1 Network representation and path generation

pathmatrix Generate the matrix corresponding to all paths between end nodes,

dijkstra Run Dijkstra's algorithm from a single source.

pathv Generate path vectors using the output from Dijkstra's algorithm.

Networks are represented by a pair of sparse adjacency matrices. The cost matrix C represents distances between nodes: C_{ij} is the cost to traverse an edge between node i and j, or is zero if there is no edge between the nodes. The label matrix has the same nonzero pattern, but L_{ij} represents the one-based index of the edge between i and j. Matlab sparse matrices are stored in compressed sparse column format: for each column in the matrix, there is a list of row numbers and values for the nonzeros in that column. From the perspective of performance and storage, then, Matlab's sparse matrix format is actually an adjacency list representation for the graph, not a dense adjacency matrix.

The path generation step is done using the usual Dijkstra's algorithm. The distances are kept in a binary heap, so that the run time for a single source is $O(V \log E)$. I maintain an auxiliary array so that I can go from the heap order to initial label order and back with equal ease; this simplifies the step in Dijkstra where you have to update distances at frontier nodes.

2 Row and column selection

colselect Eliminates unused edges, as well as any edges that appear on all the same paths. Eliminating edges that always appear together is the simplest form of "network virtualization." Preprocessing the path matrix with this routine will decrease the cost of the row selection, but strictly speaking it is not necessary.

rowselect Chooses a set of linearly dependent rows ($\bar{G}$) from the path matrix G. Also generates the Cholesky factor R of $\bar{G}\bar{G}^T$. Depending on the options chosen, R may be produced in single or double precision using a packed or unpacked layout.

resizeR Used by `rowselect` to expand the storage used for R.

addpaths Used by `rowselect` to scan through a piece of G and add any new (independent) rows to $\bar{G}$. Calls different auxiliary routines (`addpathsd` and `addpathsf`) to handle different layouts and precisions of R.

The column selection eliminates obviously redundant edges – edges that never appear, and edges that always appear together (corresponding to repeated columns in G). Eliminating these edges from G should not affect the later path loss rate computations, except to decrease their cost somewhat. Column selection is optional.

The row selection routine is a variant of the QR decomposition with column pivoting (the Gram-Schmidt algorithm). All of Q and most of R are discarded as soon as they are computed; they are not needed explicitly, and they would take a substantial amount of memory. Row selection using column-pivoted QR is described in Golub and Van Loan's book on Matrix Computations, among other places; the Gram-Schmidt procedure is described in any introductory linear algebra text.

The R factor produced by `rowselect` can be stored either in a full (column-major) format or a block packed format. For this paragraph, assume all indices are zero-based. In full format, R_{ij} is stored at position `i + j*ldR`, where `ldR` is the leading dimension of the storage allocated for R. In the packed format, R_{ij} is stored at `i + (2*j - J*kb)*(J+1)*kb/2`, where `kb` is the block size, and `J = floor(j/kb)`. If `kb` is 2, for example, the block packed layout for a 4-by-4 matrix is

$$\begin{bmatrix} 0 & 2 & 4 & 8 \\ 1 & 3 & 5 & 9 \\ & & 6 & 10 \\ & & 7 & 11 \end{bmatrix}$$

The R factor can also be computed using either single or double precision. Matlab works naturally with full storage double precision matrices, but packed single precision storage takes roughly a quarter of the space. Ordinary (unblocked) packed storage is also described in the LAPACK manual.

3 Loss rate vector calculation

compute_xG Compute the $x_{\bar{G}}$ vector which is the minimal norm solution to $\bar{G}x_{\bar{G}} = \bar{b}$.

trsolve Solve triangular systems using the R factor produced by `rowselect`. In the simplest form, `trsolve` is equivalent to applying the backslash operator in Matlab. However, the `trsolve` routine handles single precision and packed storage as well.

The loss rate calculation is basically

$$\begin{aligned} \bar{G}x_{\bar{G}} &= \bar{b} \\ x_{\bar{G}} &= \bar{G}^T y \end{aligned}$$

Using the factorization $\bar{G}\bar{G}^T = R^T R$, y can be computed using two triangular solves; to compute x_G then only requires a multiplication by $\bar{G}^T$.

In single precision, it is useful to do a single step of *iterative refinement*. Essentially, iterative refinement involves solving a system in finite arithmetic, then solving a system for the error. Iterative refinement is described in Golub and Van Loan's book, Demmel's book, and other books on numerical linear algebra.

Note that $x_{\bar{G}}$ will be different if different $\bar{G}$ matrices are chosen. Even if the $x_{\bar{G}}$ matrix is different, though, the loss rate computations will be the same, so long as the difference is in the null space of G.