New features of PROC NETFLOW:
Using IBM's Optimization Subroutine Library (OSL)

Trevor D. Kearney, SAS Institute Inc., Cary, NC

ABSTRACT
This paper describes new features in PROC NETFLOW, a procedure in SAS/OR® software. These features can decrease solution times of your optimization problems.

PROC NETFLOW can now use Optimization Subroutine Library (OSL) to solve linear and network programming problems. For network programming problems, you can enlist OSL to do some or all optimization. The input data, output of PRINT statements and solution data sets are the same whether you use PROC NETFLOW's optimizer or OSL. OSL is much easier to use when controlled from within a SAS® procedure. This paper will illustrate how PROC NETFLOW can interactively run OSL.

WHAT IS OSL
The Optimization Subroutine Library (OSL) is an IBM® software product. It is a collection of FORTRAN mathematical subroutines currently available on IBM mainframes and IBM workstations. To improve performance, OSL takes advantage of architectural capabilities of these machines, such as the IBM 3090® and ES/9000® Vector Facilities, if available, and the IBM POWER (Performance Optimization and Enhanced RISC) Architecture on the IBM RISC System/6000®.

OSL can be used to solve linear, mixed integer, and quadratic programming problems. There are a variety of algorithms available, including Simplex and Interior-point methods. Besides the main optimization subroutines, OSL also consists of subroutines that can be called to perform tasks associated with any optimization process, such as scaling the coefficient matrix, and assigning values to optimizer tuning parameters.

At present, for SAS version 6.08, the PROC NETFLOW OSL interface for doing linear and network programming is available on MVS®. PROC NETFLOW will be able to use OSL to solve mixed integer programming problems in a future release of SAS. Models with special ordered sets and arcs and nonarc variables that have fixed charges will also be able to be solved. Future releases of SAS will allow you to get PROC NETFLOW to use OSL on platforms other than MVS. IBM has announced that OSL will be available in the future for other manufacturers' computers and for personal computer operating systems.

The documentation to access OSL through PROC NETFLOW will be in the next edition of the SAS/OR Reference Guide currently under development.

ADVANTAGES OF USING PROC NETFLOW TO CONTROL OSL
Ordinarily, to use OSL for mathematical programming, you would be required to write an application program in FORTRAN, PL/1, C, or APL that would call the OSL subroutines. You would need to know how to read or supply the data, what the structure of your program should be (the order in which the OSL subroutines should be called), and how to compile and link to the OSL library. You would have to be familiar with the subroutines themselves (what each does, their arguments, return codes, and so forth). Even when your optimization requirements are quite modest, the program needed to drive OSL could be fairly complicated.

It is much easier to use PROC NETFLOW. You do not need to work in FORTRAN, PL/1, C, or APL, or to be concerned with calling the OSL subroutines in a program. You can control OSL in a manner similar to controlling the PROC NETFLOW optimizer. Some special options and parameters that control only the OSL optimizer also can be specified. These are specified as you would specify any option or parameter of any SAS procedure. Furthermore, if you decide that OSL is not necessary, you can use the NETFLOW optimizer.

When used through PROC NETFLOW, OSL is a "black box" optimizer. The contents of any required input data sets, the output of PRINT statements, and the contents of output data sets that have the solution are identical whether you use PROC NETFLOW's optimizer or OSL. You can use PROC NETFLOW to control OSL interactively, just as you can with PROC NETFLOW's optimizer. You can stop the optimizer, save intermediate solutions, print solutions, and reset parameters. Before resuming optimization, you can specify that OSL is to change techniques, for example, from Interior-point to Simplex algorithms.

If you wish to solve a Linear Programming problem (LP), you can submit the LP data to PROC NETFLOW. PROC NETFLOW will determine that there is no network component to the problem and will automatically use OSL if available to perform the optimization.

Or you could use PROC LP (another procedure in the SAS/OR software) to solve linear and mixed-integer programming problems. PROC LP has its own optimizer; it does not rely on OSL being available to solve such problems.

OSL AND CONSTRAINED NETWORK PROBLEMS

THE ADVANTAGES OF USING PROC NETFLOW'S OPTIMIZER
PROC NETFLOW uses the Primal Simplex Network Algorithm and the Primal Partitioning Algorithm to solve constrained network problems. These algorithms are fast, since they take advantage of algebraic properties of the network component of the problem. In general, network algorithms can solve network programming models several times faster than LP codes.

If the network component of the model is large compared to the side constraint component, PROC NETFLOW's optimizer can store what would otherwise be a large matrix as a spanning tree data structure. Computations involving the spanning tree data structure can be performed much faster than those using matrices. Only the non-network part of the problem, hopefully quite small, needs to be manipulated by PROC NETFLOW as matrices.

In contrast, LP optimizers must contend with matrices which can be large for large problems. Arithmetic operations on matrices often
accumulate rounding errors that cause difficulties for the algorithm. So in addition to the performance improvements, network optimization is generally more numerically stable than LP optimization.

The nodal flow conservation constraints do not need to be specified in the network model. They are implied by the network structure. However, flow conservation constraints do make up the data for the equivalent LP model. If you have an LP that is small after the flow conservation constraints are removed, that problem is a definite candidate for solution by PROC NETFLOW.

THE ADVANTAGES OF GETTING PROC NETFLOW TO USE THE OSL OPTIMIZER

Some constrained network problems are solved more quickly by OSL than the network optimizer in PROC NETFLOW. Usually, they have a large number of side constraints or nonarc variables. These models are more like LPs than network problems. The network component of the problem is so small that PROC NETFLOW cannot recoup the effort to exploit that component rather than treat the whole problem as an LP. If this is the case, it is worthwhile to get PROC NETFLOW to convert a constrained network problem to the equivalent LP and use OSL to do some or all of the optimization. This conversion can be done either before any optimization has been performed (specify the USEOSL option in the PROC NETFLOW statement) or after PROC NETFLOW has done some optimization (specify the CONVERT; statement). CONVERT; is a new interactive statement of PROC NETFLOW. It can be specified once, after which OSL will perform any needed optimization.

Using the CONVERT; statement allows you to use both PROC NETFLOW's and OSL's optimizers to the best advantage. It is often beneficial to let PROC NETFLOW do some initial optimization; especially the unconstrained network optimization, the early constrained network optimization, or both. Then specify CONVERT; PROC NETFLOW will get OSL to use the current solution as its starting point, and OSL will conclude the optimization.

Even though some network problems are better solved by converting them to an LP, the input data and the output solution are more conveniently maintained as networks. You retain the advantages of casting problems as networks: ease of problem generation and expansion when more detail is required. The model and optimal solutions are easy to understand, as a network can be drawn.

The following section illustrates the use of OSL to solve a network problem. A later section describes how PROC NETFLOW and OSL can solve LP problems.

SOLVING A NETWORK PROBLEM BY USING OSL

Consider the Oil Industry example in the "Outline of Use" section in the PROC NETFLOW chapter of the SAS/OR Reference Guide (Version 6, First Edition). If the code that runs PROC NETFLOW is changed to:

```
PROC NETFLOW
* new option *
USEOSL
ARC DATA=ARC1
COND DATA=COND1
CONOUT=SOLUTION;
RUN;
```

OSL is used to solve the problem. The output data set SOLUTION created by this PROC NETFLOW run is identical to that created by PROC NETFLOW when the option USEOSL is not specified. Sometimes, differences result if multiple optimal solutions exist. However, the value of the optimal solutions will be equal.

The messages that appear on the SAS log will depend on which optimizer is used. The messages produced by the PROC NETFLOW optimizer (USEOSL is specified) are:

```
NOTE: Number of nodes 14.
NOTE: Number of supply nodes 2.
NOTE: Number of demand nodes 4.
NOTE: Total supply=180, total demand=18
NOTE: Number of arcs 18.
NOTE: Number of <= side constraints 0.
NOTE: Number of >= side constraints 2.
NOTE: Number of arc and scenario variable side constraints 0.
NOTE: Number of iterations (neglecting any constraint) = 0.
NOTE: Of these, 0 were degenerate.
NOTE: Optimal found.
NOTE: Minimum total cost=50875.
```

The number of variables may not equal the number of arcs if the problem has nonarc variables. This example has none. To convert a network to an equivalent LP problem, a flow conservation constraint must be created for each node (including any excess or bypass node). This explains why the number of equality side constraints and the number of constraint coefficients change when OSL is used.

INTERACTIVE USE OF PROC NETFLOW AND OSL

This section describes how PROC NETFLOW can be used interactively, and how PROC NETFLOW can control OSL interactively. You can control optimization in the same way regardless of whether you choose to use OSL or PROC NETFLOW's optimizer. Also, you can specify several specialized options and parameters to control the OSL optimizer interactively. If you wanted the same degree of control over OSL, you would have to write a fairly complex program in FORTRAN, PL/1, C, or APL.

Here are the steps of a typical PROC NETFLOW run:

- **The Start:** You start by giving the PROC NETFLOW statement. In this statement, you indicate options that can be set once when the procedure starts, for example, the names of the SAS data sets that contain the model data. PROC NETFLOW can have three input data sets:
  1. ARC DATA has data for arcs and nonarc variables; their objective function coefficients, and flow or value bounds. The tail and head nodes of the arcs are also specified in ARC DATA.
  2. NODE DATA has data for nodes that supply or demand flow.
  3. CON DATA has data for constraints.

You can optionally name the output data set, ARC OUT (if the problem has no side constraints), or CON OUT, in which the optimal solution will be put. If you want OSL to do the optimization from the outset, specify the option

```
TYPE=OSL
```

The messages produced by the OSL optimizer (USEOSL is specified) are:

```
NOTE: Number of nodes 14.
NOTE: Number of supply nodes 2.
NOTE: Number of demand nodes 4.
NOTE: Total supply=180, total demand=18.
NOTE: Number of arcs 18.
NOTE: Number of <= side constraints 0.
NOTE: Number of >= side constraints 2.
NOTE: Number of arc and scenario variable side constraints 0.
NOTE: Number of iterations (optimizing with constraints) = 4.
NOTE: Of these, 0 were degenerate.
NOTE: Optimal reached.
NOTE: Minimum total cost=50875.
```

The messages produced by the OSL optimizer (USEOSL is specified) are:

```
NOTE: Number of nodes 14.
NOTE: Number of supply nodes 2.
NOTE: Number of demand nodes 4.
NOTE: Total supply=180, total demand=18.
NOTE: Number of arcs 18.
NOTE: Number of <= side constraints 0.
NOTE: Number of >= side constraints 2.
NOTE: Number of arc and scenario variable side constraints 0.
NOTE: Number of iterations (neglecting any constraint) = 0.
NOTE: Of these, 0 were degenerate.
NOTE: Optimal found.
NOTE: Minimum total cost=50875.
```

The number of variables may not equal the number of arcs if the problem has nonarc variables. This example has none. To convert a network to an equivalent LP problem, a flow conservation constraint must be created for each node (including any excess or bypass node). This explains why the number of equality side constraints and the number of constraint coefficients change when OSL is used.
At any time, you can:

- **Convert the problem to the equivalent LP problem.** If the OSL optimizer has already been used and the optimization was stopped before the optimum was reached, you can specify the CONVERT; statement. This indicates that OSL should be used from then on. The solution the PROC NETFLOW optimizer reached is used as the starting solution for OSL.

- **Perform one iteration of the optimization.** If the OSL optimizer has already been used (the USEOSL option was used in the NETFLOW statement, or the CONVERT; statement had been issued) one OSL iteration will be performed. Otherwise one iteration of PROC NETFLOW’s optimizer will be performed.

- **Print output of the problem.** PRINT ARCS; makes PROC NETFLOW print information on all arcs. PRINT SOME ARCS limits this output to a subset of arcs. There are similar PRINT statements for nonarc variables and constraints: PRINT NONARCS, PRINT SOME NONARCS, PRINT CONSTRAINTS and PRINT SOME CONS. PRINT CON ARCS allows you to limit constraint information that is obtained to members of a set of arcs that have nonzero constraint coefficients in a set of constraints. PRINT CON NONARCS is the related statement for nonarc variables. For example,

\[
\text{PRINT CON ARCS (ROUTE08-ROUTE10, } \\
\text{ WARE: , } \\
\text{ SHOP2 "Customer X")} \\
\text{/NONZERO; }
\]

indicates that a line of information should be printed for each coefficient that:

- is in a constraint with the names ROUTE08, ROUTE09 or ROUTE10, and
- is associated with an arc that has a tail node with a name that starts with the characters WARE and has a head node that has the names SHOP2 or Customer X, and
- if the arc conveys nonzero flow in the current solution.

There are several other qualifiers besides NONZERO than can be specified after the slash. Some of these are:

- **SHORT,** if a short printout is required.
- **BASIC,** print information of an arc or nonarc variable provided it is basic.
- **LOWER,** print information of an arc provided the flow through the arc equals its lower flow bound. Information associated with a nonarc variable will be printed if its value equals its lower bound.
- **UPPER,** print information of an arc provided the flow through the arc equals its capacity. Information associated with a nonarc variable will be printed if its value equals its upper bound.

- **QUIT,** immediately stops PROC NETFLOW.
- **Save the current solution in SAS data sets.** The SAVE statement has options allowing you to name output data sets. Information about the current solution will be put into the data sets you name. If you then stop PROC NETFLOW, you can later use the saved solution to warm start PROC NETFLOW.
- **Examine the values of options.** Also, information about the amount of optimization that had already been done and what this has achieved (the status of the current solution) can be displayed.
- **Reset values of options and parameters.** The names of the output data sets can be altered. You can indicate the reasons why optimization should stop, for example, a maximum number of iterations that can be performed. PROC NETFLOW then has a chance to either execute the next statement, or if the next statement (the next PROC or DATA step in the SAS session) is one that it does not recognize, do any optimization it can or is allowed and finish. If no new statement has already been submitted, you are prompted for one.

Some options of the RESET statement enable you to control aspects of the optimization algorithms. From your computational experience, you may know that certain values for these options may reduce the time it takes to solve a problem you have dealt with before. Some of the specialized options new to PROC NETFLOW that control OSL’s optimization algorithm are described in the “OSL Options and Parameters” section later in this paper.

- **Start or resume optimization.**

If you are only interested in finding the optimal solution, have used SAS variables that have special names in the input data sets or wish to use default setting for everything, then all you need is

\[
\text{PROC NETFLOW} \\
\text{ARCDATA=data set and/or CONDATA=data set} \\
\text{USEOSL;}
\]

Controlling PROC NETFLOW interactively is useful if you wish to monitor progress to optimality. For example, to use PROC NETFLOW to do some preliminary optimization, and then use OSL to finish up, a run might go something like this:

\[
\text{PROC NETFLOW} \\
\text{ARCDATA=data set and/or CONDATA=data set} \\
\text{USEOSL;}
\]

351
More specifically, if you want OSL to perform 10 iterations, print the settings of the options used to control OSL, save that solution (conout=soln10) and then use OSL to find the optimal solution and put this into a data set (conout=optimum), use:

```plaintext
PROC NETFLOW
USEOSL
MAXITER=10
COMOPT=OPTIMUM
NODEDATA=NODE1 ARCDATA=ARCl CONDATA=CONl;
RUN;
SHOW OSL;
SAVE CONOUT=SOLN10;
RESET MAXITER=1000;
RUN;
```

For a constrained network problem, the log messages might look like:

```plaintext
NOTE: The data set WORK.SOLN10 has 14.2 observations and 13 variables.
```

SOLVING LINEAR PROGRAMMING PROBLEMS IN PROC NETFLOW

PROC NETFLOW can use OSL to solve LP problems. PROC NETFLOW recognizes that the problem is an LP if the problem has no arcs. All the variables are, in constrained network terms, nonarc variables. You do not need to specify USEOSL if this is the case.

There is no NODEDATA=data set. An ARCDATA=data set may not be necessary, but if used, the ARCDATA=data set has no TAIL or HEAD list variables. This data set contains observations that have variable names, objective function coefficients, and upper- and/or lower bound values.

As SHOW OSL; was specified, the options and parameters that control the OSL optimizers were listed. Where the MAXITER parameter was changed in the RESET statement, you could have also changed any of the OSL parameters.

SOLVING LINEAR PROGRAMMING PROBLEMS IN PROC NETFLOW

PROC NETFLOW can use OSL to solve LP problems. PROC NETFLOW recognizes that the problem is an LP if the problem has no arcs. All the variables are, in constrained network terms, nonarc variables. You do not need to specify USEOSL if this is the case.

There is no NODEDATA=data set. An ARCDATA=data set may not be necessary, but if used, the ARCDATA=data set has no TAIL or HEAD list variables. This data set contains observations that have variable names, objective function coefficients, and upper- and/or lower bound values.
The following code illustrates various ways in which data for this example can be supplied to PROC NETFLOW, and the associated PROC NETFLOW calls. In the first run, the dense format of CONDATA is used. In the second run, the sparse format of CONDATA is used. These data sets could have been used as input to PROC LP.

The sparse format of CONDATA resembles the MPS format that is standard for many LP packages. The sparse format of both PROC NETFLOW and PROC LP is an improved MPS format. You can have row and column names longer than 8 bytes. The data can be in any order. There are new, more convenient ways to supply data to our procedures.

```
data vardata;
input _name_ $ _cost_ _capac_ _lo_; cards;
x1 4 6 0
x2 3 5 1
;
da dense;
input _row_ $ x1 x2 _type_ $ _rhs_; cards;
con1 3 4 le 12
con2 3 3 le 10
con3 4 2 ge 6
;
proc netflow
arcdata=vardata condata=cdense;
run;
```

```
data csparse;
input _column_ $ _type_ $ _row_ $ _coef_; cards;
_type_: conl-1
_rhs_: conl 12
x1 : conl 3
x2 : con4 4
_rhs_: con2 10
x1 1e con2 3
x2 le con2 3
 . _type_ con3 1
  . rhs con3 8
  . x1 con3 4
  . x2 con3 2
;
proc netflow
arcdata=vardata condata=csparse sparsecondata;
run;
```

OSL OPTIONS AND PARAMETERS

This section contains a list and brief description of options and parameters that can be specified to PROC NETFLOW to control OSL. A complete description will be given in the new SAS/OR Reference Guide. Nearly the full complement of options that you could use to control OSL if you wrote a FORTRAN, PL/I, C, or APL program to drive OSL can be specified as options accepted by PROC NETFLOW.

Some of the options are simple OSL options. Others are values of OSL subroutine arguments used to indicate to PROC NETFLOW how you wish an OSL subroutine to be called, if at all.

For example, the OSL subroutine that performs Simplex optimization is called EKKBSLV. The SSLVALG option that you can specify when using PROC NETFLOW to run OSL is the value of the "arg" argument of EKKBSLV. You do not really need to know much about how and when PROC NETFLOW should call EKKBSLV, but if you want OSL to use the Simplex algorithm, you could specify:

- SSLVALG=0, the algorithm is based on the characteristics of your particular problem; that is, OSL chooses whether the primal or dual Simplex method will be used.
- SSLVALG=1, the primal algorithm is to be used.
- SSLVALG=2, the dual algorithm is to be used.

Besides the Simplex methods, OSL allows you to use an Interior-point Logarithmic Barrier method, abbreviated IPBM. If you indicate that IPBM should be used, specify the IPBM option in the PROC NETFLOW statement. The IPBM method can be used to do some optimization before the simplex method takes over, or can be used until the optimal solution is found.

The OSL subroutine that performs the IPBM optimization is called EKKBSLV. The BSLVALG and SSLVSWCH options that you can specify when using PROC NETFLOW to run OSL are the values of the "arg" and "sslvswch" arguments of EKKBSLV. You do not really need to know much about how and when PROC NETFLOW should call EKKBSLV, but if you want OSL to use the IPBM, you could specify (besides USEOSL and IPBM):

- BSLVALG=0, OSL chooses an IPBM method based on the problem characteristics.
- BSLVALG=1, the primal barrier algorithm is used.
- BSLVALG=2, the primal-dual barrier algorithm is used.
- BSLVALG=3, the primal-dual barrier algorithm with a predictor-corrector method is used.

To control when the IPBM method relinquishes optimization to OSL's Simplex method, specify:

- SSLVSWCH=0, never switch to the Simplex method (until SSLVSWCH is reset to another value).
- SSLVSWCH=1, switch to the Simplex method when numerical instabilities arise.
- SSLVSWCH=2, always switch to the Simplex method when the IPBM finishes or if numerical instabilities arise.
- SSLVSWCH=3, immediately switch to the Simplex method if it is appropriate after analyzing the matrix, when the IPBM finishes, or if numerical instabilities arise.
- SSLVSWCH=4, immediately switch to the Simplex method. A basis is created for the Simplex method and Simplex optimization starts.

An option you can specify that indicates whether an OSL subroutine must be called by PROC NETFLOW is called OSLSCALE. If you specify OSLSCALE=NONE, the problem will not be scaled. Other values of this option are:

- OSLSCALE=B4.IPBM, if the problem should be scaled before IPBM optimization.
- OSLSCALE=B4.SIM, if the problem should be scaled before simplex optimization.
- OSLSCALE=RECOMMENDED, (the default) if the problem should be scaled when IBM recommends this should be done. PROC NETFLOW gets OSL to scale the problem before IPBM only if the primal barrier algorithm is used; otherwise the problem is scaled before Simplex optimization is done. If OSLSCALE=RECOMMENDED and no IPBM optimization is done, the problem will be scaled for the Simplex method.
The options available with the PROC NETFLOW statement are summarized by purpose as follows:

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPBM</td>
<td>use IPBM method from outset</td>
</tr>
<tr>
<td>MAXMIN=</td>
<td>negate USEOSL</td>
</tr>
<tr>
<td>NOUSEOSL</td>
<td>if, when and how the problem is scaled</td>
</tr>
<tr>
<td>SIM</td>
<td>use the Simplex method from the outset</td>
</tr>
<tr>
<td>TOLDINF=</td>
<td>allowed amount of dual infeasibility</td>
</tr>
<tr>
<td>USEOSL</td>
<td>use the OSL optimizer from the outset</td>
</tr>
<tr>
<td>CHANGECOUNT=</td>
<td>rate of change for PWEIGHT or DWEIGHT</td>
</tr>
<tr>
<td>CRASH=</td>
<td>type of crash, if any</td>
</tr>
<tr>
<td>DWEIGHT=</td>
<td>multiplier of the feasible objective</td>
</tr>
<tr>
<td>FASTITS=</td>
<td>controls when DEVEP pricing should be used</td>
</tr>
<tr>
<td>LOGFREQ=</td>
<td>controls how often OSL prints information</td>
</tr>
<tr>
<td>LOGLEVEL=</td>
<td>controls how much information OSL prints</td>
</tr>
<tr>
<td>MAXFACTOR=</td>
<td>maximum number of iterations between refactorizations</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>maximum number of iterations</td>
</tr>
<tr>
<td>PWEIGHT=</td>
<td>multiplier of the feasible objective</td>
</tr>
<tr>
<td>SSLVALG=</td>
<td>type of IPBM algorithm used</td>
</tr>
<tr>
<td>SSLVINIT=</td>
<td>specifies when optimization should stop</td>
</tr>
<tr>
<td>STOPM=</td>
<td>specifies when to switch from IPBM</td>
</tr>
<tr>
<td>ADJACENCY=</td>
<td>controls the formation of the adjacency matrix</td>
</tr>
<tr>
<td>BSLIVALG=</td>
<td>type of IPBM algorithm used</td>
</tr>
<tr>
<td>CHOLABSTOL=</td>
<td>absolute pivot tolerance for Cholesky factorization</td>
</tr>
<tr>
<td>CHOLRELTOL=</td>
<td>obsolete option (OSL Release 1, SAS/OR Version 6.07), relative pivot tolerance for Cholesky factorization</td>
</tr>
<tr>
<td>CHOLTINYTOL=</td>
<td>maximum cut-off tolerance for Cholesky factorization</td>
</tr>
<tr>
<td>DENSECOL=</td>
<td>dense column threshold</td>
</tr>
<tr>
<td>DENSETH=</td>
<td>density threshold for Cholesky processing</td>
</tr>
<tr>
<td>DROPTWICT=</td>
<td>constraint dropping threshold</td>
</tr>
<tr>
<td>FIXVAR1=</td>
<td>phase 1 tolerance for fixing variables</td>
</tr>
<tr>
<td>FIXVAR2=</td>
<td>phase 2 tolerance for fixing variables</td>
</tr>
<tr>
<td>FORMTYPE=</td>
<td>controls the formation of the normal matrix</td>
</tr>
<tr>
<td>MAXITER=</td>
<td>maximum number of IPBM iterations</td>
</tr>
<tr>
<td>MAXPROJNS=</td>
<td>maximum null space projections</td>
</tr>
<tr>
<td>MUFAC=</td>
<td>reduction factor for μ</td>
</tr>
<tr>
<td>MUI=</td>
<td>initial value of μ for the primal barrier algorithm</td>
</tr>
<tr>
<td>MULT=</td>
<td>multiple of μ to add to the linear objective</td>
</tr>
<tr>
<td>NULIV=</td>
<td>null space checking switch</td>
</tr>
<tr>
<td>OBJWEIGHT=</td>
<td>weight given to the true objective in composite phase 1</td>
</tr>
<tr>
<td>PDGAP=</td>
<td>primal-dual gap tolerance</td>
</tr>
<tr>
<td>PDSTEP=</td>
<td>primal-dual barrier method</td>
</tr>
<tr>
<td>PERTDIAG=</td>
<td>diagonal perturbation for Cholesky factorization</td>
</tr>
<tr>
<td>POSSBASE=</td>
<td>potential basis flag</td>
</tr>
<tr>
<td>PROJUTOL=</td>
<td>projection error tolerance</td>
</tr>
<tr>
<td>RGFACTOR=</td>
<td>reduced gradient target reduction factor</td>
</tr>
<tr>
<td>RGLIMIT=</td>
<td>reduced gradient limit</td>
</tr>
<tr>
<td>ROWORD=</td>
<td>the row ordering indicator</td>
</tr>
<tr>
<td>SSLVSWITCH=</td>
<td>controls when to switch from IPBM</td>
</tr>
<tr>
<td>STOPM=</td>
<td>step-length multiplier for the primal barrier algorithm</td>
</tr>
</tbody>
</table>

**STOPPING AND STARTING OSL OPTIMIZATION**

In the “Interactive Use of PROC NETFLOW and OSL” section, there was an example PROC NETFLOW run demonstrating how PROC NETFLOW’s internal optimizer was used to perform part of the optimization, then CONVERT; was specified, the OSL Simplex optimizer was crashed with the solution at hand and the OSL Simplex optimizer went on to find the optimum. (To crash an optimizer is to supply it with a starting solution.) This section outlines how the OSL optimizers can be stopped and started, and how you can get the OSL Simplex optimizer to take over from the OSL IPBM optimizer.

The IPBM optimizer cannot be crashed. If you use the IPBM method, you must do so right from the start of all optimization; that is,

```
PROC NETFLOW
USEOSL /* Use OSL from the start */
IPBM   /* Use the IPBM optimizer */
ARCDATA=data set CONDATA=data set
RUN;
```

You cannot use the IPBM method after doing any PROC NETFLOW optimization or after doing any OSL Simplex optimization. However, you can stop and start the IPBM optimizer, you can stop the IPBM optimizer and get OSL to change to the Simplex optimizer, or you can let OSL decide when this change occurs.

Here is an example of stopping and starting the IPBM optimizer:

```
PROC NETFLOW
USEOSL /* Use OSL from the start */
IPBM   /* Use the IPBM optimizer */
SSLVSWITCH=0 /* Do not change from IPBM to */
/* the Simplex method. */
MAXITER=3 /* Limit the amount of IPBM */
/* optimization (3 iterations) */
ARCDATA=data set other options;
RUN;
/* OSL performs 3 IPBM iterations and pauses */
PRINT PROBLEM; /* Print current solution */
SAVE CONF=IPBM3; /* Save current solution */
SHOW STATUS OSL; /* What is the optimization */
/* status and what are the */
/* values the OSL options */
/* are currently set to. */
RESET MAXITER=999999 SSLVSWITCH=3; /* Prepare for more optimization, increase the */
/* number of iterations that are allowed. */
/* Because SSLVSWITCH=3, when the optimization */
/* resumes, IPBM will be used, but the simplex */
/* method could be performed later. */
RUN; /* do more optimization */
```

If in the last RESET statement you had specified RESET MAX-
ITERB=10 SSLVSWSCH=3; OSL will analyze the matrix and may switch to the Simplex method if it determined it should. Otherwise, OSL will perform some IPBM optimization and switch to the Simplex method if:

- IPBM finishes, or
- numerical instabilities arise, or
- the total number of IPBM iterations performed reaches 10 (because MAXITERB=10 was specified).

If you want to precisely control when OSL changes from the IPBM to the Simplex method, a PROC NETFLOW run might be like this:

```plaintext
PROC NETFLOW
  USEROSL /* Use OSL from the start */
  IPBM /* Use the IPBM optimizer */
  SSLVSWSCH=0 /* Do not change from IPBM to */
  /* the Simplex method */
  MAXITERB=3 /* Limit the amount of IPBM */
  /* optimization (3 iterations) */
  ARCDATA=data set other options;
RUN;
/* OSL performs 3 IPBM iterations */
RESET MAXITERB=5;
/* Get ready to do more IPBM optimization */
/* Note that SSLVSWSCH will still be 0 */
RUN;
/* OSL performs 2 more IPBM iterations */
RESET SSLVSWSCH=4;
/* Specify that OSL must stop the IPBM */
/* and use the Simplex method. */
RUN; /* OSL uses the OSL Simplex Optimizer */
/* Get ready to do Simplex optimization */
PROC NETFLOW
  USEROSL /* Use OSL from the start */
  IPBM /* Use the IPBM optimizer */
  SSLVSWSCH=0 /* Do not change from IPBM to */
  MAXITERB=3 /* Limit the amount of IPBM */
  /* optimization (3 iterations) */
  ARCDATA=data set other options;
RUN;
/* OSL performs 3 IPBM iterations */
RESET MAXITERB=5;
/* Get ready to do more IPBM optimization */
/* Note that SSLVSWSCH will still be 0 */
RUN;
/* OSL performs 2 more IPBM iterations */
RESET SSLVSWSCH=3
/* allow OSL to choose for itself when it */
/* changes from IPBM to Simplex */
MAXITERB=5 /* but do no more than 5 */
/* IPBM iterations. */
MAXITER=10 /* limit the number of */
/* Simplex iterations */
STOPMASK=3 /* stop Simplex optimization */
/* when feasible solution found */
RUN;
/* OSL does some IPBM optimization, then */
/* changes to Simplex. Optimization stopped */
/* when feasibility was reached */
RESET STOPMASK=0; /* normal termination */
RUN;
/* Optimization stopped as the total */
/* iterations equals 10 */
RESET MAXITERB=20;
RUN;
/* Optimization stopped as the total */
/* iterations equals 20 */
RESET MAXITERB=999999;
/* Optimization finishes */

Note the SSLVSWSCH=4 specification in the last RESET statement. When the IPBM subroutine is passed SSLVSWSCH=4, it immediately changes to Simplex method when optimization is next done; that is, when the next RUN; is specified (the case here), or if PROC NETFLOW detects that the next PROC of DATA step is about to commence and PROC NETFLOW is allowed to do more optimization. The only manual means IBM has provided to stop IPBM optimization is the MAXITERB= option.

Once OSL Simplex optimization commences, you can stop it by using the STOPMASK or MAXITER= options. The following code demonstrates how the IPBM and Simplex methods can be stopped and started.

```plaintext
Note the SSLVSWSCH=4 specification in the last RESET statement. When the IPBM subroutine is passed SSLVSWSCH=4, it immediately changes to Simplex method when optimization is next done; that is, when the next RUN; is specified (the case here), or if PROC NETFLOW detects that the next PROC of DATA step is about to commence and PROC NETFLOW is allowed to do more optimization. The only manual means IBM has provided to stop IPBM optimization is the MAXITERB= option.

Once OSL Simplex optimization commences, you can stop it by using the STOPMASK or MAXITER= options. The following code demonstrates how the IPBM and Simplex methods can be stopped and started.

```plaintext
INSTALLED OSL INTO SAS

For MVS, SAS will provide you with a small batch job that creates a load library called SASOSL. You need do this only once, or whenever maintenance is applied to your installation's FORTRAN runtime library.

Most of the time, to use SAS and OSL together, all you have to do above what you do now is to concatenate your installation's FORTRAN runtime library.

When you logon, specify REGION(16000K) or larger.

To invoke SAS in batch mode, I use:

```plaintext
sas path(‘/ssd.vsf230.vsf2fort’) +
  options(‘memsize=16M’)
```
CONCLUSION

PROC NETFLOW already has a powerful optimizer for solving constrained network problems; now for version 6.08, you can use OSL. OSL can advantageously be used in conjunction with, or as an alternative to PROC NETFLOW's optimizer. When dealing with an LP problem, PROC NETFLOW automatically uses OSL.

Accessing OSL through PROC NETFLOW is much easier than writing your own FORTRAN, C, PVI, or APL program that calls OSL subroutines. You do not need to know as much computer science, allowing you to concentrate on the (more important) mathematical programming aspects of your application. The data requirements, interactive statements, and output is the same whether PROC NETFLOW's optimizer, the OSL optimizer, or both, are used. You can interactively control OSL by specifying options. Doing this allows OSL to be used with a great deal of sophistication; pausing the optimization, changing and tuning optimization techniques, and so forth. On the other hand, using OSL can be as simple as specifying a single option: USEO SL. Either way, it is as if OSL is now a part of the SAS system.

SAS and SAS/OR are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. IBM, 3090, ES/9000, MVS, and RISC System/6000 are registered trademarks or trademarks of International Business Machines Corporation. * indicates USA registration.