OPERA TB 1.0 - A MATLAB Toolbox for Optimization Algorithms in Operations Research

Kenneth Holmström

Applied Optimization and Modeling Group (**TOM**) [1]

Center of Mathematical Modeling
Department of Mathematics and Physics
Mälardalen University
P.O. Box 883, S-721 23 Västerås, Sweden

Research Report in MATHEMATICS / APPLIED MATHEMATICS
Technical Report IMa-TOM-1997-1

16 November 1997 (Revised March 30, 1998)

[1] The TOM home page is **http://www.ima.mdh.se/tom**.

Abstract

The MATLAB toolbox **OPERA TB** is a set of MATLAB m-files, which solves basic optimization problems in operations research and mathematical programming. The focus is on dense problems. Included are routines for linear programming (LP), network programming (NP), integer programming (IP) and dynamic programming (DP).

LP problems are solved using the simplex method, the dual simplex method and interior points methods like Karmakar's method.

For NP problems **OPERA TB** has an implementation of the transportation simplex method together with three methods to find initial basic feasible solutions for the transport problem (TP). It also provides three shortest path algorithms and the Ford & Fulkerson's augmenting path algorithm to solve maximum flow problems. The symmetric traveling salesman problem is solved using Lagrangian relaxation and the subgradient method using the Polyak rule II.

The IP problems are either solved with a branch and bound algorithm or an implementation of the Gomory cutting plane method. A routine to solve 0/1 IP with Balas method is included.

Dynamic programming algorithms for the knapsack problem and the inventory problem are implemented. Included is also a Lagrangian relaxation method with knapsack subproblems.

Many test examples are implemented. The test examples are mainly exercises from the books by Luenberger (Linear and Nonlinear Optimization) and Winston (Operations Research - Applications and Algorithms).

OPERA TBis used in our undergraduate courses in operations research and mathematical programming for engineers and mathematicians.

1 Introduction

The MATLAB toolbox **OPERA TB** is part of **TOMLAB**[11, 12]. **OPERA TB** is a set of MATLAB m-files, which solves many basic optimization problems in operations research and mathematical programming. Included are routines for linear programming (LP), network programming (NP), integer programming (IP) and dynamic programming (DP). The focus is on dense problems, but sparse LP problems may be solved calling the commercial solver MINOS. The toolbox is running in both MATLAB 5.1 and MATLAB 4.2c and works on both PC systems (NT4.0, Win95, Windows 3.11) and on UNIX systems (SUN, HP). This paper describes the routines in the toolbox and how they are used.

Linear Programs may either be solved using an interactive menu program *lpOpt*, or solved by direct call to a parameter driven driver routine (lpRun). In either case the problem may be solved using any of the installed optimization solvers. **OPERA TB** has its own solvers, but also calls solvers in MATLAB Optimization Toolbox [4], as well as FORTRAN and C codes using MEX-file interfaces. The LP problem must be defined before running these routines. There is also a routine *simplex* to interactively define and solve LP problems.

A MEX-file interface for both PC and UNIX has been developed for the commercial optimization code MINOS, used in **OPERA TB** for linear programs and in **NLPLIB TB 1.0**[10] for nonlinear programming problems. MEX-file interfaces working on both PC and UNIX

have also been developed for the commercial codes from the Systems Optimization Laboratory (SOL) NPSOL, NPOPT, NLSSOL, QPOPT, LSSOL and LPOPT (all of them also present in the NAG library, with other names). The aim is to expand this list in the near future.

The internal solvers in **OPERA TB** solve LP problems using the simplex method, the dual simplex method and interior points methods like Karmakar's method.

For NP problems **OPERA TB** has an implementation of the transportation simplex method together with three methods to find initial basic feasible solutions for the transport problem (TP). It also provides three shortest path algorithms and the Ford & Fulkerson's augmenting path algorithm to solve maximum flow problems. Included is a routine to find the minimum spanning tree of an undirected graph and a network simplex algorithm to solve the Minimum Cost Network Flow Problem (MCNFP). The symmetric traveling salesman problem is solved using Lagrangian relaxation and the subgradient method using the Polyak rule II.

An IP problem is either solved with a branch and bound algorithm *branch* or the cutting plane method with Gomory cuts *cutplane*. A routine *balas* that solves 0/1 IP with Balas method is included.

As an example of dynamic programming one routine solving knapsack problems *dbknap* and one routine solving deterministic inventory problems *dbinvent* are implemented. Included is also a Lagrangian relaxation method with knapsack subproblems *ksrelax*.

Files which run simple tests with the different solvers are included in a separate directory operdemo. The test examples are mainly exercises from the books by Luenberger [13] and Winston [17] and from course material used in optimization courses at Linköping University [15] [16].

Some predefined linear programming test examples are included, which are used by the menu program and the driver routine. The aim is to expand this set in the future, to a more complete set of interesting test problems. It is easy to define your own problems to be solved by the menu program and the driver routine.

OPERA TB has been used for several years in optimization courses both at Mälardalen University and at Uppsala University. **OPERA TB** is distributed free of charge for academic purposes.

2 Installation and Machine Dependencies

If **OPERA TB** is installed as a standalone toolbox, the following routines from the toolbox **NLPLIB TB 1.0** must be included: *inputV*, *inputSet* and *goptions*.

2.1 Installation on PC systems

OPERA TB is normally installed as part of **TOMLAB**, with the subpath /tomlab/opera. On PC systems a normal choice is

\matlab\tomlab\opera

or

\matlab\toolbox\tomlab\opera.

This path must be added to the MATLAB search path. Before starting a session running **OPERA TB**, call *operaInit*, which sets the number of columns and declares the global variables used. If the user has a screen with more than 80 columns, the variable `MAXCOLS` in *operaInit* should be changed to the correct number.

The example files are stored in a separate directory, /tomlab/operdemo. The full path should be added to the MATLAB search path. As a possible alternative you can move to this directory when you want to run these files.

2.2 Installation on UNIX systems

OPERA TB is normally installed as part of **TOMLAB**, with the subpath /tomlab/opera. A possible full path is

/home/tomlab/opera

or

/home/matlab/toolbox/tomlab/opera. This path must be added to the MATLAB search path. Before starting a session running **OPERA TB**, call *operaInit*, which sets the number of columns and declares the global variables used. If the user has a screen with more than 80 columns, the variable `MAXCOLS` in *operaInit* should be changed to the correct number.

The example files are stored in a separate directory, usually in a directory

/home/tomlab/operdemo

or

/home/matlab/toolbox/tomlab/operdemo.

The full path should be added to the MATLAB search path. As a possible alternative you can move to this directory when you want to run these files.

3 Overview of the toolbox OPERA TB

The MATLAB toolbox **OPERA TB** was developed as a teaching tool for a course in operations research taught in the autumn of 1994. It is a collection of MATLAB m-files which solves many of the basic optimization problems in operations research and mathematical programming. Currently **OPERA TB** consists of:

- 11 200 lines of MATLAB code, in directory *opera* and *operdemo*.
- 55 files with algorithms and utilities in the directory *opera*.
- 45 example files in the directory *operdemo*.

- Code compatible with MATLAB 4.2c and MATLAB 5.1.

OPERA TB is suitable for teaching basic courses in optimization and operations research. In the following presentation the MATLAB m-file names are given in parenthesis.

4 The Functions in OPERA TB

In this section we describe **OPERA TB** by giving tables describing most MATLAB functions with some comments. All functions files are part of the directory *opera*.

There are two menu programs for linear programming. The *simplex* routine is a utility to interactively solve LP problems in canonical standard form. When the problem is defined, it simplex calls the internal **OPERA TB** solvers *lpsimp1* and *lpsimp2*.

The menu program *lpOpt* is similar to the menu programs in **NLPLIB TB 1.0** [10]. It calls the driver routine *lpRun*, which may call any of the predefined solvers written in MATLAB C or FORTRAN code. The user may either run *lpOpt* or directly call the driver routine *lpRun*.

Table 1: Menu programs and driver routines.

Function	**Description**
lpOpt.m	Menu program for LP problems
lpRun.m	Driver routine that solves predefined LP problems
simplex.m	Interactive input and solution of LP on canonical standard form

Like the MATLAB Optimization Toolbox, **OPERA TB** and **NLPLIB TB 1.0** are using a vector with optimization parameters. In Optimization Toolbox, the routine setting up the default values in a vector OPTIONS with 18 parameters is called *foptions*. Our solvers need more parameters, currently 29, and therefore the routine *goptions* is used instead of *foptions*. In **OPERA TB** the routine lpDef, see the Table 9, calls *goptions*. Both *lpOpt* and *lpRun* are using *lpDef* to setup the default parameter vector optPar.

4.1 Solving Linear Programs

There are several algorithms implemented for **linear programming**. The standard revised simplex algorithm as formulated in Goldfarb and Todd [7, page 91] is used to solve the Phase II LP problem (*lpsimp2*). A Phase I simplex strategy which formulates a LP problem with artificial variables is implemented (*lpsimp1*). This routine is using *lpsimp2* to solve the Phase I problem. The dual simplex method [7, pages 105-106], usable when a dual feasible solution is available instead of a primal feasible, is also implemented (*lpdual*).

Two polynomial algorithms for linear programming are implemented. Karmakar's projective algorithm (*karmark*) is developed from the description in Bazaraa et al [3, page 386]. There is a choice of update, either according to Bazaraa or the rule by Goldfarb

and Todd [7, chap. 9]. The affine scaling variant of Karmakar's method (*akarmark*) is an implementation of the algorithm in Bazaraa [7, pages 411-413]. As the purification algorithm a modification of the algorithm on page 385 in Bazaraa is used.

The internal linear programming solvers *lpsimp1*, *lpsimp2* and *lpdual* all have three rules for variable selection implemented. Bland's cycling prevention rule is the choice if fear of cycling exists. There are two variants of minimum reduced cost variable selection, the original Dantzig's rule and one which sorts the variables in increasing order in each step (the default choice). The linear programming routines are listed in Table 2.

Table 2: Routines for linear programming.

Function	**Description**
lpsimp1	The Phase I simplex algorithm. Finds a basic feasible solution (bfs) using artificial variables. Calls *lpsimp2*.
lpsimp2	The Phase II revised simplex algorithm with three selection rules.
lpdual	The dual simplex algorithm.
karmark	Karmakar's algorithm. Kanonical form.
lpkarma	Solves LP on equality form, by converting and calling *karmark*.
akarmark	Affine scaling variant of Karmarkar's algorithm.

4.2 Solving Transportation Programs

Transportation problems are solved using an implementation of the transportation simplex method as described in Luenberger [13, chap 5.4] (*TPsimplx*). Three simple algorithms to find a starting basic feasible solution for the transportation problem are included; the northwest corner method (*TPnw*), the minimum cost method (*TPmc*) and Vogel's approximation method (*TPvogel*). The implementation of these algorithms follows the algorithm descriptions in Winston [17, chap. 7.2]. The functions are described in Table 3.

Table 3: Routines for transportation programming.

Function	**Description**
TPnw	Find initial bfs to TP using the northwest corner method.
TPmc	Find initial bfs to TP using the minimum cost method.
TPvogel	Find initial bfs to TP using Vogel's approximation method.
TPsimplx	Implementation of the transportation simplex algorithm.

4.3 Solving Network Programs

The implementation of the **Network Programming** algorithms are based on the forward and reverse star representation technique described in Ahuja et al. [2, pages 35-36]. The following algorithms are currently implemented:

- Search for all reachable nodes in a network using a stack approach (*gsearch*). The implementation is a variation of the Algorithm SEARCH in [1, pages 231-233].
- Search for all reachable nodes in a network using a queue approach (*gsearchq*). The implementation is a variation of the Algorithm SEARCH in [1, pages 231-232].
- Find the minimal spanning tree of an undirected graph (*mintree*) with Kruskal's algorithm described in Ahuja et al. [2, page 520-521].
- Solve the shortest path problem using Dijkstra's algorithm (*dijkstra*). A direct implementation of the Algorithm DIJKSTRA in [1, pages 250-251].
- Solve the shortest path problem using a label correcting method (*labelcor*). The implementation is based on Algorithm LABEL CORRECTING in [1, page 260].
- Solve the shortest path problem using a modified label correcting method (*modlabel*). The implementation is based on Algorithm MODIFIED LABEL CORRECTING in [1, page 262], including the heuristic rule discussed to improve running time in practice.
- Solve the maximum flow problem using the Ford-Fulkerson augmenting path method (*maxflow*). The implementation is based on the algorithm description in Luenberger [13, pages 144-145].
- Solve the minimum cost network flow problem (MCNFP) using a network simplex algorithm (*NWsimplx*). The implementation is based on Algorithm network simplex in Ahuja et al. [2, page 415].
- Solve the symmetric traveling salesman problem using Lagrangian relaxation and the subgradient method with the Polyak rule II (*salesman*), an algorithm by Held and Karp [8].

The network programming routines are listed in Table 4.

4.4 Solving Integer Programs

To solve mixed linear inequality integer programs two algorithms are implemented. The first implementation (*branch*) is a branch and bound algorithm from Nemhauser and Wolsey [14, chap. 8]. The second implementation (*cutplane*) is a cutting plane algorithm using Gomory cuts. Both routines are using linear programming routines in the toolbox **OPERA TB** (*lpsimp1*, *lpsimp2*, *lpdual*) to solve relaxed subproblems. Balas method for 0/1 integer programs restricted to integer coefficients is implemented in the routine *balas* [9]. The routines for integer programming are described in Table 5.

Table 4: Routines for network programs.

Function	**Description**
gsearch	Searching all reachable nodes in a network. Stack approach.
gsearchq	Searching all reachable nodes in a network. Queue approach.
mintree	Finds the minimum spanning tree of an undirected graph.
dijkstra	Shortest path using Dijkstra's algorithm.
labelcor	Shortest path using a label correcting algorithm.
modlabel	Shortest path using a modified label correcting algorithm.
maxflow	Solving maximum flow problems using the Ford-Fulkerson augmenting path method.
salesman	Symmetric traveling salesman problem (TSP) solver using Lagrangian relaxation and the subgradient method with the Polyak rule II.
travelng	Solve TSP problems with branch and bound. Calls *salesman*.
NWsimplx	Solving minimum cost network flow problems (MCNFP) with a network simplex algorithm.

Table 5: Routines for integer programming.

Function	**Description**
cutplane	An implementation of the cutting plane method using Gomory cuts.
branch	Branch and bound algorithm for general IP.
balas	Branch and bound algorithm for binary IP using Balas method.

4.5 Solving Dynamic Programming Problems

Two simple examples of dynamic programming are included. Both examples are from Winston [17, chap. 20]. Forward recursion is used to solve an inventory problem (*dpinvent*) and a knapsack problem (*dpknap*), see Table 6. The routines are listed in Table 6.

4.6 Solving Problems Using Lagrangian Relaxation

The usage of Lagrangian relaxation techniques is exemplified by the routine *ksrelax*, which solves integer linear programs with linear inequality constraints and upper and lower bounds on the variables. The problem is solved by relaxing all but one constraint and hence solving simple knapsack problems as subproblems in each iteration. The algorithm is based on the presentation in Fischer [5], using subgradient iterations and a simple line search rule. Lagrangian relaxation is also used by the symmetric travelling salesman solver *salesman*. Also a routine to draw a plot of the relaxed function is included.

The Lagrangian relaxation routines are listed in Table 7.

Table 6: Routines for dynamic programming.

Function	**Description**
dpinvent	Forward recursion DP algorithm for the inventory problem.
dpknap	Forward recursion DP algorithm for the knapsack problem.

Table 7: Routines for Lagrangian relaxation.

Function	**Description**
ksrelax	Lagrangian relaxation with knapsack subproblems.
urelax	Lagrangian relaxation with knapsack subproblems, plot result.

5 Linear Programming Test Problems

Table 8 describes the low level test functions and the corresponding setup routines needed for the predefined linear programming test problems. The driver routine *lpRun* may also call nonlinear solvers to solve the LP problem, therefore some extra low level routines are needed, see **NLPLIB TB 1.0** for a detailed description.

Table 8: Predefined LP test problems.

Function	**Description**
lp_init	Defines the problem parameters A, b and c. Called from *lpRun*.
lp_f	Define the objective function for LP, $c^T x$ (for NLP solvers).
lp_g	Define the gradient function for LP, the vector c (for NLP solvers).
lp_H	Define the Hessian matrix for LP, A zero matrix (for NLP solvers).
lp_c	Define the constraint vector for LP, $b - Ax$ (for NLP solvers).
lp_dc	Define the constraint gradient matrix for LP, $-A^T$ (for NLP solvers).

5.1 Utilities

Table 9 describes the utility routines used in **OPERA TB**. Some of them are also used by **NLPLIB TB 1.0**. The routines starting with the two letters *LP* are used when using nonlinear solvers to solve the linear programs.

Table 9: Utility routines.

Function	**Description**
a2frstar	Convert node-arc A matrix to Forward-Reverse Star Representation.
z2frstar	Convert matrix of arcs (and costs) to Forward-Reverse Star.
lpDef	Define optimization parameters needed to run LP with lpRun or lpOpt.
mPrint	Print matrix, format: NAME$(i,:)$ $a(i,1)a(i,2)...a(i,n)$.
printmat	Print matrix with row and column labels
vPrint	Print vector in rows, format: NAME$(i_1 : i_n)$ $v_{i_1}v_{i_2}...v_{i_n}$
xPrint	Print vector x, row by row, with format.
xPrinti	Print integer vector x. Calls *xprint*.
xPrinte	Print integer vector x in exponential format. Calls *xprint*.
getAbc	Returns the matrices, A, b and c, that defines a LP problem.

5.2 MEX-file interfaces

It is possible to call FORTRAN and C/C++ code from MATLAB using dynamically linked subroutines, so called MEX-file interfaces. On PC Windows systems these interfaces were previously either built using 16-bit DLL files or 32-bit REX MEX-file interface. MATLAB 5.1 now supports 32-bit DLLs. A severe restriction is that the standard I/O routines in C and FORTRAN are not available; DLL MEX-files must use the Windows file I/O functions instead. For long it has been very difficult to develop a working MEX-file interface on PC Windows systems, especially MEX-file interfaces to FORTRAN code. Most optimization solvers are written in FORTRAN and also most of them draws heavy on a functioning I/O system.

On UNIX systems it is easier to develop working MEX-file interfaces, and therefor FORTRAN MEX-file code is available for several optimization solvers.

In **OPERA TB** there is currently one MEX-file interface developed; to the commercial solver MINOS. As standard, MINOS has an advanced I/O handling, but it is also possible to switch it off and use MINOS as a silent subroutine. The MEX-file interface is written in C and compiled and linked using the WATCOM C/C++ version 10.6 compiler. It was impossible to make it work without converting the whole MINOS FORTRAN code to C using *f2c*.

Table 10: MEX-file interface routines.

Function	**Description**
minoslp	Interface routine, which calls minosmex, the **NLPLIB TB 1.0** MEX-file interface to MINOS.

6 The OPERA TB Example Files

In this section we present the example files currently defined in **OPERA**. There are example files that illustrates the use of every algorithm defined. The example files are stored in a separate directory *operdemo*.

6.1 Test Examples for Linear Programming

A number of test examples, shown in Table 11, are implemented for linear programming. Cycling are exemplified with three examples (*excycle1*, *excycle2*, *excycle3*) and a menu routine *excycle* to choose between the cycle examples. The Klee-Minty test example (*exKleeM*), see Bazaraa et al. [3, page 376], illustrates the exponential convergence of the simplex method for different number of variables. In this example the simplex method using the original Dantzig selection rule will visit all vertices. The small example (*exfl821*) from Fletcher [6, ex. 8.21] has redundancy in the constraints.

The test examples for interior point methods are displayed in Table 12. There are three example files, *exww597*, *exstrang* and *exkarma*, illustrating the use of the two interior point solvers The routine *exKleem2* solves the Klee-Minty example using the two solvers.

Table 11: Test examples for linear programming.

Function	**Description**
exinled	First simple LP example from a course in Operations Research.
excycle	Menu with cycling examples.
excycle1	The Marshall-Suurballe cycling example. Run both the Bland's cycle preventing rule and the default minimum reduced cost rule and compare results.
excycle2	The Kuhn cycling example.
excycle3	The Beale cycling example.
exKleeM	The Klee-Minty example. Shows that the simplex algorithm with Dantzig's rule visits all vertices.
exfl821	Run exercise 8.21 from Fletcher, Practical methods of Optimization. Illustrates redundancy in constraints.
ex412b4s	Wayne Winston example 4.12 B4, using *lpsimp1* and *lpsimp2*.
expertur	Perturbed both right hand side and objective function for Luenberger 3.12-10,11.
ex6rev17	Wayne Winston chapter 6 Review 17. Simple example of calling the dual simplex solver *lpdual*.
ex611a2	Wayne Winston example 6.11 A2. A simple problem solved with the dual simplex solver *lpdual*.

Table 12: Test examples for linear programming running interior point methods.

Function	**Description**
exww597	Test of *karmark* and *lpsimp2* on Winston example page 597 and Winston 10.6 Problem A1.
exstrang	Test of *karmark* and *lpsimp2* on Strangs' *nutshell* example.
exkarma	Test of *akarmark*
exKleeM2	Klee-Minty example solved with *lpkarma* and *karmark*.

6.2 Introduction to Operations Research

A simple introduction in Swedish to Operations Research is implemented as three m-files described in Table 13.

Table 13: Operations research demonstration.

Function	**Description**
ORdemo	Demo giving simple examples of the use of operations research.
LPdemo	LP problems described. Called from *ordemo*.
NWdemo	NP problems described. Called from *ordemo*.

6.3 Test examples for transportation programming

The routine *exlu119* implements an example taken from Luenberger [13, page 119]. It runs the three routines that finds initial basic feasible solutions. For each starting basis it runs the transportation simplex algorithm *TPsimplx*. The test examples for TP problems are given in Table 14.

Table 14: Test examples for transportation programming.

Function	**Description**
extp_bfs	Test of the three routines that finds initial basic feasible solution to a TP problem, routines *TPnw*, *TPmc* and *TPvogel*.
exlu119	Luenberger TP page 119. Find initial basis with *TPnw*, *TPmc* and *TPvogel* and run *TPsimplx* for each.
exlu119U	Test unbalanced TP on Luenberger TP page 119, slightly modified. Runs *TPsimplx*.
extp	Runs simple TP example. Find initial basic feasible solution and solve with *TPsimplx*.

6.4 Test Examples for Network Programming

The test examples for network programming are shown in Table 15. There are four traveling salesman examples, *ulyss16*, *exulys16*, *exulys22* and *exgr96*. Problem N31 in routine *exflow31* is taken from [15].

Table 15: Test examples for network programming.

Function	**Description**
exgraph	Testing network routines on simple example.
exflow	Testing several maximum flow examples.
pathflow	Pathological test example for maximum flow problems.
exflow31	Test example N31.
exmcnfp	Minimum Cost Network Flow Problem (MCNFP) example from Ahuja et al.
ulyss16	Traveling salesman (TSP) example *Odyssey of Ulysses*. Calls *salesman*.
exulys16	TSP example *Odyssey of Ulysses*, 16 nodes. Calls *travelng*.
exulys22	TSP example *Odyssey of Ulysses*, 22 nodes. Calls *travelng*.
exgr96	TSP example *Africa Subproblem*, by Groetschel. 96 nodes. Calls *travelng*.

6.5 Test Examples for Integer Programming

Three simple test examples for integer programming are listed in Table 16. Test example PKorv is a simple example from Holmberg [9]. Example I39 is from [15].

Table 16: Test examples for integer programming.

Function	**Description**
expkorv	Test of *cutplane* and *branch* for example PKorv.
exIP39	Test example I39.
exbalas	Test of 0/1 IP (Balas algorithm) on simple example.

6.6 Test Examples for Dynamic Programming

The dynamic programming test examples in Table 17 are from Winston [17], Bazaraa et al. [3] and Holmberg [9]. The example I39 is from [15].

Table 17: Test examples for dynamic programming.

Function	**Description**
exinvent	Test of *dbinvent* on two inventory examples.
exknap	Test of *dbknap* (call *branch* and *cutplane*) on five knapsack examples

6.7 Test Examples for Lagrangian Relaxation

Routines illustrating the use of Lagrangian relaxation are shown in Table 18. Example I39 is from [15]. This example is also solved with dynamic programming, see Section 6.6.

Table 18: Test examples for Lagrangian relaxation.

Function	**Description**
exrelax	Test of *ksrelax* on LP example from Fischer -85.
exrelax2	Simple example, runs *ksrelax*
exIP39rx	Test example I39, relaxed. Call *urelax* and plot.

6.8 MATLAB Optimization Toolbox Solvers

The optimization solvers in the MATLAB Optimization Toolbox [4], that are possible to use in **OPERA TB** are listed in Table 19. The user must of course have a valid license. The driver routine checks if the actual routine is in the path, i.e. possible to call, and then uses *feval* to run it.

Table 19: MATLAB Optimization Toolbox routines callable from **OPERA TB** and **NLPLIB TB 1.0**.

Function	**Type of Problem**
constr	Constrained minimization
lp	Linear programming
qp	Quadratic programming

7 Printing Utilities and Print Levels

The amount of printing is determined by setting a print level for each routine. This parameter most often has the name PriLev.

Normally the zero level (PriLev = 0) corresponds to silent mode with no output. The level one corresponds to a result summary and error messages. Level two gives output every iteration and level three displays vectors and matrices. Higher levels give even more printing of debug type. See the help in the actual routine.

The main driver or menu routine called may have a PriLev parameter among its input parameters. The routines called from the main routine normally sets the PriLev parameter to optPar(1). The vector optPar is set to default values by a call to *goptions*. The user may then change any values before calling the main routine. The elements in optPar is described giving the command: help *goptions*. For linear programming there is a special initialization routine, *lpDef*, which calls *goptions* and changes some relevant parameters.

There is a wait flag in **optPar**, **optPar(24)**. If this flag is set, the routines uses the pause statement to avoid the output just flushing by.

The **OPERA TB** routines print large amounts of output if high values for the **PriLev** parameter is set. To make the output look better and save space, several printing utilities have been developed.

For matrices there are two routines, *mPrint* and *printmat*. The routine *printmat* prints a matrix with row and column labels. The default is to print the row and column number. The standard row label is eight characters long. The supplied matrix name is printed on the first row, the column label row, if the length of the name is at most eight characters. Otherwise the name is printed on a separate row.

The standard column label is seven characters long, which is the minimum space an element will occupy in the print out. On a 80 column screen, then it is possible to print a maximum of ten elements per row. Independent on the number of rows in the matrix, *printmat* will first display A(:,1:10), then A(:,11:20) and so on.

The routine *printmat* tries to be intelligent and avoid decimals when the matrix elements are integers. It determines the maximal positive and minimal negative number to find out if more than the default space is needed. If any element has an absolute value below 10^{-5} (avoiding exact zeros) or if the maximal elements are too big, a switch is made to exponential format. The exponential format uses ten characters, displaying two decimals and therefore seven matrix elements are possible to display on each row.

For large matrices, especially integer matrices, the user might prefer the routine *mPrint*. With this routine a more dense output is possible. All elements in a matrix row is displayed (over several output rows) before next matrix row is printed. A row label with the name of the matrix and the row number is displayed to the left using the MATLAB style of syntax.

The default in *mPrint* is to eight characters per element, with two decimals. However, it is easy to change the format and the number of elements displayed. For a binary matrix it is possible to display 36 matrix columns in one 80 column row.

8 Conclusions

The **OPERA TB** and **TOMLAB** is a powerful tool for computer based learning of optimization and in computer exercises. **OPERA TB** includes a large set of standard

algorithms and example files for linear and discrete optimization.

We would like to include more routines for network programs and integer programs. We have plans for the development of a mixed integer nonlinear optimization (MINLP) solver.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*. Elsevier/North Holland, Amsterdam, The Netherlands, 1989.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall Inc., Kanpur and Cambridge, 1993.

[3] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali. *Linear Programming and Network Flows*. John Wiley and Sons, New York, 2nd edition, 1990.

[4] Mary Ann Branch and Andy Grace. *Optimization Toolbox User's Guide*. 24 Prime Park Way, Natick, MA 01760-1500, 1996.

[5] Marshall L. Fisher. An Application Oriented Guide to Lagrangian Relaxation. *Interfaces 15:2*, pages 10–21, March-April 1985.

[6] Roger Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, New York, 2nd edition, 1987.

[7] D. Goldfarb and M. J. Todd. Linear programming. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*. Elsevier/North Holland, Amsterdam, The Netherlands, 1989.

[8] Michael Held and Richard M. Karp. The Traveling-Salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[9] Kaj Holmberg. Heltalsprogrammering och dynamisk programmering och flöden i nätverk och kombinatorisk optimering. Technical report, Division of Optimization Theory, Linköping University, Linköping, Sweden, 1988-1993.

[10] Kenneth Holmström. NLPLIB TB 1.0 - A MATLAB Toolbox for Nonlinear Optimization and Parameter Estimation. Technical Report IMa-TOM-1997-2, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997.

[11] Kenneth Holmström. TOMLAB - A General Purpose, Open MATLAB Environment for Research and Teaching in Optimization. Technical Report IMa-TOM-1997-3, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997. Presented at the 16th International Symposium on Mathematical Programming, Lausanne, Switzerland, August 24-29, 1997.

[12] Kenneth Holmström. TOMLAB - An Environment for Solving Optimization Problems in MATLAB. In M. Olsson, editor, *Proceedings for the Nordic MATLAB Conference '97, October 27-28*, Stockholm, Sweden, 1997. Computer Solutions Europe AB.

[13] David G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, Reading, Massachusetts, 2nd edition, 1984.

[14] G. L. Nemhauser and L. A. Wolsey. Integer programming. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science.* Elsevier/North Holland, Amsterdam, The Netherlands, 1989.

[15] LiU Optimeringslära. Optimeringsproblem för optimeringslära 1, I3, M4 och Y4. Technical report, Division of Optimization Theory, Linköping University, Linköping, Sweden, 1993.

[16] LiU Optimeringslära. Exempelsamling i kombinatorisk optimering grundkurs. Technical report, Division of Optimization Theory, Linköping University, Linköping, Sweden, 1994.

[17] Wayne L. Winston. *Operations Research: Applications and Algorithms.* International Thomson Publishing, Duxbury Press, Belmont, California, 3rd edition, 1994.