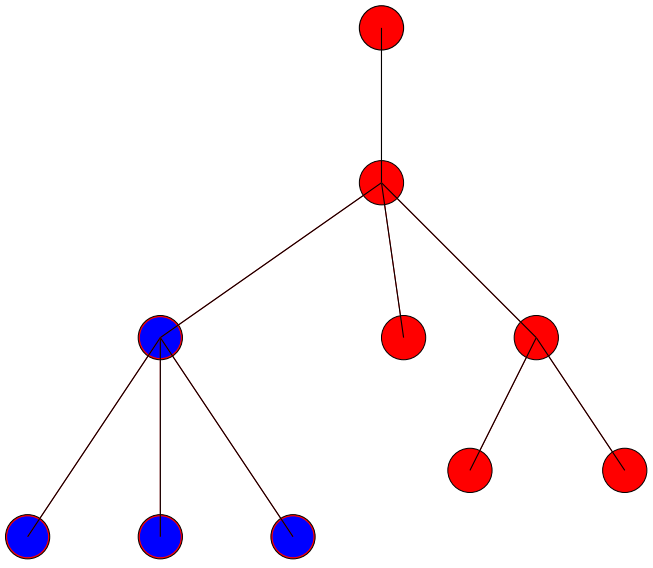


ASSEMBLY TREE

Single entry right-hand side

For back substitution (that is solving with U (L^T)) we traverse the tree from the root to the leaves.

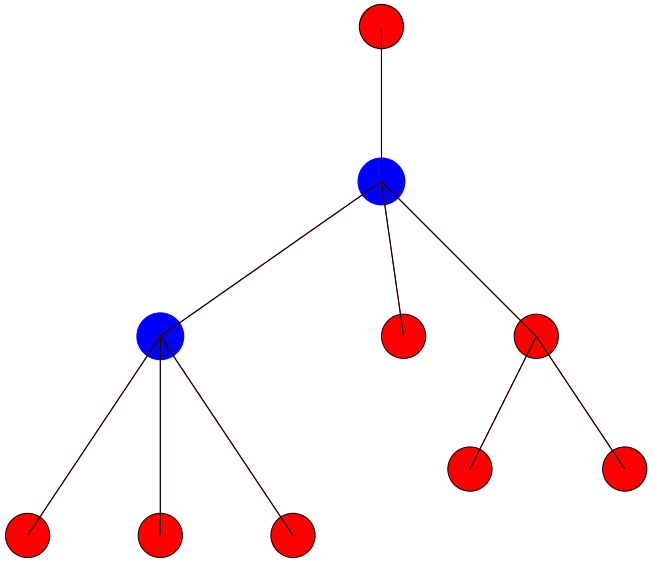
Again no operations will be performed until the node with the nonzero entry is encountered. At this point the subtree of which this is the root node will be all that is used to compute the sparse solution vector. We illustrate this for a subtree by colouring the nodes blue.



ASSEMBLY TREE

Specific component of solution

Additionally, if we are only seeking a specific entry of the solution, we can stop the substitution as soon as that component is computed. Thus we can reduce even more the part of the tree used for computing the sparse solution. We illustrate that part of the tree may be required by colouring the nodes blue.



ASSEMBLY TREE

Exploitation of sparse right-hand sides

We will examine two cases where these observations can be exploited

- The computation of entries of the inverse of the matrix

Exploitation of sparse right-hand sides

We will examine two cases where these observations can be exploited

- The computation of entries of the inverse of the matrix

See following talk from François-Henry Rouet

Exploitation of sparse right-hand sides

We will examine two cases where these observations can be exploited

- The computation of entries of the inverse of the matrix

See following talk from François-Henry Rouet
and

- The computation of null space bases

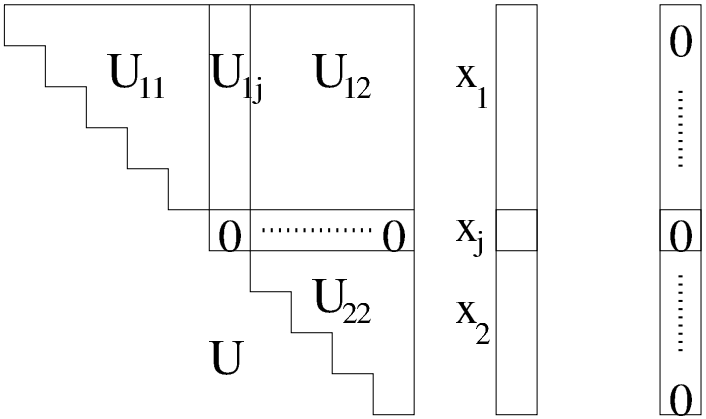
Sparse null space bases

We need to solve $Ax = LUx = O$

As L is unit triangular, we have $det(L) \neq 0$, and so

$$Ux = O$$

If null-pivots are detected (jth pivot) when factorizing the matrix, then



$$\begin{aligned} U_{11}x_1 + \quad U_{1j}x_j + \quad U_{12}x_2 \quad &= 0 \\ 0x_j + \quad 0x_2 \quad &= 0 \\ U_{22}x_2 \quad &= 0 \end{aligned}$$

We can set $x_j = 1$, so that we have to solve $U_{11}x_1 = -U_{1j}$, to yield $x = (x_1, 1, 0)$ as a null-space vector.

Sparse null space bases

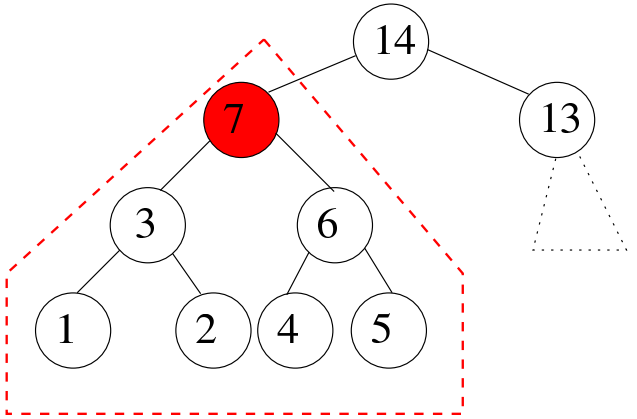
If the entry u_{jj} is set to 1 during the factorization, then solving the system

$$Ux = e_j$$

for each such j yields the same solution.

In the elimination tree this corresponds to visiting the nodes in the subtree rooted at node j.

No operation is performed for the nodes that are not in this subtree.



At each node, substitutions with the factors of the front are performed.

Results for null space basis

Matrix	Order	Entries	Defic	Factors to be loaded (Mbyte)		
				Min Size	no sp	sp
Box-cave_16x10x3	2675	15953	270	7	19	8
Box-cave_16x10x3	4419	26129	456	18	64	24
Box-cave_16x10x3	14454	89185	1653	170	1261	208
Box-cave_16x10x3	33627	212883	4056	803	9622	901

Results from thesis of Mila Slavova. Computation of null space.

Min size: A lower bound on the cost.

no sp: using solve

sp: using tree

The partitioning problem

What to partition: The nodes of the tree that correspond to the nodes containing the requested diagonal entries.

The cost: The total size of the factors loaded; the nodes that are visited in different epochs. We have to load the necessary factors at least once but want to reduce the number of times they are loaded.

Observation: Assume a node v is visited in λ epochs

We have to load the factors at node v at least once.

Therefore, $(\lambda - 1) \times w(v)$ is the extra cost is incurred, where $w(v)$ is the size of the factors of the node v.

Hypergraph partitioning

Hypergraph: $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of nodes $\mathcal{V}$ and a set of nets $\mathcal{N}$. Every net is a subset of nodes.

Cost $c(n_i)$ is associated with net n_i.

Vertex partition: $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$.

Connectivity: $\lambda(n_i)$ is the number of partitions in which n_i has nodes.

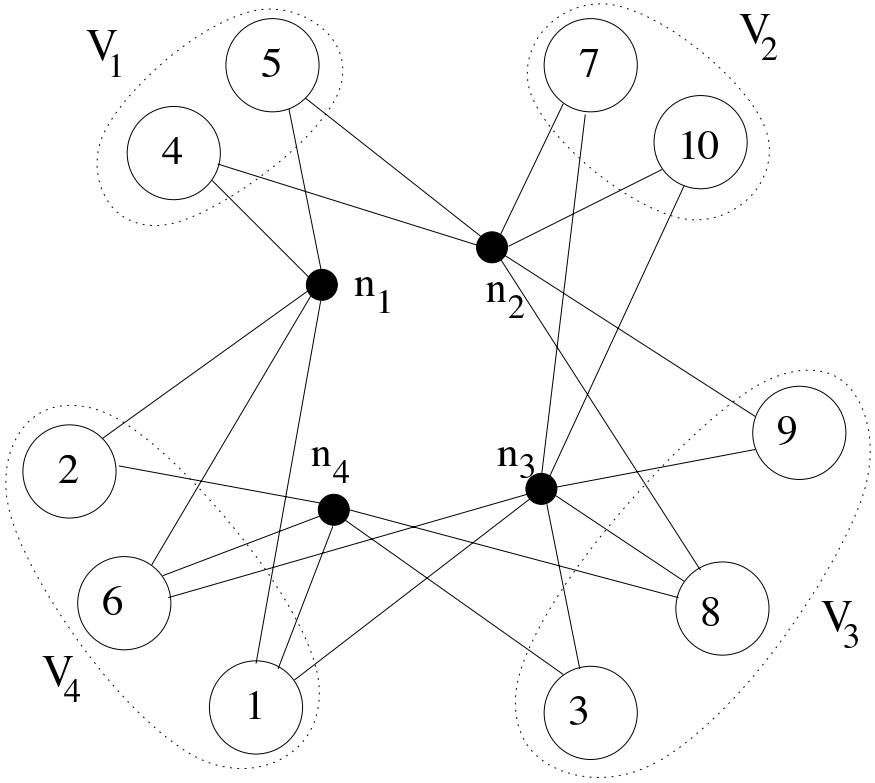
Objective: Minimize

$$cutsize(\Pi) = \sum_{n_i \in \mathcal{N}} (\lambda(n_i) - 1) c(n_i) \ .$$

Constraint: Satisfy a balance on the size of the partitions.

This problem is NP-hard.

Hypergraph partitioning



10 nodes, 4 nets.

Partitioned into 4 parts: $\{4, 5\}$, $\{7, 10\}$, $\{3, 8, 9\}$, and $\{1, 2, 6\}$,

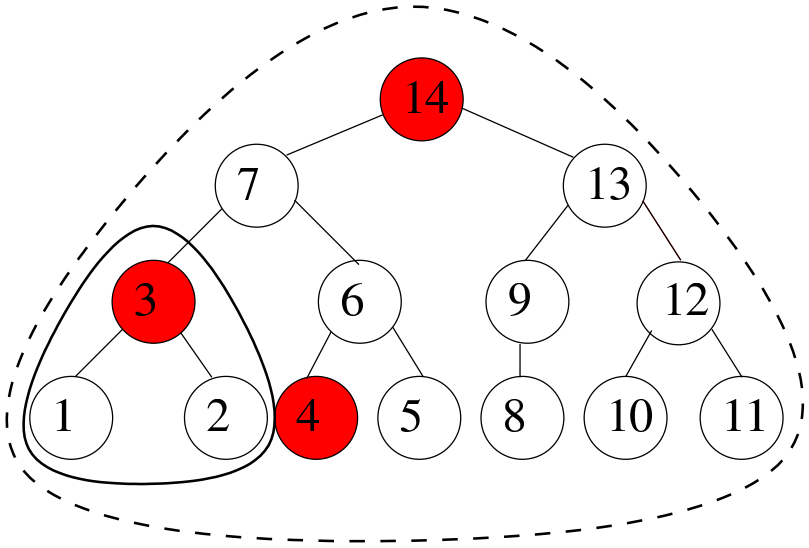
$$\lambda(n_1) = 2 \quad \lambda(n_2) = 3$$
$$\lambda(n_3) = 3 \quad \lambda(n_4) = 2$$

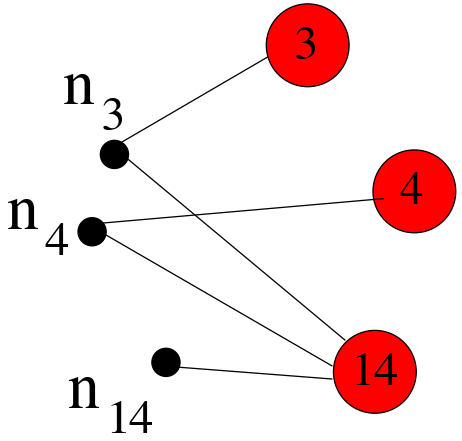
$$cutsize(\Pi) =$$
$$c(n_1) + 2c(n_2) + 2c(n_3) + c(n_4)$$

Hypergraph model for inverse entries

See following talk from François-Henry Rouet

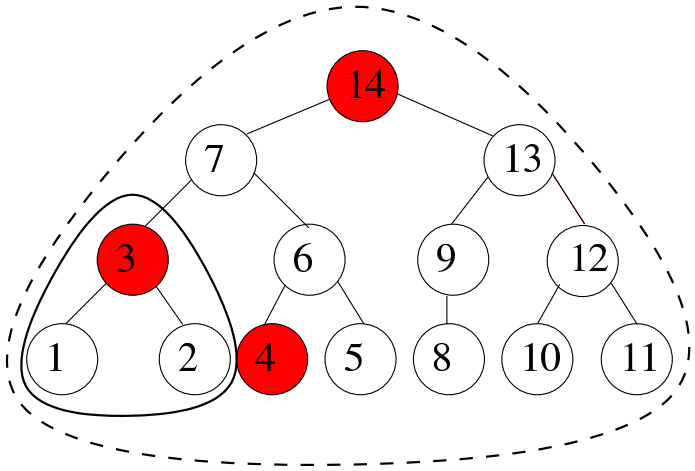
Model for the null space computations





- There is a node for each requested entry.
- There is a net for each node.
- A net is a superset of all the nets associated with nodes that are ancestors of the node with which it is associated (ie $n_j = n_j \cup n_i$, for each i that is an ancestor of j).
- The cost of a net is the sum of the sizes of the subtree of the factors rooted at the corresponding net minus the costs of the nets in the same subtree.
  We have $c(n_3)$=w(1) + w(2) + w(3), and $c(n_4)$=w(4), and hence
  $c(n_{14}) = \sum_{i \in T} w(i) - c(n_3) - c(n_4)$.

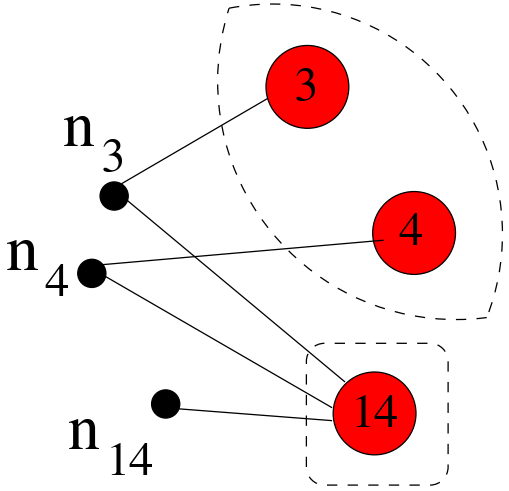
Model for the null space computations



First: solve $Ux = e_3$ and $Ux = e_4$

Second: solve for $Ux = e_{14}$

part	cost
1^{st}	$\sum_{i=1}^{4} w(i)$
2^{nd}	$\sum_{i=1}^{4} w(i) + \sum_{i=5}^{14} w(i)$



Nets 3 and 4 are cut. The cut-size is
$c(n_3) + c(n_4) = \sum_{i=1}^{4} w(i)$.

Results for null space basis

Matrix	Order	Entries	Defic	Factors to be loaded (Mbyte)				
				Min	no sp	Using sparsity		
						Po	Pr	HG
Box-cave_16x10x3	2675	15953	270	7	19	8	8	8
Box-cave_16x10x3	4419	26129	456	18	64	24	21	19
Box-cave_16x10x3	14454	89185	1653	170	1261	208	201	196
Box-cave_16x10x3	33627	212883	4056	803	9622	901	884	998

Results from thesis of Mila Slavova. Computation of null space.

Min size: A lower bound on the cost.

no sp: Ignore the sparsity.

Nat: Natural order.

Po/Pr: Postorder/preorder.

HG: Hypergraph partitioning using PaToH[†] with an allowable imbalance ratio of 0.01.

[† Çatalyürek and Aykanat,'99]

Conclusions

- We have shown that considerable gains can be obtained for the computation of sparse null space bases by only using the required part of the tree.
- Simple partitioning schemes can avoid significant reloading of matrix factors.