MOPS®

Mathematical **OP**timization **S**ystem

MOPS®

Mathematical **OP**timization **S**ystem

Benutzerhandbuch

Version 7.x, Juni 2004

Ansprechpartner für MOPS:
Prof. Dr. Uwe H. Suhl
Freie Universität Berlin
Lehrstuhl für Wirtschaftsinformatik
Garystraße 21
D-14195 Berlin (Dahlem)
FR Germany
Telefon: (49) (0)30 838 55009
Telefax: (49) (0)30 838 52027
E-mail: suhl@wiwiss.fu-berlin.de
http://www.mops-optimizer.com

Inhaltsverzeichnis

Änderungen von MOPS V4.0 gegenüber Version 3.x

- wesentlich verbessertes LP-Preprocessing (xrduce = 2)
- neues Interior Point Verfahren (IPM) zur Lösung von LP-Modellen (xlptyp = 4) mit einem „Crossover-Verfahren“ zur Bestimmung einer optimalen Basislösung.
- die Include-Dateien wurden modifiziert
- eine optimale LP-Lösung wird im Array xlpsol (anstelle xx) gespeichert

Änderungen von MOPS V5.x gegenüber Version 4.x

- weiter verbessertes LP-Preprocessing (xrduce = 2) und Interior Point Verfahren
- dynamisches Speichermanagement
- neues Eingabeformat für Triplets und lange Namen für Variablen und Restriktionen

Änderungen von MOPS V6.x gegenüber Version 5.x

- projected steepest edge pricing im primalen simplex
- verbesserte DLL-Schnittstellen

Änderungen von MOPS V7.x gegenüber Version 6.x

- neben primalem Simplex und innere Punkte Verfahren kann auch ein dualer Simplex zur Lösung von LP-Modellen eingesetzt werden. Der Parameter xlptyp legt fest, welche Engine zum Lösen des Anfangs-LPs gewählt werden soll: xlptyp = 0 impliziert primalen Simplex (default), xlptyp = 2 impliziert dualen Simplex und xlptyp = 4 führt zur Verwendung des innere Punkte Verfahrens zur Lösung des Anfangs LPs.
- Innerhalb des Branch-and-Bound-Verfahrens kann sowohl der primale Simplex (xiplpt = 0 als auch der duale Simplex (xiplpt >0) eingesetzt werden.
- Sind Range Restriktionen vorhanden, so kann das IP-Preprocessing manchmal wesentlich verbessert werden, wenn solche Restriktionen mit 0-1-Variablen in jeweils ein Paar von Restriktionen gesplittet werden (xranha = 1).
- Eine neue Schrankenreduktion Technik führt bei einigen Integer Modellen mit Fixed-Charge-Restriktionen zu verbesserten Laufzeiten (xbndrd = 2, bzw. xbndrd =3)
- Eine neue Heuristik zur Bestimmung einer Anfangslösung für ein IP-Modell wird aktiviert, wenn xnodse = -3, xheutp = 4 und xmnheu = 200, wobei xmnheu angibt wie viel Knoten maximal in der Heuristik entwickelt werden sollen.
- Das Save / Restore-Verfahren für einen Branch-and-Bound-Baum wird nicht mehr unterstützt.

0 Installation von MOPS (Windows PC)

Legen Sie ein Unterverzeichnis MOPS oder MOPS-Versions-Nr. an und kopieren Sie die Dateien mopsxxx.exe und ReadMe.txt vom Datenträger in dieses Unterverzeichnis. Anschließend muss mopsxxx.exe durch Doppel-Klick entpackt werden. Dazu muss das korrekte Kennwort eingegeben werden. Die Installation von MOPS unter Windows (Deutsch) erzeugt ein Verzeichnis c:\programme\mops mit zehn Unterverzeichnissen:

1. MOPS EXE: xmops.pro (Profile), mops.exe (32-Bit Lademodul als Windows Character Executable, Arbeitsbereichdatei mopsip.dsw für den Compaq Compiler zur Erstellung von mops.exe, xip.for (Fortran Hauptprogram)
2. lib: Objectcode-Bibliothek mopsdvf.lib erzeugt mit Visual Studio C++ und Compaq Visual Fortran 6.6 Compiler, sowie benötigte Intel MKL Bibliotheken
3. inc: alle Include-Dateien zur Compilation der Beispielprogramme
4. dll: Dynamic Link Library mops.dll mit On-Line-Hilfe mopsdll.hlp. Weiterhin werden vier Unterverzeichnisse (UV) mit Beispielprogrammen zur Verwendung der Dll angelegt:
   - UV C sample dll calls: enthält ein C-Programm, das die DLL aufruft
   - UV VB sample Burma: enthält Visual Basic Programme um die Fallstudie Burma (siehe mopsdll.hlp) auf verschiedene Weise zu lösen
   - UV VB sample dll calls: enthält ein VB-Programme, mit dem die wichtigsten DLL-Funktionen interaktiv ausgeführt werden können
   - UV VB sample sortopt: enthält eine kleine Beispielanwendung die eine Sortimentsoptimierung implementiert.
5. CaseVisualF: Fortran Modellgenerator für die Fallstudie (Kapitel 5.4), case.for, xmops.pro sowie die Arbeitsbereich- und Projektdateien für den Compaq-Compiler
6. CaseVisualC: C Modellgenerator für die Fallstudie (Kapitel 5.4), case.c, xmops.pro sowie die Arbeitsbereich- und Projektdateien für den Visual Studio C-Compiler.
7. models: Integer-Modelle im mps-Format und xcase,mps für die Modellgeneratoren
8. imr: Schnittstellendatei mit Routinen zur Modellgenerierung. Diese Routinen sind in den Objektcode-Bibliotheken enthalten und dienen nur zur Dokumentation.
9. ClipMOPS: enthält die Professional Version des MS Excel Add-In ClipMOPS. Die ClipMops.exe muss eigenständig installiert werden. Das Kennwort finset sich in der Datei ReadMe.txt.
10. Docu: enthält das aktuelle Benutzerhandbuch mops.pdf als pdf-Datei und nochmals die On-line-Hilfe zur Dll (mopsdll.hlp).

Installationstest (Ausschnitt)

- **Test des Lademoduls mops.exe im UV MOPS EXE**: Die Kommunikation zwischen MOPS und dem Anwender erfolgt über den Optimierungs-Profile xmops.pro. Dieser Profile ist auf Default-Werte eingestellt. Editieren Sie xmops.pro, um sich mit den Möglichkeiten vertraut zu machen. Dopel-Klick auf mops.exe sollte das ausgwählte Modell lösen.
- **Test des VisualC**-Modellgenerators: Starten Sie Visual Studio. Wählen Sie den Menüpunkt Datei öffnen und suchen in c:\Programme\mops\CaseVisualC nach dem Dateitypen (dsw, mdp). Sie finden case.dsw. Durch Öffnen des Arbeitsbereiches erhält man den Zugriff auf: Header, C-Sources, Bibliothek, Profile und Includes. Xcase.c enthält zwei Modellgeneratoren für die Fallstudie in Kapitel 5.4. Durch Klicken auf „Case Dateien“ mit der rechten Maustaste werden die Programme kompiliert und ccase.exe generiert. Ccase.exe kann im Menüpunkt „Erstellen (build)“ ausgeführt werden.

1 Einleitung und Übersicht

MOPS ist ein Softwaresystem zur Lösung großer linearer und gemischt-ganzzahliger Optimierungsmodelle. Diese Modellklassen werden auch mit LP (linear programming) und IP (integer programming) bezeichnet. Der Term IP wird sowohl für reine als auch für gemischt-ganzzahlige Modelle benutzt. Mathematisch können diese Problemklassen folgendermaßen definiert werden, wobei EMR für ***externe Modellrepräsentation*** steht:

$$\begin{aligned} &\textbf{Minimiere / maximiere} && \mathbf{c'\,x} \\ &&\mathbf{bl} \le \mathbf{A\,x} \le \mathbf{bu} && \textbf{(EMR)} \\ &&\mathbf{l} \le \mathbf{x} \le \mathbf{u} \\ &&\mathbf{x_j}\ \textbf{ganzzahlig für}\ \mathbf{j} \in \mathbf{JI} \end{aligned}$$

hierbei sind x, c, l, u reelle n-Vektoren, A eine reelle m x n Matrix, bl und bu reelle m-Vektoren. Die Vektoren l, u, bl, bu können auch Elemente, die plus oder minus unendlich sind, aufweisen. Dadurch können auch freie Variablen oder Restriktionen modelliert werden. Gleichungen ergeben sich dadurch, daß Elemente in bl und bu identisch sind. I = {1,..,m} und J = {1,..,n} sind die Indexmengen der Restriktionen und Variablen. JI $\subseteq$ J ist die Indexmenge der ganzzahligen Variablen. Ist JI leer, so definiert (EMR) ein LP-Modell. JI = J, so handelt es sich um ein reines ganzzahliges Problem, sonst um ein gemischt-ganzzahliges Modell.

Ganzzahlige Variablen müssen endliche untere und obere Schranken aufweisen, die im Intervall [- 32767, 32767] liegen müssen.

Das Softwaresystem MOPS [12,16] kann folgendermaßen charakterisiert werden:

- Hohe Leistung durch ausgefeilte Implementation des Simplex-Algorithmus. MOPS verwendet eine schnelle und stabile LU-Faktorisierung und LU-Update ([10, 11]), die zwischenzeitlich wesentlich verbessert wurden. Weiterhin werden bei langsamer Konvergenz automatisch verschiedene Varianten von Devex bzw. Steepest Edge Pricing aktiviert.
- Alternativ kann zum Simplex-Algorithmus ab Version 4.0 auch ein Innere Punkte Verfahren eingesetzt werden, das auf den Arbeiten von Cs. Mészáros [06,07,17] basiert. Große LP-Modelle können häufig viel schneller gelöst werden, als mit der Simplex-Engine. Da die Anzahl der Iterationen relativ unabhängig von der Modellgröße ist (typisch sind 20-40 Iterationen), wird der Arbeitsaufwand pro Iteration im wesentlichen nur durch das Lösen eines symmetrischen, positiv definiten Gleichungssystem bestimmt. Der “Fill”, d.h. wieviel neue Nichtnullelemente in der Cholesky-Faktorisierung entstehen, bestimmt maßgeblich, ob dieses Verfahren konkurrenzfähig zum Simplex-Algorithmus ist. Im Unterschied zum Simplexverfahren, ist eine Iteration im Inneren Punkte Verfahren sehr aufwendig.
- Bei vielen IP-Modellen sind durch automatisches Supernode-Processing drastische Laufzeitverkürzungen möglich. Dabei werden u.a. Schnittebenen generiert, Matrixkoeffizienten betragsmäßig reduziert und Variablen fixiert. Durch die Verschärfung der LP-Relaxation wird der Branch-and-Bound-Prozeß viel effektiver [13].

MOPS bietet verschiedene Benutzeroberflächen:

- ClipMOPS ist ein Excel Add-In mit dem kleinere Modelle in einem Excel Tabellenblatt eingegeben bzw. im MPS-Format ($\Rightarrow$ 3.2) geladen werden können. Die Optimierung kann per Mausklick erfolgen und optimale Lösungen im Tabellenblatt angezeigt werden.
- Lademodule (DOS, Windows, Unix), wobei ein Profile das Setzen von Parametern erlaubt. Modelldaten müssen hier im MPS- oder Triplet-Format vorliegen.
- Visual Basic, C++, C# und Borland Delphi-Programmen sind Optimierungsfunktionen der MOPS Dynamic Link Library (mops.dll) direkt zugänglich. Modelle, Parameter und Lösungen können im Hauptspeicher über die DLL-Schnittstellen ausgetauscht werden.

- Fortran oder C-Programme können Optimierungsfunktionen der Objectcode-Bibliothek (mops.lib) direkt rufen. Voraussetzung sind kompatible Compiler. Auf PCs wird die MOPS-Bibliothek mit Visual Studio C++ und dem Compaq Visual Fortran Compiler erzeugt. Programme können dann mit Routinen der MOPS-Bibliothek zu einem Lademodul gelinkt werden. Modelle können von Programmen direkt im internen MOPS-Format (⇒ 5.3) im Hauptspeicher generiert werden. und Lösungen direkt aus dem Haupstspeicher entnommen werden. Im Abschnitt 5.4 wird eine Fallstudie beschrieben, bei der dieser Weg beschritten wurde.

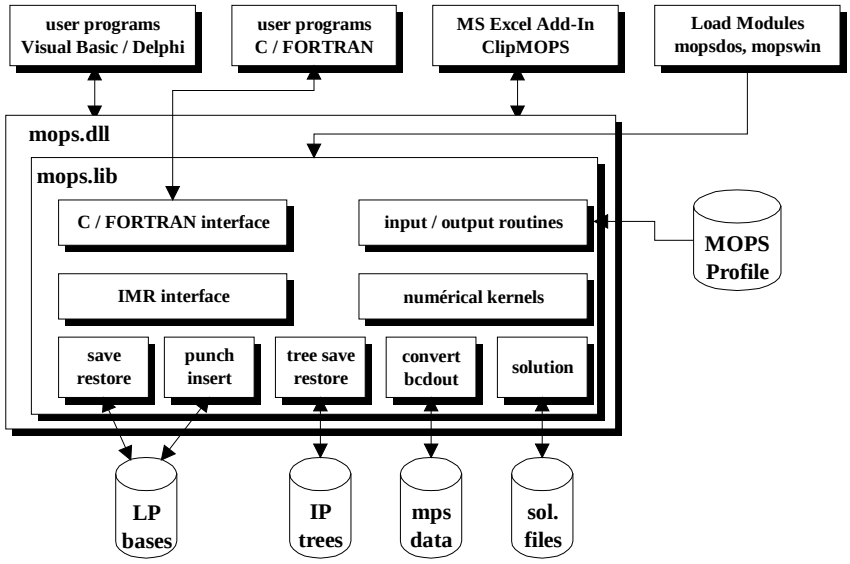


Abbildung 1: externe Architektur von MOPS

1.1 Systemrestriktionen

Da MOPS alle Daten während der Optimierung im Hauptspeicher hält, wird die maximale Problemgröße durch den zur Verfügung stehenden (virtuellen) Hauptspeicherplatz begrenzt. Bezüglich der maximalen Zeilen- bzw. Spaltenzahl gibt es keine realistischen Einschränkungen. Die maximale Modellgröße wird durch wenige Parameter festgelegt, die ggf. erhöht werden können (→ Speicherplatzallokation).

1.2 Systemkomponenten

Von einem externen Blickpunkt besteht MOPS in einer Windows oder Unix Umgebung aus:

- einer Objektcode-Bibliothek mopsdvf.lib; auf Intel-Plattformen mit Windows gibt es optional eine MOPS dynamic link library mops.dll (32-Bit)
- Lademodul mops.exe zur Lösung von LP und IP-Modellen
- Include-Dateien für benutzereigene C / FORTRAN-Programme zum Aufruf von MOPS:
- einer Profile-Datei xmops.pro zum Setzen von Systemparametern
- Hauptprogramm xip.for zur Lösung von LPs und IPs
- Modellgeneratoren für die Fallstudie in FORTRAN / C und den Modelldaten xcase.mps.

Unterschiede in Systemumgebungen werden im Kapitel 6 beschrieben.

1.3 Einbettung von MOPS in ein EUS

Der Einsatz von mathematischer Optimierungssoftware im Rahmen von entscheidungsunterstützenden Systemen (EUS) kann an Hand der Abbildung 2 erläutert werden: *Einige* Planungsfunktionen werden über LP- oder IP-Modelle ausgeführt. Vielfach werden relevante Daten vom Datenserver extrahiert und auf einer Workstation eine Datenbank (EUS-DB) für die Anwendung aufgebaut. Diese Datenbank wird in periodischen Abständen aktualisiert.

Bei der Modellimplementierung ist hohe Flexibilität bei Modelldefinition und Modellgenerierung wichtig. Dabei sollte z.B. die zeitliche Größe des Planungshorizontes und der einzelnen Planperioden frei wählbar sein. Modellierungssysteme wie z.B. AMPL, AIMMS, GAMS vereinfachen die Modellimplementation. Modellgeneratoren werden auch in prozeduralen Sprachen (z.B. C++, Fortran, Visual Basic) entwickelt, wobei mit eingebetteten SQL-Anwiesungen auf die EUS-DB zugegriffen wird. Vielfach werden Modelle für den Optimierer im MPS-Format generiert. Für zeitkritische Anwendungen sollten Modelldaten direkt im Hauptspeicher im internen Format des Optimierers generiert werden. Alternativ kann über die Schnittstellen der mops.dll ein Modell im Hauptspeicher generiert werden.

Es gibt eine Reihe von Möglichkeiten, wie ein Computermodell implementiert werden kann:

- Das Modell kann in AMPL formuliert werden zu dem eine MOPS-Schnittstelle existiert
- Ein Modellgenerator kann die Daten im MPS-Format generieren.
- Eine Variante ist das Triplet-Format. Es erlaubt lange Dateinamen, weist keine Datenredundanz auf (im Gegensatz zum mps-Format). Die langen Dateinamen erlauben alphanumerische Indizes, die z.B. durch Punkte getrennt werden können und damit die Ergebnisaufbereitung vereinfachen.

Client/Server-Systeme erlauben die Aufteilung der Prozesse Datenextraktion, Applikationslogik, Modellgenerierung, Modelloptimierung und Ergebnisvisualisierung. Die Modellgenerierung und Optimierung kann auch auf einem Server erfolgen.

Die Aufbereitung der Modellösung muß für den Planer als Ergebnis einen Entscheidungsvorschlag präsentieren. Hierbei ist die *Visualisierung der Modellergebnisse* zur Erhöhung der Benutzerfreundlichkeit vorteilhaft. Client-Server-Architekturen bieten vielfältige Möglichkeiten der Aufteilung von Datenmanagement, Präsentation, Applikationslogik, Modellgenerierung und Optimierung.

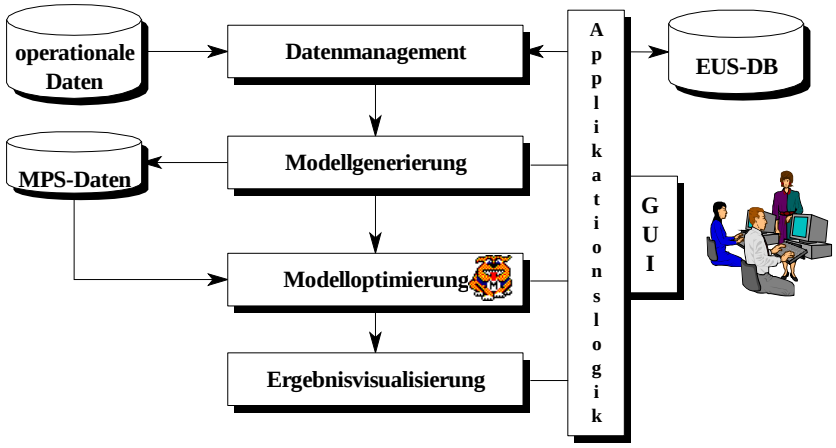


Abbildung 2: Einsatz mathematischer Optimierungssoftware in EUS

Entwurf und Implementierung von Anwendersystemen mit Mathematischer Optimierung sind aufwendig und erfordern besondere Erfahrungen. Daher ist der Einsatz von MP-Technologie in EUS nur dann sinnvoll, wenn wesentliche Vorteile realisierbar erscheinen.
Ständige Leistungssteigerungen bei Rechnern und Optimierungssoftware verbessern die Voraussetzungen für den praktischen Einsatz von MP-Software in betrieblichen Anwendungen. Vor allem bei gemischt-ganzzahliger Optimierung wurden in der letzten Dekade enorme Fortschritte bei den Algorithmen erzielt, die es erlauben, Modelle zu lösen, die noch vor wenigen Jahren auch auf den schnellsten Rechnern praktisch unlösbar waren. Besonders vorteilhaft ist der Einsatz von Hochleistungs-PCs die für interaktive Anwendungen prädestiniert sind.

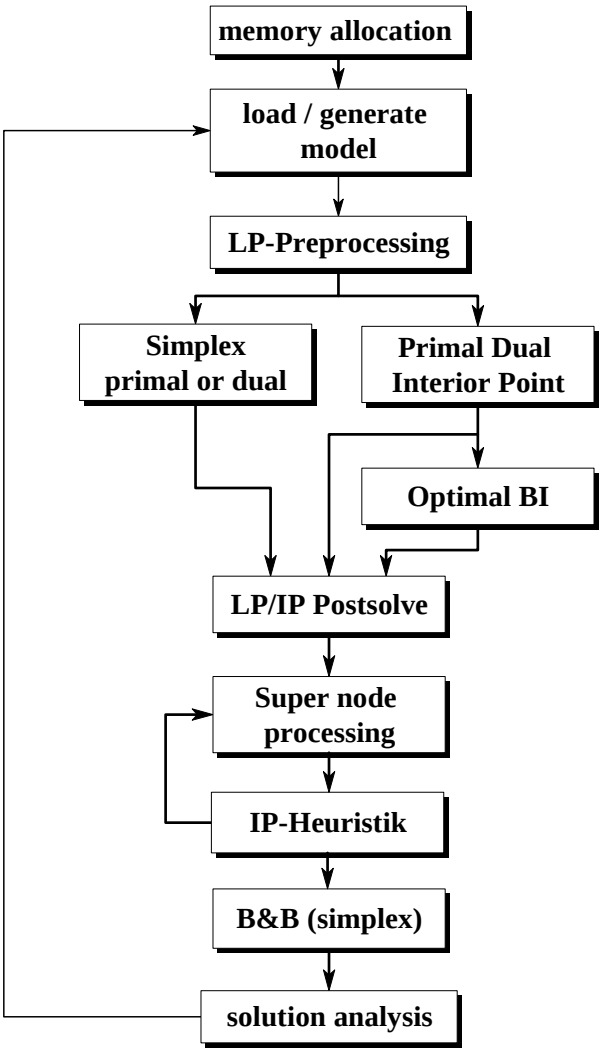


Abbildung 3: Ablaufschema einer LP / IP-Optimierung

2 Modelldaten

MOPS bietet fünf Eingabemöglichkeiten für Modelldaten:

- MPS-Format (externe Datei, 10. Anhang A)
- Triplet-Format (externe Datei, 11. Anhang B)
- internes Format direkt in MOPS-Arrays im Hauptspeicher gespeichert (Abschnitt 5.3)
- über IMR-Schnittstellenroutinen im Hauptspeicher (Abschnitt 5.7)
- über DLL-Routinen (Abschnitt 6)

Das Eingabeformat wird durch den Wert der Variablen *xinfor* bestimmt. Hat xinfor den Wert 1, dann werden die Daten im MPS-Format erwartet und die Routine xcnvrt aufgerufen. Andernfalls müssen die Modelldaten durch einen Modellgeneratot im internen Format oder durch Verwendung der Schnittstellenroutinen zur Verfügung gestellt werden. Ein Verständnis über die externe und interne Modell-Repräsentation ist bei der Benutzung von MOPS hilfreich. Ausgangspunkt ist die externe Modelldarstellung (EMR) von Kapitel 1.

3 MOPS-Dateien

MOPS benutzt bzw. erzeugt folgende Dateien:

MOPS-Dateien	
xfnmps	MPS Modelldaten bzw. Triplet-Modelldaten (Input)
xfnpro	MOPS-Profile (Input)
xfnlps	LP-Lösung (Output)
xfnips	IP-Lösung (Output)
xfnbai	Eingabebasis (Input)
xfnbao	Ausgabebasis (Output)
xfnsav	Interne Basis (Input / Output)
xfnsta	Statistiken (Output)
xfnlog	Iterationen-Log (Output)
xfnmsg	Fehlermeldungen bzw. Warnungen (Output)
xfntre	Bound-Baum (Input / Output)

Darüber hinaus werden noch einige interne Dateien erzeugt, die nach der Optimierung gelöscht werden. Alle Namen sind als Zeichenketten der Länge 64 definiert. Der Benutzer kann alle gültigen Namen für die Ein- und Ausgabedateien benutzen. ***Es darf allerdings kein Leerzeichen im Dateinamen enthalten sein***. Besteht ein Name nur aus Leerzeichen, so wird bei einer Ausgabe (und nur dann) vom Betriebssystem standardmäßig eine Datei FORunit-Nr bzw. unter VM/CMS :FTNrF001 erzeugt.

Bewährt hat sich folgende Namensgebung:

1. Jeder Dateiname besteht aus einem Namen und einer Dateierweiterung aus drei Buchstaben, die durch einen Punkt voneinander getrennt sind. Unter VM/CMS ist die Dateierweiterung durch den Filetyp definiert. Darüber hinaus können Dateideklarationen nur in der Datei MOPS EXEC vorgenommen werden (siehe Abschnitt 7.3)
2. als Dateierweiterungen (Dateityp, Filetype, Extension) wird verwandt:
   - .mps - MPS- oder Triplet- Datenbestand
   - .lps - LP-Lösung
   - .ips - IP-Lösung
   - .bas - Basis im MPS-Format
   - .sav - Basis im internen Format
   - .sta - Statistikdaten
   - .log - Iterationszeilen, wenn keine Bildschirmausgabe

.msg - Nachrichten und Fehler
.tre - Branch-and-Bound Baum

Es gibt unter DOS / Windows und Unix eine Routine xgenfn, die automatisch die oben definierten Dateierweiterungen erzeugt. Es gelten dabei folgende Regeln:

- Erweiterungen werden nur für Dateinamen erzeugt, denen der Benutzer keinen Dateinamen (Default-Wert: Leerzeichen) vergeben hat.
- *Der generierte Dateiname ergibt sich aus dem Namen für den MPS-Datenfile (XFNMPS), wobei der Pfadname nicht berücksichtigt wird. Die Erweiterung für xfnmps muß mps sein.*
- Die generierten Dateien werden im aktiven Verzeichnis gespeichert. Beispiel: Das aktive Verzeichnis sei MOPS. xfnmps möge den Dateinamen e:\user\probl\har2.mps enthalten. Im Normalfall werden die Dateien har2.lps, har2.ips, har2.sta, har2.msg, har2.bas und har2.sav im Unterverzeichnis MOPS generiert.
- Die Generierung der Dateien wird durch die Parameter xoutlv, xoutsl und xfrbas, die im Abschnitt 5.4 beschrieben werden, gesteuert.

4 Benutzung des MOPS Lademoduls (LP und IP)

Sieht man von ClipMOPS (nur Windows) ab, ist dies die einfachste Möglichkeit MOPS zu benutzen und bietet dennoch die volle Funktionalität der Optimierung. Die Eingabedaten müssen hier als MPS-Datei oder Triplet-Datei vorliegen. Dateinamen und Input-Parameter werden hierbei aus einer Profile-Datei **xmops.pro** eingelesen. Sie kann mit jedem Editor verändert werden. Ist ein Parameter oder Dateiname nicht im Profile gesetzt, bleibt der Default-Wert bestehen. Der Lademodul wird durch Eingabe von mops aufgerufen. Profile und mops.exe müssen sich im gleichen Verzeichnis befinden. Die Eingabedaten (mps oder triplet) können in einem beliebigen Verzeichnis gespeichert sein. Der volle Pfadnamen zu diesen daten muss in xfnmps spezifiziert werden.

4.1 Speicherplatzallokation

Hauptspeicherallokation und das Anlegen des internen Speicherplatzes erfolgen ab MOPS Version 5.0 automatisch. Im Normallfall braucht der MOPS Nutzer hierüber nichts zu wissen. Für sehr große Modelle kann ein Eingriff in das Speichermanagement erforderlich bzw. sinnvoll sein. Ein wichtiger Parameter ist ***xmreal*** (Integer*4), dessen Defaultwert 300 MB beträgt. Dieser Wert gibt an, wieviel ***virtueller Speicher*** maximal für alle Optimierungsdaten im Hauptspeicher zur Verfügung steht, d.h. die Maschine kann weniger Realspeicher als xmreal aufweisen. Dann muss jedoch der Swap-Speicher auf der Festplatte hinreichend groß sein. Xmreal kann gegenüber dem Default erhöht oder erniedrigt werden. Z.B. kann durch die Angabe xmreal = 1000 (z.B. im MOPS Profile) der Speicherplatz zur Lösung extrem großer Modelle erhöht werden.

In engem Zusammenhang mit xmreal stehen die Parameter **xmmax** (maximale Zeilenanzahl), **xnmax** (maximale Spaltenanzahl), **xnzmax** (maximale Anzahl von Nichtnullelementen), **xnenmx** (maximale Anzahl von Nichtnullelementen für die LU-Faktorisierung) und **xrcmax** (maximale Anzahl von Zeichen zur Speicherung von Namen). Auch diese Parameter können im Profile gesetzt werden. Die interne Aufteilung des durch xmreal angelegten Speicherplatzes erfolgt durch diese Parameter. Nach dem Einlesen von mps- bzw. Triplet-Daten kann kann die Speicherallokation an Hand der tatsächlichen Modellgröße vorgenommen werden. Dies erfolgt durch den Parameter xdimcn. Default für xdimcn = 1. In diesem Fall wird aus der tatsächlichen Größe xn, xm, xnzero xnmax, xmmax und xnzmax wie folgt bestimmt: xnmax = xn + xadcol, xmmax = xm + xadrow und xnzmax = xnzmax + xadnon. Damit soll sichergestellt werden, dass später z.B. in der IP-Optimierung zusätzliche Zeilen und Nichtnullelemente eingeführt werden können ohne die Daten im Hauptspeicher zu verschieben. Die

Parameter xadcol, xadrow und xadnon können ebenfalls im Profile gesetzt werden. Folgende Parameter können im Profile die Speicherallokation verändern:

- **Einstellungen zur Lösung von LP und IP-Modellen**

  xmreal: Größe des virtuellen Hauptspeichers für alle MOPS-Arrays in Megabyte
  xnmax: Maximalzahl der Strukturvariablen
  xmmax : Maximalzahl der Restriktionen
  xnzmax: Maximalzahl der Nichtnullelemente in der Koeffizientenmatrix **ohne** die Nichtnullelemente der rechten Seite
  xnenmx: Maximalzahl der Nichtnullelemente in der LU-Faktorisierung; Standardeinstellung ist xnenmx = xnzmax
  xrcmax: Maximalzahl der Zeichen, die insgesamt zur Speicherung der Zeilen und Spaltennamen vorgesehen sind.

- **Zusätzliche Einstellungen zur Lösung von IP-Modellen**

  xmncli: Maximalzahl der Cliques und Implikationen
  xmnzcl: Maximalzahl der möglichen Einträge in der Clique- und Implikationentabelle
  xmaxno: Maximalzahl der Knoten in der Hauptspeicherknotentabelle

4.2 Der MOPS Profile xmops.pro

- jede Zeile (Datensatz) des Profiles, die in Spalte 1 einen *, ein C oder c aufweist, wird vollständig als Kommentarzeile betrachtet.
- nach einem **!** wird der Rest einer Zeile als Kommentar betrachtet.
- alle Buchstaben - außer denen in Zeichenketten - werden in Großbuchstaben umgewandelt.
- Wertzuweisungen von MOPS-Variablen erfolgen durch Gleichheitszeichen. Zwischen Variablenname und Gleichheitszeichen kann, muß aber kein Leerzeichen stehen.

Beispiel-Profile xmops.pro für LP und IP-Optimierung

```
xfnmps = ‘opti\probl\mops.mps’
xminmx = 'max'   !  optimization direction min or max  (default: 'min')
xstart= 1!  start from crash basis
Xfnsta = 'statist'   ! file name for statistic file
xmreal = 100 xnzmax = 50000 xadnon = 40000  xstorn = 0
```

4.3 Benutzer-Inputparameter

Dieser Begriff bezeichnet Toleranzen, Dateinamen und Parameter zur Strategienauswahl, die verändert werden können. Bis auf den Dateinamen für die MPS-Daten weisen alle Parameter Standard-Einstellungen auf. Sie können vor dem Aufruf von MOPS-Routinen verändert werden, entweder im Profile oder im Anwenderprogramm (⇒ 6.).

4.4 Lösung von LP-Modellen

MOPS ist so konzipiert, daß praktisch alle LP-Modelle mit den Standardeinstellungen gelöst werden. Dies beinhaltet in folgender Reihenfolge den Aufruf folgender Module: Convert, LP-Preprocessing, Skalierung, Crash, Primaler Simplex, Lösungsoutput-Prozedur.

Im Profile xmops.pro muß lediglich in xfnmps der volle Pfadname der MPS-Eingabedatei spezifiert werden, wenn die Eingabedaten nicht im MOPS Verzeichnis gespeichert sind, z.B. xfnmps = 'e:\probl\sample.mps'. Standardmäßig wird minimiert. Die Variable xminmx muß bei Maximierung auf 'max' gesetzt werden. *Bis auf Zeichenketten werden alle Buchstaben des Profiles in Großbuchstaben umgewandelt.* Nach der Optimierung wird die LP-Lösung - sofern der Dateiname xfnlps nicht redefiniert wurde - in das lokale Verzeichnis, von dem aus die Optimierung gestartet wurde, unter dem Dateinamen (ohne Pfadangabe) von xfnmps mit der Erweiterung lps gespeichert. Für das o.g. Beispiel also unter sample.lps. In der Datei xfnmps.sta befindet sich eine Statistik über den Optimierungsprozeß.

Im folgenden werden die wichtigsten Variablen für die LP-Optimierung mit einer Kurzbezeichnung vorgestellt. Eine ausführliche Beschreibung findet sich im Kapitel 8.

Name	Kurzbezeichnung
xautom	automatische Anpassung der Pricing-Strategie
xbatyp	Typ der Basisdatei
xbound	Name des Boundvektors bei der MPS-Dateneingabe
xbndha	Schrankenstrategie des LP-Preprozessors
xdata	Name des Modells bei der MPS-Dateneingabe
xfnbai	Dateiname einer Insert Basis (Inputdatei)
xfnbao	Dateiname einer Punch Basis (Outputdatei)
xfnlps	Dateiname einer optimalen LP-Lösung
xfnmps	Dateiname von Eingabedaten im MPS-Format
xfnmsg	Dateiname der Fehlerdatei
xfnpro	Dateiname des Profiles
xfnsav	Dateiname einer Basis im MOPS-internen Format
xfnsta	Dateiname der erzeugten Statistikdatei
xdjscl	Devex-Strategie
xfrbas	Häufigkeit mit der Basen gesichert werden
xinfor	bestimmt das Datenformat der Eingabedaten
xlptyp	bestimmt den Algorithmus zur Lösung des Anfangs-LPs
xmaxel	größter erlaubter Modellkoeffizient
xminmx	Min- oder Maximierung
xmiter	Maximale Anzahl der LP-Iterationen
xobj	Name der Zielfunktion bei der MPS-Dateneingabe
xoutlv	bestimmt das Outputlevel der Datenausgabe
xoutsl	bestimmt die Erstellung von Lösungsdateien
xrange	Name des Rangevektors bei der MPS-Dateneingabe
xrduce	bestimmt, ob LP-Preprozessing erfolgen soll
xrhs	Name des Rhs-Vektors bei der MPS-Dateneingabe
xscale	bestimmt die Skalierung von Modellen
xstart	bestimmt die Auswahl der Startbasis

4.5 LP-Output-Dateien

MOPS legt während oder nach der Optimierung standardmäßig folgende Dateien an: Statistikdatei, LP-Lösungsdatei, LP-Basisdatei und die Datei für die Aufzeichnung einzelner LP-Iterationen, bei IP-Modellen auch die IP-Lösungsdatei und ggfs. einen Branch-and-Bound-Baum. Die einzelnen Dateien werden im folgenden beschrieben.

4.5.1 Statistikdatei

Die von MOPS erzeugte Statistikdatei zeichnet Informationen über die Phasen Convert, LP-Preprocessing, Crash und Optimierung auf. Die meisten Angaben erklären sich selbst. Für zukünftige Optimierungsläufe mit dem gleichen Modell kann die *LP-Optimization Statistic* wichtig sein, die angelegt wird, wenn der Input-Parameter xoutlv > 0 spezifiziert wurde. Die Angaben lost feasibility, singular bases und reinversion (accuracy) sagen etwas über die numerische Stabilität während des Simplex oder des Crossovers aus. Wenn die entsprechenden Werte signifikant sind (z.B. größer als 5), dann sollte ggfs. die Modellformulierung, Lösungsstrategien oder die Toleranzen überprüft werden. Wurde das Modell ohne Skalierung (xscale = 0) oder nur mit zeilenweiser Skalierung (xscale = 1) gelöst, dann sollte die Optimierung mit xscale = 2 wiederholt werden.

Die Angaben reinversion (space), row / column compressions geben Aussagen darüber, inwieweit der vorhandene Speicherplatz für die LU-Faktorisierung ausreichend bemessen war. Sind die Werte signifikant größer als Null, dann kann durch Vergrößerung des Wertes xmxnen in der Include-Datei xlpdim.inc i.d.R. eine Verringerung der Laufzeit erreicht werden.

4.5.2 Lösungsdatei (MPS-Format)

Die LP-Lösung wird durch Aufruf der Prozedur xsolut in die Datei xfnlps geschrieben, wenn im Profile xoutsl > 0 gesetzt wurde. Bei einer IP-Optimierung werden auch IP-Lösungen in die Datei xfnips geschrieben. Die MPS-Lösungsausgabe kann in drei Sektionen unterteilt werden:

- **Identifikationsteil:** Hier wird das gelöste Modell identifiziert, dessen Status (optimal, unzulässig, unbeschränkt) und ggfs. dessen optimaler Zielfunktionswert.
- **Row-Section:** Für jede Restriktion werden folgende Informationen ausgegeben:
  - **Number:** interner Zeilenindex (Indizierung xn + 1, ... xn + xm)
  - **Status:** BS Variable befindet sich in der Basis
    UL Variable befindet sich an der oberen Schranke
    LL Variable befindet sich an der unteren Schranke
    EQ Variable ist fixiert, d.h. UL = LL
    IN Restriktion ist inaktiviert.
  - **Row:** Zeilenname. Ist der Zeilenname länger als acht Zeichen, so werden nur die ersten acht Zeichen ausgegeben.
  - **Activity:** Wert der Restriktion bei Substitution der Lösung
  - **Slack:** Differenz zwischen dem Wert der Restriktion und dem beschränkenden Wert der rechten Seite.
  - **Lower limit:** untere Schranke für den Wert der Restriktion.
  - **Upper limit:** obere Schranke für den Wert der Restriktion.
  - **Dual activity:** Dualwert der Restriktion in einer optimalen LP-Lösung. Er gibt die marginale Veränderung in der Zielfunktion an, wenn die rechte Seite b_i um eine Einheit verändert wird. Dies ist eine Grenzwertbetrachtung, bei der kein Basiswechsel vorkommen und keine duale Degeneriertheit vorliegen darf.
- **Column- Section:** Für jede Strukturvariable werden folgende Informationen ausgegeben:
  - **Number:** interner Spaltenindex (Indizierung: 1, ..., xn)
  - **Status:** wie oben, während der IP-Optimierung erhalten ganzzahlige Variablen den Status IV (Integer Variable)
  - **Columns:** Spaltenname. Ist der Spaltenname länger als acht Zeichen, so werden nur die ersten acht Zeichen ausgegeben.
  - **Activity:** Wert der Strukturvariablen in der Lösung
  - **Input cost:** Zielfunktionskoeffizient.
  - **Lower limit:** untere Schranke für die Strukturvariable.
  - **Upper limit:** ober Schranke für die Strukturvariable.
  - **Reduced cost:** Grenzkosten für die Strukturvariable, d.h. um wieviel der Zielfunktionskoeffizient der Variablen reduziert bzw. vergrößert werden muß, damit die Variable in die Basis kommt.

Anmerkung: Bei Einsatz des LP-Preprozessors (xrduce = 1) können Schranken von Strukturvariablen verändert werden. Die LP-Lösung wird dadurch nicht verändert. Durch xbndha = 0 wird bei eingeschränktem LP-Preprocessing keine Schrankenveränderung vorgenommen.

4.5.3 Basisdateien (nur Simplex)

MOPS startet die Optimierung in diesem Fall von einer Ausgangsbasis. Die Wahl dieser Basis hat einen großen Einfluß auf die Lösungszeit und zwar sowohl durch die Anzahl der Sim-

plexiterationen als auch durch die Sparsity der Basis und ihrer LU-Zerlegung. Die unterschiedlichen Strategien bei der Handhabung von Startbasen können vom Benutzer durch die Variable xstart gesteuert werden . MOPS bietet drei Möglichkeiten zur Auswahl der Startbasis:

1. Die Startbasis besteht aus allen logischen Variablen, d.h. der mxm Einheitsmatrix. Sie wird auch als *all logical Basis* bezeichnet. Im allgemeinen ist diese Basis keine gute Wahl.
2. Eine Heuristik, implementiert in der Routine *xcrash,* versucht solche Strukturvariablen in die Basis aufzunehmen, die viele Freiheitsgrade aufweisen, dünn besetzt sind und *Artificials* (Schlupfvariablen zu Gleichungen) aus der logischen Basis eliminieren. Meistens wird durch das Crash-Verfahren die Lösungszeit, verglichen mit Methode 1, erheblich verkürzt. Es ist daher das Default-Verfahren.
3. Es wird eine Start-Basis von einer vorangegangenen Optimierung, entweder vom gleichen oder einem anderen Modell verwandt. Diese Basis kann entweder im MOPS-internen Format oder im MPSX-kompatiblen Format vorliegen. Das interne Format erfordert weniger Speicherplatz und ist auch schneller zu verarbeiten. Dafür darf das Modell **strukturell nicht verändert werden**. Bei ähnlichen Modellen, bei denen die Anzahl von Restriktionen und Variablen nicht gleich sind, muß das MPS Basisformat verwandt werden.

Da im MPS-Format Namen positionsgebunden definiert werden, sind Leerzeichen in den Namen signifikant. Sind Restriktionen bzw. Variablennamen kürzer als acht Zeichen, so ist eine Punch-Basis nicht kompatibel zum internen Format.

xstart
0 all logical basis
1 Crash
2 Einlesen einer Basis im MPS-Format (xfnbai)
3 Einlesen einer Basis im MOPS-Format (xfnsav)

Das **Erzeugen** von Basisdateien wird durch die Variablen xbatyp und xfrbas gesteuert. xbatyp spezifiziert, in welchem Format die Basis gelesen oder geschrieben wird :

xbatyp
1 erzeugt die Datei xfnsav im MOPS-Format
2 erzeugt die Datei xfnbao im MPS-Format
3 erzeugt zwei Dateien, eine im MPS- und die andere im MOPS-Format

xfrbas bestimmt die Iterationsfrequenz, nach der die aktuelle Basis gespeichert wird :

xfrbas
-1 die Basis wird nie gespeichert
0 die Basis wird bei Terminierung gespeichert
p die Basis wird nach jeweils p Iterationen und bei Terminierung gespeichert

Anmerkung: Im MOPS-Format muß die Anzahl der Restriktionen und Variablen von Basis und Modell übereinstimmen, während dies im MPS-Format nicht der Fall zu sein braucht. Bei größeren Modelländerungen muß überprüft werden, ob nicht besser von einer Crash-Basis gestartet werden soll.

4.5.4 Log-Datei

MOPS bietet die Möglichkeit den Optimierungsprozeß mittels sogenannter Log-Meldungen zu protokollieren. Standardmäßig werden die Meldungen auf dem Bildschirm ausgegeben. Die dazugehörende Unitnummer für die Bildschirmausgabe ist maschinenabhängig, für die

meisten Fortran-Compiler ist es die Sechs. Dieser Wert wird in der Datei *xlpdim.inc* der Variablen xterun zugewiesen. Diese Variable weist den gleichen Wert auf, wie die Variable xunlog, die in xlpinit.inc definiert wird.

Anhand der Log-Meldungen kann man überprüfen, in welcher Phase sich die Optimierung befindet oder ob die gewünschten Optionen auch gesetzt und ausgeführt werden, wie z.B. LP-Preprozessing, Skalierung, Einlesen einer Startbasis, Speichern von Basen, usw.. Die Steuerparameter für die Log-Datei werden im Profile gesetzt. Welche Meldungen ausgegeben werden, steuert die Variable xoutlv, welche zugleich auch das Schreiben der Statistikdatei steuert. Folgende Optionen sind möglich :

xoutlv
0 keinerlei Ausgaben
1 keine Log-Meldungen, nur Statistiken bei Terminierung
2 Log-Meldungen nach jeder LU-Faktorisierung, bei Terminierung Statistiken
3 es werden zusätzliche Informationen nach xfrlog Iterationen geschrieben.

Die Option xoutlv = 0 ist z.B. dann von Interesse, wenn ein Anwender seine eigenen Ergebnisroutinen aufrufen will oder nur die Lösungsdatei benötigt wird.

Achtung : Die Option xoutlv = 3 dient ausschließlich der Analyse von Fehlersituationen, da sie die iterative Ausgabe der Informationen die CPU-zeit signifikant erhöht.

4.6 Leistungstuning bei der LP-Optimierung

Werden LP-Modelle gleicher Struktur, jedoch mit geänderten Daten häufig gelöst oder gibt es unbefriedigende Laufzeiten, dann sollte mit den verschiedenen in MOPS vorhandenen Lösungsstrategien experimentiert werden. Beim Einsatz des Simplexverfahrens dient als Maßstab vor allem *die Anzahl der benötigten LP-Iterationen in Relation zur Anzahl der Restriktionen des Modells*. Im Normalfall beträgt die Anzahl der LP-Iterationen weniger als das Zwei- bis Vierfache der Anzahl der Restriktionen. Einige Modelle benötigen u.U. weniger LP-Iterationen als die Anzahl der Restriktionen im Modell. Ist der Normalfall nicht gegeben, dann lohnt es sich nach Verbesserungen zu suchen. Modelle, die dem Normalfall nicht entsprechen, weisen häufig eine oder mehrere der folgenden Eigenschaften aus:

- mehrperiodische Modelle mit Treppenstruktur
- hohe strukturell bedingte primale bzw. duale Degeneration
- signifikant höhere Dichte der Koeffizientenmatrix oder Matrixteile die sehr dicht sind
- extremer Fill während der LU-Faktorisierung und des LU-Updates

Zu beobachten ist, daß dichte Modelle und solche mit viel Fill in der LU-Faktorisierung neben langsamerer Iterationsgeschwindigkeit auch mehr Iterationen benötigen. Folgende Module sind für Verbesserungen relevant:

- **LP-Preprocessing**: der LP-Preprocessor wird aufgerufen, wenn xrduce ≥ 0 gesetzt ist (Default = 2). Durch das Entfernen trivialer Modellteile kann die die Laufzeit i.d.R. drastisch verkürzt werden. Zu beachten ist, daß über ein „Postsolve“ eine vollständige primale und duale Lösung bestimmt wird.
- **Skalierung des Modells**: standardmäßig wird eine zeilen- und spaltenweise Skalierung des Modells (Matrix, Schranken, Zielfunktion) vorgenommen und ist auch unbedingt zu empfehlen. xscale = 2 (Default) gibt i.a. die besten Resultate, auch bei der Lösung von IP-Modellen. Der Skalierungsalgorithmus ist in [12] beschrieben. xscale = 1 bietet eine vereinfachte Skalierung, bei der jede Zeile der Koeffizientenmatrix A durch das betragsmäßig größte Element dividiert wird. Diese Skalierung hat den Vorteil, daß die Werte der Strukturvariablen von der Skalierung nicht berührt werden. In seltenen Fällen, bei denen das

Modell bereits optimal skaliert formuliert wurde, z.B. Set-Partitioning-Modellen, ist von einer MOPS-Skalierung abzuraten, indem xscale = 0 gesetzt wird. Es gibt allerdings LP bzw. IP-Modelle, die ohne xscale = 2 nicht lösbar sind.

- **Auswahl des Algorithmus zur Lösung des Anfangs-LPs**. Diese Auswahl wird über den Parmeter xlptyp getroffen (default: 0). Standardmäßig wird ein primaler Simplexcode benutzt. Optional kann auch ein dualer Simplexcode (xlptyp = 2) oder ein innere Punkte Verfahren (xlptyp = 4) eingesetzt werden. In den meisten Fällen – zumindestens bei sehr großen LP-Modellen wird das LP-Modell mit IPM schneller gelöst (xlptyp = 4). Muß eine Basis-Lösung (BI, Basis Identifikation) bestimmt werden (z.B. ist dies für IP-Modelle sinnvoll), dann kann dies in ungünstigen Fällen mehrfach so lange dauern wie die reine Lösungszeit mit dem IPM. Deswegen kann manchmal letztlich doch die Simplex-Engine (xlptyp = 0, 2) schneller sein. Zu berücksichtigen ist, daß duale Simplexiterationen prinzipbedingt ca. langsamer sind. Der Einsatz des dualen Simplexcode zur Lösung des Anfangs-LPs ist ist aber bei einer Reihe von Modellen am schnellsten.
- **Startbasis (nur Simplex)**: Bei Vorliegen einer Startbasis eines ähnlichen Modells kann die Optimierungszeit u.U. drastisch reduziert werden. Dazu muß xstart = 3 spezifiziert werden und in xfnbai der Name der Datei angegeben werden, die die Basis enthät. Diese Angabe ist nur dann erforderlich, wenn der Dateiname nicht automatisch generiert wurde.

  Ist keine Startbasis vorhanden, dann kann mit der Angabe xstart = 1 eine Heuristik (Crash) aufgerufen werden, die eine nichttriviale Startbasis erzeugt. Es gibt zwei Varianten, die unterschiedliche Bewertungsfunktionen für die Strukturvariablen enthalten. Sie werden durch den Parameter xcratp (Crash type), der den Wert 0 bzw. 1 haben darf, gesteuert. Default ist xstart = 1 und xcratp = 0.
- **Auswahl der Pivotzeile in Phase 1 (nur Simplex)**: Diese Auswahl wird durch den Parameter xchztp (Chuzr type) definiert, der 0 oder 1 sein kann (default: 1). Ist xchztp = 0 gesetzt, dann erfüllt die Variable, die in die Basis aufgenommen wird, folgende Bedingungen: alle bereits zulässigen Basisvariablen behalten ihre Zulässigkeit nach einem Basiswechsel. Bei der Bestimmung der Variablen, die die Basis verläßt, werden degenerierte und fixierte Variablen bevorzugt ausgewählt. Bei Gleichheit wird die Zeile mit dem betragsmäßig größten Pivotelement gewählt.

  Ist xchztp = 1, dann wird ein anderes Verfahren zur Bestimmung der Pivotzeile gewählt: Die Summe der Unzulässigkeiten aller Basisvariablen als Funktion der Schrittweite ist eine konvexe, stückweise lineare Funktion mit endlich vielen Stützpunkten. Es wird diejenige Variable gewählt, die die Summe der Unzulässigkeiten minimiert [12], dabei dürfen zulässige Basisvariablen auch unzulässig werden. Default ist xchztp = 1. Es gibt jedoch Modelle, bei denen xchztp = 0 bessere Resultate liefert.
- **Auswahl der Pivotspalte (nur Simplex)**: wird durch den Wert der Variablen xdjscl (Default xdjscl = 1) bestimmt. Ist xdjscl 0 oder 1, so wird sogenanntes partial pricing durchgeführt. Dabei wird eine Liste von ca. 20 Variablen mit attraktiven reduzierten Kosten fortgeschrieben, die aus fortlaufenden Segmenten der Matrix ermittelt wird. Ist xdjscl = 1, so wird modified Devex [12] benutzt, d.h. die reduzierten Kosten einer Variablen werden skaliert.

  Ist xdjscl = 2, so wird die Devex-Strategie benutzt, bei der in jeder Iteration skalierte reduzierte Kosten aller Variablen bestimmt werden. Volles Devex ist sehr aufwendig und nur für solche Modelle empfehlenswert, bei denen eine drastische Reduktion der Iterationszahl den Aufwand überkompensiert. In diesem Zusammenhang sind weitere Parameter erwähnenswert: Die Variable xautom hat standardmäßig den Wert 1. Nach jeweils mehr als xfrchk (Default 300) Iterationen wird in Phase 2 überprüft, ob die Zielfunktion sich hinreichend verbessert. Ist dies nicht der Fall so wird Devex dann aktiviert, wenn der Quotient

aus der Anzahl Strukturvariablen und der Anzahl Restriktionen kleiner gleich xxndxm (Default 3) und der Fill während der letzten LU-Faktorisierung mindestens xminfi (Default 0.2, d.h. 20%) betrug. Ist die automatische Steuerung nicht erwünscht, dann kann xautom = 0 gesetzt werden.

- **LU-Faktorisierung und Update (Simplex, BI)** werden durch folgende Parameter beeinflußt: xpivlu ist die Threshold Pivot Tolerance (Default 0.1), die im offenen Intervall]0, 1[liegen muß. Je größer xpivlu, desto größer ist die numerische Stabilität und der Fill, d.h. die Erzeugung neuer Nichtnullelemente in der Faktorisierung. Xdropf ist eine relative Tolernz, die tendenziell angibt, ab welchem Wert ein Element in der LU-Faktorisierung Null gesetzt wird. Der Wert der Variablen xmxnen bestimmt, wieviel Speicherplatz für die Etavektoren der LU-Faktorisierung zur Verfügung gestellt werden. Ist dieser Wert zu klein, so kann u.U. ein Abbruch erfolgen oder die Leistung des Optimierers durch zu häufige Kompressionen reduziert werden. Die Anzahl der Zeilen- und Spaltenkompressionen wird im Statistikfile nach der Optimierung angegeben. Die Werte sind im Idealfall Null. Werte unter 50 sind noch akzeptabel. Viel größere Werte können durch Erhöhen von xmxnen in xlpdim.inc und Compilation von xlp.f bzw. xip.f mit anschließendem Linken vermieden werden.

4.7 Lösung von IP-Modellen

Fast jedes quantifizierbare deterministische Entscheidungsproblem läßt sich als IP-Modell formulieren. Die wichtigste Modellklasse sind gemischte 0-1-Modelle mit linearer Zielfunktion. Solche Modelle entstehen *auch* durch Transformation nichtlinearer Optimierungsprobleme in MIP-Modelle. Dies kann dann sinnvoll sein, wenn das Modell eine schwache Nichtlinearität und viele lineare Nebenbedingungen aufweist.

Obwohl IP-Modelle eine große Ähnlichkeit zu LP-Modellen aufweisen, handelt es sich um eine grundsätzlich andere mathematische Problemklasse. IP Modelle ohne besondere Struktur gehören in die Klasse der NP-harten Probleme. *Dies bedeutet, daß alle bekannten Algorithmen im schlechtest möglichen Fall IP-Modelle nur in exponentieller Laufzeit als Funktion der Anzahl ganzzahliger Variablen lösen können*. Viele praktische IP-Modelle können dennoch optimal oder nahezu optimal gelöst werden. Generell schwanken jedoch die Lösungszeiten mit IP-Software sehr stark. Die Größe des Modells ist selten ein Indikator. Es gibt Anwendungen mit tausenden von 0-1-Variablen, die routinemäßig (nahezu) optimal gelöst werden. Andererseits gibt es wichtige industrielle Anwendungen (i.d.R. mit Reihenfolgebedingungen), die nicht in akzeptabler Zeit gelöst werden können.

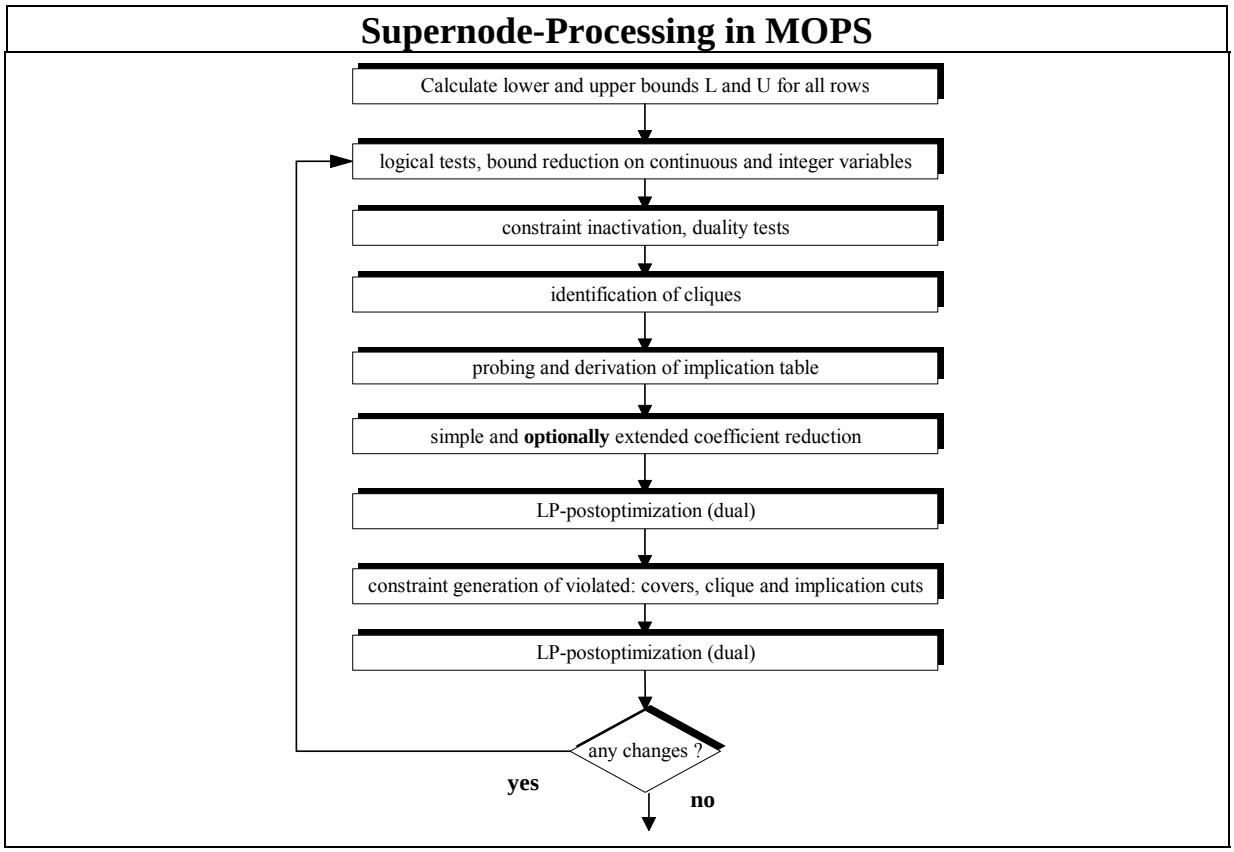
Alle erfolgreichen Algorithmen zur Lösung von IP-Modellen basieren auf dem Branch-and-Bound Prinzip bzw. dem Branch-and-Cut-Prinzip. Bei dem in MOPS implementierten Branch-and-Bound-Algorithmus wird an jedem Knoten die LP-Relaxation gelöst. Ein Knoten wird nur dann eliminiert wenn:

- das zugehörige LP-Problem unzulässig ist
- das zugehörige LP-Problem eine ganzzahlige Lösung aufweist
- der LP-Zielfunktionswert schlechter ist als der Wert der bisher besten gefundenen Lösung.

Daher ist die Qualität der LP-Relaxation unter dem Aspekt der Elimination eines Knotens von zentraler Bedeutung. Ein guter Anhaltspunkt hierfür ist der prozentuale Abstand zwischen dem Zielfunktionswert des Anfangs-LP's vom Wert einer optimalen IP-Lösung. Maßgebend ist hier sowohl die Modellformulierung als auch die Fähigkeit des IP-Optimizers durch Supernode-Processing die LP-Relaxation zu verbessern. Ist der prozentuale Abstand zwischen LP und IP-Optimalwert größer als wenige Prozent, dann ist die Suche nach einer optimalen IP-Lösung, bzw. der Beweis der Optimalität, i.d.R. sehr schwierig.

Die Modellierung hat bei IP-Modellen für die Lösbarkeit eine sehr große Bedeutung. Es gibt Beispiele für anfangs nicht lösbare Modelle, die nach Reformulierung in wenigen Minuten optimal lösbar waren. Ganz allgemein ist es bei IP-Modellen vorteilhaft, die Restriktionen so eng wie möglich zu formulieren, um den Lösungsraum klein zu halten.

Algorithmische Fortschritte der letzten Dekade basieren alle auf dem Konzept der *strengen LP-Relaxation*. Dabei wird durch die Modellformulierung oder das IP-Preprocessing erreicht, daß die LP-Relaxation sich so weit wie möglich der konvexen Hülle der ganzzahligen Gitterpunkte annähert. Insbesondere wird dadurch der Abstand zwischen dem optimalen LP-Wert und der IP-Lösung reduziert. Die Konsequenz ist, daß mit sehr viel höherer Wahrscheinlichkeit die LP-Lösung zur Elimination eines Knotens führt. Das Supernode-Processing in MOPS besteht aus einer Reihe komplexer Algorithmen, die in [13] genauer beschrieben sind. Hier soll nur der Ablauf der einzelnen Module, sowie die Parametereinstellungen beschrieben werden.



Standardeinstellung in MOPS ist volles Supernode-Processing. In seltenen Fällen kann ein reduziertes Supernode- Processing bessere Ergebnisse liefern.

4.8 Profile-Einstellungen bei der IP-Optimierung

Enthält ein Modell 0-1-Variablen, dann wird nach dem Lösen des LPs standardmäßig das zugehörige IP gelöst. Soll die IP-Lösung unterdrückt werden, d.h. nur das zugehörige LP gelöst werden, dann muß die Variable xlpmip den Wert 0 erhalten. Ein minimaler Profile zur Lösung von IP-Modellen erfordert also nur einen Eintrag für den Namen der MPS-Datei, z.B. xfnmps = ‘d:\mops20\models\har2.mps’. Soll das Modell maximiert werden, dann muß zusätzlich xminmx = ‘max’ gesetzt werden.

Im folgenden werden die wichtigsten Variablen für die IP-Optimierung mit einer Kurzbezeichnung vorgestellt. Eine ausführliche Beschreibung findet sich im Kapitel 9.

Name	Kurzbezeichnung
xfnips	Dateiname für IP-Lösungen
xlpmip	legt fest, ob ein IP-Modell nur als LP gelöst wird
xbrheu	Bestimmt das Auswahlverfahren für eine Branchingvariable
xipbnd	definiert eine untere / obere Schranke für den Wert einer IP-Lösung
xlpgap	definiert xipbnd über einen Prozentwert vom LP-Wert
xtolin	Toleranz für die Ganzzahligkeit einer IP-Lösung
xtolqi	Toleranz für die Quasi-Ganzzahligkeit einer IP-Lösung
xrimpr	gewünschte Verbesserung für den Wert der nächsten IP-Lösung
xninso	Zahl der zu speichernden IP-Lösungen (eine oder mehrere)
xprlev	wenn positiv, wird ein Probing der 0-1-Variablen durchgeführt
xiplpt	bestimmt mit welchem Algorithmus LPs an einem Knoten reoptimiert werden
xcored	wenn positiv, wird eine Koeffizientenreduktion durchgeführt
xlotst	bestimmt die Ausführung von logischen Tests an einem Knoten
ximpli	bestimmt Ableitung und Einsatz von Implikationen
xclict	bestimmt Ableitung und Einsatz von Cliques
ximbnd	bestimmt Ableitung und Einsatz von konditionellen Schranken
xnodse	bestimmt die Knotenauswahlregel im Branch-and-Bound-Prozeß
xcovct	bestimmt Ableitung und Einsatz von Cover-Schnittebenen
xmxdsk	bestimmt den max. erlaubten Speicherbedarf der Knotentabelle auf der Festplatte
xmxnod	Maximalzahl erlaubter Knoten für den Branch-and-Bound-Prozeß
xmxmin	Maximale CPU-Zeit in Minuten für den Branch-and-Bound-Prozeß
xmnheu	Maximalzahl von Knoten in der Heuristik, wenn xheutp > 0
xheutp	bestimmt, ob und mit welcher Strategie die Heuristik benutzt werden soll

Im Gegensatz zur LP-Optimierung läßt sich die Lösungszeit für IP-Modelle nicht prognostizieren. Durch die Variablen xmxnod, xmxmin und xmxdsk läßt sich die maximal erlaubte Knotenzahl, CPU-Zeit in Minuten bzw. der Festplattenspeicherplatz in Megabytes begrenzen. Wird eine der Grenzen erreicht, dann wird die Optimierung vorzeitig abgebrochen und der Branch-and-Bound-Baum unter dem Namen xfnmps.tre gespeichert, wenn xfrtre > 0 gesetzt wurde. Die Optimierung kann dann mit folgender Profile-Einstellung ggf. beliebig oft fortgesetzt weren, sofern die Branch-and-Bound-Suche nicht vorzeitig beendet wird. Dabei müssen jedoch alle Supernode-Processing-Strategien unverändert bleiben. Es dürfen nur folgende Parameter geändert werden: xbrheu, xmxmin, xmxnod.

Profile-Einstellung für eine Fortsetzung der B&B-Suche

```
xfnmps = \u\opti\probl\mops.mps
xminmx = 'max'   !  optimization direction min or max  (default: 'min')
xstart= 3! start from optimal save basis
xmxmin = 600.    xmxnod = 100000    ! limit search to either 600 minutes or 100000 nodes
```

IP-Output-Dateien

MOPS legt während oder nach der IP-Optimierung zusätzlich zu den LP-Output-Dateien standardmäßig eine IP-Lösungsdatei und, falls die IP-Optimierung aus Zeitgründen vorzeitig beendet wird, einen Branch-and-Bound-Baum an, von dem die IP-Optimierung fortgesetzt werden kann. Die Statistikdatei enthält zusätzliche Informationen über das Supernode-Processing und den Verlauf der IP-Optimierung.

4.10 Leistungstuning bei der IP-Optimierung

Das Laufzeitverhalten wird stark von Branching und Knotenauswahlregeln beeinflußt. Unbefriedigende Laufzeiten sind ein großes Problem bei der IP-Optimierung. Im folgenden werden

einige Möglichkeiten aufgezeigt, die einen entscheidenden Einfluß auf die Lösungszeit haben können:

- Den wichtigsten Einfluß hat die Modellformulierung. Mit ihr muß bereits im Vorfeld einer möglichen Anwendung experimentiert werden. Kann eine optimale IP-Lösung ermittelt werden, dann gibt der relative Abstand zwischen optimalem LP und IP-Optimum einen Hinweis auf die Güte der Modellformulierung. Ein relativer Abstand von wenigen Prozent ist meistens ausreichend für eine schnelle Lösung. Eine empfehlenswerte Literaturquelle für IP-Modellierung ist [19].

  Bei der IP-Modellierung empfiehlt sich zunächst eine logisch korrekte Modellierung einer gegebenen Problemstellung mit Hilfe von Standardtechniken, bei der nicht unbedingt auf eine effiziente Formulierung geachtet wird. Erst danach sollte eine Überarbeitung des IP-Modells unter dem Aspekt der effizienten Lösbarkeit erfolgen. Dabei sollte der Effekt des Supernode-Processing in der Statistikdatei studiert werden.

- Während des Branch-and-Bound-Algorithmus werden standardmäßig logische Tests vom Grad 1 nach der Auswahl eines Knotens und dem Setzen einer Branching-Variablen ausgeführt. Bei den meisten IP-Modellen ist diese Strategie optimal. Eine ähnliche Aussage gilt auch für die Bestimmung von konditionellen Schranken (ximbnd = 2). Hier kann ggf. durch ximbnd = 1 eine Laufzeitreduktion erfolgen (Default ximbnd = 2).

- Die verwendete Knotenauswahlregel hat ebenfalls einen wesentlichen Einfluß auf die Anzahl der entwickelten Knoten. MOPS erlaubt folgende Knotenauswahlregeln, die durch den Wert der Variablen xnodse definiert werden (Default: 3):

  0. Last In First Out (LIFO): der zuletzt bearbeitete Knoten wird zur Auswahl herangezogen; die LIFO-Regel hat folgende Merkmale:
     - einzige Regel, die keine Neubestimmung der LU-Faktorisierung am Knoten k erfordert, daher ist der Overhead für einen Node-Restore minimal
     - bei schwierigen IP-Modellen meistens nicht konkurrenzfähig, da Fehlentscheidungen bei der Auswahl der Branchingvariablen nur über entsprechendes *Backtracking* korrigiert werden können.
  1. bester ZF-Wert: es wird ein Knoten mit minimalem bzw. maximalem Zielfunktionswert gewählt
  2. Summe der Integer Unzulässigkeiten: $s_k = \sum_j \min(f_{kj}, 1 - f_{kj})$ wobei f_{kj} der fraktionelle Teil der Integer Variablen j der LP-Lösung am Knoten k ist und j alle Integer Variablen durchläuft. Es wird ein Knoten mit minimalem s_k gewählt.
  3. Best Projection Kriterium: $BP_k = (z^* - z(LP)_k) / s_k$. Es wird ein Knoten mit maximalem BP_k gewählt.
  4. Ursprüngliches Best Projection Kriterium: $BP_k = z(LP)_k + p\, s_k$. p wird wie folgt initialisiert: $p = 0.4\, z(LP)_0 / s_0$ und Knoten 0 das Anfangs-LP definiert. Wird eine bessere Integer-Lösung mit dem ZF-Wert z* gefunden, dann wird p neu bestimmt: $p = (z^* - z(LP)_0) / s_0$. Änderungen von p bewirken die Änderungen von BP_k. Es wird ein Knoten mit minimalem bzw. maximalem BP_k gewählt.

- Der Optimierungsalgorithmus zur Lösung der LP-Relaxation eines Knotens (xiplpt) kann ebenfalls einen großen Einfluß auf die Gesamtlaufzeit ausüben. Standardmäßig wird ein primaler Simplexcode gewählt (xiplpt = 0). Optional kann auch ein dualer Simplexcode eingesetzt werden. Zu beachten ist, daß der duale Simplexcode prinzipbedingt bei gleicher Iterationszahl langsamer ist. Eine Einsparung kann nur dann erzielt werden, wenn im Mittel die Anzahl der LP-Iterationen für den dualen Algorithmus etwa um den Faktor zwei

kleiner ist, als die des primalen Codes. Das IPM-Verfahren ist nicht sinnvoll, da keine gutenReoptimierungsmöglichkeit besteht. Es kann daher nicht gewählt werden.

- Durch die Spezifikation einer realistischen absoluten Schranke (xipbnd) für einen optimalen IP-Wert bzw. eines prozentualen Wertes (xlpgap) für die Berechnung einer Schranke, basierend auf dem LP-Wert nach dem Supernode-Processing, kann manchmal die Suche verkürzt werden. Dabei ist jedoch die Gefahr gegeben, daß die Suche erfolglos bleibt, wenn xipbnd bzw. xlpgap zu streng gewählt werden.

- Empfehlenswert ist, den Parameter xrimpr, der die relative Verbesserung, ausgehend von einer gefundenen IP-Lösung zur nächsten, festlegt, nicht zu klein zu wählen. Man sollte ihn von seinem Default-Wert xrimpr = 1.d-4 auf 1.d-3 erhöhen. Meistens rechtfertigt die Qualität der Modelldaten keinen kleineren Wert.

- Besondere Schwierigkeiten bereiten manchmal IP-Modelle, die als LP optimal lösbar sind, jedoch als IP-unzulässig sind. Dies ist darin begründet, daß ein Knoten nur dann eleminiert wird, wenn das LP unzulässig ist.

5 Einsatz der MOPS Objekt-Bibliothek

Alle globalen Variablen von MOPS fangen mit dem Buchstaben X an, der daher als Anfangsbuchstabe für Benutzervariablen nicht verwendet werden sollte. Die skalaren Variablen, (nicht indizierte Variablen) sind im Common-Block xcr.inc (communication region) gespeichert. Die Variablen in xcr.inc können klassifiziert werden in: Input- und Output-Dateien, Benutzer-Inputparameter, MOPS-interne Variablen und Speicherallokationsparameter.

5.1 Parameter zur Speicherallokation

- **Einstellungen zur Lösung von LP und IP-Modellen**

xnmax: Maximalzahl der Strukturvariablen
xmmax : Maximalzahl der Restriktionen
xnzmax: Maximalzahl der Nichtnullelemente in der Koeffizientenmatrix **ohne** die Nichtnullelemente der rechten Seite
xnenmx: Maximalzahl der Nichtnullelemente in der LU-Faktorisierung; Standardeinstellung ist xnenmx = xnzmax
xrcmax: Maximalzahl der Zeichen, die insgesamt zur Speicherung der Zeilen und Spaltennamen vorgesehen sind.

- **Zusätzliche Einstellungen zur Lösung von IP-Modellen**

xmncli: Maximalzahl der Cliques und Implikationen
xmnzcl: Maximalzahl der möglichen Einträge in der Clique- und Implikationentabelle
xmaxno: Maximalzahl der Knoten in der Hauptspeicherknotentabelle

5.2 MOPS-interne Variablen

Diese dienen dem parameterlosen Informationsaustausch zwischen den MOPS-Routinen. Durch Verwendung der Include-Dateien xcr.inc, und xtypes.inc kann eine Benutzerprozedur auf alle MOPS-internen Daten zugreifen. Eine Veränderung dieser internen Daten sollte nur mit äußerster Vorsicht vorgenommen werden. Die globalen Variablen von MOPS sind in Common-Blöcken deklariert, die als Include-Dateien in die Programme eingebunden werden müssen. Ein Hauptprogramm zur LP-Optimierung muß folgende Include-Dateien beinhalten:

xcr.inc (Deklaration der skalaren Variablen)
xinit.inc (Initialisierung der Default-Werte)

Zur IP-Optimierung muß zusätzlich die Include-Datei xipinit.inc eingebunden werden. Ein Beispiel ist in xcase.for enthalten.

Unterprogramme, die auf globale Variablen von MOPS zugreifen wollen, müssen für die skalaren Variablen die Include-Datei xcr.inc und die Include-Datei xtypes.inc einbinden.

Folgende Routinen sind von einem Benutzerprogramm direkt aufrufbar: xcnvrt, xlpopt, xipopt, xrprof, xchkpa.

Jede Routine setzt einen Return Code (RC) in xrtcod, der nach dem Aufruf immer abgeprüft werden sollte. Im Fall eines Fehlers ist xrtcod ungleich Null. Wenn das Program durch einen Fehler terminiert, kann durch Aufruf von xerror eine Fehlermeldung in die Message-Datei (Dateiname xfnmsg) geschrieben werden.

5.3 MOPS internes Modellformat (IMR)

In der internen Modelldarstellung (LP oder IP) wird jeder Restriktion der EMR eine logische Variable hinzugefügt. Der m-Vektor der logischen Variablen wird mit s bezeichnet. Die Variablen der IMR bestehen also aus dem Vektor (x,s)' mit der Dimension n+m. Je nach Typ der Restriktion erhalten die logischen Variablen untere und obere Schranken, die gemeinsam mit denen für die Strukturvariablen in den n+m dimensionalen Vektoren lb und ub gespeichert werden. In der IRM werden die Restriktionen in die Form A x + E s = $\underline{0}$ transformiert, wobei E die m-dimensionale Einheitsmatrix ist. Für die internen Datenstrukturen ist es sinnvoll, alle Variablen einheitlich zu indizieren. Da die Strukturvariablen von 1,...,n indiziert sind, ist es zweckmäßig die logischen Variablen von n+1,..,n+m zu indizieren. Für die i-te Restriktion ergeben sich als untere bzw. obere Schranken für die zugehörige logische Variable s_i: lb(n+i) = -bu(i), ub(n+i)= -bl(i). Die IMR sieht also folgendermaßen aus:

Minimiere / maximiere $\quad c' x$

$$A x + E s = \underline{0} \qquad \textbf{(IMR)}$$
$$lb \le (x, s)' \le ub$$

Hierbei sind x, c reelle n-Vektoren, A eine reelle m x n Matrix, E die m x m Einheitsmatrix, $\underline{0}$ der m-dimensionale Nullvektor, lb und ub reelle n+m-Vektoren, wobei für $1 \le j \le n$ gilt: lb(j) = l(j), ub(j) = u(j). Die Schranken für $n+1 \le j \le n+m$ ergeben sich aus den ursprünglichen Schranken bl und bu und zwar ist lb(j) = -bu(j-n), ub(j) = -bl(j-n). Die Vektoren lb und ub können auch Elemente, die plus oder minus unendlich sind, aufweisen. Zu beachten ist die vertauschte Darstellung der Schranken mit Vorzeichenwechsel für die logischen Variablen in der IMR gegenüber den Schranken für die Restriktionen in der EMR, die sich aus A x + E s = $\underline{0}$ ergibt. In der IMR hat das Gleichungssystem (d.h. die Matrix (A,E) den Rang m, unabhängig welchen Rang A hat. Grafisch läßt sich die IMR folgendermaßen darstellen:

1	**n**	**n+1**	**n+m**		
c		0			
A		E		=	0
l		-bu			lb
u		-bl			ub

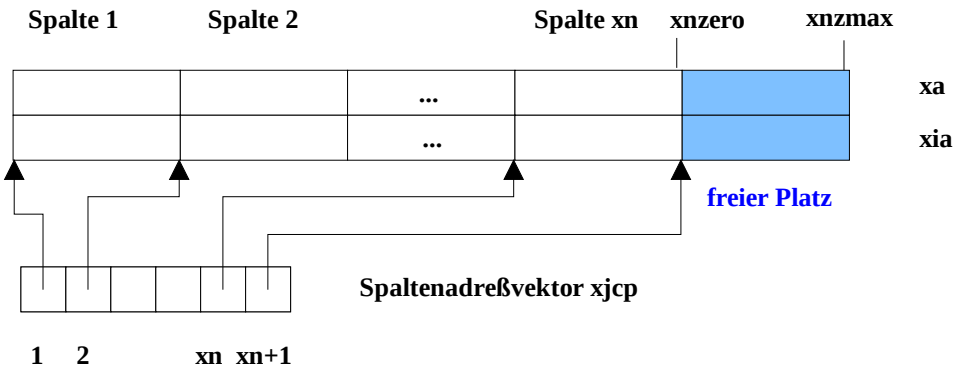
Die Modelldaten können auch **direkt** in internen Arrays von MOPS in Form der IMR gespeichert werden. Der Modellgenerator für die LP-Daten muß in Fortran oder C geschrieben sein, um mit Hilfe der Include-Dateien xcr.inc, xlp.inc und xip.inc Zugriff zu den internen Feldern von MOPS zu erhalten. Selbstverständlich muß entweder der gleiche oder ein kompatibler Compiler zu den Objektcodes in der MOPS-Bibliothek verwandt werden. Die interne Modelldarstellung bietet gegenüber dem MPS-Format eine Reihe von Vorteilen:

- Die aufwendige Konvertierung von MPS-Daten in das interne Format wird vermieden; insbesondere werden Datentypkonversionen vermieden.
- Modelldaten benötigen im MPS-Format mehr Speicherplatz als in der IMR.

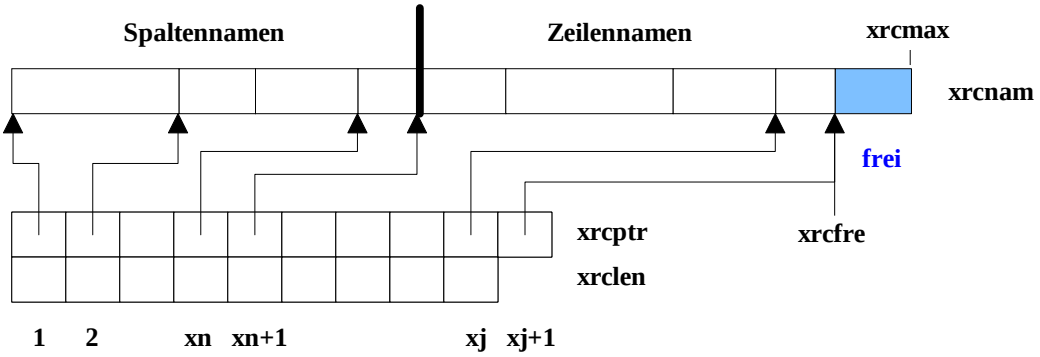
- Die numerischen Werte sind genauer, weil die doppelte Datentypkonversion (Modellgenerator - MPS-Format- Convert) vermieden wird.
- Die interne Modelldarstellung erlaubt variabel lange Zeilen- und Spaltennamen bis zu 64 Zeichen (MPS-Format max. 8 Zeichen). Bei Modellen mit vielen Indizes erleichtert dies die Implementation des Modellgenerators und erlaubt lesbarere Namen.

Die Implementation der IMR in interne Datenstrukturen von MOPS erfordert:

- Die Dimensionen m und n werden in den Variablen xm, xn gespeichert. Die Variable xj enthält n+m. Obwohl redundant, ist der Wert für xj eine Eingabegröße, da die rechte Seite in den Positionen xn+1, .., xj von xlb bzw. xub gespeichert wird. Die Anzahl der Nichtnullelemente wird in der Variablen xnzero gespeichert. Alle (skalaren) Variablen sind im Common Block xcr.inc gespeichert.
- der Vektor c ist im Array xcost in den Positionen 1,...,xn gespeichert
- die Vektoren lb, ub sind in den Arrays xlb, xub in den Positionen 1,...,xj gespeichert.
- die dünn besetzte Matrix A wird *lückenlos* spaltenweise gespeichert und zwar werden die *Nichtnullelemente* im Feld xa (R8) und die zugehörigen Zeilenindizes im Feld xia (I4) gespeichert. Innerhalb einer Spalte ist die Reihenfolge der Elemente beliebig. Der Zeigervektor xjcp definiert für $1 \le j \le xn+1$ den Anfangspunkt einer Spalte in den Feldern xa und xia. Spalte j belegt also in diesen Feldern die Positionen xjcp(j),...xjcp(j+1)-1. Der xn+1-te Wert für xjcp dient nur der Adressierung der Spalte xn, wobei xjcp(xn+1) = xnzero+1 gelten muß. Die m-dimensionale Einheitsmatrix der IMR wird nicht gespeichert. Die Variable xnzmax definiert die Maximalzahl der Nichtnullelemente, die gespeichert werden können. Die Positionen xnzero+1,...,xnzmax in den Arrays xa und xia sind ungenutzt.



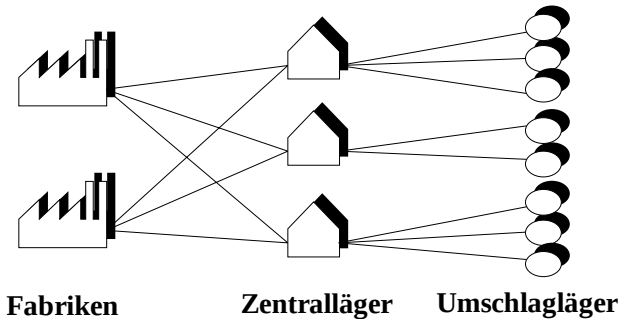
- nach dem Konvertieren einer MPS-Datei werden die Namen sequentiell in Strings von jeweils acht Zeichen gespeichert, wobei xrcfre die Position des letzten besetzten Elementes in xrcnam angibt:



- die Länge jedes Namens j wird durch xrclen(j) definiert und ist auf 64 Zeichen beschränkt. Nach Einlesen einer MPS-Datei gilt xrclen(j) = 8 ($1 \le j \le xj$). Die Namen können in beliebiger Reihenfolge angeordnet sein. Der Zeiger xrcfre dient zur Identifikation der letzten belegten Speicherstelle im Feld xrcnam. Namen sind für die Optimierung bedeutungslos.
- bei IP-Modellen muß für jede ganzzahlige Variable mit dem Index j das Bit xinteg im Array xkey(j) gesetzt werden. (vgl. die Programme xcase.f, bzw. xcase.c).

5.4 Fallstudie zur Modellgenerierung im IMR-Format

Wir betrachten ein mehrstufiges Distributionssystem mit Fabriken, Zentrallägern und Umschlaglägern. Die Fabriken produzieren jeweils ein bestimmtes Artikelsortiment. Von den Fabriken werden die Artikel zu den Zentrallägern transportiert. Jedes Zentrallager muß das gesamte Artikelsortiment führen, d.h. es wird u.U. von mehreren Fabriken beliefert. Die Umschlagläger werden ausschließlich von den Zentrallägern beliefert. Im Planungszeitraum darf jedes Umschlaglager nur von genau einem Zentrallager beliefert werden.



Alle Kostenfunktionen (Transportkosten und Herstellkosten) werden als linear angenommen.

Der Planungszeitraum beträgt ein Jahr. Es sind die Bedarfsmengen der Umschlagläger der einzelnen Artikeln gegeben, sowie die Produktions- und Lagerkapazitäten. Die Planung hat als Ziel, die Produktions- und Transportkosten zu minimieren, wobei die gegebenen Kapazitäten für Fabriken und Zentralläger sowie die Bedarfsforderungen der Umschlagsläger einzuhalten sind. Nach Definition der Indizes werden die Daten definiert, die als gegeben vorausgesetzt werden:

Indizes und Indexmengen	
w:	Fabrik, $w \in W$
i:	Artikel, $i \in I$
z:	Zentrallager, $z \in Z$
u:	Umschlaglager, $u \in U$

I_w: Artikelsortiment, das Fabrik w produzieren kann
I: gesamtes Artikelsortiment, d.h. $I = \bigcup_{i \in I_w} I_w$

Daten

Alle Daten beziehen sich auf den einjährigen Planungshorizont. Tonnen werden mit t abgekürzt. Mit ZE wird eine kurze repräsentative Bezugsperiode (z.B. eine Woche) des Planungshorizontes definiert. Die Kapazität eines Zentrallagers wird als Stromgröße definiert, um den unterschiedlichen Umschlagsgeschwindigkeiten Rechnung zu tragen.

b(u,i): Bedarf von Umschlaglager u vom Artikel i (t)
s(i) : Umschlagsgeschwindigkeit vom Artikel i (1/ZE)
k(w,i): Produktionskapazität der Fabrik w von Artikel i (t)
h(w,i): Herstellkosten des Artikels i in Fabrik w (DM/t)
K(z) : Kapazität des Zentrallagers z (t/ZE)
t(i) : Einheitstransportkosten für Artikel i pro km und t (DM/t)
d(a,b): Entfernung (KM) vom Knoten a zum Knoten b

Mathematische Modellformulierung

Es wird ein einperiodisches gemischt-ganzzahliges Optimierungsproblem formuliert.

Entscheidungsvariablen des Modells

x(w,z,i) Menge (t) des Artikels i, die von Fabrik w nach Zentrallager z transportiert wird; $w \in W$, $i \in I_w$, $z \in Z$; Da wir keine Lagerhaltung in den Fabriken unterstellen, ist die zu transportierende Menge x(w,z,i) gleich der zu produzierenden Menge

y(z,u) = 1 falls Umschlaglager u dem Zentrallager z zugeordnet wird, sonst 0;

Modellrestriktionen

$$\sum_{z \in Z} x(w,z,i) \le k(w,i) \qquad w \in W,\ i \in I_w$$

$$\sum_{w \in W} x(w,z,i) - \sum_{u \in U} b(u,i)\, y(z,u) = 0 \qquad z \in Z,\ i \in I$$

$$\sum_{z \in Z} y(z,u) = 1 \qquad u \in U$$

$$\sum_{i \in I} \sum_{u \in U} b(u,i) s(i)\, y(z,u) \le K(z) \qquad z \in Z$$

$x(w,z,i) \ge 0$, $z \in Z$, $w \in W$, $i \in I_w$

$y(z,u) \in \{0,1\}$, $u \in U$, $z \in Z$

Zielfunktion

$$\textit{minimiere:} \sum_{z \in Z} \sum_{w \in W} \sum_{i \in I(w)} x(w,z,i)\,(h(w,i) + t(i)\,d(w,z)) + \sum_{z \in Z} \sum_{u \in U} \sum_{i \in I(w)} b(u,i)\,t(i)\,d(z,u)\,y(z,u)$$

Implementation des Modells

C und Fortran-Programme befinden sich in den Unterverzeichnisssen CaseVisualC und CaseVisualF. Die Datei xcase.c bzw. case.for enthält Quellenprogramme zur Modellgenerierung. Es sind jeweils zwei Modellgeneratoren vorhanden: einer, der direkt die MOPS-IMR generiert, während der Zweite die IMR-Schnittstellenroutinen in 5.7 benutzt. Die Daten für die Modellgeneratoren befinden sich im UV models in der Datei xcase.mps. Die Dateierweiterung von xcase wurde nur deshalb mps genannt, um die automatische Generierung aller Output-Dateien unter dem Dateinamen xcase zu ermöglichen.

Subroutine xcase zur Modellgenerierung

xredat dient zum Einlesen der Daten aus der Datei xcase.mps in Hauptspeicherfelder. In xcase werden die Matrixkoeffizienten als Triplets (i,j,a_{ij}) generiert. Es wird empfohlen, sich diese Routine genau anzuschauen, um die wichtigsten Prinzipien der Modellgenerierung im internen Format zu verstehen. Zur Speicherung von Zeilen- bzw. Spaltennamen wird die Subroutine xsname verwandt. Anschließend wird die Subroutine xcordr aufgerufen, um eine spaltenweise Sortierung der Matrixkoeffizienten vorzunehmen. Nach dieser Routine liegt das Modell im IMR-Format vor und die LP-und IP-Optimierung kann durchgeführt werden.

5.5 LP-Optimierung

xrprof

Diese Prozedur dient dazu Parameter und Dateinamen aus der Profile-Datei xfnpro einzulesen. xfnpro hat den Default-Wert 'xmops.pro'. Falls die Datei mit dem Namen xfnpro nicht existiert, so werden für alle Parameter die Default-Werte genommen.

xrtcod 0: ok; 1: ungültiger Name im Profile oder eine syntaktisch inkorrekte Wertzuweisung.

xchkpa

Diese Routine dient zur Überprüfung der wichtigsten Benutzer-Inputdaten (vom Profile oder Programm) abzuprüfen. Es wird getestet, ob die Werte im zulässigen Bereich liegen.

xrtcod 0: alles ok; 1: ein Input-Datum liegt im nicht zulässigen Bereich.

xcnvrt (b)

xcnvrt liest die MPS-Datei, die in xfnmps angegeben wurde und erzeugt eine interne Modelldarstellung (IMR) des Modells. Falls Syntax-Fehler erkannt werden, wird ein positiver Return-Code gesetzt.

xlpopt (b)

Mit xlpopt wird ein LP oder IP-Modell, das in der IMR vorliegt, als LP gelöst. In xlpopt wird die IMR und die gegebenen Parameter zuerst extensiv getestet. Falls fatale Fehler entdeckt werden, wird mit einem positiven Return-Code terminiert. Wenn die syntaktischen Modelltests keine Fehler entdecken, wird die Optimierung standardmäßig mit der primalen Simplex-Methode gestartet. Der Status bei der Terminierung wird mit Hilfe der Variablen xrtcod und xlpsta ausgedrückt. Wird das Problem gelöst, so hat xrtcod den Wert Null und xlpsta zeigt den Status der Lösung. Zu beachten ist, daß xrtcod auch dann den Wert Null erhält, wenn das Problem als unzulässig oder unbeschränkt erkannt wird.

LP-Status bei xrtcod = 0
xlpsta = 0 optimale Lösung
xlpsta = 1 das Problem hat keine zulässige Lösung
xlpsta = 2 unbeschränkte Lösung

Wird das **Problem nicht gelöst**, so sind folgende Werte für xrtcod und xertyp möglich:

Mögliche Return-Codes bei abnormalem Ende
xrtcod = 1 Iterationslimit erreicht
xertyp = 1: LP iteration limit xmiter reached
xertyp = 2: LP time limit xmxtlp reached
xertyp = 3: IP-time limit xmxmin for B&B reached
xertyp = 4: node LP iteration limit xmitip reached
xertyp = 5: insufficient space xmxnen for LU factors
xertyp = 6: node limit xmxnod reached
xertyp = 7: nodes buffer size too small
xertyp = 8: disk full
xertyp = 9: granted disk space xmxdsk exhausted
xertyp =10: xmxnen must be larger than xnzero

xrtcod = 2	Eingabefehler bei der IMR oder Basis
xrtcod = 3	Ausgabefehler
xrtcod = 4	Numerische Probleme, die Benutzereingriffe erfordern
xrtcod = 5	nicht benutzt
xrtcod = 6	interner Systemfehler
xrtcod = 7	nur wenn LGS gelöst werden: Matrix ist singulär

In diesem Fall zeigt xlpsta den Status der aktuellen Lösung bei der Terminierung an:

LP-Status bei abnormalem Ende	
xlpsta = 3	primal zulässig, aber dual unzulässig
xlpsta = 4	dual zulässig, primal unzulässig
xlpsta = 5	weder primal noch dual zulässig

Ist xoutsl = 1 gesetzt, so wird die optimale Lösung im MPS-Format in eine Datei geschrieben. Bei jedem anderen Wert wird keine Datei generiert.

Eine optimale LP-Lösung kann auch direkt aus Arrays im Hauptspeicher entnommen werden. Zu beachten ist, daß die Informationen nur im Fall xlpsta = 0 diese Bedeutung haben! Die Indizierung der Strukturvariablen läuft von 1,...,xn die der logischen Variablen von xn+1,..,xj, wobei xj = xn + xm (siehe auch 5.3 MOPS internes Modellformat (IMR)). Bei Arrays der Dimension xj sind also die Werte der Strukturvariablen in den Positionen 1,...,xn und die der logischen Variablen in den Positionen xn+1,...,xj gespeichert. Der Zugriff auf diese Informationen erfordert die Einbindung der Include-Datei xcr.inc.

xfunct (R) Zielfunktionswert einer optimalen LP-Lösung.

xlpsol(xj) (R)

Dieser Array enthält die Werte aller Variablen einer optimalen LP-Lösung. Zu beachten ist, daß sich der Wert s einer logischen Variablen (Positionen xn+1,...,xj) aus der der Gleichung s = -ax ergibt, wobei a der Zeilenvektor der zugehörigen Restriktion ist.

xdjsc(xj) (R)

Dieser Array enthält die reduzierten Kosten aller Variablen einer optimalen LP-Lösung. Die reduzierten Kosten einer Variablen j mit $1 \le j \le xj$ sind definiert als $d_j = c_j - \pi' a_j$, wobei π der Vektor der Dualwerte und a_j der Spaltenvektor und c_j der Zielfunktionswert der Variablen j ist. Für logische Variablen ist $c_j = 0$.

xpi(xm) (R)

Dieser Array enthält die Dualwerte einer optimalen LP-Lösung. Für eine Restriktion i definiert der i-te Dualwert die marginale Veränderung in der Zielfunktion, wenn die rechte Seite b_i um eine Einheit verändert wird. Diese Grenzwertbetrachtung hat nur eingeschränkte Gültigkeit und ist insbesondere bei dualer Degeneration (Normalfall) sehr problematisch.

xkey(xj) (I)

Alle statischen und dynamischen Statusinformationen einer Variablen J werden in einzelnen Bits von xkey(j) gespeichert und können bei Bedarf abgerufen werden. Ist p der Index eines bestimmten Bits, so wird es durch die Operation *btest(xkey(j),p)* getestet. Die logische Funktion btest erhält den Wert wahr, wenn Bit p gesetzt ist. Die Indizes sind mit aussagefähigen Namen versehen, um die Tests zu erleichtern. Im Rahmen einer Ergebnisausgabe sind folgende Bits relevant: Das Bit xbasic ist gesetzt, wenn die Variable in der Basis ist. Ist die Variable nicht in der Basis, so definiert das Bit xatub, ob sich die Variable an der oberen Schranke befindet. Für fixierte Variablen ist dieses Bit nicht gesetzt. Für eine freie Variable ist dieses Bit ebenfalls nicht gesetzt, obwohl keine endliche untere Schranke existiert.

5.6 IP-Optimierung

Das Hauptprogramm xip.for zeigt den grundsätzlichen Ablauf einer IP-Optimierung. Der erste Teil des Programms, vom Einlesen des Profiles bis zum Lösen des LP's, ist identisch mit xlp.f. Nach der Lösung des LPs und der LP-Lösungsausgabe, wird überprüft, ob eine optimale LP-Lösung vorliegt, Integer-Variablen vorhanden sind (xmxj > 0) und ob das IP-Modell gelöst werden soll (xlpmip > 0). Wenn alle diese Bedingungen erfüllt sind, dann wird die Routine xipopt zur Lösung des IPs aufgerufen. Wird das Problem gelöst, so hat xrtcod den Wert 0 und xipsta hat den Wert 0. xrtcod erhält auch dann den Wert Null, wenn das Problem als unzulässig erkannt wird. Der Status der IP-Optimierung wird in der Variablen xipsta gespeichert.

IP-Status	
xipsta = 0	Branch&Bound-Suche ist vollständig abgeschlossen.
xipsta = 1	Branch&Bound-Suche ist nicht vollständig abgeschlossen, da ein Knoten- oder Zeit- oder Speicherplatzlimit erreicht wurde, oder Fehler auftraten

Ist die **Branch-&-Bound-Suche vorzeitig beendet**, so hat xrtcod einen der folgenden Werte:

Mögliche Return-Codes bei abnormalem Ende	
xrtcod = 1	Knoten-, Iterations- oder CPU-Zeitlimit erreicht
xrtcod = 2	Eingabefehler bei der IMR
xrtcod = 3	Ausgabefehler
xrtcod = 4	Numerische Probleme, die Benutzereingriffe erfordern
xrtcod = 5	Zu wenig Speicherplatz für die LU-Matrizen
xrtcod = 6	interner Systemfehler

Lösungen werden während der IP-Optimierung analog zur LP-Lösung in die Datei xfnips geschrieben, sofern xoutsl > 0 ist. Outputinformationen über eine optimale oder die beste gefundene IP-Lösung sind in folgenden Variablen gespeichert:

xnints (I) Anzahl gefundener IP-Lösungen.

xzbest (R) Zielfunktionswert der besten gefundenen IP-Lösung, wenn xnints > 0.

xintso(xj) (R)

Dieser Array enthält die Werte aller Variablen der besten gefundenen IP-Lösung, wenn xnints > 0 ist. Der Wert s einer logischen Variablen (Positionen xn+1,..,xj) ergibt sich aus der der Gleichung s = -ax, wobei a der Zeilenvektor der zugehörigen Restriktion ist. Das Bit xinteg in xkey gibt an, ob eine Variable ganzzahlig oder kontinuierlich ist.

5.7 IMR-Routinen (Zugriff zu Modell- und Lösungsinformationen)

Hilfsroutinen bilden die Schnittstelle zur IMR (→ 5.3). Mit ihnen können Informationen über einzelne Zeilen, Spalten und Matrixelemente gelesen bzw. neue Zeilen, Spalten und Matrixelemente eingegeben werden. Auch die Lösungswerte der aktuellen LP- bzw. IP-Lösung können gelesen werden. Alle IMR-Routinen stehen im Modul imr.for im UV IMR als Quellenprogramme zur Verfügung. Eine vorhandene LP-Basis wird durch die IMR-Routinen nicht modifiziert.

b bezeichnet im folgenden den Speicherblock (real*8), der vorher angelegt wurde, um die Modelldaten aufzunehmen.

Xmorig definiert die Anzahl der Restriktionen im ursprünglichen Modell. Vor einer LP/IP-Optimierung ist xmorig = xm. Ein LP-Prprocessing entfernt ggf. Restriktionen, d.h. xm ≤ xmorig. Nach einem Supernode-Processing können jedoch Schnittebenen generiert werden, sodaß xm wieder erhöht wird. Die IMR-Routinen fügen Zeilen und Elemente grundsätzlich im Bereich 1,..,xmorig ein.

subroutine xnwmod	
dient zur Aufnahme eines neuen Models im Speicher. Ein vorhandenes Modell wird gelöscht. Vorher müssen ggf. xmmax, xnmax, xnzmax, usw. (vgl. 4.1) Werte für die maximale Modellgröße zugewiesen werden, wenn von den Standardwerten abgewichen werden soll.	
Fortran Calling Sequence: call xnwmod (b)	
Input Parameter	
Output Parameter	
Xrtcod (I4)	Returncode: 0: ok, <>0: Fehler

subroutine xnwcol	
Diese Routine definiert eine neue Strukturvariable.	
Fortran Calling Sequence: call xnwcol (j,mode,name,len,type,cj,lb,ub,b)	
Input Parameter	
j (I4)	Spaltenindex
mode (I4):	0: neue Variable, <>0: Modifikation einer Variable
name (C64)	Variablenname
len (I4)	Anzahl Zeichen von cname
type (I4)	Typ der Variablen: 0: kontinuierlich, <>0 Integer Variable
cj (R8)	Zielfunktionskoeffizient
lb (R8)	Untere Schranke für den Variablenwert
ub (R8)	Obere Schranke für den Variablenwert
b (R8)	Speicherblock
Output Parameter	
Xrtcod (I4)	Returncode: 0: ok, <>0: Fehler

subroutine xnwrow	
Diese Routine definiert eine neue Restriktion.	
Fortran Calling Sequence: call xnwrow (i,mode,rowname,len,rhsl,rhsu,b)	
Input Parameter	
i (I4):	Zeilenindex einer Variablen, wobei $1 \leq j \leq xn$
mode (I4):	0: neue Zeile, <>0: Modifikation einer Zeile
name (C64)	Zeilenname
len (I4)	Anzahl Zeichen von cname
rhsl (R8)	Untere Schranke für den Zeilenwert
rhsu (R8)	Obere Schranke für den Zeilenwert
b (R8)	Speicherblock
Output Parameter	
Xrtcod (I4)	Returncode: 0: ok, <>0: Fehler

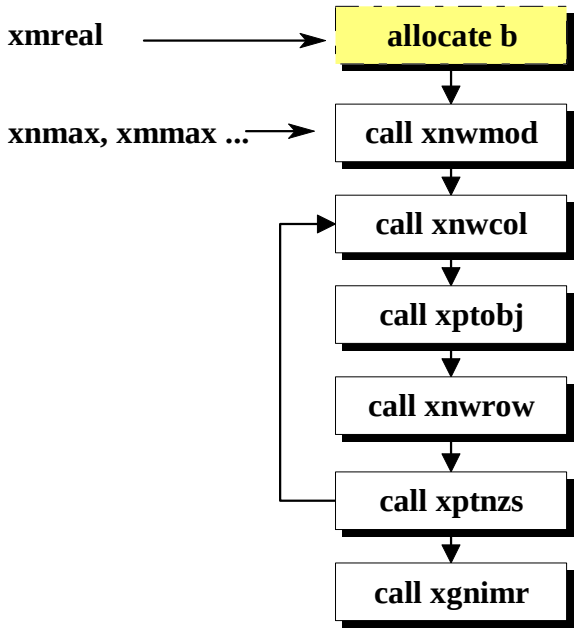
subroutine xgtinx	
Liefert den Index eines gespeicherten Zeilen- bzw. Spaltenamens	
Fortran Calling Sequence: call xgtinx (name,len,mode,index,b)	
Input Parameter	
Name (C64)	Zeilen- oder Spaltenname
Output Parameter	
len (I4)	Anzahl Zeichen von Name
mode (I4)	0: Suche nach Zeilennamen, <>0: Suche nach Spaltennamen
index (I4):	Index, wenn Name vorhanden, 0 sonst
b (R8)	Speicherblock
Xrtcod (I4)	Returncode: 0: ok, <>0: Fehler

subroutine xgnimr	
Erzeugt aus einem in Triplets generierten Modell eine interne Modelldarstellung für die nachfolgende Optimierung.	
Fortran Calling Sequence: call xgnimr(b)	
Input Parameter	
b (R8)	Speicherblock
Output Parameter	
Xrtcod (I4)	Returncode: 0: ok, <>0: Fehler

subroutine xptnzs	
Speichert Triplets (i,j,a_{ij}) in der internen Modellstruktur.	
Fortran Calling Sequence: call xptnzs (nzero,rowi,coli,value,b)	
Input Parameter	
nzero (I4)	Anzahl der Nichtnullelemente in rowi, coli, value
rowi (1:nzero) (I4)	Zeilenindizes
coli (1:nzero) (I4)	Spaltenindizes
Value (1:nzero) (R8)	Nichtnullelemente
b (R8)	Speicherblock
Output Parameter	
Xrtcod (I4)	Returncode: 0: ok, <>0: Fehler

subroutine xptobj	
Weist einer Strukturvariablen einen Zielfunktionskoeeffizienten zu.	
Fortran Calling Sequence: call xptobj (j,cj,b)	
Input Parameter	
j (I4)	Spaltenindex
cj (R8)	Zielfunktionskoeffizient
b (R8)	Speicherblock
Output Parameter	
Xrtcod (I4)	Returncode: 0: ok, <>0: Fehler

Die Generierung eines Modells kann nach folgendem Schema erfolgen.



5.8 C-Schnittstelle zur MOPS-Bibliothek (PC-Version)

Die Moduln der MOPS-Bibliothek sind in der PC-Version auch mit dem binärkompatiblen Visual Studio C++-Compiler zugänglich. In MOPS werden globale Variablen durch Common-Blöcke definiert, deren Adresse relativ zum Anfang des Blocks ermittelt wird. Dieses Konstrukt entspricht in C dem einer Struktur. Der Zugriff auf die wenigen benötigten Variablen in xcr.inc wird bei der hier verwendeten Lösung unter C über globale Zeiger realisiert.

Im Unterverzeichnis CaseVisualC befinden sich entsprechende Beispielprogramme (xip.c, xcase.c, xjoin.h).

Die Include-Datei xjoin.h zeigt, welche Variablen und Funktionen die Schnittstelle zur Verfügung stellt.

6 Einsatz der MOPS Dynamic Link Library

Die MOPS-DLL basiert auf der statischen MOPS-Biblithek (mops.lib) und ermöglicht Windows-Programmen einen interaktiven Zugriff auf das Optimierungsmodell im Hauptspeicher. Eine DLL ist charakterisiert durch:

- „gelinkte“ DLL-Funktionen, die in einem eigenen – von der rufenden Anwendung separaten - (virtuellen) Adreßraum ausgeführt werden,
- Anwenderprogramme unterschiedlicher Programmiersprachen können über die definierten APIs eine DLL nutzen. Es muß der 32-Bit-Aufruf von Windows DLL-Funktionen unterstützt werden. Dies erfolgt u.a. für Delphi, Visual Basic, Visual C++ und C#.
- DLL-Funktionen können nur unter Windows (98, 2000, NT, XP) eingesetzt werden.

6.1 Schema einer Optimierung mittels DLL-Funktionen

Primärdaten werden aus einer Datenbank extrahiert. Anschließend generiert ein Visual Basic Programm (Modellgenerator) das Modell in den Arrays ia, ja, a, lb, ub, c und typ.

Reservierung eines Speicherblocks von 300 MB
rc =AllocateMemory(0)
rc = InitModel()
If rc <> 0 Then MsgBox " Fehler in InitModel: " & rc
Vergrößerung der maximalen Modelldimensionen bei sehr großen LP/IP-Modellen (optional)
rc = SetMaxLPDim (mmax, nmax, nzmax)

```
If rc <> 0 then exit
Initialisierung des Speicherblocks zur Aufnahme eines neuen Modells
rc = InitModel()
If rc <> 0 then exit
Übergabe eines Modells an die DLL
Rc = PutModel (intyp, inf, m, n, nz, ia(1), ja(1), a(1), lb(1), ub(1), c(1), typ(1))
Es wird das LP gelöst.
Do
    rc = OptimizeLP(100, xiter, xnif, xlpsta, xfunct, xsif)
    If rc <> 0 Then Errorhandling
Until rc <> 999
Msgbox " Der Zielfunktionswert beträgt:" & xfunct
Speichere die LP-Lösung in VB-Arrays
rc = GetLPSolution(xlpsta, xfunct, xs(1), dj(1), stat(1))
If rc <> 0 Then
   meldung1 = "Fehler in GetLPSolution " & rc
   Exit Function
End If
Es wird das IP gelöst
rc = OptimizeIP(xfrnod, xnodes, xnints, xresou, xzbest, xzubnd)
If rc <> 0 Then exit
rc = GetIPSolution(ipsta, xzbest, xs(1), dj(1), stat(1))
```

6.2 Beschreibung der DLL-Funktionen

6.2.1 AllocateMemory(): Anlegen eines Speicherblocks für das Modell.
6.2.2 Break(): Vorzeitiges Beenden einer Optimierung.
6.2.3 DelCol(): Löscht Spalte des Modells.
6.2.4 DelRow(): Löscht Zeile des Modells.
6.2.5 FindName(): Liefert den Zeilen- oder Spaltenindex eines Namens.
6.2.6 FreeMemory(): gibt angelegten Speicherblock frei.
6.2.7 GetCol(): Liest die Informationen einer Strukturvariablen.
6.2.8 GetDim(): Liest die aktuellen Dimensionen des Modells.
6.2.9 GetIPSolution() übergibt die beste gefundene IP-Lösung an das rufende Programm
6.2.10 GetLPSolution () übergibt eine optimale LP-Lösung an das rufende Programm
6.2.11 GetMaxDim(): Liest die maximalen Dimensionen der DLL.
6.2.12 GetModel(): überträgt ein LP/IP-Modell von der DLL in das rufende Programm
6.2.13 GetNonzero(): Liest ein Element der Koeffizientenmatrix.
6.2.14 GetParameter(): Liest den aktuellen Wert eines MOPS-Parameters.
6.2.15 GetRow(): Liest die Informationen einer Restriktion.
6.2.16 Mops():Optimierungsprozeß mittels Profile und MPS-Datei.
6.2.17 InitModel(): Initialisierung einer Optimierung bzw. eines Modells.
6.2.18 OptimizeIP(): IP-Optimierung.
6.2.19 OptimizeLP(): LP-Optimierung.
6.2.20 PutCol(): Modifiziere (Schreibe) Informationen einer (neuen) Strukturvariablen.
6.2.21 PutRow(): Modifiziere (Schreibe) Informationen einer (neuen) Restriktion.
6.2.22 PutModel() Übergibt ein LP/IP-Modell an die DLL
6.2.23 PutNonzeros(): Modifizieren oder Schreiben von Matrixkoeffizienten.
6.2.24 PutRow(): Modifiziere (Schreibe) Informationen einer (neuen) Restriktion.
6.2.25 ReadMpsFile(): Einlesen einer klassischen MPS-Datei.
6.2.26 ReadProfile(): Einlesen einer Profile-Datei.
6.2.27 ReadTripletFile() Einlesen einer Triplet-Datei
6.2.28 SetParameter(): Setzen eines MOPS-Parameters.

6.2.29 WriteMpsFile(): Schreiben einer MPS-Datei.

6.2.30 WriteTripletFile(): Schreiben einer Triplet-Datei

Hinweis: In den Funktionsbeschreibungen werden folgende Abkürzungen für die Übergabeparameter verwendet: I4: Integer 4 Bytes; R4: Floating-Point 4 Bytes; R8: Floating-Point 8 Bytes; Cs: Zeichenkette mit s Zeichen.

Jede Funktion liefert einen Return Code an das rufende Programm zurück, der mit rc bezeichnet wird. Rc ist 0, wenn die Funktion fehlerfrei ausgeführt wurde. Werte ungleich null signalisieren Fehlersituationen.

Im Anhang B „Deklarierung der DLL-Funktionen“ sind beispielhaft die Schnittstellen für Visual Basic beschrieben.

6.2.1 DLL-Funktion AllocateMemory()

AllocateMemory (mem) muss aufgerufen werden, um einen Speicherblock zur Aufnahme eines Modells und aller zugehörigen Optimierungsdaten anzulegen. AllocateMemory muss vor allen anderen Funktionen die Modelldaten lesen oder schreiben ausgeführt werden.

Ist ein Speicherblock bereits angelegt und wird AllocateMemory erneut aufgerufen, so wird der bisherige Speicher freigegeben und der gewünschte Speicher neu angelegt. In jedem Fall gehen alle Daten des Blockes verloren. Ist die Allokation erfolgreich, so werden Standardeinstellungen für LP und IP-Dimensionen vorgenommen. Ggf. muss vorher die Funktion FreeMemorye() aufgerufen werden.

Mit den Routinen SetMaxLPDim() bzw. SetMaxIPDim() können die in AllocateMemory voreingestellten Dimensionen redefiniert werden.

Input

mem (I4): Speichergröße des Blockes in Megabyte [MB], falls mem > 0; ist mem = 0, so wird der Default (300 MB) verwendet.

Output

Rc (I4): 0: alles OK, <> 0 nicht genügend verfügbarer Speicher vorhanden

Beispiel:

Rc = AllocateMemory (0): legt einen Datenspeicher von 300 MB an.

Rc = AllocateMemory (800): legt einen Datenspeicher von 800 MB an.

6.2.2 DLL-Funktion Break()

Break () muß aufgerufen werden, wenn die Optimierung vorzeitig beendet werden soll, d.h. wenn nach Beendigung der Funktion OptimizeLP() oder OptimizeIP() der Returncode 999 zurückgeliefert wird, die Optimierung aber nicht mehr weitergeführt werden soll. Der Optimierer erhält die Notwendigkeit, eventuell durchgeführte Skalierungen rückgängig zu machen und geöffnete Dateien zu schließen.

Input

Output

Rc: 0: alles OK, ≠ 0: Fehlersituation

Beispiel: Rc = Break()

6.2.3 DLL-Funktion DelCol()

DelCol (ColIndex) löscht die Spalte des Modells, deren Index angegeben wird.

Input

ColIndex (I4): Spaltenindex

Output

(I4): 0: alles OK, <> 0 Fehlersituation

Beispiel: Rc = DelCol(8).

6.2.4 DLL-Funktion DelRow()

DelCol (RowIndex) löscht die Zeile des Modells, deren Index angegeben wird.

Input

RowIndex (I4): Zeilenindex

Output

(I4): 0: alles OK, <> 0 Fehlersituation

Beispiel: Rc = DelRow(8).

6.2.5 DLL-Funktion Findname()

FindName (sName, lMode, lIndex) sucht den Namen einer Restriktion oder Strukturvariablen (Zeile oder Spalte) und liefert den entsprechenden Index.

Input

Name(C64): Zeilen- oder Spaltenname, nach dem gesucht werden soll
Mode (I4): 0 suche einen Zeilennamen ansonsten einen Spaltennamen

Output

Index (I4): Zeilen- oder Spaltenindex oder Null, falls Name nicht gefunden wird.
Rc: 0: alles OK, ≠ 0: Fehlersituation

Beispiel: Rc = FindName("Restriktion_Kap", 0, lIndex).

6.2.6 DLL-Funktion FreeMemory

FreeMemory() muss aufgerufen werden, um einen angelegten (virtuellen) Speicherblock freizugeben, d.h. dem Betriebssystem wieder zur Verfügung zu stellen. Dabei gehen alle Daten des Blockes verloren.

Input

Output

Rc (I4): 0: alles OK, <> 0 Fehler

Beispiel: Rc = FreeMemory().

6.2.7 DLL-Funktion GetCol()

GetCol (ColIndex, ColName, ColTyp, NoElements, Cost, LoBound, UpBound, Status, Activity, RedCost) liest Informationen einer Strukturvariablen.

Input

ColIndex (I4): Index

Output

ColName: Variablenname (max. 64 Zeichen lang)
ColTyp (I4): Variablentyp (0: kontinuierlich, 1: ganzzahlig)
NoElements (I4): Anzahl der Nichtnull-Elemente in der Spalte
Cost (R8): Zielfunktionskoeffizient
LoBound (R8): untere Schranke
UpBound (R8): obere Schranke

Status (I4): Status der Variablen im Optimum (1: basis; 2: lower; 3: upper; 4: fix)
Activity (R8): Aktivität der Variablen im Optimum
RedCost (R8): Reduzierte Kosten der Variablen im Optimum
Rc (I4): 0: alles OK, 1: ungültiger Index, 2: Name ist länger als 64 Zeichen

Wurde das Modell noch nicht optimal gelöst, sind die Werte für Status, Activity und RedCost nicht definiert.
Die Variable sColName muß lang genug sein, um den vollständigen Variablennamen aufnehmen zu können (z.B. durch *Dim sColName as string * 64* oder durch die Funktion *String$()*). Eine „Verlängerung“ der Variablen durch die DLL ist nicht möglich.

Beispiel: Rc = GetCol(17, ColName, ColTyp, NoElements, Cost, LoBound, UpBound, Status, Activity, RedCost) liest Spalte 17.

6.2.8 DLL-Funktion GetDim()

GetDim (NoRows, NoCols, NoNz) liefert die aktuellen Modelldimensionen.

Input

Output
NoRows (I4): Anzahl der Restriktionen
NoCols (I4): Anzahl der Strukturvariablen
NoNz (I4): Anzahl der Nichtnull-Elemente der Koeffizientenmatrix
Rc: 0: alles OK, ≠ 0: Fehlersituation

Beispiel: Rc = GetDim(NoRows, NoCols, NoNz).

6.2.9 DLL-Funktion GetIPSolution()

GetIPSolution (ipsta, Activity, RedCost, Status) liefert die bisher beste gefundene Integer Lösung, falls Lösungen gefunden wurden (xnints > 0).

Input
Die folgenden Arrays dienen zur Aufnahme einer IP-Lösung
Activity(1:xj) (R8), RedCost(1:xj) (R8), Status(1:xj) (I4)

Output
ipsta (I4): Status der IP-Lösung: 0: optimal (search completed), 1: Optimalität nicht bewiesen
Activity(1:xj) (R8): Aktivitäten der Variablen und Restriktionen
RedCost(1:xj) (R8): reduzierte Kosten von Variablen und Restriktionen (Dualwerte)
Status(1:xj) (I4): Status der Variablen in der Lösung: 0: Integer Variable, > 0 kontinuierliche Variable mit folgender Bedeutung: 1: basic, 2: lb, 3:ub, 4: fixed, 5: superbasic
Rc (I4): 0: alles OK, <> 0: Fehler

Beispiel: Es sollen die Werte aller Integer-Variablen > 0 ausgegeben werden:

```
if xnints > 0 then
   Rc = GetIPSolution (lipsta, dActivity, dRedCost, lStatus)
  do j = 1 to xn
     if Status(j) = 0 and dActivity(j) > 0.001 then
       Ausgabe der Werte
     End if
   Enddo
End if
```

6.2.10 DLL-Funktion GetLPSolution()

GetLPSolution (lpsta, Activity, RedCost, Status) liefert die Werte einer optimalen LP-Lösung, d.h. wenn lpsta = 0

Input

Die folgenden Arrays dienen zur Aufnahme einer optimalen LP-Lösung
Activity(1:xj) (R8), RedCost(1:xj) (R8), Status(1:xj) (I4)

Output

lpsta (I4): Status der LP-Lösung: 0: optimal, 1: infeasible, 2: unbounded, 3: nicht optimal
Activity(1:xj) (R8): Aktivitäten der Variablen und Restriktionen
RedCost(1:xj) (R8): reduzierte Kosten von Variablen und Restriktionen (Dualwerte)
Status(1:xj) (I4): Status der Variablen in der Lösung mit folgender Bedeutung: 1: basic, 2: lb, 3: b, 4: fixed, 5: superbasic
Rc (I4): 0: alles OK, <> 0: Fehler

Beispiel: Es sollen die Werte aller Basis-Variablen > 0 ausgegeben werden:

```
Rc = GetLPSolution (lpsta, Activity, RedCost, Status)
do j = 1 to xj
   if Status(j) = 1 and Activity(j) > 0.001 then
     Ausgabe der Werte
   End if
Enddo
```

6.2.11 DLL-Funktion GetMaxDim()

GetMaxDim (lMaxRows, lMaxCols, lMaxNz) liefert die maximal zulässigen Matrixdimensionen. Die Größen werden beim Generieren der DLL festgelegt und sind vom Anwender nicht dynamisch änderbar.

Input

Output

NoRows (I4): maximal mögliche Anzahl der Restriktionen
NCols (I4): maximal mögliche Anzahl der Strukturvariablen
NoNz (I4): maximal mögliche Anzahl der Nichtnull-Elemente der Koeffizientenmatrix
Rc (I4): alles OK, ≠ 0 Fehlersituation

Beispiel: Rc = GetMaxDim(MaxRows, MaxCols, MaxNz).

6.2.12 DLL-Funktion GetModel()

GetModel (intyp, inf, m, n, nz, ia, ja, a, lb, ub, c, typ) übergibt ein komplettes LP/IP-Modell von der MOPS Dll in gegebene Arrays des rufenden Programms. Die Beschreibung des Modellformats ist analog wie in PutModel.

Input

intyp (I4): gewünschte Speicherungsart der Modellmatrix in den Arrays ia, ja, a:
0: Triplets in ia, ja, a, Triplets (i,j,aij) können beliebig sortiert sein
1: Zeilenweise, wobei ia in den Positionen 1,..,m die Startpositionen der Zeilen in den Arrays ja und a enthält.
2: Spaltenweise, wobei ja in den Positionen 1,..,n die Startpositionen der Spalten in den Arrays ia und a enthält

Output

inf (R8): Wert repräsentiert „unendlich (infinity)“, d.h. eine Schranke ist nicht existent, wenn ihr Wert den Betrag von inf hat.
m (I4): Anzahl der Zeilen des Modells
n (I4): Anzahl der Spalten des Modells
nz (I4): Anzahl der Nichtnullelemente des Modells
ia(1:nz) (I4): Zeilenindizes (intyp = 0,2) bzw. Startadressen der Zeilen (intyp = 1)
ja(1:nz) (I4): Spaltenindizes (intyp = 0,1) bzw. Startadressen der Spalten (intyp = 2)
a(1:nz) (R8): Nichtnullelemente der Matrix (Triplets, Zeilenweise, Spaltenweise)

lb(1:n+m) (R8): untere Schranken für Strukturvariablen und Restriktionen
ub(1:n+m) (R8): obere Schranken für Strukturvariablen und Restriktionen
c(1:n) (R8): Zielfunktionskoeffizienten
typ(1:n) (I4): Variablentyp (0: kontinuierlich, <>0 integer)
Rc (I4): 0: alles OK, 2: Kein oder ein fehlerhaftes Modell vorhanden (z.B. n oder m = 0). Dieser Return Code wird auch gesetzt, wenn das interne Modell bereits einem Preprocessing oder einer Skalierung unterzogen wurde.

Bemerkung: alle Arrays müssen ausreichend dimensioniert sein, um das Modell aufzunehmen. Dazu kann vorher z.B. die Dll-Funktion GetDim ausgeführt werden. Weiterhin ist zu beachten, dass die Funktion vor der Optimierung aufgerufen werden muss, da andernfalls das Model stark transformiert (Preprocessing, Skalierung) sein kann.

6.2.13 DLL-Funktion GetNonzero()

GetNonzero(RowIndex, ColumnIndex, Element) liest einen Koeffizienten aus der Matrix.

Input

RowIndex (I4): Zeilenindex
ColumnIndex (I4): Spaltenindex

Output

Element (R8): Matrixelement der Position RowIndex, ColumnIndex.
Rc (I4): 0: alles OK, <> 1: Ungültiger Index

Beispiel: Rc = GetNonzero(1, 2, d) liest das Element in Zeile 1, Spalte 2 und speichert den Koeffizienten in Element.

Hinweis: Durch ein LP/IP-Preprocessing bzw. Skalierung kann das Modell sehr stark vom Ursprungsmodell abweichen. GetNonzero macht i.d.R. nur einen Sinn, wenn xrduce = 0 (kein LP-Preprocssing), xscale = 0 (keine Skalierung) und xnopre 0 = 0 (kein IP-Preprocessing) gesetzt wurde.

6.2.14 DLL-Funktion GetParameter()

GetParameter(Parameter, Wert) ermöglicht das Auslesen von MOPS-Parametern. Der Parametername wird in sParameter übergeben, der Wert des Parameters in Wert als Zeichenkette zurückgeliefert. Parameter können mit der Funktion SetParameter() geändert werden.

Input

Parameter (C6): Name des Parameters

Output

Wert(C80): Aktueller Wert des Parameters
Rc (I4): 0: alles OK, <> 0 Fehler
Beispiel: Rc = GetParameter("xmxmin", Wert) liefert den aktuellen Wert des Parameters xmxmin.

6.2.15 DLL-Funktion GetRow()

GetRow (RowIndex, RowName, RowTyp, LoRhs, UpRhs, Status, Activity, RedCost) liest Informationen einer Restriktion.

Input

RowIndex (I4): Index der Restriktion

Output

RowName: Name (max. 64 Zeichen lang)
RowTyp (I4): Restriktionstyp (1: unbeschränkt; 2: ”<=”; 3: ”>=”; 4: ”=”; 5: ranged)

LoRhs (R8): untere Schranke
UpRhs (R8): obere Schranke
Status (I4): Status der Restriktion im Optimum (1: basis; 2: lb; 3: ub; 4: fix)
Activity (R8): Aktivität der Restriktion im Optimum
RedCost (R8): Reduzierte Kosten der Restriktion im Optimum
Rc (I4): 0: alles OK, 1: ungültiger Index, 2: Name länger als 64 Zeichen

Beispiel: Rc = GetRow(17, RowName, RowTyp, LoRhs, UpRhs, Status, Activity, RedCost) liest die 17te Zeile.

Achtung: Wurde das Modell nicht optimal gelöst, sind die Werte für Status, Activity und RedCost nicht definiert.

Die Variable RowName muß lang genug sein, um den vollständigen Restriktionsnamen aufnehmen zu können (z.B. durch *Dim sRowName as string * 64* oder durch die Funktion *String$()*). Eine „Verlängerung“ der Variablen durch die DLL ist nicht möglich.

6.2.16 DLL-Funktion Mops()

Mops(fnpro) ruft eine standardsierte MOPS-Optimierung analog zur mops.exe auf. Der Parameter fnpro enthält den Dateinamen (mit Pfad) des Profiles. Die Funktion MOPS führt zu Beginn alle Initialisierungen eines Windows- oder DOS- Hauptprogramms aus, d. h. alle bis dahin übergebenen Parameter werden reinitialisiert und es werden **ausschließlich** die Default- bzw. die Parametereinstellungen aus der Datei fnpro verwendet. Es werden folgende Funktionen ausgeführt:

1. Profile einlesen (optional)
2. MPS-Datei eingelesen (bzw. Modellgenerator xcase aufgerufen)
3. LP-Optimierung
4. IP-Optimierung(optional)

Input

fnpro: Dateiname mit maximal 64 Zeichen

Output
Rc (I4):
0 Funktion erfolgreich durchgeführt.
1 Das Einlesen des Profiles verursachte einen Fehler.
2 Das Generieren der Default-Dateinamen verursachte einen Fehler.
3 Das Testen der MOPS-Parameter ist eine ungültige Einstellung entdeckt worden.
4 Es ist ein Fehler beim Einlesen der MPS-Datei aufgetreten.
5 Es ist ein Fehler bei der Ausführung des Modellgenerators aufgetreten.
6 Es ist ein Fehler während der LP-Optimierung eingetreten. (s.a. OptimizeLP()).
7 Das Schreiben der Lösungsdatei (XFNLPS) verursachte einen Fehler.
8 Es ist ein Fehler während der IP-Optimierung eingetreten. (s.a. OptimizeIP()).

LP-Beispiel: Folgendes Modell soll als LP gelöst werden (Parametereinstellungen befinden sich in der Datei c:\mops\dll\xmops.pro). Nach der Optimierung wird der Returncode , der LP-Status und der Zielfunktionwert abgefragt:

```
fnpro = ”c:\mops\dll\xmops.pro”
rc = Mops(fnpro)
rc = GetParameter(”xrtcod”, rc)
If (rc = ”0”) Then
    rc = GetParameter(”xlpsta”, lpsta)
    If (lpsta = ”0”) Then
        rc = GetParameter(”xfunct”, funct)
```

```
    End if
End if
```

IP-Beispiel: In dem Profile sind sämtliche Optimierungsparameter enthalten. Es muß der vollständige Dateiname des Profiles übergeben werden. Nach der Optimierung wird der Returncode, der IP-Status und der Zielfunktionwert der besten ganzzahligen Lösung abgefragt:

```
fnpro = ”c:\mops\xmops.pro”
rc = Mops(fnpro)
rc = GetParameter(”xrtcod”, sreturncode)
rc = GetParameter(”xipsta”, lpsta)
rc = GetParameter(”xzbest”, zbest)
```

Eine ausführliche Beschreibung der MOPS-Parameter erfolgt in einem der nachfolgenden Hilfethemen. In den Beispielen ist auf die Abfrage der Returncodes **verzichtet** worden.

6.2.17 DLL-Funktion InitModel()

InitModel() initialisert die internen Datenstrukturen des Optimierers. Die Funktion muß vor dem Einlesen einer MPS- oder Triplet-Datei und vor der Übergabe eines neuen Modells gerufen werden.

Input

Output

Rc (I4): 0: alles OK, ≠ 0 Fehlersituation

Beispiel: Rc = InitModel()

6.2.18 DLL-Funktion OptimizeIP()

OptimizeIP (FrNod, Nodes, NoIntSol, Errortype, Zbest, LpBound) ruft den IP-Optimierer. Vor Aufruf dieser Funktion muss das zugehörige LP mit OptimizeLP ***optimal*** gelöst worden sein. Die Optimierung wird nach *FrNod* Knoten bzw. wenn eine IP-Lösung gefunden wurde, unterbrochen. Die Funktion kann in einer Schleife so oft ausgeführt werden, bis das Modell vollständig gelöst ist. Soll die Optimierung nach einer Unterbrechung vorzeitig abgebrochen werden, muss die Funktion Break aufgerufen werden.

Input

FrNod (I4): Anzahl Knoten bis zur nächsten Unterbrechung (break)

Output

Nodes (I4): aktuelle Anzahl der Knoten im Branch&Bound-Baum
NoIntSol (I4): Anzahl gefundener IP-Lösungen
Zbest (R8): bisher bester Zielfunktionswert einer IP-Lösung
LpBound (R8): bestmöglicher Zielfunktionswert einer IP-Lösung
Rc (I4): 0: Suche erfogreich beendet
-1: Das LP ist unbeschränkt, unzulässig oder OptimizeLP konnte durch nicht erfolgreich ausgeführt werden z.B. durch Speicherbeschränkung
-2: Exit weil keine ganzzahligen Variablen vorhanden sind
2: Input Error (z.B. Basis oder Suchbaum-Datei)
3: Output Error während der Optimierung
4: numerische Probleme
6: Systemfehler
999: Optimierung planmäßig unterbrochen

Beispiel

Es soll eine IP-Optimierung durchgeführt werden, die nach spätestens 100 Knoten unterbrochen werden soll. Es werden nur einige RC abgefragt.

```
xfrnod = 100
```

```
Do
rc = OptimizeIP(xfrnod, xnodes, xnints, xertyp, xzbest, xzubnd)
If rc = 0 Then
  MsgBox " Die IP-Optimierung ist erfolgreich beendet"
Else If rc = 1 Then
  MsgBox "Ressourcenproblem (Speicher, Zeit, Iterationen)"
End If
Loop Until rc <> 999
```

6.2.19 DLL-Funktion OptimizeLP()

OptimizeLP (FrLog, Iter, NoInfeas, LpStatus, Zfunct, SumInfeas) startet den LP-Optimierer. Diese Funktion muss auch vor einer IP-Optimierung (mit OptimizeIP()) gerufen werden. Die Optimierung wird bei Einsatz der Simplex-Engine (xlptyp <> 4) nach *FrLog* Iterationen unterbrochen. Die Funktion kann in einer Schleife so oft ausgeführt werden, bis das Modell vollständig gelöst ist. Soll die Optimierung vorzeitig abgebrochen werden, muss die Funktion Break aufgerufen werden.

Input

FrLog (I4): Anzahl der Simplex-Iterationen bis zum nächsten break

Output

Iter (I4): Anzahl Iterationen (kumuliert)
NoInfeas (I4): Anzahl Unzulässigkeiten
LpStatus (I4): Status der Optimierung (0: optimal, 1:infeasible, 2: unbounded)
Zfunct (R8): aktueller Zielfunktionswert
SumInfeas (R8): Summe der Unzulässigkeiten
Rc (I4): 0: Modell ist gelöst
999: Optimierung planmäßig unterbrochen
1: Ressourcenlimit erreicht (z.B. Speicher, Iterationslimit, Zeitlimit)
2: Input Error
3: Output Error
4: numerische Probleme
5: noch nicht vergeben
6: Systemfehler

Anmerkung: Mit GetParameter kann mittels xertyp, xline1 und xline2 eine genaue Fehlermeldung ermittelt werden, insbesondere wenn gilt: rc = 1.

Beispiel: :Es wird ein LP gelöst, nach 100 Iterationen wird die Optimierung unterbrochen.

```
xfrlog = 100
Do
    rc = OptimizeLP(xfrlog, xiter, xnif, xlpsta, xfunct, xsif)
   If rc = 0 Then
     MsgBox " Die LP-Optimierung ist erfolgreich beendet.”
   Else If rc = 1 Then
     MsgBox " Ressourcenproblem (z.B. Iterationslimit)"
   Else If rc = 2 Then
     MsgBox " Es ist ein Input-Fehler eingetreten."
   Else If rc = 3 Then
     MsgBox " Es ist ein Output-Fehler eingetreten."
   End If
Loop Until rc <> 999
```

6.2.20 DLL-Funktion PutCol()

PutCol (ColIndex, Mode, ColName, ColTyp, Cost, LoBound, UpBound) fügt eine Strukturvariable (Spalte) in das Modell ein (Mode = 0) bzw. aktualisiert eine Spalte (Mode≠0). Spaltenindexe mit einem Index größer oder gleich ColIndex werden beim Einfügen um 1 erhöht, d.h. wenn Spalte 17 eingefügt wird, wird die bisherige Spalte 17 zu Spalte 18, Spalte 18 zu Spalte 19, usw.

Input

ColIndex (I4): Spaltenindex (Wertebereich 1 bis Anzahl Spalten +1)
Mode (I4): 0: Einfügen, 1: update
ColName: Spaltenname (max. 64 Zeichen)
ColTyp (I4): Variablentyp (0: kontinuierlich, 1: ganzzahlig)
Cost (R8): Zielfunktionskoeffizient
LoBound (R8): untere Schranke
UpBound (R8): obere Schranke

Output

Return-Code (I4):

0 Funktion erfolgreich durchgeführt
1 Fehlersituation: Kein gültiger Spaltenindex
2 Fehlersituation: statisches Limit der DLL erreicht
3 Fehlersituation: Untere Schranke ist größer als obere Schranke
4 Warnung: Namensgleichheitstest kann nicht durchgeführt werden.
5 Warnung: Mindestens eine weitere Spalte besitzt den gleichen Namen

Beispiel: lRc = PutCol(17, 0, "Spalte_17", 1, 10.0, 0, 1) fügt an Position 17 eine neue Binärvariable mit dem Zielfunktionskoeffizienten von 10.0 ein.
Achtung: Die Return-Codes 4 und 5 besitzen nur informativen Charakter. Der Einfüge- bzw. Updateprozeß wird dennoch durchgeführt.

6.2.21 DLL-Funktion PutRow()

PutCol (ColIndex, Mode, ColName, ColTyp, Cost, LoBound, UpBound) fügt eine Strukturvariable (Spalte) in das Modell ein (Mode = 0) bzw. aktualisiert eine Spalte (Mode≠0). Spaltenindexe mit einem Index größer oder gleich ColIndex werden beim Einfügen um 1 erhöht, d.h. wenn Spalte 17 eingefügt wird, wird die bisherige Spalte 17 zu Spalte 18, Spalte 18 zu Spalte 19, usw.

Input

ColIndex (I4): Spaltenindex (Wertebereich 1 bis Anzahl Spalten +1)
Mode (I4): 0: Einfügen, 1: update
ColName: Spaltenname (max. 64 Zeichen)
ColTyp (I4): Variablentyp (0: kontinuierlich, 1: ganzzahlig)
Cost (R8): Zielfunktionskoeffizient
LoBound (R8): untere Schranke
UpBound (R8): obere Schranke

Output

Return-Code (I4):

0 Funktion erfolgreich durchgeführt
1 Fehlersituation: Kein gültiger Spaltenindex
2 Fehlersituation: statisches Limit der DLL erreicht
3 Fehlersituation: Untere Schranke ist größer als obere Schranke

4 Warnung: Namensgleichheitstest kann nicht durchgeführt werden.
5 Warnung: Mindestens eine weitere Spalte besitzt den gleichen Namen

Beispiel: lRc = PutCol(17, 0, "Spalte_17", 1, 10.0, 0, 1) fügt an Position 17 eine neue Binärvariable mit dem Zielfunktionskoeffizienten von 10.0 ein.
Achtung: Die Return-Codes 4 und 5 besitzen nur informativen Charakter. Der Einfüge- bzw. Updateprozeß wird dennoch durchgeführt.

6.2.22 DLL-Funktion PutModel()

PutModel (intyp, dinf, m, n, nz, ia, ja, a, lb, ub, c, typ) übergibt ein komplettes LP/IP-Modell an die MOPS Dll

Input

intyp (I4): Speicherungsart der Modellmatrix in den Arrays ia, ja, aij:
0: Triplets in ia, ja, a, Triplets (i,j,aij) können beliebig sortiert sein
1: Zeilenweise, wobei ia in den Positionen 1,..,m die Startpositionen der Zeilen in den Arrays ja und aij enthält.
2: Spaltenweise, wobei ja in den Positionen 1,..,n die Startpositionen der Spalten in den Arrays ia und aij enthält

- inf (R8): Wert repräsentiert „unendlich (infinity)“, z.B. 1.d20.
- m (I4): Anzahl der Zeilen
- n (I4): Anzahl der Spalten
- nz (I4): Anzahl der Nichtnullelemente
- ia(1:nz) (I4): Zeilenindizes (intyp = 0,2) bzw. Startadressen der Zeilen (intyp = 1)
- ja(1:nz) (I4): Spaltenindizes (intyp = 0,1) bzw. Startadressen der Spalten (intyp = 2)
- a(1:nz) (R8): Nichtnullelemente der Matrix (Triplets, Zeilenweise, Spaltenweise)
- lb(1:n+m) (R8): untere Schranken für Strukturvariablen und Restriktionen
- ub(1:n+m) (R8): obere Schranken für Strukturvariablen und Restriktionen
- c(1:n) (R8): Zielfunktionskoeffizienten
- typ(1:n) (I4): Variablentyp (0: kontinuierlich, <>0 integer)

Output

Rc (I4): 0: alles OK, ≠ 0 Fehlersituation

6.2.23 DLL-Funktion PutNonzeros()

PutNonzeros (NoElements, RowIndices, ColIndices, Aij) fügt *NoElements* Nichtnull-Elemente in die Koeffizientenmatrix ein. PutNonzeros() kann erst verwendet werden, nachdem die Restriktionen und Strukturvariabeln definiert worden sind, d.h. die entsprechenden Indexe in den Arrays müssen vor dem Aufruf von PutNonzeros definiert worden sein.
Für die Modifizierung (Ändern, Löschen) eines Koeffizienten muß lNOElements=1 sein: Ein vorhandenes Element wird gelöscht, wenn es betragsmäßig kleiner als xdropm ist (DEFAULT von xdropm = 1.d-06).

Input

NoElements (I4): Anzahl Elemente, die beim aktuellen Aufruf übergeben werden.
RowIndices (I4): Array für die Zeilenindexe
ColIndices (I4): Array für die Spaltenindexe
Aij (R8): Array für die Nichtnullelemente

Output

Return-Code (I4):
0 Funktion erfolgreich durchgeführt

1 Fehlersituation: keine Restriktionen definiert
2 Fehlersituation: keine Variablen definiert
3 Fehlersituation: ungültiger Zeilenindex
4 Fehlersituation: ungültiger Spaltenindex
5 Fehlersituation: statisches Limit der DLL erreicht

Beispiele

Beispiel für die Übergabe aller Matrixkoeffizienten in Visual Basic:
Dim lRowIndices() As Long, lColIndices() As Long, dAij() As Double, lRc&, lNoElements&
...
Redim lRowIndices(MAX_ARRAYDIM)
Redim lColIndices(MAX_ARRAYDIM)
Redim dAij(MAX_ARRAYDIM)
...
lRc = PutNonzeros(lNoElements, lRowIndices(1), lColIndices(1), dAij(1)).

Beispiel für die Modifikation eines einzelnen Koeffizienten:
lRc = PutNonzeros(1, 4, 2, 10.0) fügt in Zeile 4, Spalte 2 eine 10 ein bzw. überschreibt einen vorhandenen Koeffizienten.

Beispiel für das Entfernen eines Nichtnullelements:
lRc = PutNonzeros(1, 4, 2, 0.0) löscht das Nichtnullelement in Zeile 4, Spalte 2.

6.2.24 DLL-Funktion PutRow()

PutRow (RowIndex, Mode, RowName, Lb, Ub) fügt eine Restriktion (Zeile) in die Koeffizientenmatrix ein (Mode = 0) bzw. aktualisiert eine Zeile (Mode <> 0). Zeilenindexe mit einem Index größer oder gleich lRowIndex werden beim Aktualisieren um 1 erhöht, d.h. wenn Zeile 17 eingefügt wird, wird die bisherige Zeile 17 zu Zeile 18, Zeile 18 zu Zeile 19, usw.

Input

RowIndex (I4): Zeilenindex
Mode (I4): 0: Einfügen, 1: update
RowName: Zeilenname (max. 64 Zeichen)
Lb (R8): untere Schranke
Ub (R8): obere Schranke

Output

Return-Code (I4):
0 Funktion erfolgreich durchgeführt
1 Fehlersituation: ungültiger Zeilenindex
2 Fehlersituation: statisches Limit der DLL erreicht
3 Fehlersituation: Untere Schranke ist größer als obere Schranke
4 Warnung: Namensgleichheitstest kann nicht durchgeführt werden
5 Warnung: Mindestens eine weitere Restriktion besitzt den gleichen Namen

Beispiele

Beispiel einer nicht beschränkten Restriktion. Zuerst wird der MOPS interne Wert für ”unendlich” eingelesen:
DIM xinf As Double, lb As Double, ub As Double
lRc = GetParameter("xinf", sWert)
xinf = CDbl(sWert)
lb = -xinf
ub = xinf
lRc = PutRow(lRowIndex, 0, ”Zeile_1”, lb, ub).

Beispiel einer „≤ 5“ - Restriktion:
lb = -xinf
ub = 5
lRc = PutRow(lRowIndex, 0, ”Zeile_2”, lb, ub).

Beispiel einer „≥ 6“ - Restriktion:
lb = 6
ub = xinf
lRc = PutRow(lRowIndex, 0, ”Zeile_3”, lb, ub).

Beispiel einer „= 0“ - Restriktion:
lb = 0
ub = 0
lRc = PutRow(lRowIndex, 0, ”Zeile_4”, lb, ub).

Beispiel einer Range Row 0 ≤ ... ≤ 5 :
lb = 0
ub = 5
lRc = PutRow(lRowIndex, 0, ”Zeile_5”, lb, ub).

Achtung: Die Return-Codes 4 und 5 besitzen nur informativen Charakter. Der Einfüge- bzw. Updateprozeß wird dennoch durchgeführt.

6.2.25 DLL-Funktion ReadMpsFile()

ReadMpsFile(fnmps) liest eine MPS-Datei und überführt sie in ein internes Modell. Fnmps muss den vollständiger Dateinamen spezifizieren.. Neben AllocateMemory muss vorher auch InitModel aufgerufen werden.

Input

fnmps (C64): Dateiname (inklusive Pfad) der MPS-Datei

Output

Rc (I4): 0: Funktion erfolgreich durchgeführt, ≠ 0 Fehlersituation

Beispiel

Lies und konvertierte die mit SetPatrameter zuvor spezifizierte MPS-Datei:

```
lRc = ReadMpsFile()
If lRc <> 0 Then
   lRc = GetParameter (”XERTYP”, sFehlermeldung)
   If lRc = 0 Then
      Msgbox sFehlermeldung
   End If
   lRc = GetParameter (”XLINE1”, sFehlermeldung)
   If lRc = 0 Then
      Msgbox sFehlermeldung
   End If
   lRc = GetParameter (”XLINE2”, sFehlermeldung)
   If lRc = 0 Then
      Msgbox sFehlermeldung
   End If
End If
```

6.2.26 DLL-Funktion ReadProfile()

ReadProfile(fnpro) liest einen MOPS Profile. Dies ist eine editierbare ASCII-Datei, in der Optimierungsparameter gesetzt werden können.

Input

fnpro (C64): Dateiname (inklusive Pfad) des Profiles

Output

Rc (I4): 0: alles OK, ≠ 0 Fehlersituation

Beispiel

```
lRc = ReadProfile(”c:\mops\dll\burma\xmops.pro”)
If lRc <> 0 Then
   lRc = GetParameter (”XERTYP”, sFehlermeldung)
   If lRc = 0 Then
      Msgbox sFehlermeldung
   End If
   lRc = GetParameter (”XLINE1”, sFehlermeldung)
   If lRc = 0 Then
      Msgbox sFehlermeldung
   End If
   lRc = GetParameter (”XLINE2”, sFehlermeldung)
   If lRc = 0 Then
      Msgbox sFehlermeldung
   End If
End If
```

6.2.27 DLL-Funktion ReadTripletFile()

ReadTripletFile(fntriplet) liest eine Triplet-Datei und überführt sie in ein internes Modell. Fntriplet muss den vollständiger Dateinamen spezifizieren. Neben AllocateMemory muss vorher auch InitModel aufgerufen werden.

Input

fntriplet (C64): Dateiname (inklusive Pfad) der Triplet -Datei

Output

Rc (I4): 0: alles OK, <> 0 Fehlersituation

6.2.28 DLL-Funktion SetParameter()

SetParameter(s) ermöglicht das Setzen von MOPS-Parametern. Alle Parameter werden nach einem Aufruf von NewModel() initialisiert und müssen gegebenenfalls erneut gesetzt werden.

Input

s (max. 80 Zeichen): Parameter und Wert. Syntax: <Parameter>=<Wert>

Output

Rc (I4): 0: Funktion erfolgreich durchgeführt, ≠ 0 Fehlersituation

Beispiel

lRc = SetParameter ("xoutlv = 1 ") setzt den Parameter xoutlv auf 1, d.h. es werden keine Log-Meldungen geschrieben.

6.2.29 DLL-Funktion WriteMpsFile()

WriteMpsFile (**xfnmps**) generiert eine MPS-Datei, wobei xfnmps den vollständigen Dateinamen mit Pfad definiert.

Input

fnmps (C64): Dateiname (inklusive Pfad) der MPS-Datei

Output

Rc (I4): 0: Funktion erfolgreich durchgeführt, ≠ 0 Fehlersituation

6.2.30 DLL-Funktion WriteTripletFile()

WriteTripletFile (**xfnmps**) generiert eine Triplet-Datei, wobei xfnmps den vollständigen Dateinamen mit Pfad definiert.

Input

fnmps (C64): Dateiname (inklusive Pfad) der Triplet-Datei

Output

Rc (I4): 0: Funktion erfolgreich durchgeführt, ≠ 0 Fehlersituation

6.3 MOPS-Fehlermeldungen

Ist der MOPS-Return-Code ungleich Null, dann spezifizieren i.d.R. die MOPS-Parameter xertyp, xline1 und xline2 eine kurze Fehlerbeschreibung (Ausnahme: interne Systemfehler werden nur durch eine Fehlernummer gekennzeichnet). Sie können mittels der Funktion GetParameter() gelesen werden; z. B. GetParameter (xertyp, inhalt), wobei ”inhalt” eine Variable von Typ „Zeichenkette mit 80 Zeichen fester Länge“ ist.

6.4 Die forxxx-Dateien

Die Defaulteinstellungen eines Optimierungslaufs generieren u.U. einige Dateien im aktuellen Unterverzeichnis. Im folgenden wird kurz erläutert, mit welchen MOPS-Parametern entweder das Anlegen verhindert bzw. ein eigener Name verwendet werden kann (mittels der DLL-Funktion SetParameter):

Dateiname	Dateityp	Eigener Name	nicht Anlegen
for007	Log-Datei	XFNLOG = myName.lps	XOUTLVL = 1
for013	LP-Lösung	XFNLPS = myName.lps	XOUTSL = 0
for014	externe Basis	XFNBAO = myName.bao	XFRBAS = -1
for016	IP-Lösung	XFNIPS = myName.ips	XOUTSL = 0
for017	interne Basis	XFNSAV = myName.sav	XFRBAS = -1
for019	Statistik	XFNSTA = myName.sta	XOUTLV = 0
for020	Fehlernachricht	XFNMSG = myName.msg	XOUTLV = 0
for024	Suchbaum	XFNTRE= myName.tre	XFRTRE = -1
for033	Debug	ist nicht vorgesehen	XDEBUG = 0

6.5 Hinweise zur Fehlerbehebung

6.5.1 Falsche oder unvollständige Parameterübergabe

Die Parameter bei dem Aufruf einer DLL-Funktion sind von signifikanter Bedeutung für die Stabilität der Applikation. Die Deklarationen im rufendem Programm muß exakt den Definitionen des Funktionsaufruf entsprechen, ansonsten kann nicht nur die Applikation sondern sogar das Betriebssystem „abstürzen“.

6.5.2 Übergabe von Zeichenketten

Eine häufige Fehlermöglichkeit besteht bei der Übergabe von Zeichenketten an eine DLL-Funktion. Eine Zeichenkette muß vom rufenden Programm ”by value” übergeben werden. Es ist weiterhin sicherzustellen, daß die Zeichenkette ausreichend ”dimensioniert” ist: entweder mit fester oder als variabel lange Zeichenkette, wobei die Zeichenkette vor der Übergabe mit

der maximalen Länge der Rückgabezeichenkette initialisiert werden muß (z.B. mit Leerzeichen).

6.5.3 Arbeiten mit dem MOPS-Parameter xoutlv

Its der Parameter xoutlv (Default 2) > 1, dann wird eine „log Datei“ mit dem Namen fort07 angelegt, in der alle gerufenen MOPS-Hauptfunktionen protokolliert werden. Xoutlv sollte höchstens auf 2 gesetzt werden, da sonst die Datei sehr groß werden kann und die Optimierung durch die Schreibvorgänge auf der Festplatte stark verlangsamt wird.

7 Besonderheiten von MOPS unter diversen Plattformen

Unter dem Betriebssystem VM/CMS sind folgende Besonderheiten zu beachten:

- die Dateinamen in den Variablen xfnxxx müssen Leerzeichen enthalten (Default). Dazu dürfen den Dateinamen xfn*** im Profile keine Namen zugewiesen werden. Die Zuordung der Dateinamen zwischen Unit-Nummer auf der Fortran-Ebene und den physischen Dateinamen auf CMS-Ebene wird in der Kommando-Datei MOPS EXEC vorgenommen.
- Die File-Typen der Standard MOPS-Dateien (siehe 4.1) lauten hier:

fn datmpsx:	MPS Datendatei
fn profile:	Profile
fn lpsolut:	LP-Lösungsdatei
fn ipsolut:	IP-Lösungsdatei
fn basisout:	Output Basisdatei im punch MPS-Format
fn basis:	Input / Output Basisdatei im MOPS-Format
fn tree:	B&B-Baum
fn errors:	Datei für Fehlermeldungen und Warnungen
fn log:	Log-Datei, falls nicht der Bildschirm benutzt wird.

- **MOPS Systemdateien:**

xmops profile:	MOPS Profile
mops exec:	Kommandodatei

- Alle Include Dateien haben den Dateityp copy. Darüber hinaus existiert eine Maclib xmops maclib, in der die Includedateien abgelegt sind.
- der Aufruf von MOPS erfolgt in der Form **MOPS fn**, wobei fn ein Dateiname ist, der die Inputdaten festlegt. Hat xinfor im xmops Profile den Wert 1, so werden die Daten von der MPS-Datei “fn datmpsx *“ gelesen. Bei anderen Werten von xinfor werden keine Inputdaten gelesen.
- Zu beachten ist, daß in jedem Fall die von MOPS generierten Output-Dateien unter dem Namen **fn** mit den entsprechenden Dateierweiterungen (s.o.) gespeichert werden. Dies gilt auch bei der Generierung der Daten im internen Format, so daß die Spezifikation der Dateinamen in jedem Fall erfolgen muß.
- Die MOPS-Kommandoprozedur mops exec generiert einen *nicht permanenten Lademodul,* der anschließend ausgeführt wird. In bestimmten Situationen ist es besser einen permanenten Lademodul *xmops module* zu erzeugen. Dies erfolgt durch Aufruf des Execs genmops. Anschließend kann der Mops Exec wie folgt benutzt werden, um den Lademodul xmops module zu starten: **mops fn go**.
- Wenn Log-Meldungen **nicht** auf dem Bildschirm erscheinen sollen, muß in MOPS EXEC in der Zeile mit der Datei-Definition **FILEDEF FT06F001 DISK &1 LOG** der Stern in Spalte 1 entfernt werden.

8 Fehlermeldungen

Jeder Fehler hat einen eindeutigen Fehlercode, der von der entsprechenden MOPS-Routine in der Variablen xertyp gespeichert wird. Beim Auftreten eines Fehlers kann die Routine xerror gerufen werden. Diese Routine ermittelt anhand des Fehlercodes entsprechende Fehlermeldungen, die sowohl in den Zeichenketten xline1, xline2 als auch in die Fehlerdatei unter dem Namen xfnmsg geschrieben wird, falls xoutlv > 1 ist. In den vorgenerierten Lademoduln mopdos.exp, mopswin.exe wird xerror nur im Hauptprogramm gerufen. Von MOPS-Subroutinen wird xerror nicht gerufen. Im folgenden werden die Fehlermeldungen mit ihrem Fehlercode beschrieben.

0001 LP iteration limit xmiter reached; xmiter is:
0002 LP time limit xmxtlp reached; xmxtlp is:
0003 IP time limit xmxmin reached; xmxmin is:
0004 node LP iteration limit xmitip reached; xmitip is:
0005 insufficient space for LU factors - increase xmxnen; xmxnen is:
0006 node limit xmxnod reached; xmxnod is:
0007 nodes buffer size to small - increase xmxnin; xmxnin is:
0008 disk full - optimization terminated
0009 granted disk space exhausted - increase xmxdsk; xmxdsk is:
0010 xmxnen must be larger than xnzero; xmxnen is.
0011 pivot element in LU-factorization is too small; xpivot is:
0012 disk capacity insufficient to restart IP-optimization!
0015 basis incorrect - basis size is:
0016 basis incorrect - basic bit not set for:
0017 basis incorrect - duplicate index in basis list xh
0018 basis incorrect - index in xh out of range:
0030 open error on file with unit:
0031 read error on file with unit:
0032 write error on file with unit:
0033 rewind error on file with unit:
0034 input file incorrect for your model - unit:
0035 error closing file with unit:

Tritt ein Fehler beim Lesen des Profiles (Fehlernummer 50 bis 55), beim Konvertieren der Daten vom MPS-File (Fehlernummer 100 bis 299) auf, so wird zusätzlich zu der Fehlermeldung noch die Zeilennummer in der Datei angegeben.

0050 name in profile unknown - line:
0054 double precision value in profile incorrect - line:
0055 integer value in profile incorrect - line:

0110 premature end of input file; fn is:
0112 Name card missing in mps-data file
0113 Rows section card missing in mps-data file
0122 No valid rows specified
0123 No objective function specified
0125 illegal row type:
0126 row name blank
0128 row name x at record y already present; row index of duplicate:
0129 row hash table too small - increase xmxnen; xmxnen is:
0130 too many rows - increase xmaxro; xmaxro is:
0131 name space too small - increase xmaxrc; xmaxrc is:
0132 column section card missing.
0152 column name missing.
0156 too many nonzeros - increase xmaxnz.
0157 Column name already present.
0158 Column hash table too small - increase xmxnen.
0159 too many columns - increase xmaxco.
0161 duplicate row entry in column section.

0162 Row name unknown.
0163 illegal numerical value in mps-file at record: x incorrect value:
0164 Nonzero different from 1.0 in sos row.
0165 a non sos variable has a nonzero in sos row.
0166 wrong section card after column section.
0167 magnitude of nonzero too large.
0170 preceeding marker not correctly closed
0171 row name in sosorg marker card not found.
0172 sos row was already defined.
0173 sos row is not equality row.
0180 rhs name missing.
0181 too many nonzeros increase xmaxnz.
0182 rhs value is different from 1.0 in sos row.
0183 rhs value for an N-row is illegal.
0184 xrhs appears twice.
0185 Wrong section card after RHS section.
0186 duplicate row entry in rhs section.
0201 xrange appears twice.
0202 wrong section card after range section.
0203 duplicate row entry in range section.
0222 xbound appears twice.
0223 bound vector name missing.
0224 illegal bound-type.
0225 column name unknown.
0226 bound redefined with different value or type.
0227 problem infeasible due to non-integer bound.
0228 wrong section card after bounds section.
0301 first basis card is not a name card.
0303 Wrong type for basis card.
0353 wrong type for basis card.
0355 end marker E is missing in save basis file
0357 premature end of save basis file

Bei einer Syntaxüberprüfung der IMR (Fehlernummer 400 bis 428) durch die Routine xchkmo oder xchimr können folgende Fehler auftreten:

0400 xm out of range:
0401 xn out of range:
0402 xj is not equal to xm + xn.
0403 xnzero out of range.
0404 xjcp(xn + 1) - 1 is not equal to xnzero.
0405 Pointer xjcp not increasing.
0406 Problem is unbounded
0407 Duplicate row entry in column section.
0408 Row index out of range.
0409 Magnitude of matrix element is smaller than xdropm.
0410 Magnitude of matrix element is larger than xmaxel.
0411 Ratio of biggest to smallest element in a column too large.
0412 System error by checking logicals.
0413 System error by checking bounds.
0414 Inconsistency between number of stored characters and xrcfre.
0415 Inconsistency between number of stored nonzeros and xnzero.
0419 Column zero but xnenta not equal zero.
0420 Fixed bound is plus or minus infinity.
0421 Lower bound larger than upper bound.
0422 Inactivated row in active part of column.
0423 Activate row in inactivated part of column.
0424 xjcp(1) has to be positive.
0425 xrcfre is greater than xrcmax.
0426 xcoptr(1) has to be positive.
0427 Pointer xjcp not increasing.

Die Fehler mit den Fehlernummern 600 bis 602 treten bei der Überprüfung der Wertebereiche von Eingabeparametern durch die Routine xchkpa auf.

0600 floating value is out of range - value is: x; range is from: y to z
0601 integer value is out of range - value is: x; range is from: y to z
0602 xsscal must be 1.d0 or -1.d0; its value is:

0650 input matrix is singular'
0651 dense matrix is singular
0750 a fixed integer variable is nonintegral - index is: x value is:
0759 tree file not correct - wrong tree or tree clobbered
0760 Unresolvable num. problems in heuris at node
0761 Unresolvable num. problems in xipopt at node
0763 disk space too small to restart - increase xmxdsk!
0764 unknown node selection strategy
0800 A fixed bound is plus or minus xinf - xaprow
0801 row index out of range - xaprow
0802 row index not increasing - xaprow
0803 col index out of range - xaprow
0804 lower bound is larger than upper bound
0900 fatal error in model generator

Bei Systemfehlern müssen Modell und Profile zur Fehlerbeseitigung eingesandt werden. Bis zu einer Beseitigung des Fehlers ist es meistens möglich, mit anderen Parametereinstellungen weiterzuarbeiten, da dann meistens ein anderer Lösungspfad genommen wird, der nicht zur Fehlersituation führt. Änderungen folgender Parameter für die LP- und IP-Optimierung könnten dieses Ziel ermöglichen: xlptyp, xautom, xrduce, xbndha, xscale, xstart, xchztp, xdjscl. Während der IP-Optimierung sind alle Parameter zum Supernode-Processing Änderungskandidaten und xiplpt (Art der LP-Reoptimierung an einem Knoten).

9 Übersicht der wichtigsten Eingabeparameter

Im folgenden werden wichtige Parameter mit ihren Default-Werten beschrieben. Hinter dem Namen jeder Variablen steht in Klammern ihr Datentyp. Es bedeuten: Cn Zeichenkette der Länge n, I: Integer (4 Bytes), R: Gleitkommazahl, doppelt genau (8 Bytes), R4: einfache Gleitkommazahl (4 Bytes). Sofern dies nicht anders vermerkt ist, kann jede dieser Variablen auch im MOPS-Profile spezifiziert werden.

xautom (I)
Standardeinstellung: 1
Ist xautom = 1, so wird bei zu langsamer Konvergenz in Phase 2 volles Devex eingeschaltet, wenn der Fill in der letzten Faktorisierung > xminfi ist und xn / xm < xxndxm ist. In Phase 1 wird xchztp = 0 gesetzt, wenn vorher xchztp = 1 gesetzt war.
1: die Pricingstrategie soll automatisch angepaßt werden. sonst: keine Anpassung der eingestellten Pricingstrategie.
siehe auch: xfrchk, xminfi, xxndxm, xdjscl, xchztp

xbatyp (I)
Standardeinstellung: 1
Enthält xfrbas einen Wert größer gleich Null, dann bestimmt xbatyp in welchem Format eine Basis gespeichert wird. Beim internen Basisformat (default) darf das Modell beim Re-Start nicht verändert werden. Das externe Basisformat kann auch bei ähnlichen Modellen, sowie beim Austausch von Basen zwischen unterschiedlichen LP-Systemen verwandt werden.
1: Internes Basis Format, es wird die Datei xfnsav erzeugt. 2: Externes Basis Format (MPS-Format), es wird die Datei xfnbao erzeugt. 3: Es wird sowohl eine interne als auch eine externe Basis erzeugt.
siehe auch: xfrbas, xfnbao, xfnsav, xstart

xbndha (I)
Standardeinstellung: 0
Bestimmt ob im LP-Preprocessing reduzierte Schranken beibehalten werden sollen. In der Regel arbeitet der Duale Simplex am besten, wenn xbndha = 1 und xpreme = 4095
0. reduzierte Schranken werden am Ende des LP-Preprocessings wieder relaxiert 1. reduzierte Schranken werden am Ende des LP-Preprocessings nicht relaxiert
siehe auch: xpreme

xbound (C32)
Standardeinstellung: Leerzeichen
definiert für MPS-Eingabedaten, welcher Bound-Vektor verwendet werden soll. Ist xbound blank, so wird der erste Vektor in der BOUND-Sektion gewählt.
siehe auch: xfnmps, xdata, xobj, xrhs, xrange

xbrheu (I)
Standardeinstellung: 3
Bestimmt das Auswahlverfahren für eine Branchingvariablen.
0. es wird bei Minimierung (Maximierung) eine fraktionelle IP-Variable mit maximalem (minimalem) Zielfunktionswert auf Null gesetzt 1. es wird bei Minimierung (Maximierung) eine fraktionelle IP-Variable mit minimalem (maximalem) Zielfunktionswert auf Eins gesetzt 2. es wird die am stärksten fraktionelle IP-Variable auf Null gesetzt, wenn der fraktionelle Teil kleiner gleich 0.5 ist und sonst auf Eins gesetzt. 3. Bei Minimierung (Maximierung) wird eine fraktionelle IP-Variable mit maximalem (minimalem) Zielfunktionswert auf Null gesetzt, wobei Fixed-Charge-Variablen bevorzugt werden. FC-Variablen haben die Form $\sum_{i \in J_i} x_i \le U \bullet y$ bzw. analog als ≥ Restriktion.

xchksw (I)
Standardeinstellung: 1
legt fest, in welchem Maße das interne Modell syntaktisch überprüft wird.
0: keine Überprüfung >0: Überprüfung aller numerischen Werte und der Zeilen- und Spaltennamen auf Duplizität.
siehe auch: -

xdata (C32)
Standardeinstellung: Leerzeichen
definiert nach welchem Datennamen im MPS-File gesucht werden soll. Ist der xdata blank, so wird das erste Datendeck im MPS-File, gekennzeichnet durch die erste Namen-Karte, gewählt. Abdernfalls wird nach einer Namen-Karte gesucht, deren Namensfeld gleich xdata ist.
siehe auch: xfnmps, xobj, xrhs, xrange, xbound

xdimcn (I)
Standardeinstellung: 1
0: die maximalen Modelldimensionen xnmax, xmmax, xnzmax sind voreingestellt >0: im MPS-Convert werden die maximalen Dimensionen bestimmt
siehe auch:

xdjscl (I)
Standardeinstellung: 1
0: die reduzierten Kosten werden im Pricing nicht skaliert. 1: reduzierte Kosten von Nichtbasis-Variablen werden skaliert (modified Devex) 2: volles Devex
siehe auch: xautom

xdropf (R)
Standardeinstellung: 1.0D-11
Ein Element in einer Zeile / Spalte der LU-Faktorisierung wird null gesetzt, wenn es kleiner gleich xdropf, multipliziert mit dem betragsmäßig größtem Element der Zeile / Spalte ist.
Zulässiger Bereich: [0 - 1.0D-9]

xdropm (R)
Standardeinstellung: 1.0D-7
Ein Datenelement in den Eingabedaten, das kleiner als xdropm ist, wird gleich null gesetzt.
Zulässiger Bereich: [0 - 1.0D-4]
siehe auch: -

xfnbai (C64)
Standardeinstellung: Leerzeichen
Dateiname mit einer Basis im externen MPS-Format, von der die Optimierung gestartet werden soll. Ist diese Datei nicht im lokalen Unterverzeichnis, so muß der volle Pfadname spezifiziert werden. Zu beachten ist, daß Basen nur für ähnliche Modelle sinnvoll sind.
siehe auch: xstart

xfnbao (C64)
Standardeinstellung: Leerzeichen
Dateiname unter dem eine Basis gespeichert wird, falls xfrbas gleich Eins oder Drei ist.
siehe auch: xfrbas, xbatyp

xfnips (C64)
Standardeinstellung: Leerzeichen
Dateiname für zu speichernde Integer-Lösungen wenn xoutsl > 0 gesetzt wurde. Die Struktur ist wie bei LP-Lösungen, wobei jedoch die ganzzahligen Variablen den Status IV erhalten.
siehe auch: xoutsl, xninso

xfnlps (C64)
Standardeinstellung: Leerzeichen
Dateiname für die LP-Lösung, sofern diese in eine Datei ausgegeben werden soll. Diese Ausgabe wird durch den Parameter xoutsl gesteuert.
siehe auch: xoutsl

xfnmps (C64)
Standardeinstellung: Leerzeichen
Dateiname mit MPS-Daten. Ist die Datei nicht im lokalen Unterverzeichnis, so muß der volle Pfadname spezifiziert werden. Beispiel: xfnmps = \u\opti\probl\pilot.mps.
siehe auch: xdata, xobj, xrhs, xrange, xbound

xfnmsg (C64)
Standardeinstellung: Leerzeichen
Name der Fehlermeldungs-Datei.

xfnpro (C64)
Standardeinstellung: 'xmops.pro'
Dateiname für den Profile, in dem fast alle Benutzer-Input-Parameter spezifiziert werden können. Dieser Name kann **nicht** im Profile geändert werden, jedoch in xlpinit.inc.

xfnsav (C64)
Standardeinstellung: Leerzeichen
Dateiname für MOPS-interne Basis. Ist die Datei nicht im lokalen Unterverzeichnis, so muß der volle Pfadname spezifiziert werden.
siehe auch: xbatyp

xfnsta (C64)
Standardeinstellung: Leerzeichen
Name der Statistik-Datei.

xfntre (C64)
Standardeinstellung: Leerzeichen
Dateiname für zu speichernden Branch-and-Bound-Baum bei vorzeitigem Abbruch der IP-Optimierung durch Knoten- oder Zeitbeschränkungen.
siehe auch: xipbeg, xmxnod, xmxmin

xfrbas (I)
Standardeinstellung: 0
Bestimmt die Iterationsfrequenz, nach der die aktuelle Basis gespeichert wird.
-1: die Basis wird nie gespeichert 0: die Basis wird nur bei der Terminierung gespeichert p: wenn p eine positive ganze Zahl ist, wird die Basis nach p Iterationen gespeichert.
siehe auch: xbatyp, xfnbao, xfnsav

xfrinv (I)
Standardeinstellung: 100
p: p muß eine positive ganze Zahl sein. Nach p Iterationen wird eine neue LU-Faktorisierung der Basis bestimmt.
Zulässiger Bereich: (1 - 200)
siehe auch: xluctr, xtolel

xfrlog (I)
Standardeinstellung: 0
Bestimmt nach wieviel LP-Iterationen eine Zeile über Informationen bei der Durchführung einer Simplex-Iteration ausgegeben werden soll. Eine Log-Meldung wird nur dann zu jeder p-ten Iteration geschrieben (p > 0, xfrlog = p), wenn xoutlv = 3 gesetzt wurde. Zu beachten ist, daß die CPU-Zeit zur Optimierung steigt, je häufiger eine Ausgabe erfolgt.
siehe auch: xoutlv

xfrnod (I)
Standardeinstellung: 100
Bestimmt nach wievielen Knoten Zeilenschranken neu berechnet werden sollen, um Rundungsfehler und damit unbeabsichtigtes Abschneiden von Knoten zu vermeiden.
Zulässiger Bereich: [0 - xinf]
siehe auch: -

xfrtre (I)
Standardeinstellung: 0
Bestimmt, ob im Falle einer Zeit- oder Knotenüberschreitung im B&B-Prozeß der B&B-Baum für eine spätere Wiederaufnahme der IP-Optimierung gespeichert werden soll.
≥ 0: der B&B-Baum wird bei vorzeitigem Abbruch unter dem Namen xfntre gespeichert sonst: der Baum wird nicht gespeichert
siehe auch: xfntre, xmxnod, xmxmin, xipbeg

xheutp (I)
Standardeinstellung: 4
Bestimmt ob und welche IP-Heuristik zur Bestimmung einer Anfangslösung verwendet werden soll.
0. die Heuristik wird nicht benutzt 1. eine Teilmenge der ganzzahligen Nichtbasisvariablen wird auf ihren Nichtbasiswert im LP gesetzt 2. 0-1-Variablen, die quasi eins sind werden auf 1 gesetzt, d.h. x(j) wird auf 1 gesetzt, wenn x(j) ≥ 1°- xtolqi 3. nach Option 1 wird Option 2 ausgeführt 4. nach Option 1 werden quasi-ganzzahlige Variablen gerundet und zwar wird x(j) auf 0 gesetzt wenn x(j) ≤ xtolqi ist; x(j) wird auf 1 gesetzt, wenn x(j) ≥ 1°- xtolqi ist.
siehe auch: xmnheu, xtolqi

xinf (R)
Standardeinstellung: 1.0D15
Dient zur Darstellung von plus (bzw. minus) unendlich. Der Wert kann **nicht** im Profile geändert werden, jedoch in xlpinit.inc.
Zulässiger Bereich: (1.0D10 - 1.0D30)

xinfor (I)
Standardeinstellung: 1
Bestimmt das Datenformat der Modelldaten:
1: MPS-Format, sonst: internes Format
siehe auch: xfnmps

xipbnd (R)
Standardeinstellung: bei Minimierung xinf , bei Maximierung -xinf
Es wird nur nach Integer-Lösungen gesucht, die einen Zielfunktionswert aufweisen, der besser als xipbnd ist. Bei Minimierung muß also der Wert einer IP-Lösung kleiner, bei Maximierung größer als xipbnd sein.
Zulässiger Bereich: [-xinf - +xinf]
siehe auch: xrimpr, xlpgap, xglgap

xiplpt (I)
Standardeinstellung: 0
Mit dem Wert der Variablen xiplpt wird festgelegt, mit welchem Algorithmus das LP an einem Knoten in der IP-Phasegelöst werden soll.
0: primaler Simplexalgorithmus
>0: dualer Simplexalgorithmus
siehe auch: xlptyp

xlotst (I)
Standardeinstellung: 1
Bestimmt, ob im Branch-and-Bound-Prozeß an einem Knoten nach dem Setzen einer 0-1-Branching-Variablen (vor der LP-Reoptimierung) logische Tests vom Grad 1 ausgeführt werden sollen
0: es werden keine logischen Tests durchgeführt sonst:es werden logische Tests ausgeführt, wobei Zeilensummen aktualisiert werden. Anmerkung: Bei der LIFO-Knotenauswahlregel werden die logischen Testst grundsätzlich durchgeführt, d.h. es wird intern xlotst = 1 verwendet.
siehe auch: xnodse

xlpgap (R)
Standardeinstellung: xinf
Ist xipbnd = |xinf| und ist xlpgap kleiner als xinf, dann wird nach dem ersten Supernod-Prozessing xzubnd = xfunct * |1. + xlpgap| * xsscal gesetzt. Zu beachten ist, daß ein zu kleiner Wert für xlpgap einen erfolglosen Branch-and-Bound-Prozeß verursachen kann.
Zulässiger Bereich: [0. - xinf]

xlpmip (I)
Standardeinstellung: 1
Mit xlpmip wird festgelegt, ob ein IP-Modell nur als LP gelöst werden soll, oder ob im Anschluß an die LP-Optimierung das IP gelöst werden soll. Der IP-Optimierer wird nur dann aufgerufen, wenn die xlpmip größer Null ist und das Modell Integer-Variablen enthält.
0: ein IP-Modell soll *nur* als LP gelöst werden 1: ein IP-Modell soll im Anschluß an die LP-Optimierung auch als solches gelöst werden.
siehe auch: Kapitel 2.1.1

xlptyp (I)
Standardeinstellung: 0
Mit dem Wert der Variablen xlptyp wird festgelegt, mit welchem Algorithmus das Anfangs-LP gelöst werden soll.
0: primaler Simplexalgorithmus
2: dualer Simplexalgorithmus
4: innere Punkte Verfahren
siehe auch: Kapitel 4.5

xluctr (R)
Standardeinstellung: 2.0D0
Ist kein anderer Faktorisierungsgrund gegeben, so wird eine neue LU-Faktorisierung berechnet, wenn die Anzahl der Elemente in der momentaten LU-Zerlegung dividiert durch die Anzahl der Elemente in der letzen LU-Faktorisierung größer als xluctr ist.
Zulässiger Bereich: [1.1 - 5]
siehe auch: xfrinv, xtolel

xmallc (I)
Standardeinstellung: 1
Ist xmallc > 0, so wird bei der Speicherallokation ein Speicherblock in der Größe von xmreal MB angelegt. Ist xmallc = 0, so wird ein Speicherblock fester Größe ***bsize*** benutzt, der in der Subroutine xmopss definiert wird. Eine Änderung von ***bsize*** muss durch Recompilation von xmopss und Update der Bibliotheken mops.lib bzw. mops.dll wirksam gemacht werden.
Zulässiger Bereich: [0 - maxint]
siehe auch: xmxmin, xfntre, xipbeg

xmreal (I)
Standardeinstellung: 300
Xmreal legt bei der dynamischen Allokation fest, wie viel Megabytes der Speicherblock umfassen soll. acht werden.
Zulässiger Bereich: [100 - 2000]
siehe auch: xmallc

xmxlpt (R)
Standardeinstellung: xinf
Bestimmt die maximale CPU-Zeit in Minuten zur Lösung des Anfangs-LP, d.h. die LP-Optimierung wird nach maximal xmxlpt Minuten terminiert. Ist xfrbas ≥ 0, so wird die Basis gespeichert und die Optimierung kann durch xipbeg = 1 fortgesetzt werden.
Zulässiger Bereich: [0 - xinf]
siehe auch: xmiter, xfnbai, xfnsav, xstart

xmxnod (I)
Standardeinstellung: 100000
Bestimmt die maximale Knotenanzahl für den B&B-Prozeß, d.h. die IP-Optimierung wird nach maximal xmxnod Knoten terminiert. Ist xfrtre ≥ 0, so wird der Baum unter xfntre gespeichert und die Optimierung kann durch xipbeg = 1 fortgesetzt werden.
Zulässiger Bereich: [0 - maxint]
siehe auch: xmxmin, xfntre, xipbeg

xmxdsk (R)
Standardeinstellung: 100. [MB]
Bestimmt die maximale Größe in [MB] des erlaubten Festplattenspeicherplatzes der für die Knotentabellen im Branch-and-Bound-Prozeß benutzt werden darf. xmxdsk ist nur für xnodse > 0 relevant, da für xnodse = 0 die LIFO Knotenauswahlregel verwandt wird, bei der kein Festplattenspeicherplatz benötigt wird. Wird die festgelegte Speichergröße erreicht, dann wird bei xfrtree ≥ 0 der Baum unter xfntre gespeichert (wenn noch genügend Speicherplatz vorhanden ist!) und die Optimierung kann durch xipbeg = 1 fortgesetzt werden.
Zulässiger Bereich: [0 - xinf]
siehe auch: xmxnod, xfntre, xipbeg

xmxmin (R)
Standardeinstellung: xinf
Bestimmt die maximale CPU-Zeit in Minuten für den B&B-Prozeß, d.h. die IP-Optimierung wird nach maximal xmxmin Minuten terminiert. Ist xfrtre ≥ 0, so wird der Baum unter xfntre gespeichert und die Optimierung kann durch xipbeg = 1 fortgesetzt werden.
Zulässiger Bereich: [0 - xinf]
siehe auch: xmxnod, xfntre, xipbeg

xmxpsu (I)
Standardeinstellung: 10
Maximalzahl der Durchgänge im Supernode-Processing bei der alle Techniken ausgeführt werden, um die Schärfe der LP-Relaxation zu verbessern. Das Supernode-Processing wird terminiert nach xmxpsu Durchgängen oder wenn sich die Zielfunktion nicht mehr ändert.
Zulässiger Bereich: [0 - maxint]
siehe auch:

xmxtlp (R)
Standardeinstellung: xinf
Bestimmt die maximale CPU-Zeit in Minuten zur Lösung des Anfangs-LP, d.h. die LP-Optimierung wird nach maximal xmxtlp Minuten terminiert. Ist xfrbas ≥ 0, so wird die Basis gespeichert und die LP-Optimierung kann später fortgesetzt werden.
Zulässiger Bereich: [0 - xinf]
siehe auch: xmxnod, xfrbas, xbatyp

xmnheu (I)
Standardeinstellung: 100
Bestimmt wieviele Knoten des B&B-Prozesses in der Heuristik entwickelt werden sollen. Diese Angabe ist nur relevant, wenn die Heuristik benutzt wird, d.h. xheutp > 0 ist.
Zulässiger Bereich: [0 - maxint]
siehe auch: xheutp

xmaxel (R)
Standardeinstellung: 1.0D-2 * xinf
Ein Matrixkoeffizient darf höchstens so groß sein wie xmaxel. Andernfalls wird ein Fehler diagnostiziert. Der Wert kann **nicht** im Profile geändert werden, jedoch in xlpinit.inc.
siehe auch: xinf

xminfi (R)
Standardeinstellung: 0.2
Ist xautom = 1, so wird bei zu langer Konvergenz volles Devex eingeschaltet, wenn der Fill in der letzten Faktorisierung > xminfi ist und xn / xm < xxndxm ist.
siehe auch: xautom, xdjscl, xxndxm

xminmx (C8)
Standardeinstellung: 'min'
Legt die Optimierungsrichtung fest: Minimierung ('min') oder Maximierung ('max').

xmiter (I)
Standardeinstellung: 100000

Iterationslimit, d.h. nach xmiter Iterationen wird die Optimierung abgebrochen.

xninso (I)
Standardeinstellung: 1
Bestimmt, ob alle gefundenen IP-Lösungen in die Lösungsdatei (definiert durch xfnips) übernommen werden.
≤ 1: es wird nur die beste Lösung gespeichert sonst:es werden alle gefundenen Lösungen gespeichert
siehe auch: xfnips, xoutsl

xnodse (I)
Standardeinstellung: 3
Bestimmt die Knotenauswahlstrategie im Branch-and-Bound-Prozeß
0: LIFO 1: wähle einen Knoten mit bestem Zielfunktionswert je nach Minimierung o. Maximierung 2: wähle einen Knoten mit der kleinsten Summe der Integer Unzulässigkeit 3: Best Projection, wobei xnodse = 2 bis zur ersten IP-Lösung gewählt wird 4. Best Projektion (Originalversion)
siehe auch: Abschnitt 5.9, Leistungstuning bei der IP-Optimierung

xcored (I)
Standardeinstellung: 1
Bestimmt, ob eine Koeffizientenreduktion durchgeführt werden soll.
0: keine Koeffizientenreduktion 1: einfache Koeffizientenreduktion 2: erweiterte Koeffizientenreduktion basierend auf Probing (sehr aufwändig)

ximpli (I)
Standardeinstellung: 2
Bestimmt, in welchem Maße Implikationen, die durch ein Probing ermittelt werden, während des Supernode-Processing eingesetzt werden sollen.
0. es werden keine Implikationen abgeleitet 1. es werden alle Implikationen abgeleitet und gespeichert. Die Implikationen werden während aller logischen Tests benutzt. 2. zusätzlich zu Option 1 wird überprüft, ob die Implikationen-Schnittebenen die LP-Lösung an einem Superknoten verletzen. Ist dies der Fall, werden die Schnittebenen abgeleitet und in der IMR gespeichert.

xclict (I)
Standardeinstellung: 2
Bestimmt in welchem Maße Cliques während des Supernode-Processing eingesetzt werden sollen.
0. es werden keine Cliques abgeleitet 1. es werden alle Cliques gespeichert und während der logischen Tests benutzt. 2. zusätzlich zu Option 1 werden in der LP-Lösung verletzte Clique-Schnittebenen an einem Superknoten in der IMR gespeichert.

ximbnd (I)
Standardeinstellung: 2
Schranken für kontinuierliche Variablen können ggf. durch konditionelle Bound-Implikationen eingeschränkt werden (nur für gemischt-ganzzahlige Modelle).
0: Bound-Implikationen werden nicht benutzt 1: Bound-Implikationen werden nur im Supernode-Processing benutzt 2: Bound-Implikationen werden auch im Branch-and-Bound-Prozeß benutzt

xcovct (I)
Standardeinstellung: 1
Bestimmt, ob durch die LP-Lösung verletzte Überdeckungsschnittebenen an einem Superknoten abgeleitet werden sollen.
0: es werden grundsätzlich keine Cover Cuts abgeleitet 1: verletzte Cover Cuts werden nur am ersten Superknotenam abgeleitet 2: Cover Cuts werden nur bei Verbesserung der Zielfunktion abgeleitet 3: Cover Cuts werden an jedem Superknoten abgeleitet
Beispiel: aus $5\ y_1 + 6\ y_2 + 7\ y_3 \leq 11$, mit $0 \leq y_i \leq 1$ können die Cover Cuts $y_1 + y_3 \leq 1$ und $y_2 + y_3 \leq 1$ abgeleitet werden, sofern sie die LP-Lösung verletzen

xobj (C32)
Standardeinstellung: Leerzeichen
legt für MPS-Eingabedaten fest, welcher Zeilenname als Zielfunktion verwendet werden soll. Diese Zeile muß vom Typ unrestricted (N-Row) sein. Ist XOBJ blank, so wird die erste N-Row im Datendeck als Zielfunktion gewählt.
siehe auch: xfnmps, xdata, xrhs, xrange, xbound

xoutlv (I)
Standardeinstellung: 2
Kontrolliert, ob LOG-Meldungen ausgegeben werden sollen (log-file, Statistiken)
0: keinerlei Ausgaben 1: es werden nur Statistiken bei Terminierung ausgegeben 2: es werden iterative Informationen am Bildschirm ausgegeben 3: nach XFRLOG Iterationen werden Einzelinformationen über eine LP-Iteration ausgegeben
siehe auch: xfnsta, xfrlog

xoutsl (I)
Standardeinstellung: 1
Bestimmt, ob und wie die LP-Lösung in die Datei xfnlps ausgegeben wird.
1: die Lösung wird im MPS-Format ausgegeben; sonst: die Lösung wird nicht ausgegeben
siehe auch: xfnlps, xfnips

xprlev (I)
Standardeinstellung: 0
Bestimmt, in welchem Maße Probing durchgeführt werden soll. Beim Probing werden nicht fixierte 0-1-Variablen versuchsweise auf 0 bzw. 1 gesetzt und alle Implikationen untersucht. Diese Technik kann bei reinen 0-1-Problemen sehr wirkungsvoll sein. Für gemischte Probleme ist sie i.a. weniger geeignet. Daher wird xprlev für gemischte 0-1-Probleme auf 0 gesetzt.
0: Probing wird nur für fraktionelle 0-1-Variablen ausgeführt. 1: für alle 0-1-Variablen, jedoch nur an einem Superknoten 2: an jedem Knoten des B&B-Baumes

xrange (C32)
Standardeinstellung: Leerzeichen
legt für MPS-Eingabedaten fest, welcher Vektor als Range-Vektor verwendet werden soll. Ist XRANGE blank, so wird der erste Vektor in der RANGE-Sektion gewählt.
siehe auch: xfnmps, xdata, xobj, xrhs, xbound

xrduce (I)
Standardeinstellung: 2
Bestimmt, ob LP-Preprocessing aufgerufen wird.
0: kein Preprocessing, 2: Preprocessing
siehe auch: xbndha

xrhs (C32)
Standardeinstellung: Leerzeichen
legt für MPS-Datei fest, welcher Vektor als rechte Seite verwendet werden soll. Ist XRHS blank, so wird der erste Vektor in der RHS-Sektion gewählt.
siehe auch: xfnmps, xdata, xobj, xrange, xbound

xrimpr (R)
Standardeinstellung: 1.0D-4
Nach jeder gefundenen Integer-Lösung mit dem Wert xzbest wird die Schranke xzubnd auf xzbest - xrimpr * |xzbest| * xsscal gesetzt, d.h. die nächste Integer-Lösung soll um xrimpr * 100% besser sein, als die gefundene Lösung. Wird der Branch-and-Bound-Prozeß beendet, dann ist bewiesen, daß die beste gefundene Lösung maximal xrimpr * 100 % vom globalen Optimum entfernt liegt. xsscal ist 1 für Minimierung und -1 für Maximierung.
Zulässiger Bereich: [0 - 1.0D]

xscale (I)
Standardeinstellung: 2
Bestimmt, ob die Koeffizientenmatrix vor der Optimierung skaliert wird.
0: nicht skaliert, 1: zeilenweise skaliert, 2: zeilenweise und spaltenweise Skalierung
siehe auch: -

xstart (I)
Standardeinstellung: 1
Bestimmt die Ermittlung einer Startbasis
0: Die Basis wird durch alle logischen Variablen definiert 1: Es wird eine Crash-Basis erzeugt 2: Es wird von einer externen MPS-basis (xfnbai) gestartet 3: Es wird von einer internen Basis (xfnsav) gestartet
siehe auch: xfnbai, xfnsav

xtold1 (R)
Standardeinstellung: 1.0D-7
Duale Zulässigkeitstoleranz in Phase 1. Reduzierte Kosten müssen in Phase 1 des Simplex- größer als xtold1 sein, damit Variablen in die Basis aufgenommen werden dürfen.
Zulässiger Bereich: [1.0D-10 - 1.0D-1]

xtold2 (R)
Standardeinstellung: 1.0D-7
Duale Zulässigkeitstoleranz in Phase 2. Reduzierte Kosten müssen in Phase 2 des Simplex- größer als xtold2 sein, damit eine Variable in die Basis aufgenommen werden darf.
Zulässiger Bereich: [1.0D-10 - 1.0D-1]

xtolpv (R)
Standardeinstellung: 1.0D-6
Mindestpivotgröße während des Simplex-Algorithmus.
Zulässiger Bereich: [1.0D-8 - 1.0D-3]

xtolx (R)
Standardeinstellung: 1.0D-4
Primale, absolute Zulässigkeitstoleranz, die bei **Beendigung** der LP-Optimierung zusammen mit xtolre Zulässigkeit festlegt, d.h. eine Variable x mit der unteren Schranke lb und der oberen Schranke ub wird dann und nur dann als zulässig betrachtet wenn gilt: $lb-xtolx-xtolre*\|lb\| \le x \le ub+xtolx+xtolre*\|ub\|$. Dies gilt auch für die logischen Variablen.
Zulässiger Bereich: [1.0D-9 - 1.D-3]
siehe auch: xtolre

xtolin (R)
Standardeinstellung: 1.0D-5
Ein Lösungswert für eine Integer-Variable wird nur dann als ganzzahlig betrachtet, wenn er maximal um xtolin vom nächsten ganzzahligen Wert entfernt ist.
Zulässiger Bereich: [1.d-3 - 1.0D-6]

xtolqi (R)
Standardeinstellung: 0.1D
Ein Lösungswert für eine Integer-Variable wird nur dann als quasi ganzzahlig betrachtet, wenn er maximal um xtolqi vom nächsten ganzzahligen Wert entfernt ist.
Zulässiger Bereich: [1.d-3 - 1.0D-6]
siehe auch: xheutp

xtolre (R)
Standardeinstellung: 1.0D-14
Primale, relative Zulässigkeitstoleranz die bei **Beendigung** der LP-Optimierung zusammen mit xtolx Zulässigkeit festlegt, d.h. eine Variable x mit der unteren Schranke lb und der oberen Schranke ub wird dann und nur dann als zulässig betrachtet wenn gilt: lb-xtolx-xtolre*|lb| ≤ x ≤ ub+xtolx+xtolre*|ub|. Dies gilt auch für die logischen Variablen.
Zulässiger Bereich: [0.0 - 1.D-7]
siehe auch: xtolx

xtolx1 (R)
Standardeinstellung: 1.0D-4
Primale, absolute Zulässigkeitstoleranz (Phase I): Eine Variable x mit der unteren Schranke lb und der oberen Schranke ub wird dann und nur dann als zulässig betrachtet wenn gilt: lb-xtolx1-xtolr1*|lb| ≤ x ≤ ub+xtolx1+xtolr1* |ub|. Dies gilt auch für die logischen Variablen.
Zulässiger Bereich: [1.0D-9 - 1.D-3]
siehe auch: xtolr1

xtolr1 (R)
Standardeinstellung: 1.0D-12
Primale, relative Zulässigkeitstoleranz (Phase I): Eine Variable x mit der unteren Schranke lb und der oberen Schranke ub wird dann und nur dann als zulässig betrachtet wenn gilt: lb - xtolx1 - xtolr1 * |lb| ≤ x ≤ ub + xtolx1 + xtolr1 * |ub|. Diese Aussage gilt auch für die logischen Variablen.
Zulässiger Bereich: [0.0 - 1.D-7]
siehe auch: xtolx1

xtolx2 (R)
Standardeinstellung: 1.0D-4
Primale Zulässigkeitstoleranz (Phase II): Eine Variable x mit der unteren Schranke lb und der oberen Schranke ub wird dann und nur dann als zulässig betrachtet wenn gilt: lb-xtolx2-xtolr2*|lb| ≤ x ≤ ub+xtolx2+xtolr2*|ub|. Dies gilt auch für die logischen Variablen.
Zulässiger Bereich: [1.0D-9 - 1.D-3]
siehe auch: xtolr2

xtolr2 (R)
Standardeinstellung: 1.0D-14
Primale, relative Zulässigkeitstoleranz (Phase II): Eine Variable x mit der unteren Schranke lb und der oberen Schranke ub wird dann und nur dann als zulässig betrachtet wenn gilt: lb-xtolx2-xtolr2*|lb| ≤ x ≤ ub+xtolx2+xtolr2*|ub|. Dies gilt auch für die logischen Variablen.
Zulässiger Bereich: [0.0 - 1.D-7]
siehe auch: xtolx2

xtolzr (R)
Standardeinstellung: 1.0D-12
Eine berechnete Gleitkommazahl wird auf Null gesetzt, wenn ihr Betrag nicht größer als xtolzr ist.
Zulässiger Bereich [1.0D-13 - 1.0D-9]

xxndxm (I)
Standardeinstellung: 3
Ist xautom = 1, so wird bei zu langsamer Konvergenz volles Devex eingeschaltet, wenn der Fill in der letzten Faktorisierung > xminfi ist und xn / xm < xxndxm ist
Zulässiger Bereich: [1 - maxint]
siehe auch: xfrchk, xminfi, xxndxm, xdjscl, xchztp

10 Anhang A MPS-Format

Das MPS-Format wird von allen kommerziellen MP-Systemen verarbeitet und sichert daher die höchstmögliche Kompatibiltät zwischen verschiedenen MP-Systemen. Beim MPS-Format werden die Daten in einer Datei gespeichert, die folgenden Aufbau haben muß:

- Datensätze sind sequentiell gespeichert, wobei ein Datensatz maximal 80 Bytes umfaßt
- Ein Datenbestand beginnt mit einem NAME und endet mit einem ENDATA Datensatz
- Der Datenbestand besteht aus einzelnen Sektionen, die durch Schlüsselwörter identifiziert werden: NAME, ROWS, COLUMNS, RHS, RANGES und BOUNDS.
- Folgende Sektionen müssen vorhanden sein: NAME, ROWS und COLUMNS.
- Ein Datenbestand kann mehrere RHS, RANGES und BOUNDS Vektoren aufweisen.
- Kommentare sind durch einen Stern (*) in Spalte eins zu kennzeichnen.

Beispiel (LP-Modell): Min x_4

$0.5\,x_1 - 1\,x_2 + 1\,x_3 + 1\,x_4 = 1$

$0.5\,x_1 - 1\,x_2 - 1\,x_4 = 0$

$-2 \le x_1 \quad x_2 \ge 0,\ x_3,\ x_4$ unbeschränkt

Variablen und Restriktionen werden mit C1, C2, C3, C4 bzw. R1, R2 und R3 bezeichnet:

```
1    5          15          25              40          50
NAME           EXAMPLE
ROWS
 E  R1
 N  R3
 E  R2
COLUMNS
    C1         R2             0.5          R1           5.E-01
    C2         R1          -1.0000E+00     R2          -1.
    C3         R1           1.0000E+00
    C4         R2          -1.0000E+00     R3           1.
    C4         R1           1.0000E+00
RHS
    RHS        R1           1.00000E+00
BOUNDS
 LO B1234      C1          -2.00000E+00
 FR B1234      C3
 FR B1234      C4
ENDATA
```

Die allgemeine Struktur eines Datenbestandes im MPS-Format sieht folgendermaßen aus:

NAME Name des Modells

ROWS

Hier werden Restriktionen und Zielfunktion spezifiziert.

COLUMNS

Hier werden die Variablen sowie deren Nichtnullelemente spaltenweise angegeben.

RHS

Hier werden die Nichtnullelemente von bl oder bu angegeben.

RANGES

Hier können individuelle Schranken für Restriktionen definiert zusätzlich definiert werden.

BOUNDS

Hier werden individuelle Schranken für einzelne Variablen spezifiziert.

ENDATA

Ein Datensatz ist in sechs Felder unterteilt:

	Feld 1	Feld 2	Feld 3	Feld 4	Feld 5	Feld 6

Spalte	2-3	5-12	15-22	25-36	40-47	50-61
Inhalt	Typ	Name 1	Name 2	Wert 1	Name 3	Wert 2

NAME-Sektion

Die NAME-Sektion wird durch einen Datensatz mit dem Schlüsselwort NAME in Spalte 1 definiert. Ab Spalte 15 muß ein Name (Zeichenkette ungleich Blank) beliebiger Länge stehen. Nur die ersten 32 Zeichen sind signifikant.

ROWS-Sektion

Die ROWS-Sektion wird durch einen Datensatz mit dem Schlüsselwort ROWS in Spalte 1 eingeleitet. Danach werden die Restriktionen des Modells durch jeweils einen Datensatz definiert. In Feld 1 wird der Typ einer Restriktion definiert: L bzw. LE für ≤, G bzw. GE für ≥ , E bzw. EQ für = und N für unbeschränkte Restriktionen. Dieser letzte Typ wird für die Zielfunktion vergeben. Darüber hinaus können jedoch auch andere unbeschränkte Restriktionen - z.B. zur Berechnung bestimmter Werte oder als alternative Zielfunktion - diesen Typ erhalten. Die Restriktionstypen müssen in Großbuchstaben definiert werden.

Jede Restriktion muß durch einen eindeutigen Namen als Zeichenkette mit maximal 8 Zeichen in Feld 2 definiert werden. Diese Zeichenkette darf beliebige Zeichen enthalten. Gespeichert werden alle 8 Zeichen, auch die Leerzeichen am Anfang oder am Ende.

COLUMNS-Sektion

Die COLUMNS-Sektion wird durch einen Datensatz mit dem Schlüsselwort COLUMNS in Spalte 1 eingeleitet. In den folgenden Datensätzen werden die Nichtnullelemente der Koeffizientenmatrix des Modells spezifiziert. Ein Nichtnullelement wird durch den Zeilen-Namen der Restriktion, so wie er in der ROWS-Sektion bereits definiert wurde, und durch einen eindeutigen Namen für den Spaltenvektor bestimmt, der nach den gleichen Regeln wie den Namen für die Restriktionen gebildet wird. Der Spaltenname wird in Feld 2 der Zeilenname in Feld 3 und der Wert des Nichtnullelementes in Feld 4 spezifiziert. Ein gültiger Wert besteht aus einer Ziffernfolge, ggfs. mit einem vorangestellten Vorzeichen, einem Dezimalpunkt (Option), einer Ziffernfolge nach dem Dezimalpunkt (Option) und einem Exponenten (optional), (d.h. einer Ziffernfolge die mit D bzw. E beginnt, optional ein Vorzeichen hat und mit einer ganzen Zahl endet). Der Wert braucht nicht links- oder rechtsbündig im Feld ausgerichtet zu sein. Gültige Beispiele sind 27, 0.27, + 0.27, -2., -27.E+5. Wird kein Dezimalpunkt angegeben, so wird die Zahl als ganzzahlig angenommen. In Feld 5 und 6 kann ein weiteres Nichtnullelement derselben Spalte angegeben werden. Alle Nichtnullelemente einer Spalte müssen zusammenhängend gruppiert werden. Innerhalb einer Spalte ist die Reihenfolge der Zeilen jedoch beliebig.

Die Spezifikation von ganzzahligen Modellen Modelle wird im Abschnitt 9.1.1 erläutert.

RHS-Sektion

Die RHS-Sektion wird durch einen Datensatz mit dem Schlüsselwort RHS in Spalte 1 eingeleitet. In den folgenden Datensätzen werden die Nichtnullelemente eines Vektors der rechten Seite spezifiziert. Mehrere Vektoren sind möglich. Im Normalfall wird nur ein Vektor spezifiziert. Jeder Vektor muß einen eindeutigen Namen aufweisen, der in Feld 2 angegeben wird. In den Feldern 3-6 werden die Nichtnullelemente des Vektors aufgeführt. Ein oder mehrere Vektoren werden genauso wie die Spaltenvektoren in der COLUMNS-Sektion gespeichert. Dabei kommen die gleichen Regeln wie in der COLUMNS-Sektion zum Tragen.

RANGES-Sektion

Die RANGES-Sektion wird durch einen Datensatz mit dem Schlüsselwort RANGES in Spalte 1 eingeleitet. In den folgenden Datensätzen werden die Nichtnullelemente eines RANGES-Vektors spezifiziert. Jeder Vektor muß einen eindeutigen Namen aufweisen, der in Feld 2 angegeben wird. In den Feldern 3-6 werden die Nichtnullelemente des Vektors aufgeführt. Da-

bei kommen die gleichen Regeln wie in der COLUMNS-Sektion zum Tragen. Ein RANGES-Vektor dient zur Definition einer zweiseitigen Restriktion:

Ist $l \leq a\ x \leq u$ eine Restriktion, wobei a, x reelle Vektoren und l, u reelle Skalare sind, so kann diese Restriktion mit Hilfe eines RANGES-Vektors definiert werden. Es sei r = u-l. Wir setzen r > 0 voraus. In der ROWS-Sektion kann diese Restriktion entweder als GE oder LE-Restriktion definiert werden. Entsprechend wird in der RHS-Sektion entweder l oder u, sofern ungleich Null, als Koeffizient dieser Zeile definiert. In der RANGES-Sektion wird dann für die entsprechende Zeile als Koeffizient r angegeben. Ein Range-Element kann auch für Gleichungen definiert werden. Es sei b der Koeffizient einer Restriktion, der in der RHS-Sektion definiert wurde, und r der Koeffizient, der in der RANGES-Sektion spezifiziert wurde. Es seien l bzw. u die resultierenden unteren bzw. oberen Schranken der betrachteten Restriktion. Generell gelten folgende Regeln:

Typ	Vorzeichen von r	l	u
G	+ oder -	b	b + |r|
L	+ oder -	b - |r|	b
E	+	b	b + |r|
E	-	b - |r|	b

BOUNDS-Sektion

Die BOUNDS-Sektion wird durch einen Datensatz mit dem Schlüsselwort BOUNDS in Spalte 1 eingeleitet. In den folgenden Datensätzen werden in einem BOUNDS-Vektor individuelle Schranken definiert, die sich vom Standardwert unterscheiden. Im Normalfall, d.h. wenn in der BOUNDS-Sektion für eine Strukturvariable keine Eintragung vorhanden ist, wird eine untere Schranke von Null und eine obere Schranke von unendlich angenommen.

Jeder Vektor muß einen eindeutigen Namen aufweisen, der in Feld 2 angegeben wird. In Feld 3 wird der Name der Strukturvariablen angegeben, für den Schranken redefiniert werden sollen. In Feld 1 steht der Typ der Schranke. Für bestimmte Typen wird im Feld 4 der numerische Wert angegeben. Folgende Fälle sind möglich:

Typ	**Bedeutung / Intervall**	**Wertangabe (J/N)**
LO	untere Schranke	j
UP	obere Schranke	j
FX	fixierter Wert	j
FR	freie Variable	n
PL	$[0, \infty)$	n
MI	$(-\infty, 0]$	n
BV	Binärvariable (0-1)	n
LI	Integer Variable, untere Schranke	j
UI	Integer Variable, obere Schranke	j

Für eine Variable sind mehrere (konsistente) Eintragungen möglich. Z.B. $-20 \leq x \leq 30$ erfordert LO x -20 und UP x 30; $-\infty < x \leq 10$ erfordert MI x und UP x 10.

Spezifikation ganzzahliger Variablen

Die Deklaration ganzzahliger Variablen ist auf zwei Arten möglich:

1. durch Einschluß der Variablen in sogenannten Marker-Datensätzen in der COLUMNS-Sektion. Ein Marker-Datensatz enthält in Feld 1 nur Leerzeichen, in Feld 2 einen eindeutigen Namen in Feld 3 'MARKER', Feld 4 ist Blank und in Feld 6 steht ein Schlüsselwort. Folgende Schlüsselwörter sind erlaubt: 'INTORG', 'INTEND' und 'SOSORG' und 'SOSEND'. Sie haben folgende Bedeutung:

Alle Variablen, die zwischen zwei Marker-Datensätzen mit den Schlüsselwörtern 'INTORG' und 'INTEND' eingeschlossen sind, werden als ganzzahlig betrachtet. Werden keine Spezifikationen über diese Variablen in der BOUNDS-Sektion getroffen, so werden diese Variablen als 0-1-Variablen definiert. Um allgemeine ganzzahlige Werte zu erreichen, müssen also entsprechende Schranken definiert werden.

Variablen, die zwischen Marker-Datensätzen mit den Schlüsselwörtern 'SOSORG' und 'SOSEND' eingeschlossen sind, gehören zu einem sogenannten "special ordered set", kurz SOS genannt. Dies bedeudet, daß eine Restriktion über die eingeschlossenen Variablen generiert wird, in der jeder Koeffizient 1 ist. Darüber hinaus wird automatisch für den Koeffizienten der Rechten Seite der Wert 1 angesetzt. Der Name im Feld 2 des SOSORG-Datensatzes muß in der ROWS-Sektion als EQ-Zeile definiert worden sein. Variablen in einem SOS sind zugleich 0-1-Variablen. Darüber hinaus muß in einer zulässigen ganzzahligen Lösung exakt eine Variable aus dem SOS den Wert Eins aufweisen. Beispiel:

```
ROWS
.     .
 E  SON1102
 .     .
COLUMNS
 .     .
    SON1102   'MARKER'                 'SOSORG'
    Y2N1100
    Y2N1101   KSN1101        1.00000
    Y2N1101   RTN1101       -5.00000
    Y2N1102   RTN1102        5.00000
    Y2N1103   RTN1103        4.00000
    Y2N1104   KSN1104        1.00000
    Y2N1105   RTN1105        2.00000
    Y2N1106   KSN1106       -1.00000
    SEN1999   'MARKER'                 'SOSEND'
 .     .
    IONW999   'MARKER'                 'INTORG'
    XXNW100   GGNW100    -2500.00000
    XXNW100   GONW100   -90000.00000
    IENW999   'MARKER'                 'INTEND'
 .     .
```

Der Name eines SOSEND-Markers, bzw. die Namen von INTORG und INTEND-Markern (in Feld 2) werden im Gegensatz zu MPSX/370 ignoriert, d.h. sie brauchen nicht mehr eindeutig zu sein.

2. durch Spezifikation der Variablen in der Bounds-Sektion. Das Schlüsselwort BV definiert 0-1-Variablen. Durch LI bzw. UI wird eine Variable als ganzzahlig mit der entsprechenden unteren bzw. oberen Schranke definiert.

Konvertierung von MPS-Daten

Die Konvertierung von MPS-Daten und der Aufbau der IMR im wird intern durch Aufruf der Subroutine **xcnvrt** durchgeführt. Dabei wird insbesondere die erforderliche Indizierung der Restriktionen und Variablen durchgeführt und die Daten in entprechenden Feldern im Hauptspeicher gespeichert. Wie diese Speicherung vorgenommen wird, ist in Kapitel 6.3 beschrieben. Die Indizierung der Restriktionen und Variablen erfolgt exakt in der Reihenfolge, wie die Zeilen- und Spaltennamen im MPS-Datendeck angeordnet sind. Während der Konvertierung wird eine Syntaxüberprüfung der MPS-Daten durchgeführt. Der Benutzer benötigt bei Verwendung des MPS-Formates nur dann Kenntnisse über die interne Modell-Repräsentation, wenn das Modell nach einem Convert oder einer Optimierung im Hauptspeicher modifiziert werden soll.

Anmerkungen zu MPS-Daten

- Ein MPS-Datenbestand sollte immer nur ein Datenmodell, ggfs. mit mehreren N-Restriktionen, RHS, RANGES und BOUNDS-Vektoren enthalten.
- Namen werden durch Zeichenketten fester Länge (8 Zeichen) definiert. Bei unterschiedlichen Leerzeichen am Anfang oder Ende eines Namens führen zu Unterschieden. Die Namen bbEGONbb bzw.bbbEGONb wobei b ein Leerzeichen symbolisiert, werden unterschieden.
- Spaltennamen und Zeilennamen werden getrennt gespeichert. Obwohl davon abgeraten wird, für Zeilen und Spalten gleiche Namen zu verwenden, müssen die Mengen der Zeilennamen und der Spaltennamen nicht disjunkt sein.
- In der ROWS-Sektion werden keine Dx-Rows, d.h. Zeilen die lineare Kombinationen anderer Restriktionen sind, unterstützt. Skalierungsfaktoren für Zeilen mit dem Schlüsselwort 'SCALE' in Feld 3 und den Skalierungswert in Feld 4 werden ignoriert.
- In der COLUMNS-Sektion werden keine Skalierungs-Datensätze unterstützt.
- Ganzzahlige Variablen müssen endliche untere und obere Schranken aufweisen die im Intervall [-32767, 32767] liegen müssen. Ist dies nicht der Fall, so werden die entsprechden Schranken ohne Fehlermeldung auf diese Werte gesetzt.

11Anhang B Triplet-Dateiformat

Ab MOPS-Version 5.0 gibt es ein neues Dateiformat, das wie das MPS-Format aus einer sequentiellen Datei mit einzelnen Sektionen besteht.

Sektion 1 enthält eine beliebige Anzahl von Datensätzen mit Kommentaren (***Kommentardatensätze)***, die einen * in Position 1 eines Datensatzes aufweisen müssen. Diese Datensätze sind für die Optimierung belanglos. Die Informationen werden nicht eingelesen.

Sektion 2 besteht aus einem Datensatz, der (***in dieser Reihenfolge***) 4-5 Informationen beinhaltet: die Anzahl der Modellrestriktionen (xm), die Anzahl der Modellvariablen (xn), die Maximalzahl der Nichtnullelemente des Modells (xnzmax), die Maximalzahl der 1-Byte-Zeichen des Modells (xrcmax) und optional einen Gleitkommawert der +∞ repräsentiert, z.B. 1.e+20. Dieser Wert legt fest, ob eine endliche Schranke existiert oder nicht. Die ersten vier Werte müssen ganzzahlig sein. Für xnzmax kann also auch die aktuelle Anzahl der Nichtnullelemente übergeben werden. Das gleiche gilt für xrcmax. Hier gibt es noch folgende Besonderheit: Ist xrcmax = 0, dann werden keine Namen in MOPS gespeichert, auch wenn diese in der Datei vorhanden sind. Ist xrcmax = 1, dann wird xrcmax folgendermaßen berechnet: xrcmax = (xn + xm) * xdfnal, wobei xdfnal standardmäßig den Wert 8 enthält. Ist xrcmax > 1, dann wird dieser Wert zur Dimensionierung des Zeichenarrays verwendet.

Sektion 3 besteht aus xn Datensätzen mit den notwendigen Informationen für alle ***Strukturvariablen***. Jeder Datensatz muss (in dieser Reihenfolge) folgende Daten enthalten: untere Schranke für die Variable, obere Schranke für die Variable, Kostenkoeffizient, Typ (0: kontinuierlich, >0: Integer) und ***optional*** den Namen der Variablen (max. 64 Zeichen). Der Name einer Variablen kann mit " beginnen und enden. Dieses Zeichen sowie Blanks werden von MOPS nicht gespeichert.

Sektion 4 besteht aus xm Datensätzen mit den erforderlichen Informationen für alle ***Restriktionen***. Jeder Datensatz muss (in dieser Reihenfolge) folgende Daten enthalten: untere (interne) Schranke für die entsprechende logische Variable, obere (interne) Schranke für die entsprechende logische Variable, ***optional*** den Namen der Restriktion (max. 64 Zeichen).
Zu beachten ist (vgl. xx.x), dass sich die interne untere bzw. obere Schranke einer logischen Variablen aus dem ***negativen Wert*** von ***bu*** bzw. ***bl*** d.h. den entsprechenden Werten der rechten bzw. linken Seite der Modellrestriktion ergibt (vgl. 5.3).

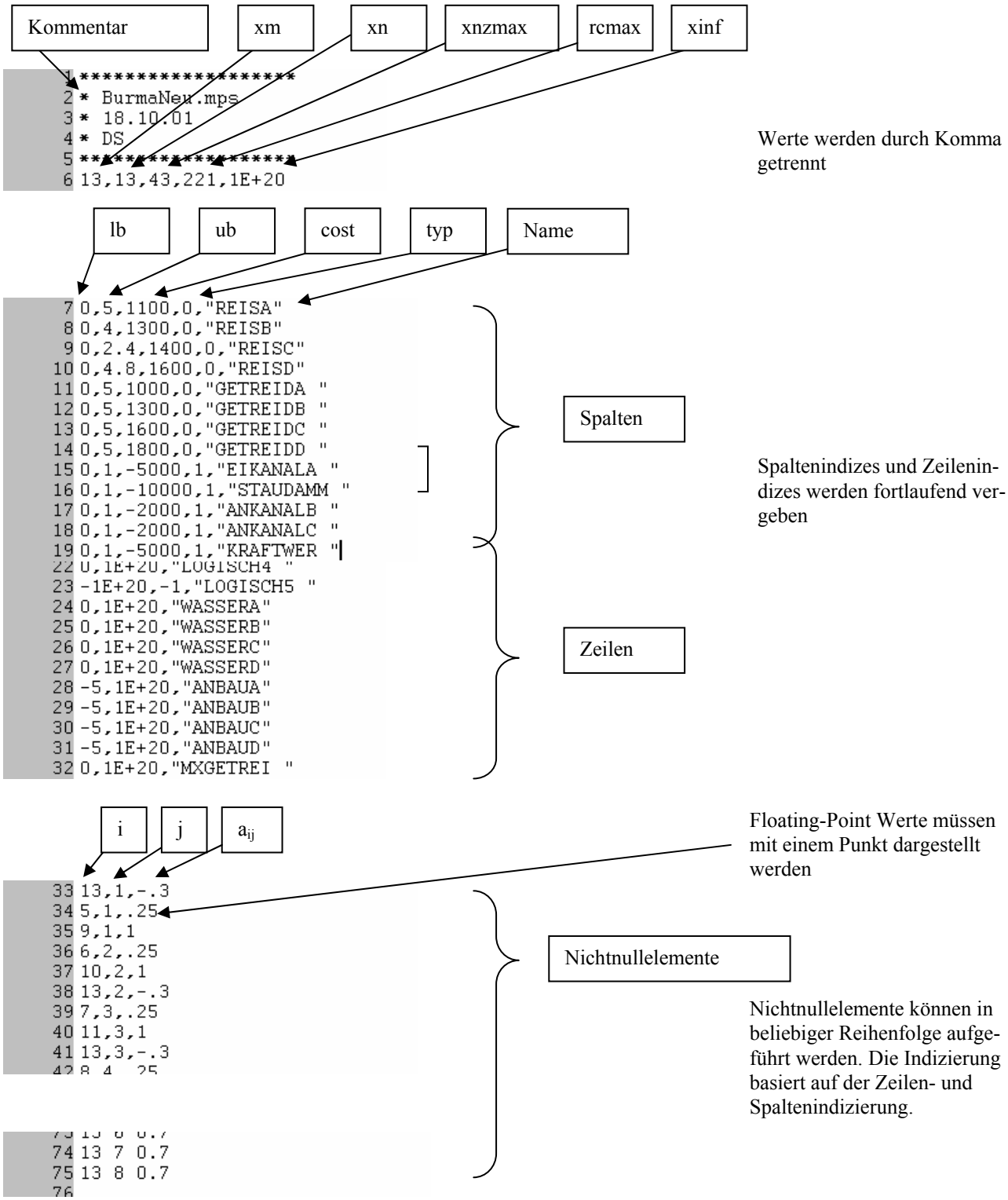
Sektion 5 besteht aus einer beliebigen Anzahl von Datensätzen mit den Nichtnullelementen der Koeffizientenmatrix. Jedes Nichtnullelement wird als Triplet (i,j,a_{ij}) in einem Datensatz gespeichert. Die Indizierung bezieht sich auf die Reihenfolge, wie Strukturvariablen und Restriktionen in Sektion 3 und 4 (indirekt) nummeriert wurden. Während i und j ganzzahlige Größen sein müssen mit $1 \le i \le xm$ und $1 \le j \le xn$, muss a_{ij} ein gültiger Gleitkommawert sein.

Weitere Aspekte bei der Verwendung von Triplet-Dateien sind:

- Kommentardatensätze dürfen nur am Anfang einer Triplet-Datei stehen.
- In allen anderen Datensätzen werden die Daten (Werte, Namen) in Form von Tokens, d.h. zusammenhängenden Zeichenketten gespeichert, die durch Separatoren getrennt werden. Als Separatoren sind möglich: ***Blank, Komma, Semikolon***. Diese Zeichen dürfen nicht in einem Token vorhanden sein. Führende Separatoren werden ebenfalls ignoriert.
- Ganzzahlige Werte dürfen keinen Punkt enthalten, Gleitkommawerte müssen einen Punkt enthalten.
- Sind in der Triplet Datei Namen gespeichert, dann werden diese nur dann eingelesen und in den MOPS-Arrays gespeichert, wenn die Zelle xstorn einen Wert größer Null und enthält. Als Default ist xstorn = 1. Wurde xrcmax = 0 spezifiziert, dann wird xstorn = 0 gesetzt.

- Zum Einlesen einer Triplet-Datei muss ***xfnmps*** deren vollständigen Pfadnamen enthalten, Weiterhin muss die MOPS-Variable xinfor auf den Wert 2 gesetzt werden. Bei Verwendung der DLL muss die Routine ReadMPSFile mit xinfor = 2 verwendet werden.
- Fehlersituationen: 1. Dateifehler (open, rewind, read), 2. maximale Modelldimensionen definiert durch xmmax, xnmax, xnzmax, xrcmax überschritten, 3. Syntaxfehler in den Modelldaten (inkorrekte Integer oder Gleitkommawerte, Token länger als 64 Zeichen, Zeilen oder Spaltenindex inkorrekt), 3. Logische Datenfehler (z.B. lb > ub), 4.Bei doppeltem Eintrag eines Triplet-Wertes, d.h. gleiches Paar (i,j) wird zu einem späteren Zeitpunkt (zu Beginn der LP-Optimierung) eine Fehlermeldung generiert, wenn die Zelle xchksw einen Wert größer Null aufweist.
- Tritt ein Fehler beim Einlesen auf, so ist xrtcod = 2 und xertyp enthält den Fehlercode.

Beispielmodell Burma



12Anhang C: Einbindung der DLL-Funktionen in Visual Basic

Die folgenden Declare-Anweisungen werden benötigt, wenn MOPS-DLL-Funktionen aus einer Visual Basic 4.0 oder höher (32-Bit-Applikation) heraus aufgerufen werden sollen. Die Anweisungen können mit dem Menüpunkt *Bearbeiten|Kopieren* über die Zwischenablage in den Quellcode kopiert werden. Die DLL-Funktionen sind vom Typ 4-Byte Integer.

Warnung: Die Namen der Funktionen sind **case-sensitiv!** Die Groß/Kleinschreibweise der Funktionsnamen darf **nicht** modifiziert werden. Dies gilt ebenso für den Aufruf (Call) der Funtionen.

Hinweis: Folgende Visual Basic Abkürzungen werden verwendet:

&: Variablentyp 4-Byte Integer (Festpunktzahl)
#: Variablentyp 8-Byte Floating Point (Fließkommazahl)
$: Variablentyp x-Byte String (Zeichenkette)

Declare Function **AllocateMemory** Lib "mops.dll" (ByVal lMemory&) As Long

Declare Function **Break** Lib "mops.dll" () As Long

Declare Function **FindName** Lib "mops.dll" (ByVal sName$, ByVal lMode&, lIndex&) As Long

Declare Function **FreeMemory** Lib "mops.dll" () As Long

Declare Function **GetCol** Lib "mops.dll" (ByVal lColIndex&, ByVal sColName$, lColTyp&, lNoElements&, dcost#, dLoBound#, dUpBound#, lStatus&, dActivity#, dRedCost#) As Long

Declare Function **GetDim** Lib "mops.dll" (lNoRows&, lNoCols&, lNoNz&) As Long

Declare Function **GetIPSolution** Lib "mops.dll" (lipsta&, dLPfunct#, dxs#, ddj#, lsta&) As Long

Declare Function **GetLPSolution** Lib "mops.dll" (llpsta&, dLPfunct#, dxs#, ddj#, lsta&) As Long

Declare Function **GetNonzero** Lib "mops.dll" (ByVal lRowIndex&, ByVal lColIndex&, dElement#) As Long

Declare Function **GetMaxDim** Lib "mops.dll" (lMaxRows&, lMaxCols&, lMaxNz&) As Long

Declare Function **GetModel** Lib "mops.dll" (ByVal lintyp&, inf#, m&, n&, nz&, ia&, ja&, a#, lb#, ub#, c#, typ&) As Long

Declare Function **GetParameter** Lib "mops.dll" (ByVal sparameter As String, ByVal swert As String) As Long

Declare Function **GetRow** Lib "mops.dll" (ByVal lRowIndex&, ByVal sRowName$, lRowTyp&, dLoRhs#, dUpRhs#, lStatus&, dActivity#, dRedCost#) As Long

Declare Function **InitModel** Lib "mops.dll" () As Long

Declare Function **Mops** Lib "mops.dll" (ByVal fnpro As String) As Long

Declare Function **OptimizeIP** Lib "mops.dll" (ByVal lFrNod&, lNodes&, lNoIntSol&, lErtyp&, dZf#, dLpBound#) As Long

Declare Function **OptimizeLP** Lib "mops.dll" (ByVal lFrLog&, lIter&, lNoInfeas&, lLpStatus&, dZf#, dSumInfeas#) As Long

Declare Function **PutCol** Lib "mops.dll" (ByVal lColIndex&, ByVal lMode&, ByVal sColName$, ByVal lColTyp&, dcost#, dLoBound#, dUpBound#) As Long

Declare Function **PutModel** Lib "mops.dll" (ByVal lintyp&, ByVal inf#, ByVal m&, ByVal n&, ByVal nz&, ia&, ja&, a#, lb#, ub#, c#, typ&) As Long

Declare Function **PutNonzeros** Lib "mops.dll" (ByVal lNoElements&, lRowIndices&, lColIndices&, dAij#) As Long

Declare Function **PutRow** Lib "mops.dll" (ByVal lRowIndex&, ByVal lMode&, ByVal sRowName$, dLoRhs#, dUpRhs#) As Long

Declare Function **ReadMpsFile** Lib "mops.dll" (ByVal fnmps As String) As Long

Declare Function **ReadProfile** Lib "mops.dll" (ByVal fnpro As String) As Long

Declare Function **ReadTripletFile** Lib "mops.dll" (ByVal fnmps As String) As Long

Declare Function **SetMaxIPDim** Lib "mops.dll" (ByVal lncli&, ByVal limp&, ByVal lnzcl&, ByVal lmaxno&) As Long

Declare Function **SetMaxLPDim** Lib "mops.dll" (ByVal lm&, ByVal ln&, ByVal lnz&, By-Val llunz&, ByVal lrcn&, ByVal ladm&, ByVal ladn&, ByVal ladnz&) As Long

Declare Function **SetParameter** Lib "mops.dll" (ByVal s$) As Long

Declare Function **WriteMpsFile** Lib "mops.dll" (ByVal fnmps As String) As Long

Declare Function **WriteTripletFile** Lib "mops.dll" (ByVal fnpro As String) As Long

13Anhang D: Einbindung der DLL-Funktionen in Visual C

Die folgende Header-Datei muss in einem C/C++ Programm eingebunden werden, wenn MOPS-Funktionen aufgerufen werden sollen.

```
//-----------------------------------------------------------------------------------------------
// mopsdll.h
// This header has to be included, when the MOPS dll is used from c/c++ programs
//-----------------------------------------------------------------------------------------------
typedef UINT (__stdcall *ALLOCATEMEMORY)(int);
typedef UINT (__stdcall *BREAK)();
typedef UINT (__stdcall *DELCOL)(int);
typedef UINT (__stdcall *DELROW)(int);
typedef UINT (__stdcall *FINDNAME)(char*, int, int*);
typedef UINT (__stdcall *FREEMEMORY)(void);
typedef UINT (__stdcall *GETCOL)(int, char*, int*, int*, double*, double*, double*, int*,
double*, double*);
typedef UINT (__stdcall *GETDIM)(int*, int*, int*);
typedef UINT (__stdcall *GETIPSOLUTION)(int*, double*, double*, double*, int*);
typedef UINT (__stdcall *GETLPSOLUTION)(int*, double*, double*, double*, int*);
typedef UINT (__stdcall *GETMAXDIM)(int*, int*, int*);
typedef UINT (__stdcall *GETMODEL)(int, double*, int*, int*, int*, int*, int*, double*,
double*, double*, double*, int*);
typedef UINT (__stdcall *GETNONZERO)(int, int, double*);
typedef UINT (__stdcall *GETPARAMETER)(char*, char*);
typedef UINT (__stdcall *GETROW)(int, char*, int*, double*, double*, int*, double*, dou-
ble*);
typedef UINT (__stdcall *INITMODEL)(void);
typedef UINT (__stdcall *MOPS)(char*);
typedef UINT (__stdcall *NEWMODEL)();
typedef UINT (__stdcall *OPTIMIZEIP)(int, int*, int*, int*, double*, double*);
typedef UINT (__stdcall *OPTIMIZELP)(int, int*, int*, int*, double*, double*);
typedef UINT (__stdcall *PUTCOL)(int, int, char*, int, double*, double*, double*);
typedef UINT (__stdcall *PUTMODEL)(int, double, int, int, int, int*, int*, double*, double*,
double*, double*, int*);
typedef UINT (__stdcall *PUTNONZEROS)(int, int*, int*, double*);
```

```
typedef UINT (__stdcall *PUTOBJ)(int, double);
typedef UINT (__stdcall *PUTROW)(int, int, char*, double*, double*);
typedef UINT (__stdcall *READMPSFILE)(char*);
typedef UINT (__stdcall *READPROFILE)(char*);
typedef UINT (__stdcall *READTRIPLETFILE)(char*);
typedef UINT (__stdcall *SETMAXIPDIM)(int, int, int, int);
typedef UINT (__stdcall *SETMAXLPDIM)(int, int, int, int, int, int, int, int);
typedef UINT (__stdcall *SETPARAMETER)(char*);
typedef UINT (__stdcall *WRITEMPSFILE)(char*);
typedef UINT (__stdcall *WRITETRIPLETFILE)(char*);
```

14Literatur

[1] Brearley, A.L., Mitra, G. and Williams, H.P., Analysis of mathematical programming problems prior to applying the simplex algorithm, Math. Prog. 8, 54-83, 1975.

[2] Crowder, H., E.L. Johnson and M.W. Padberg, Solving large-scale zero-one linear programming problems, Oper. Res. 31, 803-834, 1983.

[3] Dietrich, B.L., L.F. Escudero and F. Chance, Efficient reformulation for 0-1 programs - methods and computational results, Discrete Applied Mathematics 42, 147-175, 1993.

[4] Hoffman, K. and M.W. Padberg, Improving LP-representations of zero-one linear programs for branch-and-cut, ORSA Journal on Computing 3, 121-134, 1991.

[5] Johnson, E.L., M.M. Kostreva and U.H. Suhl, Solving 0-1 integer programming problems arising from large-scale planning models, Oper. Res. 35, 803-819, 1985.

[6] Mészáros, Cs. (1996), The Efficient Implementation of Interior Point Methods for Linear Programming and their Applications, Ph.D. Thesis, Eötvös Loránd University of Sciences, Ph.D. School of Operations Research, Budapest

[7] Mészáros, Cs.(1996), Fast Cholesky factorization for interior point methods of linear programming. Computing & Mathematics with Application, 31(4/5), 49-54

[8] Mészáros, C. and U.H. Suhl, (2003) Advanced preprocessing techniques for linear and quadratic programming, OR Spectrum, 25(4), Springer, 575-595

[9] Savelsbergh, M.W.P., Preprocessing and Probing Techniques for Mixed Integer Programming Problems, ORSA Journal on Computing, 6(4), 445-454, 1994.

[10] Suhl, U.H., Solving large-scale mixed integer programs with fixed charge variables, Math. Programming 32, 165-182, 1985.

[11] Suhl, U.H. and Suhl, L.M., Computing Sparse LU Factorizations for Large-Scale Linear Programming Bases, ORSA Journal on Computing 2, 325-335, 1990.

[12] Suhl, L.M. and Suhl, U.H., A Fast LU-update for linear programming, Annals of Operations Research 43, 33-47, 1993.

[13] Suhl, U.H., MOPS - **M**athematical **OP**timization **S**ystem, European Journal of Operational Research 72, 312-322, 1994.

[14] Suhl, U.H. and R. Szymanski, Supernode Processing of Mixed-Integer Models, Computational Optimization and Applications 3 (1994), 317-331.

[15] Suhl, U.H. and Hilbert, H., A Branch-and-Cut-Algorithm for Solving Generalized Multiperiod Steiner Problems in Graphs, Networks 31(4), 273-282, 1998, John Wiley & Sons, New York

[16] Suhl, U.H.; Suhl, L.M., Solving Airline Fleet Scheduling Problems with Mixed-Integer Programming, in: Ciriani, T.; Gliozzi, S., Johnson, E.L., Tadei, R. (Eds.): Operational Research in Industry, MacMillan Press, London 1999, 135-156.

[17] Suhl, U.H., MOPS - **M**athematical **OP**timization **S**ystem, OR News, 8 (2000), 11-16.

[18] Terlaky, T. [Ed.], Interior point methods of mathematical programming", Kluwer Academic Publishers, 1996

[19] Vanderbei, R.J. (1997), Linear Programming: Foundations and Extensions, Kluwer Academic Publishers

[20] Williams, H.P., Model Building in Mathematical Programming, John Wiley, 1984.

[21] Wolsey, L.A., Tight formulations for mixed integer programming: A survey, Math. Programming 45, 173-191, 1989..