RC 20453 (90427) May 6, 1996
Computer Science/Mathematics

IBM Research Report

WGPP: Watson Graph Partitioning
(and sparse matrix ordering)
Package

Users Manual: Version 1.2

Anshul Gupta

IBM Research Division
T.J. Watson Research Center
P. O. Box 218
Yorktown Heights, New York 10598

anshul@watson.ibm.com

IBM Research Division
Almaden · Austin · China · Haifa · T.J. Watson · Tokyo · Zurich

WGPP: Watson Graph Partitioning
(and sparse matrix ordering)
Package

Users Manual: Version 1.2*

Anshul Gupta

anshul@watson.ibm.com

IBM Research Report RC 20453

May 6, 1996

1 Introduction

Graph partitioning is an important problem with extensive application in scientific computing, optimization, VLSI design, task partitioning for parallel processing, etc. The Watson Graph Partitioning Package (*WGPP*) is a package of subroutines for partitioning graphs and computing fill-reducing orderings of sparse matrices for their direct factorization. The graph partitioning problem requires dividing the set of nodes of a weighted graph into k disjoint subsets or partitions such that the sum of weights of nodes in each subset is nearly the same (within a user supplied tolerance) and the total weight of all the edges connecting nodes in different partitions is minimized. In the case of unweighted graphs, each node and edge is assumed to have a unit weight. Computing a fill-reducing ordering of a sparse matrix seeks to generate a permutation of the columns (or rows) of a sparse matrix that minimizes the storage and computation required for factoring the sparse matrix. The *WGPP* library contains a collection of fast and high quality routines for unstructured graph partitioning and for generating robust fill-reducing orderings for sparse matrices arising in different application domains.

*Since this document describes software that might be upgraded from time to time, *WGPP* users and the readers of this report are encouraged to request the latest version of the manual via email from the author.

2 Performance, Quality, and Usability Comparisons

At the time of its release, *WGGP* is either faster or generates graph partitions with a smaller edgecut (for small number of partitions, it is both faster and better) than the other state-of-the art graph partitioning packages available, such as, *Chaco*[7] and *Metis*[10].

Currently, two of the best and most well known codes available for computing fill-reducing ordering for sparse matrices are that of Harwell *MC47B/BD* routine based on the approximate minimum degree (*AMD*) algorithm[4] (if both ordering time and quality are taken into account) and the one in *Metis* (if only ordering quality is considered). In many practical applications, sparse matrices with the same structure need to be factored more than once. In such situations, ordering quality (which determines the factorization time) becomes more important than ordering time. Our experiments suggest that for sparse matrices arising in two- and three-dimensional finite element problems, the *WMMRB* ordering routine of our package is typically 2 to 8 times slower than *AMD*, but better in quality on an average. However, for sparse matrices arising from highly irregular graphs, such as those in linear programming problems, *WMMRB* significantly outperforms *AMD*, both in terms of ordering time and quality. In [6], we present extensive experimental results that show that replacing the minimum degree based ordering in the IBM Optimization Subroutine Library (*OSL*) by *WMMRB* reduces the total time for solving a suite of 53 medium to large problems by a factor of 2.2. *WMMRB* is better than *Metis* in both ordering time and quality for most matrices. Recent reports by Ashcraft and Liu [2] and Hendrickson and Rothberg [8] suggest that their graph-partitioning based ordering algorithms may generate competitive orderings for certain types of finite-difference and finite-element matrices. The ordering software reported in [2] and [8] is not available in public domain for direct comparison with WGPP, but is reported to generate better orderings than Metis. To the best of our knowledge, *WMMRB* is one of the most general purpose routines for fast, robust, and high quality ordering of sparse matrices arising in different applications.

The memory requirement of *WGPP* and *WMMRB* routines is typically less than that of *Chaco*, equivalent to *Metis*, and more than that of *AMD*.

The *WGPP* routines provide some useful options and added functionality that is absent from other packages. Some examples are:

- options for preprocessing high-degree nodes and for compressing graphs with multiple degrees of freedom.
- ability to make multiple calls to generate different partitions/orderings without changing the input.
- ability to generate the better of minimum degree and graph partitioning based ordering in a single call.
- handling graphs (matrices) with or without self-edges (diagonal indices).
- limited access to internal data-structures through an advanced interface.
- user control over maximum tolerable imbalance among partitions.
- choice of static or dynamic memory allocation.

3 Availability

The WGPP library is currently available for the following platforms (it will soon be available for other systems):

- IBM RS6000 family (XLF 2.3.xx), request code WGPPX2.
- IBM RS6000 POWER (XLF 3.2.xx), request code WGPPP1.
- IBM RS6000 POWER 2 (XLF 3.2.xx), request code WGPPP2.
- IBM RS6000 POWER PC (XLF 3.2.xx), request code WGPPPC.
- SGI MIPS 1 and 2, IRIX 5.3 and higher, request code WGPPSG.
- SUN/Sparc, F77 2.0.1, Solaris 2.0 and higher, request code WGPPS2.
- SUN/Sparc, F77 3.0, Solaris 4.0 and higher, request code WGPPS3.

While better ways of distributing the software are being worked upon, for the time being, the library can be obtained by sending email to anshul@watson.ibm.com with the subject line containing the request code followed by one or more blanks followed by the email address at which the appropriate library and the manual will be emailed. The manual is sent in the postscript format and the library is emailed as a uuencoded file that should yield a binary file with a *.a* extension upon successful decoding. It may be required to run *ranlib* on the decoded file.

3.1 Use of WGPP in Conjunction with OSL

The users of the IBM Optimization Subroutine Library (*OSL*) can benefit from *WGPP* by simply linking this library before *OSL* while compiling their programs. For example,

```
xlf -O2 -o foo.out foo.f -lwgppp2 -losl
```

will automatically replace the sparse matrix ordering routines of *OSL* by the *WMMRB* routine (called automatically with appropriate arguments) described in Section 4.4.3.

4 *WGPP* Subroutines

This section addresses some issues in the use of *WGPP* subroutines and describes their calling sequences in detail. *The original code for these routines is in Fortran and expects the parameters to be passed by reference.*

4.1 Types of graphs/matrices accepted

WGPP routines work only with undirected graphs (hence, symmetric structure matrices). Therefore, if the adjacency list of node I contains J, then the adjacency list of node J must include I. The graphs can be unweighted, or can have weights assigned to vertices, or edges, or both.

WGPP subroutines prefer graphs without self-edges, but are capable of handling graphs with self-edges (i.e., adjacency list of node I contains I) with some restrictions. A *WGPP* routines checks the first node for a self-edge and turns on a flag that warns it of the presence of self-edges in the remainder of the graph. Even if all the other nodes do not have self-edges, correct results will be produced. However, if the first node does not have a self-edge, the *WGPP* routine will assume that self-edges are not present and will crash if it encounters one. To summarize, if a user wishes a *WGPP* subroutine to accept a graph with self-edges, then the first node must have a self-edge. The *WGPP* routines' capability of handling self-edges is particularly useful for ordering sparse matrices that are stored in compressed-sparse-row or compressed-sparse-column formats because the data-structures can be used unchanged without the user having to create the XADJ and ADJNCY arrays by deleting the diagonal entries of the matrix.

4.2 Indexing options

WGPP offers users access to three subroutines. Further variations of the algorithms implemented in *WGPP* are available to the user by setting different options in the *options* input parameter to these routines. *WGPP* supports both C-style indexing starting from 0 and Fortran-style indexing starting from 1. C-style indexing is preferred and is used in the internal routines. Fortran-style numbering forces a conversion in the beginning and a reconversion at the end.

4.3 Memory requirements

The user has the option of having the *WGPP* routines allocate memory dynamically by specifying NAUX $= 0$, or by supplying an integer array AUX of size NAUX > 0 and having the *WGPP* routines use AUX as the work space. The exact value of NAUX required is data dependent but in this section, we will give some rough estimates. In general, these might be overestimates for very regular graphs (nodes have nearly uniform degrees) and underestimates for highly irregular graphs (node degrees have large variations). *WGPP* with WTFLAG $= 0$ (unweighted graphs) would typically require about $20 \times N + 4 \times E + C$, where N and E are the number of vertices and edges in the graph, respectively, and C is a constant of about 10000. Also, the larger the number of partitions, the smaller is the memory requirement. For very small number of partitions (especially bisections) and if WTFLAG > 0, (weighted graphs) a NAUX of $20 \times N + 5 \times E + 20000$ is recommended. For *WMMRB* with not too irregular graphs, $30 \times N + 6 \times E + 30000$ should be sufficient. The memory requirement can be a lot less if compression is opted for (see OPTIONS parameter in Section 4.4.3) and is achieved. The memory requirements are higher if preprocessing of high degree nodes (see OPTIONS parameter in Section 4.4.3) is opted for.

In general, using NAUX $= 0$ is somewhat more robust (especially for ordering LP matrices) because memory requirement is problem dependent and may exceed the above estimates. However, dynamic memory allocation resulting from supplying NAUX $= 0$ may adversely affect the run time of WGPP subroutines on some machines. Also, see Section 5.

4.4 Calling sequences

The following subsection describe the calling sequences of the three subroutines, namely, *WGPP* (graph partitioning), *WGPPAD* (graph partitioning with an advanced interface), and *WMMRB* (sparse matrix ordering). The common parameters are described only once in Section 4.4.1. The original code for these routines is in Fortran and expects the parameters to be passed by reference. Therefore, if a *WGPP* subroutine is called from a C program, the user must explicitly pass the addresses of the data-structures described below.

4.4.1 *WGPP*: the graph partitioning routine

The calling sequence and description of the parameters of *WGPP* is as follows:

WGPP (N, XADJ, ADJNCY, VWGTS, EWGTS, WTFLAG, NPARTS, OPTIONS, NUMBERING, IECUT, PART, AUX, NAUX)

```
*******************************************************************
*                                                                 *
*  INPUT =                                                        *
*                                                                 *
*       SYMBOLIC NAME: N                                          *
*       POSITION:      1st    parameter.                          *
*       ATTRIBUTES:    INTEGER*4                                  *
*       DESCRIPTION:   Number of nodes in the graph.              *
*                                                                 *
*       SYMBOLIC NAME: NUMBERING                                  *
*       POSITION:      9th    parameter.                          *
*       ATTRIBUTES:    INTEGER*4                                  *
*       DESCRIPTION:   Numbering or indexing option.              *
*                      If NUMBERING = 1, then Fortran-style indexing  *
*                      is used; i.e., all array indices start from 1  *
*                      and the partitions of the graph are labeled    *
*                      from 1 to NPARTS.                          *
*                      If NUMBERING = 0, then C-style indexing is     *
*                      used; i.e., all array indices start from 0 and *
*                      the partitions of the graph are labeled from   *
*                      0 to NPARTS - 1.                           *
*                                                                 *
*       SYMBOLIC NAME: XADJ                                       *
*       POSITION:      2nd    parameter.                          *
*       ATTRIBUTES:    INTEGER*4 XADJ (NUMBERING : N + NUMBERING)     *
*       DESCRIPTION:   XADJ(I) points to the starting location in     *
*                      ADJNCY of the indices of the vertices adjacent *
*                      to vertex I in the graph.  The vertices        *
*                      adjacent to vertex I+1 must be stored          *
*                      immediately after the vertices adjacent to     *
*                      vertex I in ADJNCY.  Thus, XADJ(I+1) - XADJ(I) *
*                      is the degree of vertex I in the graph.        *
*                                                                 *
*       SYMBOLIC NAME: ADJNCY                                     *
```

```
*       POSITION:       3rd    parameter.                             *
*       ATTRIBUTES:     INTEGER*4 ADJNCY(NUMBERING:XADJ(N+NUMBERING)-1)*
*       DESCRIPTION:    The adjacency list of the graph.  The vertices *
*                       adjacent to vertex I+1 must be stored          *
*                       immediately after the vertices adjacent to     *
*                       vertex I.                                      *
*                                                                      *
*       SYMBOLIC NAME: VWGTS                                           *
*       POSITION:       4th    parameter.                             *
*       ATTRIBUTES:     INTEGER*4 VWGTS (NUMBERING : N+NUMBERING-1),   *
*                       if WTFLAG is 2 or 3.                           *
*       DESCRIPTION:    VWGTS(I) = weight of vertex I, if WTFLAG is 2  *
*                       or 3.  VWGTS is NULL otherwise.                *
*                                                                      *
*       SYMBOLIC NAME: EWGTS                                           *
*       POSITION:       5th    parameter.                             *
*       ATTRIBUTES:     INTEGER*4 EWGTS(NUMBERING:XADJ(N+NUMBERING)-1),*
*                       if WTFLAG is 1 or 3.                           *
*       DESCRIPTION:    EWGTS(J) = weight of edge corresponding to the *
*                       entry in location J of ADJNCY, if WTFLAG is 1  *
*                       or 3.  EWGTS is NULL otherwise.                *
*                                                                      *
*       SYMBOLIC NAME: WTFLAG                                          *
*       POSITION:       6th    parameter.                             *
*       ATTRIBUTES:     INTEGER*4                                      *
*       DESCRIPTION:    WTFLAG = 0 for unweighted graphs (both VWGTS   *
*                       and EWGTS are NULL and all nodes and edges are *
*                       assumed to have identical weights.             *
*                                                                      *
*                       WTFLAG = 1 if only edges have weights (VWGTS is*
*                       NULL and all nodes are assumed to have the same*
*                       weights.                                       *
*                                                                      *
*                       WTFLAG = 2 if only nodes have weights (EWGTS is*
*                       NULL and all edges are assumed to have the same*
*                       weights.                                       *
*                                                                      *
*                       WTFLAG = 3 if both nodes and edges have        *
*                       weights.                                       *
*                                                                      *
*       SYMBOLIC NAME: NPARTS                                          *
*       POSITION:       7th    parameter.                             *
*       ATTRIBUTES:     INTEGER*4                                      *
```

```
*       DESCRIPTION:   Number of parts into which the graph needs to  *
*                      be partitioned.                                *
*                                                                     *
*       SYMBOLIC NAME: OPTIONS                                        *
*       POSITION:      8th    parameter.                              *
*       ATTRIBUTES:    INTEGER*4 OPTIONS (NUMBERING : NUMBERING + 4)  *
*       DESCRIPTION:   OPTIONS(NUMBERING) = 0:  Use all default       *
*                      options.  The remaining entries in the options *
*                      array are ignored.                             *
*                      OPTIONS(NUMBERING) = 1:  Use the following     *
*                      options instead of the defaults.               *
*                                                                     *
*                      OPTIONS(NUMBERING+1) = the maximum percentage  *
*                      of tolerable load imbalance; default is 3%.    *
*                      Note that only whole number percentages for    *
*                      tolerable imbalance can be specified because   *
*                      OPTIONS is an integer array.
*                      Imbalance is computed as the ratio of the      *
*                      weight of the heaviest part to average part    *
*                      weight.  For example, consider a graph with 6  *
*                      vertices with weights 8, 3, 4, 5, 11, and 9.   *
*                      Suppose the graph is to be partioned into four *
*                      parts.  The average weight of a part is (8+3+4+*
*                      5+11+9)/4 = 10.  If the distribution of the    *
*                      weights in the 4 parts is (8, 3), (4, 5), (11),*
*                      and (9), then the load imbalance is 11/10,     *
*                      which is equal to 10%.  This example also shows*
*                      it is not always possible to limit the         *
*                      imbalance to the user defined tolerance; e.g., *
*                      in the above example, 10% is the best possible.*
*                                                                     *
*                      OPTIONS(NUMBERING+2) = type of matching to be  *
*                      used in coarsening.  A value of 2 instructs    *
*                      WGPP to perform heavy edge matching (HEM), and *
*                      value of 4 for this options results in heavy   *
*                      triangle matching (HTM).  HTM is faster than   *
*                      HEM, but results in somewhat worse quality     *
*                      partitioning.                                  *
*                                                                     *
*                      OPTIONS(NUMBERING+3) = random number seed.     *
*                      One can use different values of the seed to    *
*                      force WGPP to generate a different initial     *
*                      permutation of the graph.  This is useful if   *
```

```
*                    one needs to generate a few different          *
*                    partitionings of the same graph (perhaps to    *
*                    chose the best) without having to change the   *
*                    input.                                         *
*                                                                   *
*                    OPTIONS(NUMBERING+4) = type of refinement.     *
*                    A value of 11 results in greedy refinement (GR)*
*                    and a value of 12 results in Kernighan-Lin (KL)*
*                    refinement.  KL is significantly slower than   *
*                    GR, especially moderate to large values of     *
*                    NPARTS.  Fortunately, the benefit of using KL  *
*                    over GR is the maximum for small number of     *
*                    parts, where the run-time penalty is not       *
*                    excessive.  For large number of parts, GR does *
*                    as well as KL and is much faster.  For small   *
*                    number of parts relative to the size of the    *
*                    graph, KL is recommended.                      *
*                                                                   *
*      SYMBOLIC NAME: NUMBERING                                     *
*      POSITION:      9th    parameter.                             *
*      ATTRIBUTES:    INTEGER*4                                     *
*      DESCRIPTION:   See after the description of N.               *
*                                                                   *
*      SYMBOLIC NAME: NAUX                                          *
*      POSITION:      13th   parameter.                             *
*      ATTRIBUTES:    INTEGER*4                                     *
*      DESCRIPTION:   NAUX = 0:  AUX is NULL and WGPP should allocate*
*                    the memory for its internal use dynamically.   *
*                    NAUX > 0:  NAUX is the size of work space AUX. *
*                    If NAUX is insufficient or less than 0, an     *
*                    error message is generated.                    *
*                                                                   *
*                                                                   *
*  OUTPUT =                                                         *
*                                                                   *
*      SYMBOLIC NAME: IECUT                                         *
*      POSITION:      10th   parameter.                             *
*      ATTRIBUTES:    INTEGER*4                                     *
*      DESCRIPTION:   The total weight (number, if WTFLAG is 0 or 2) *
*                    of the edges that connect nodes belonging to   *
*                    different partitions.                          *
*                                                                   *
*      SYMBOLIC NAME: PART                                          *
```

```
*       POSITION:      11th   parameter.                            *
*       ATTRIBUTES:    INTEGER*4 PART (NUMBERING : N + NUMBERING - 1) *
*       DESCRIPTION:   PART(I) contains an integer value between     *
*                      NUMBERING and NUMBERING + NPARTS - 1 that is   *
*                      label of the partition to which node I belongs.*
*                                                                    *
*                                                                    *
*  WORK SPACE =                                                      *
*                                                                    *
*       SYMBOLIC NAME: AUX                                           *
*       POSITION:      12th   parameter.                            *
*       ATTRIBUTES:    INTEGER*4 AUX (NUMBERING : NAUX+NUMBERING-1),  *
*                      if NAUX > 0.                                  *
*       DESCRIPTION:   Temporary work space.  Used only if NAUX > 0.  *
*                                                                    *
**********************************************************************
```

4.4.2 *WGPPAD*: graph partitioning routine with advanced interface

WGPPAD is identical to WGPP, except that it provides some additional information about the partitioning with the aid three extra output parameters. In *WGPPAD*, AUX and NAUX become the 15th and 16th parameters respectively and are preceded by the three output parameters described after the calling sequence. The other parameters and their attributes and descriptions are the same as in *WGPP*.

WGPPAD (N, XADJ, ADJNCY, VWGTS, EWGTS, WTFLAG, NPARTS, OPTIONS, NUMBERING, IECUT, PART, PWGTS, INDEG, ITER, AUX, NAUX)

```
**********************************************************************
*                                                                    *
*  OUTPUT =                                                          *
*                                                                    *
*       SYMBOLIC NAME: PWGTS                                         *
*       POSITION:      12th   parameter.                            *
*       ATTRIBUTES:    INTEGER*4 PWGTS(NUMBERING : NPARTS+NUMBERING-1)*
*       DESCRIPTION:   Upon completion of execution, PWGTS(J) contains*
*                      the total weight (or number if WTFLAG is 0     *
*                      or 1) of nodes that belong to the partition    *
*                      labeled J, NUMBERING <= J < NPARTS + NUMBERING *
*                                                                    *
*       SYMBOLIC NAME: INDEG                                         *
*       POSITION:      13th   parameter.                            *
*       ATTRIBUTES:    INTEGER*4 INDEG( NUMBERING : NUMBERING + 1,    *
*                                NUMBERING : N + NUMBERING + 1)*
*       DESCRIPTION:   This two dimensional array contains the       *
```

```
*                      internal degree of each vertex in the         *
*                      partitioned graph.  The value INDEG ( ITER +   *
*                      NUMBERING, I) is the sum of weights (or the    *
*                      number, if WTFLAG is 0 or 2) of all the edges  *
*                      that connect node I with other nodes in the    *
*                      same partition.  INDEG can also be used to     *
*                      determine if a particular node is internal to  *
*                      a partition (i.e., all its neighbors belong to *
*                      the same partition) or if it is connected to   *
*                      nodes in other than its own partition.  The    *
*                      former is true iff the internal degree of a    *
*                      is equal to the total degree of that node,     *
*                      which is the sum of weights (or the number, if *
*                      WTFLAG is 0 or 2) of all the edges incident on *
*                      that node.                                     *
*                                                                     *
*       SYMBOLIC NAME: ITER                                           *
*       POSITION:      14th   parameter.                              *
*       ATTRIBUTES:    INTEGER*4                                      *
*       DESCRIPTION:   ITER always returns a 0 or a 1.  It helps      *
*                      determine the internal degree of a node using  *
*                      the output array INDEG (see description of     *
*                      INDEG above).                                  *
***********************************************************************
```

4.4.3 *WMMRB*: sparse matrix ordering routine

The calling sequence and description of the parameters of *WMMRB* is given below. The parameters N, XADJ, ADJNCY, NUMBERING, AUX, and NAUX have the same attributes and descriptions as in *WGPP*, and are omitted from the following description.

WMMRB (N, XADJ, ADJNCY, OPTIONS, NUMBERING, PERM, INVP, AUX, NAUX)

```
***********************************************************************
*                                                                     *
*  INPUT =                                                            *
*                                                                     *
*       SYMBOLIC NAME: OPTIONS                                        *
*       POSITION:      4th    parameter.                              *
*       ATTRIBUTES:    INTEGER*4 OPTIONS (NUMBERING : NUMBERING + 4)  *
*       DESCRIPTION:   OPTIONS(NUMBERING) = 0:  Use all default       *
*                      options.  The remaining entries in the options *
*                      array are ignored.                             *
*                      OPTIONS(NUMBERING) = 1:  Use the following     *
*                      options instead of the defaults.               *
```

```
*                                                                      *
*                      OPTIONS(NUMBERING+1) = the maximum number of    *
*                      nodes that a subgraph must have before it is    *
*                      ordered by using a minimum degree based         *
*                      algorithm without further subpartition.  The    *
*                      user can obtain a pure minimum degree ordering *
*                      by specifying OPTIONS(NUMBERING+1) greater than*
*                      N.  A value of 0 in this field lets WMMRB chose*
*                      its own default.  Typically, it is best to use *
*                      the default, but advanced users may experiment *
*                      with this parameter to find out what best suits*
*                      their application.  Sometimes a value larger   *
*                      than the default, which is between 50 and 200, *
*                      may result in a faster ordering without a big  *
*                      compromise in quality.  The default value for  *
*                      this element of OPTIONS is 0.                  *
*                                                                      *
*                      OPTIONS(NUMBERING+2) = 0: Default; has no      *
*                      effect.                                        *
*                      OPTIONS(NUMBERING+2) = 1: This forces WMMRB to *
*                      to compute a minimum degree based ordering     *
*                      in addition to the ordering based on recursive *
*                      graph bisection.  It then computes the amount  *
*                      of fill-in that each ordering would generate   *
*                      in Cholesky factorization and returns the      *
*                      permutation corresponding to the better        *
*                      ordering.  The use of this option increases the*
*                      ordering time (in most cases the increase is   *
*                      not significant), but is very useful when one  *
*                      ordering is used for multiple factorizations.  *
*                                                                      *
*                      OPTIONS(NUMBERING+3) = random number seed.     *
*                      One can use different values of the seed to    *
*                      force WGPP to generate a different initial     *
*                      permutation of the graph.  This is useful if   *
*                      one needs to generate a few different          *
*                      orderings of the same sparse matrix (perhaps to*
*                      chose the best) without having to change the   *
*                      input.                                         *
*                                                                      *
*                      OPTIONS(NUMBERING+4) = type of sparse matrix.  *
*                      This option can be used to aid WMMRB chose     *
*                      appropriate values of some internal parameters *
```

```
*                      and to chose appropriate algorithms in various *
*                      stages of ordering.  If the user has no        *
*                      information about the type of sparse matrix or *
*                      if the matrix does not fall into one of the    *
*                      following categories, then the default value 0 *
*                      should be used.                                *
*                      Certain sparse matrices have a few rows/columns*
*                      that are much denser than most of the rows/col-*
*                      umns.  Most sparse matrices used in interior-  *
*                      point problems in linear programming fall in   *
*                      this category.  For such matrices, the integer *
*                      value supplied in OPTIONS(NUMBERING+4) should  *
*                      be odd (e.g., 1 or 3).  This instructs WMMRB to*
*                      split the graph based on the high degree nodes *
*                      before proceeding with the ordering.           *
*                      Sometimes, sparse matrices arise from finite-  *
*                      element graphs in which many or most vertices  *
*                      have more than one degree of freedom.  In such *
*                      graphs, there are a many small groups of nodes *
*                      that share the same adjacency structure.  If   *
*                      the sparse matrix comes from a problem like    *
*                      this, then a value of 2 or 3 should be used in *
*                      OPTIONS(NUMBERING+4).  This instructs WMMRB to *
*                      construct a compressed graph before proceeding *
*                      with the ordering.                             *
*                                                                     *
*                      To summarize, OPTIONS(NUMBERING+4) should      *
*                      contain 0 for default, 1 for a preprocessing   *
*                      based on high degree nodes, 2 for preprocessing*
*                      the graph to compress it, and 3 if both of the *
*                      above preprocessing steps are desired.         *
*                      Preprocessing the high degree nodes may result *
*                      in a significant improvement in the quality of *
*                      ordering (if the matrix is suitable for this), *
*                      and preprocessing for compression may result in*
*                      a much faster ordering if there are significant*
*                      (enough to compensate for the time spent in    *
*                      performing compression) number of nodes with   *
*                      multiple degrees of freedom.                   *
*                                                                     *
*                      An Additional Function of OPTIONS(NUMBERING+4) *
*                      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ *
*                      The user may add 4 (i.e., set the 3rd least    *
```

```
*                      significant bit) in OPTIONS(NUMBERING+4) to    *
*                      generate a permutation that would generate     *
*                      an identical amount of fill-in during          *
*                      factorization, but would tend to force the most*
*                      dense columns of the factor towards the lower  *
*                      right hand side corner.  Many sparse           *
*                      factorization codes switch to a dense matrix   *
*                      algorithm  towards the end of factorization if *
*                      they detect that the lower right corner of the *
*                      factor is nearly dense.  Using a value of 4, 5,*
*                      or 7 instead of 0, 1, or 3, respectively, in   *
*                      OPTIONS(NUMBERING+4) may improve the           *
*                      performance of the numerical factorization code*
*                      by providing it a larger nearly dense block at *
*                      the end.  The total amount of fill-in or the   *
*                      operation count remains unchanged.             *
*                                                                     *
*                      Summary and Recommended Options                *
*                      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~                *
*                      The recommended contents of OPTIONS are        *
*                      (1,0,1,0,5) for interior-point algorithms,     *
*                      (1,0,1,0,2) for finite-element problems that   *
*                      can benefit from compression; i.e., there is   *
*                      reasonable fraction of vertices with multiple  *
*                      degrees of freedom, and (1,0,1,0,0) for other  *
*                      matrices.  If the user does not know the source*
*                      of the matrix, then (1,0,1,0,7) will attempt   *
*                      all the optimizations (at the cost of extra run*
*                      time).  If ordering time is important (i.e.,   *
*                      there is only one factorization per ordering,  *
*                      then OPTIONS(NUMBERING+2) should be set to 0.  *
*                      For even faster ordering of finite-element or  *
*                      finite difference matrices only, a large       *
*                      integer (between N/j and N+1, where j is small)*
*                      may be used if ordering time is more important *
*                      than ordering quality.                         *
*                                                                     *
*                                                                     *
*  OUTPUT =                                                           *
*                                                                     *
*       SYMBOLIC NAME: PERM                                           *
*       POSITION:      6th    parameter.                              *
*       ATTRIBUTES:    INTEGER*4 PERM (NUMBERING : N + NUMBERING - 1) *
```

```
*       DESCRIPTION:   The permutation vector (as defined in        *
*                      Sparspak).  If J = PERM(I), then J should be  *
*                      the I-th column in the PAP^T matrix to be     *
*                      factored.                                     *
*                                                                    *
*       SYMBOLIC NAME: INVP                                          *
*       POSITION:      7th    parameter.                             *
*       ATTRIBUTES:    INTEGER*4 INVP (NUMBERING : N + NUMBERING - 1) *
*       DESCRIPTION:   The inverse permutation vector (as defined in *
*                      Sparspak).  If J = INVP(I), then I should be  *
*                      the J-th column in the PAP^T matrix to be     *
*                      factored.                                     *
*                                                                    *
**********************************************************************
```

5 Bugs and Problems

WGPP performs minimal error-checking and it is the user's responsibility to call WGPP subroutines with correct arguments, especially, valid symmetric graphs/matrices.

Version 1 of *WGPP* is a beta release and is not guaranteed to be bug free. Any bugs or other problems with the software or this manual may be reported via email to *anshul@watson.ibm.com*. If NAUX is 0 and memory problems are encountered, one may attempt to increase the size of the data partition and/or the heap (e.g., by using the -bmaxdata option during compilation on the IBM RS6000 family of workstations). In such cases it may help to increase the stack size (e.g., by using the -bmaxstack option during compilation on the IBM RS6000 family of workstations). On very rare occasions, a routine might crash due to insufficient memory without generating the appropriate error message. This problem can be overcome by increasing NAUX. Also, on some platforms, WGPP occasionally fails to return the original data structures intact on graphs with self edges (diagonal entries).

6 Acknowledgments

Details of the algorithms implemented in *WGPP* can be found in [5, 6]. Many algorithms implemented in *WGPP* are adapted from or improved upon the ones discussed in [9, 11, 1, 3]. *WMMRB* uses the *AMD* code written by Amestoy, Davis, and Duff [4] to label small subgraphs containing about 100 nodes.

7 Notice: Terms and Conditions for Use of *WGPP*

The *WGPP* library may be used solely for educational, research, and benchmarking purposes by non-profit organizations and the U.S. government. Commercial and other organizations may make use of the *WGPP* library solely for benchmarking purposes only. Any other use of the libraries requires prior written permission. As unestablished research software, the *WGPP* library is provided free of charge on an "as is" basis without warranty of any kind, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose. IBM does not warrant that the functions contained in this library will meet the user's requirements or that the operation of its routines will be uninterrupted or error-free. Acceptance and use of this program constitutes the user's understanding that he/she will have no recourse to IBM for any actual or consequential damages, including, but not limited to, lost profits or savings, arising out of the use or inability to use this library. Even if the user informs IBM of the possibility of such damages, IBM expects the user to accept the risk of any such harm, or the user shall not attempt to use this library for any purpose. The library may not be sold. One may make copies of the library, provided that the copies are not distributed and are used under the same conditions. The use of the *WGPP* library is an implicit agreement by the user to these conditions.

References

[1] Cleve Ashcraft. Compressed graphs and the minimum degree algorithm. *SIAM Journal of Scientific Computing*, 16:1404–1411, 1995.

[2] Cleve Ashcraft and Joseph W.-H. Liu. Robust ordering of sparse matrices using multisection. Technical Report CS 96-01, Department of Computer Science, York University, Ontario, Canada, 1996.

[3] Cleve Ashcraft and Joseph W.-H. Liu. Generalized nested dissection: Some recent progress. In *Proceedings of Fifth SIAM Conference on Applied Linear Algebra*, Snowbird, Utah, June 1994.

[4] Timothy A. Davis, Patrick Amestoy, and Iain S. Duff. An approximate minimum degree ordering algorithm. Technical Report TR-94-039, Computer and Information Sciences Department, University of Florida, Gainesville, FL, 1994.

[5] Anshul Gupta. Fast and effective algorithms for graph partitioning and sparse matrix reordering. Technical Report (Number to be assigned), IBM T. J. Watson Research Center, Yorktown Heights, NY, 1996.

[6] Anshul Gupta. Graph partitioning based sparse matrix ordering algorithms for interior-point methods. Technical Report RC 20467 (90480), IBM T. J. Watson Research Center, Yorktown Heights, NY, May 21, 1996. Also available at *ftp://ftp.cs.umn.edu/users/kumar/anshul/IP-ordering.ps.Z.*

[7] Bruce Hendrickson and Robert Leland. Chaco user's guide. Technical Report SAND93-2339, Sandia National Laboratories, Albuquerque, NM, October 1993.

[8] Bruce Hendrickson and Edward Rothberg. Improving the runtime and quality of nested dissection ordering. Technical Report SGI manuscript, Silicon Graphics, Inc., Mountain View, CA, 1996.

[9] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995.

[10] George Karypis and Vipin Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota, 1995.

[11] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report TR 95-064, Department of Computer Science, University of Minnesota, 1995.