

STOREQ Users Manual

Storage Requirement Estimation and Optimization

Version 0.1, December 14, 2000

Author: Per Gunnar Kjeldsberg

1. Introduction

The STOREQ program performs storage requirement estimation and optimization given a polyhedral description of a data intensive application and a possibly partially fixed execution ordering. This manual describes the format of its input and output, as well as the interactive steps performed by the designer.

STOREQ is developed using MatlabTM version 5.3.1 (R11.1), and requires Matlab to run. It is not necessary to have any additional Matlab toolboxes installed.

See [Kjeldsberg00] for more detailed information of STOREQ's theoretical background.

2. Input Format

The main input to STOREQ is two matrixes. One defines the polytopes in iteration space in which the statement of the Application Under Development (AUD) produce array elements. This matrix is denoted the Set Of Statement Polytopes (SetOfSP). The second matrix defines the dependencies between these polytopes and is denoted the Set Of Dependencies (SetOfDep). In addition, two more input matrixes is automatically generated by the program, which can afterwards be changed by the user before parts of STOREQ is rerun. These matrixes are the Fixed Dimensions (FD) matrix and the Ignore Dependency (IgnoreDep) matrix.

The inputs are defined so as to be compatible with future output information from the Atomium and Masai-3 tools currently being developed at IMEC.

COPYRIGHT

© Copyright 2000: Author and IMEC, Leuven, Belgium

Permission to use, copy and modify this software and the supporting documentation without any fee, is hereby granted, provided that

1. Both the above copyright notice and this permission notice appear in all copies of both the software and the supporting documentation.
2. No commercial use is made out of it.

WARRANTY - LIABILITY

The software is supplied on an "as is" basis without warranty of any kind, and the author shall have no liability or obligation for damages, including but not limited to actual, indirect, consequential and incidental damages occurring out of or in connection with the use or performance of the software, without regard to whether such damages were foreseeable.

The author can not be obliged to give any technical or training with respect to the above mentioned software.

2.1. Set Of Statement Polytopes Matrix

The Set Of Statement Polytopes (SetOfSP) matrix is a two dimensional matrix where each row defines one statement polytope. For each dimension in the iteration space, the row has one positive start value and one negative end value. The end value will always be greater or equal to the start value. The order of the dimensions is of now consequence, but the same order must be used for all rows in the matrix, and also for all other input and output matrixes where dimensions are used. When references are made to the dimensions, the first dimension used in the SetOfSP is denoted dimension 1, the second dimension 2, and so on. Figure 1 defines the format of the SetOfSP matrix.

SetOfSP = [StartDim1 -EndDim1 StartDim2 -EndDim2 --- StartDimN -EndDimN;	% SP 1

StartDim1 -EndDim1 StartDim2 -EndDim2 --- StartDimN -EndDimN];	% SP M

Figure 1: Format of SetOfSP matrix for M statement polytopes and N dimensions

Figure 2 presents a general code example where all statements are placed in a common iteration space. The graphical representation of the common iteration space with dependency relations is given in Figure 3.

```

for i = 0 to 15
  for j = 0 to 7
    for k = 0 to 4
      {
        if (j <= 4) & (k <= 3)
          A[i][j][k] = f1( input );
        if (i <= 7) & (j >= 1) & (j <= 5) & (k <= 3)
          B[i][j][k] = f2( A[i][j-1][k] );
        if (i >= 8) & (j >= 1) & (j <= 5) & (k >= 1)
          C[i][j][k] = f3( A[i][j-1][k-1] );
        if (i >= 8) & (j == 6) & (k >= 1)
          D[i][j][k] = f4( B[i-8][j-1][k-1], C[i][j-1][k] );
        if (i >= 8) & (j == 7) & (k >= 1)
          E[i][j][k] = f4( D[8+15-i][j-1][k] );
      }
    }
  }
}

```

Figure 2: General Code Example

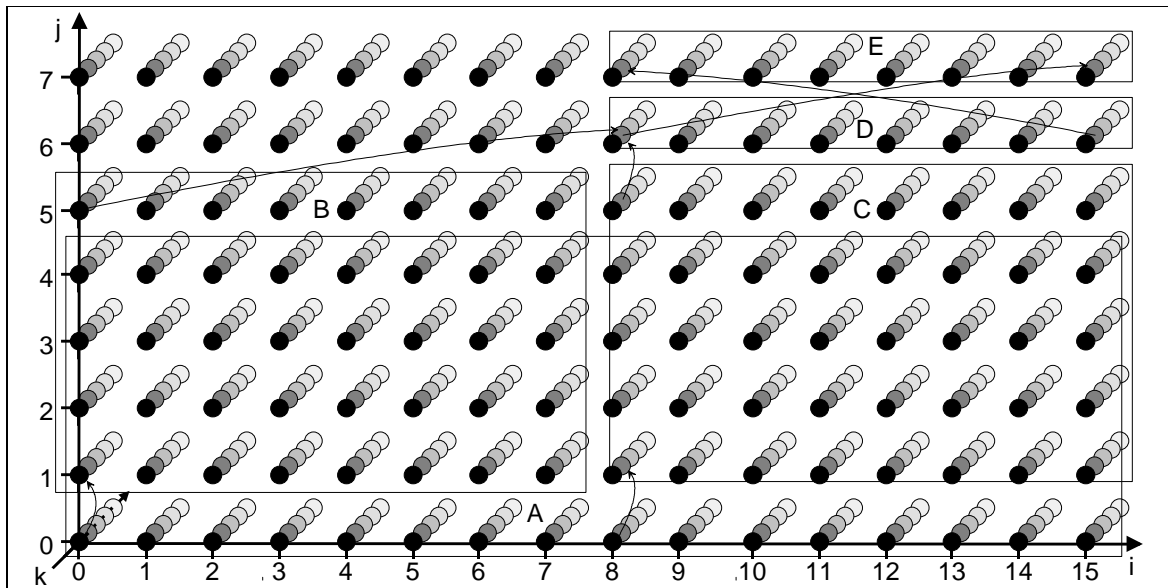


Figure 3: Common Iteration Space for General Code Example

In this example each statement produces elements of different arrays and with array locations of each element equal to its production location in the iteration space. This is in general not necessary, as long as the input code is single assignment and has manifest affine dependencies.

The SetOfSP matrix for this example is presented in Figure 4. Note the Matlab format where different rows are separated with a semicolon, and where ... signifies that the input continues on the next line. In this case the order of dimensions follow the sequential order of the code. The start and end points in the i-dimension is listed first, then the j-dimension, and finally the k-dimension. When references are made to the polytopes of the SetOfSP, the first row is denoted polytope 1, the second row polytope 2, and so on.

```
SetOfSP = [ 0 -15 0 -4 0 -3; ... %Polytope A
           0 -7 1 -5 0 -3; ... %Polytope B
           8 -15 1 -5 1 -4; ... %Polytope C
           8 -15 6 -6 1 -4; ... %Polytope D
           8 -15 7 -7 1 -4]; %Polytope E
```

Figure 4: SetOfSP matrix for General Code Example

2.2. Set Of Dependencies Matrix

The Set Of Dependencies (SetOfDep) matrix is a two dimensional matrix where each row defines one dependency between two statement polytopes. The first column in the matrix contains the polytope number from which the dependency stems. The second column contains the polytope number to which the dependency goes. The third column contains the number of Dependency Vectors (DVs) for each dependency. For a uniform dependency this will always be 1, while for a non-uniform dependency it may be larger than one. For non-uniform dependencies, only the extreme DVs need to be included. In each row, the correct number of DVs then follow. Again the positive start point and negative end point are listed for each dimension. The order of dimensions must be the same for all dependencies, and also the same as the order used in the SetOfSP matrix. For rows that are shorter than the maximum length row, a number of zeros are added so that all rows have the same length. Figure 5 defines the format of the SetOfDep matrix.

```
SetOfDep = [ FSP TSP NDV SD1DV1 -ED1DV1 SD2DV1 -ED2DV1 --- SDnDV1 -EDnDV1 %Dep 1
            ---
            SD1DVm -ED1DVm SD2DVm -ED2DVm --- SDnDVm -EDnDVm ...
            ---
            FSP TSP NDV SD1DV1 -ED1DV1 SD2DV1 -ED2DV1 --- SDnDV1 -EDnDV1 %Dep k
            ---
            SD1DVm -ED1DVm SD2DVm -ED2DVm --- SDnDVm -EDnDVm];
```

Figure 5: Format of SetOfDep matrix for k dependencies with n dimensions and maximum m DVs

FSP = From SP, TSP = To SP, NDV = Number of DVs,
SDiDVj = Start Dimension i of DV j, EDiDVj = End Dimension i of DV j

The SetOfDep matrix for the general code example of Figure 2 is given in Figure 6. All but the last dependency are uniform and have hence only one DV. When references are made to the dependencies of the SetOfDep, the first row is denoted dependency 1, the second row dependency 2, and so on. When references are made to the DVs of the dependencies of the SetOfDep, the first DV of a dependency is denoted DV 1, the second DV 2, and so on.

```
SetOfDep = [ 1 2 1 0 -0 0 -1 0 -0 0 0 0 0 0; ... %Dependency 1 (between A and B)
           1 3 1 8 -8 0 -1 0 -1 0 0 0 0 0; ... %Dependency 2 (between A and C)
           2 4 1 0 -8 5 -6 0 -1 0 0 0 0 0; ... %Dependency 3 (between B and D)
           3 4 1 8 -8 5 -6 1 -1 0 0 0 0 0; ... %Dependency 4 (between C and D)
           4 5 2 8 -15 6 -7 1 -1 15 -8 6 -7 1 -1]; %Dependency 5 (between D and E)
```

Figure 6: SetOfDep matrix for General Code Example

2.3. Fixed Dimensions Matrix

The Fixed Dimensions (FD) matrix is a matrix with one row and a number of columns equal to the number of dimensions in the common iteration space. The number of dimensions in the common iteration space is equal to the number of levels in the corresponding common loop nest. In the FD matrix, the first column represents the outermost nest level of the execution ordering. It can either contain a zero, indicating that no dimension is fixed at this position, or the number of the dimension fixed at this position. As explained in Section 2.1, the numbering of the dimensions follows the order of which they are used in the SetOfSP matrix. The following columns of the FD represent the second outermost nest level, the third outermost nest level, and so on, until the innermost nest level is reached. As mentioned the number of nest levels is the same as the number of dimensions in the iteration space. While the ordering of dimensions can be unfixed, nest level 1 is always the outermost level in the loop nest. Placing a dimension at a given nest level will thus fix this dimension at a given level in the loop nest. Figure 7 defines the format of the FD matrix.

FD = [DimNumAtNestLevel1 DimNumAtNestLevel2 --- DimNumAtNestLevelN]

Figure 7: Format of FD matrix for common iteration space with N dimensions/nest levels

A fully unfixed execution ordering for the general code example of Figure 2 is thus indicated with a FD matrix as shown in Figure 8 a). The FD matrix for an execution ordering with the second dimension fixed at the innermost nest level, is given in Figure 8 b). Note that the second dimension is the j-dimension when the ordering of Figure 3 is used. Finally Figure 8 c) shows the FD for a fully fixed ordering with dimension 3 (the k-dimension) outermost, dimension 1 (the i-dimension) second outermost, and dimension 2 (the j-dimension) innermost.

a) FD = [0 0 0]	b) FD = [0 0 2]	c) FD = [3 1 2]
----------------------	----------------------	----------------------

Figure 8: Different FD matrixes for the General Code Example

As will be explained in Chapter 4, the first step to perform when STOREQ is being used, is an initialization step. The FD matrix is then initialized to have the correct number of columns, and with a fully unfixed execution ordering. The user can afterwards change the FD matrix to reflect the (partially) fixed execution ordering before the next steps of the STOREQ are performed. or repeated.

2.4. Ignore Dependency Matrix

The Ignore Dependency (IgnoreDep) matrix is a matrix with one row and a varying number of columns equal to the number of dependencies that should be ignored in the next run of STOREQ. If no dependencies are to be ignored, the IgnoreDep matrix should be empty with length zero. Each column contains the dependency number, see Section 2.2, of a dependency that should be ignored. The same initialization step that initializes the FD matrix, also initialize the IgnoreDep as an empty matrix. No dependencies are hence ignored unless the IgnoreDep matrix is changed by the user. If more than one dependency is to be ignored, they must be listed in increasing dependency numbers. Figure 9 defines the format of the IgnoreDep matrix

IgnoreDep = [IgnoreDep1 IgnoreDep2 --- IgnoreDepN]
--

Figure 9: Format of IgnoreDep matrix when N dependencies are ignored

The IgnoreDep is typically useful when the user wants to experiment with optimization of the execution ordering for only a subset of the original dependencies, but do not want to change the input matrixes. Figure 10 a) shows an empty IgnoreDep, while Figure 10 b) and c) show IgnoreDep matrixes that will remove dependency 5 and dependencies 3 and 5 for the SetOfDeps respectively. Note that dependency 3 is listed before dependency 5 in Figure 10 c). The opposite order would result in an error.

a)	IgnoreDep = []	b)	IgnoreDep = [5]	c)	IgnoreDep = [3 5]
----	----------------	----	-----------------	----	-------------------

Figure 10: Different IgnoreDep matrixes for the General Code Example

2.5. Current Restrictions to the STOREQ Input

The SetOfSP matrix should in general contain the polytopes of each statement of the application. This is the kind of information that will be available from the MASAI-3 tool. For uniform dependencies this is also a legal input format. For non-uniform dependencies the current version of the program requires that a polytope contains only the part of the statement polytope where array elements are produced that are read by the outgoing dependency. If several dependencies read partly the same array elements, it is hence necessary to define several polytopes, one for each dependency. These reduced polytopes correspond to the Dependency Parts (DPs) defined in [Kjeldsberg00]. The Atomium tool will be able to output these DPs. Additional input information would then however be needed for the STOREQ program when it is extended to detect the combined size of simultaneously alive dependencies as outlined in [Kjeldsberg00]. The input must include a data structure that indicates which polytopes that stem from the same array elements. There are no reasons why this should not be possible, and the Atomium tool will again be able to produce the necessary information. Alternatively the automatic generation of DPs, as it is now performed for uniform dependencies, must be extended to also hold for non-uniform dependencies. This requires a somewhat more complex dependency detection technique however.

3. Output Format

The main output to STOREQ is two matrixes. One presents the estimated upper and lower bounds on the storage requirement for each dependency. This matrix is denoted the Set Of Bounds (SetOfBounds). The second matrix presents the optimal ordering of dimensions for each dependency as found by the program, and is denoted the Set Of Fixed Dimensions Best Case Ordering (SetOfFDBC). In addition the program prints out guiding hints to the user regarding how to organize the dimensions legally for negative dependencies, and also a suggestion for the next FD to use.

3.1. Set Of Bounds Matrix

The Set Of Bounds (SetOfBounds) matrix is a two dimensional matrix where each row lists the estimated lower bound (LB) and upper bound (UB) on the storage requirement for a single dependency. The first column of each row contains the dependency number as defined in Section 2.2. The second column contains the LB and the third column contains the UB. Figure 11 defines the format of the SetOfBounds matrix.

SetOfBounds =	DepNum	LowerBound	UpperBound	% Dep 1

	DepNum	LowerBound	UpperBound	% Dep M

Figure 11: Format of SetOfBounds matrix for with M dependencies

Figure 12 shows the SetOfBounds matrix for the general code example, when the estimation has been performed with no execution ordering fixed.

SetOfBounds =	1	1	32
	2	5	48
	3	16	32
	4	1	32
	5	1	32

Figure 12: SetOfBounds matrix for General Code Example

3.2. Set Of Fixed Dimensions Best Case Ordering Matrix

The Set Of Fixed Dimensions Best Case Ordering (SetOfFDBC) matrix is a two dimensional matrix where each row lists the optimal ordering of dimensions for a single dependency. The first column of each row contains the dependency number as defined in Section 2.2. The second column represents the outermost nest level of the execution ordering. It contains the number of the dimension fixed at this position. As explained in Section 2.1, the numbering of the dimensions follows the order of which they are used in the SetOfSP matrix. The following columns of the SetOfFDBC represent the second outermost nest level, the third outermost nest level, and so on, until the innermost nest level is reached. If a minus sign is put in front of the dimension number, it signifies that this dimension is a nonspanning dimension (ND) for this dependency. Similarly, if there is no minus sign in front of the dimension number, it signifies that the dimension is a spanning dimension (SD) for this dependency. If a zero is encountered at any iteration level, it signifies that the ordering of the remaining dimensions has no consequence for the total storage requirement. Figure 13 defines the format of the SetOfFDBC matrix.

SetOfFDBC =	DepNum	DimNumAtNestLevel1	DimNumAtNestLevel2	---	DimNumAtNestLevelN	%Dep 1

	DepNum	DimNumAtNestLevel1	DimNumAtNestLevel2	---	DimNumAtNestLevelN	%Dep M

Figure 13: Format of SetOfFDBC matrix with M dependencies and a common iteration space with N dimensions/nest levels

Figure 14 shows the SetOfFDBC matrix for the general code example, when the estimation has been performed with no execution ordering fixed.

SetOfFDBC =	1	-1	-3	2
	2	-1	2	3
	3	3	1	0
	4	-1	-3	2
	5	1	-3	2

Figure 14: SetOfFDBC matrix for General Code Example

3.3. Printed Guiding Hints

The STOREQ program presents a number of guiding hints to the user during execution. When the first initialization step is performed, any negative dependency vectors (DVs) are detected. The negative dimension(s) are printed together with all the spanning dimensions (SDs) of this DV. When all negative DVs have been reported, the user is reminded that a negative dimension can not be placed outermost among the SDs of this DV. This must be taken into account when the Fixed Dimensions (FD) matrix is changed later on.

During an ordinary run of the STOREQ program, the current FD matrix is first printed. If this FD matrix results in an illegal ordering of any negative dimensions, a warning regarding this is printed. This can happen even if a fully unfixed FD matrix is given as input, since the program assumes that any negative dimensions are taken care of using a partially fixed ordering. With no ordering fixed, an illegal best case ordering may be found, and this ordering will then result in the warning.

Since a possibly illegal best case ordering is used during the estimation of a dependency with a negative DV, the lower bound (LB) may not be correct. The program detects this situation, and prints out a legal FD matrix to be used to find the LB. A legal FD matrix for LB calculation is only printed if a fully unfixed FD matrix has been used as input. Any partially fixed execution ordering specified by the user is assumed to be legal so that both upper and lower bounds are correct. If FD matrix generates an illegal ordering, a warning about this will still be printed as indicated in the previous paragraph.

After having printed the SetOfBounds and SetOfFDBC matrixes, STOREQ finally prints a suggestion for the next FD matrix to use as input. This FD matrix uses any fixed dimensions as already specified in the current FD matrix, and places the remaining dimensions at the unfixed nest levels.

3.4. Current Restrictions to the STOREQ Output

The heuristic that finds the next FD matrix to use is currently not very advanced. It is assumed that the ordering for any negative dependencies is legal. The current version only takes into account the placement of the SDs in the optimal ordering of each dependency. It does not take into account the size of the penalty for each dependency when its optimal ordering is not used. This would be a relatively simple extension. Note that then it is probably only possible to indicate the next dimension to fix innermost, not the full FD, since the change in bounds will differ. Finally, when the simultaneously alive dependency detection is included in the main program, this information should also be taken into account here.

The function that generates the FD matrix to use for calculation of the LB currently assumes that only one negative dimension is present. For the negative dimension, it also assumes that the starting value of the statement polytope to which the dependency is going is smaller or equal to the starting value of the statement polytope from which the dependency is coming. This means that the placement of statement polytope D and E in Figure 3 is legal. Assuming that D stays at its current location it is also legal to move E to the left, but illegal to move it to the right.

4. Typical Sequence of Execution Steps

The STOREQ program can be used either as a pure estimation tool, given a partially fixed execution ordering, or as an interactive tool for exploration of the execution order search space.

4.1. Pure Estimation Tool

When used as a pure estimation tool, it is only necessary to define the four input matrixes in the Matlab workspace. This can be done simply by generating a text file with extension “.m” that contains the matrixes in the format described in Chapter 2. After Matlab has been started, the following commands must be executed:

```
>>“filename”.m  
>>RunSTOREQ
```

The resulting upper and lower bounds on the storage requirement of individual dependencies can then be found inspecting the resulting SetOfBounds matrix dumped in the Matlab workspace.

4.2. Interactive Use for Optimization of Execution Ordering

When STOREQ is used interactively, some more steps are needed. Each step, and the resulting output, will be described in somewhat detail in the following subsections.

4.2.1. Input Matrixes

The input will typically be the two SetOfSP and SetOfDep matrixes. A text file with the extension “.m” is normally the easiest input format. Appendix A lists the text file “GenCodeEx.m” with the input matrixes for the general code example used in this User Manual. To read the content of the text file into the Matlab workspace, Matlab is started and then the following command is executed:

```
>>GenCodeEx
```

4.2.2. Initialization

Since the input text file does not include the FD and IgnoreDim matrixes, these must be given initial values before the main body of the STOREQ program is run. This is done using the following command:

```
>>InitSTOREQ
```

In addition to generating a fully unfixed FD matrix and an empty IgnoreDim matrix, this function detects and reports any negative dependencies. The output from running InitSTOREQ on the GenCodeEx.m input is shown in Appendix B.

4.2.3. First Run of STOREQ

The next step after having initialized the FD and IgnoreDim matrixes, is to run the main body of STOREQ using the initial content of the FD and IgnoreDim matrixes. This can be useful even if some storage requirement information is available, especially if any negative dependencies were detected during the initialization. This ensures that STOREQ produces FD matrixes that can be used to find the lower bounds on the storage requirement of the negative dependencies. The following command can be used:

```
>>RunSTOREQ
```

The output from running RunSTOREQ on the GenCodeEx.m input and with the initial FD and IgnoreDim matrixes is shown in Appendix C.

4.2.4. Exploring Negative Dependency

As can be seen from Appendix B and Appendix C, dependency 5 has one Dependency Vector (DV) with a negative dimension 1. The output presented in Appendix B also informs the user that this DV has two Spanning Dimensions (SDs), dimension 1 and 2, and that it is illegal to place dimension 1 outermost among them. Furthermore, the output in Appendix C suggests that the user should use an FD = [0 0 1] matrix to find the lower bound on the storage requirement for dependency 5. The following command can be used:

```
>>FD= [0 0 1]
```

```
>>RunSTOREQ
```

The output from these commands is given in Appendix D, and shows that the lower bound on the storage requirement of dependency 5 is 8, not 1 as found with a fully unfixed execution ordering. What Appendix D also shows however, is the large size penalty inflicted on the other dependencies to be able to have a legal ordering for dependency 5. This indicates that it may be useful to perform a loop body split, placing dependency 5 in a separate loop nest. Dependency 1 through 4 can then be optimized without having to take into account the constraints of the legal ordering of dependency 1. (This is of course not the only transformation that will solve this problem, but it is used here for demonstration purposes.)

4.2.5. Exploring Optimal Ordering for Dependency 1 through 4

To be able to optimize the ordering for dependency 1 through 4 ignoring dependency 5, it is necessary to change the IgnoreDep matrix. To start from scratch it is also best to reset the FD matrix to the initial fully unfixed value before STOREQ is run again. The following commands can be used:

```
>>IgnoreDep = [5]
```

```
>>FD = [0 0 0]
```

```
>>RunSTOREQ
```

The output from these commands is given in Appendix E. As can be seen from the optimal ordering of the different dependencies, there are several conflicts. For dependency 2, dimension 3 should be placed innermost, while for the other dependencies dimension 2 should be placed innermost. Note that the zero in dependency 3 indicates that when dimension 1 has been placed at a given nest level, it has no consequence for the storage requirement how dimensions are ordered at nest levels inside it. In this case there are only one dimension remaining, 2, which must obviously be placed innermost.

The program suggests that and $FD = [1\ 3\ 2]$ should be used for the next run of STOREQ. The following commands can be used:

```
>>FD = [1 3 2]
```

```
>>RunSTOREQ
```

The output from these commands is given in Appendix F. There is a relatively large penalty on dependency 3, while the other dependency sizes are at or close to their optimal lower bounds. Further exploration will reveal that this is indeed the optimal ordering for the combined size of these dependencies, assuming that all of them are alive simultaneously.

5. References

[Kjeldsberg00] P.G. Kjeldsberg, "Storage Requirement Estimation and Optimization for Data Intensive Applications", draft of Ph.D. Thesis, NTNU, December 2000

Appendix A: Input Text file, GenCodeEx.m

```
%Statement Polytopes and DVs for STOREQ User Manual example
```

```
SetOfSP = [ 0 -15 0 -4 0 -3; ...  
           0 -7 1 -5 0 -3; ...  
           8 -15 1 -5 1 -4; ...  
           8 -15 6 -6 1 -4; ...  
           8 -15 7 -7 1 -4];
```

```
SetOfDep = [ 1 2 1 0 -0 0 -1 0 -0 0 0 0 0 0; ...  
            1 3 1 8 -8 0 -1 0 -1 0 0 0 0 0; ...  
            2 4 1 0 -8 5 -6 0 -1 0 0 0 0 0; ...  
            3 4 1 8 -8 5 -6 1 -1 0 0 0 0 0; ...  
            4 5 2 8 -15 6 -7 1 -1 15 -8 6 -7 1 -1];
```

Appendix B: Output from InitSTOREQ

```
>> InitSTOREQ
```

```
*****  
Dimension 1 of DV 2 of dependency 5 is negative  
DV 2 has the following SDs: 1 2
```

```
In any legal ordering, the negative dimension can not  
be fixed outermost among the SDs. Run the EstStorageReq  
twice to find the UB and LB. Use first a fully unfixed  
FD and then an FD as indicated by the first run.
```

```
*****
```

```
*****  
The FD is now initialized without any dimensions fixed.  
Customize it according to fixed dimensions and then run  
EstStorageReq(SetOfSP,SetOfDep,FD,IgnoreDep)  
The IgnoreDep is initialized so that no dependencies are ignored.  
*****
```

Appendix C: Output From RunSTOREQ with initial FD and IgnoreDim matrixes

```
>> RunSTOREQ
```

```
FD =
    0    0    0
```

```
*****
For dependency 5 it is not legal to place
dimension 1 outermost among its SDs.
*****
```

```
*****
For dependency 5 the current estimation only
produces a correct UB. To find the LB use
FD = [ 0 0 1 ]
*****
```

```
*****
DependencyNumber LowerBound UpperBound
s =
    1     1    32
    2     5    48
    3    16    32
    4     1    32
    5     1    32
```

```
DependencyNumber Best-CaseOrderingOfDimensions
- indicate ND
s =
    1    -1    -3    2
    2    -1     2    3
    3     3     1    0
    4    -1    -3    2
    5     1    -3    2
```

A suggestion for the next FD to try out is

```
FD = [ 3 1 2 ]
```

```
*****
```

Appendix D: Output From RunSTOREQ with FD = [0 0 1]

```

>> FD = [0 0 1]

FD =
    0    0    1

>> RunSTOREQ

FD =
    0    0    1

*****
DependencyNumber  LowerBound  UpperBound
s =
    1     8    32
    2    40    48
    3    16    32
    4     8    32
    5     8    32

DependencyNumber  Best-CaseOrderingOfDimensions
                  - indicate ND
s =
    1    -3     2    -1
    2     2     3    -1
    3     3     2     1
    4    -3     2    -1
    5    -3     2     1

A suggestion for the next FD to try out is
FD = [ 3 2 1 ]
*****

```

Appendix E: Output From RunSTOREQ with IgnoreDep = [5] and FD = [0 0 0]

```

>> IgnoreDep = [5]

IgnoreDep =
  5

>> FD = [0 0 0]

FD =
  0    0    0

>> RunSTOREQ

FD =
  0    0    0

*****
DependencyNumber  LowerBound  UpperBound
s =
  1      1      32
  2      5      48
  3     16      32
  4      1      32

DependencyNumber  Best-CaseOrderingOfDimensions
                  - indicate ND
s =
  1     -1     -3     2
  2     -1      2     3
  3      3      1     0
  4     -1     -3     2

A suggestion for the next FD to try out is
FD = [ 1 3 2 ]
*****

```

Appendix F: Output From RunSTOREQ with IgnoreDep = [5] and FD = [1 3 2]

```
>> FD = [ 1 3 2]
```

```
FD =
  1      3      2
```

```
>> RunSTOREQ
```

```
FD =
  1      3      2
```

```
*****
```

```
DependencyNumber  LowerBound  UpperBound
```

```
s =
  1      1      1
  2      6      6
  3     32     32
  4      1      1
```

```
DependencyNumber  Best-CaseOrderingOfDimensions
                  - indicate ND
```

```
s =
  1     -1     -3      2
  2     -1      3      2
  3      1      3      2
  4     -1     -3      2
```

```
A suggestion for the next FD to try out is
```

```
FD = [ 1 3 2 ]
```

```
*****
```