Direct Solvers for Sparse Matrices

Direct solvers for sparse matrices involve much more complicated algorithms than for dense matrices. The main complication is due to the need for efficiently handling the *fill-in* in the factors L and U. A typical sparse solver consists of four distinct steps as opposed to two in the dense case:

1. An ordering step that reorders the rows and columns such that the factors suffer little fill, or that the matrix has special structure, such as block-triangular form.
2. An analysis step or symbolic factorization that determines the nonzero structures of the factors and creates suitable data structures for the factors.
3. Numerical factorization that computes the L and U factors.
4. A solve step that performs forward and back substitution using the factors.

There is a vast variety of algorithms associated with each step. The review papers by Duff [137] (see also [135, Chapter 6]) and Heath, Ng, and Peyton [219] can serve as excellent references for various algorithms. Usually steps 1 and 2 involve only the graphs of the matrices, and hence only integer operations. Steps 3 and 4 involve floating-point operations. Step 3 is usually the most time-consuming part, whereas step 4 is about an order of magnitude faster. The algorithm used in step 1 is quite independent of that used in step 3. But the algorithm in step 2 is often closely related to that of step 3. In a solver for the simplest systems, i.e., symmetric and positive definite systems, the four steps can be well separated. For the most general unsymmetric systems, the solver may combine steps 2 and 3 (e.g. SuperLU) or even combine steps 1, 2 and 3 (e.g. UMFPACK) so that the numerical values also play a role in determining the elimination order.

In the past 10 years, many new algorithms and much new software have emerged which exploit new architectural features, such as memory hierarchy and parallelism. In Table 10.1, we compose a rather comprehensive list of sparse direct solvers. It is most convenient to organize the software in three categories: the software for serial machines, the software for SMPs, and the software for distributed memory parallel machines.

Table 10.1: Software to solve sparse linear systems using direct methods.

Code	Technique	Scope	Contact	
Serial platforms				
MA27	Multifrontal	Sym	HSL	[140]
MA41	Multifrontal	Sym-pat	HSL	[6]

MA42	Frontal	Unsym	HSL	[144]
MA47	Multifrontal	Sym	HSL	[141]
MA48	Right-looking	Unsym	HSL	[142]
SPARSE	Right-looking	Unsym	Kundert	[281]
SPARSPAK	Left-looking	SPD	George	[191]
SPOOLES	Left-looking	Sym and Sym-pat	Ashcraft	[21]
SuperLLT	Left-looking	SPD	Ng	[339]
SuperLU	Left-looking	Unsym	Li	[126]
UMFPACK	Multifrontal	Unsym	Davis	[103]
Shared memory parallel machines				
Cholesky	Left-looking	SPD	Rothberg	[405]
DMF	Multifrontal	Sym	Lucas	[308]
MA41	Multifrontal	Sym-pat	HSL	[7]
PanelLLT	Left-looking	SPD	Ng	[211]
PARASPAR	Right-looking	Unsym	Zlatev	[471]
PARDISO	Left-right looking	Sym-pat	Schenk	[395]
SPOOLES	Left-looking	Sym and Sym-pat	Ashcraft	[21]
SuperLU_MT	Left-looking	Unsym	Li	[127]
Distributed memory parallel machines				
CAPSS	Multifrontal	SPD	Raghavan	[220]
DMF	Multifrontal	Sym	Lucas	[308]
MUMPS	Multifrontal	Sym and Sym-pat	Amestoy	[8]
PaStiX	Left-right looking *	SPD	CEA	[224]
PSPASES	Multifrontal	SPD	Gupta	[210]
SPOOLES	Left-looking	Sym and Sym-pat	Ashcraft	[21]
SuperLU_DIST	Right-looking	Unsym	Li	[306]

S+	Right-looking †	Unsym	Yang	[182]

* In spite of the title of the paper
† Uses QR storage to statically accommodate any LU fill-in

Abbreviations used in the table: SPD = symmetric and positive definite; Sym = symmetric and may be indefinite; Sym-pat = symmetric nonzero pattern but unsymmetric values; Unsym = unsymmetric; HSL = Harwell Subroutine Library: `http://www.cse.clrc.ac.uk/Activity/HSL`

There is no single algorithm or software that is best for all types of linear systems. Some software is targeted for special matrices such as symmetric and positive definite, some is targeted for the most general cases. This is reflected in column 3 of the table, ``Scope.'' Even for the same scope, the software may decide to use a particular algorithm or implementation technique, which is better for certain applications but not for others. In column 2, ``Technique,'' we give a high-level algorithmic description. For a review of the distinctions between left-looking, right-looking, and multifrontal and their implications on performance, we refer the reader to the papers by Heath, Ng, and Peyton [219] and Rothberg [370]. Sometimes the best (or only) software is not in the public domain, but available commercially or in research prototypes. This is reflected in column 4, ``Contact,'' which could be the name of a company or the name of the author of the research code.

Another complication is due to the diverse applications ... Old code like SPARSE, distinctive advantage ...

In the context of shift-and-invert spectral transformation (SI) for eigensystem analysis, we need to factorize $A - \sigma I$, where A is fixed. Therefore, the nonzero structure of $A - \sigma I$ is fixed. Furthermore, for the same shift σ, it is common to solve many systems with the same matrix and different right-hand sides (in which case the solve cost can be comparable to factorization cost). It is reasonable to spend a little more time in steps 1 and 2 to speed up steps 3 and 4. That is, one can try different ordering schemes and estimate the costs of numerical factorization and solutions based on symbolic factorization, and use the best ordering. For instance, in computing the SVD, one has the choice between shift-and-invert on AA^*, A^*A, and $\begin{bmatrix} 0 & A \\ A^* & 0 \end{bmatrix}$, all of which can have rather different factorization costs, as discussed in Chapter 6.

Some solvers have the ordering schemes built in, but others do not. It is also possible that the built-in ordering schemes are not the best for the target applications. It is sometimes better to substitute an external ordering scheme for the built-in one. Many solvers provide well-defined interfaces so that the user can make this substitution easily. One should read the solver documentation to see how to do this, as well as to find out the recommended ordering methods.