7 Implementierungsaspekte

7.1 MOPS – ein Mathematisches OPtimierungsSystem

Implementiert wurden die vorgestellten Techniken im Optimierungssystem MOPS.

MOPS ist eine Hochleistungssoftware zur Lösung von linearen und gemischt-ganzzahligen Modellen. Im Jahr 1987 von U.Suhl ins Leben gerufen, wird MOPS seitdem fortlaufend weiter entwickelt. Durch Ergänzung von neuen mathematischen Konzepten, der Verbesserung von Algorithmen oder durch die Umsetzung von Fortschritten im Datenmanagement, können immer wieder bessere Lösungszeiten erzielt werden. Tabelle 7.1 gibt einen Überblick über die Entwicklung von MOPS.

Year	Version	Description
1987	1.0	primal simplex, LU-factorization and PFI-update
1988	1.1	LU-update of the basis factorization
1989	1.2	LP-preprocessing, update
1991	1.3	new pivot row selection minimizing the sum of infeasibilities
1992	1.4	new scaling, ftran, devex
1994	2.0	Mixed 0-1-programming with IP Preprocessing
1995	2.5	Mixed integer programming with general node selection
1997	3.0	additional dual algorithm for branch-and-bound phase
1998	3.5	Improved Supernode processing
1999	4.0	additional interior point algorithm to solve the initial LP
2001	5.0	new memory management, improved numerical kernels
2003	6.0	Improved Supernode processing with lifted cover cuts
2003	7.0	Fixed charge and general bound reduction by solving LPs
2004	7.5	New dual simplex algorithm for initial LP and branch-and-bound
2004	7.6	Improved Supernode processing with Gomory cuts
2005	7.7	Improved dual simplex algorithm, extended dll interface
2006	7.8	improved primal simplex algorithm, AMPL interface
2007	8.0	Improved Supernode processing with MIR cuts

Tabelle 7.1: Entwicklung der MOPS Versionen

In verschiedenen internationalen wissenschaftlichen Publikationen sind die einzelnen in MOPS angewandten Techniken und Algorithmen dokumentiert. Dazu zählen u. a. [Suhl94; Suhl00] über den Primalen Simplex und die Systemarchitektur, [MéSu03] über das LP-Preprocessing, [SuSz94; SuWa04] über das Supernode processing und [KoSu06] über den Dualen Simplex und die Duale Phase I.

Die über die Jahre hinweg erzielten Fortschritte sollen anhand der Lösungszeiten des Modells OIL hier nochmals veranschaulicht werden. Es handelt sich dabei um ein verhältnismäßig kleines Modell mit 5563 Restriktionen, 6181 Variablen, wovon 74 Binärvariablen sind, und 39597 Nicht-Null-Elementen in der Koeffizienten Matrix. In Tabelle 7.2 werden die Fortschritte der LP-Optimierung dargestellt und in Tabelle 7.3 die der IP-Optimierung.

Jahr	**Version**	**Oil (5563 x 6181)**	**Sec.**
1991	1.4	I486 (25 MHz)	612,4
1995	2.5	P133 Win 3.11	20,7
1999	4.0	PIII (400 MHz), Win 98	5,1
2001	5.0	PIII (500 MHz), Win 98	3,9
2002	6.0	PIV (2,2 GHz), Win 2000	0,9
2004	7.6	PIV (3,0 GHz), Win 2000, primal	1,1
2004	7.8	PIV (3,0 GHz), Win 2000, dual	1,6
2007	8.0	PIV (3,0 GHz), Win 2000, IPM	0,3

Tabelle 7.2: Fortschritte der LP-Optimierung am Modell Oil

Jahr	Version	Oil (5563 x 6181)	Sec.
1994	2.0	PII (500 MHz) LIFO-MIP	1794,3
1995	2.5	PII (500 MHz), general node selection	450,1
1999	4.0	PIV (2,2 GHz), IPM for initial LP	75,2
2003	6.3	PIV (2,2 GHz) various improvements	39,6
2005	7.8	PIV (3,0 GHz) Gomory cuts, dual in	9,8
2007	8.0	PIV (3,0 GHz) MIR-Cuts	7,8

Tabelle 7.3: Fortschritte der IP-Optimierung am Modell Oil

Die im Laufe der Jahre durch Forschung und Entwicklung entstandenen über 1000 Prozeduren sind hauptsächlich in FORTRAN77 geschrieben. MOPS kann auf einer großen Auswahl von Systemplattformen angewandt werden. Darüber hinaus kommt es bereits in einer Vielzahl

von praktischen Anwendungen zum Einsatz. Es handelt sich um eines der weltweit schnellsten Systeme zur Lösung von Optimierungsproblemen.

7.1.1 Zugriffsmöglichkeiten auf MOPS

In folgendem Abschnitt soll auf die externen Zugriffsmöglichkeiten auf MOPS eingegangen werden. Für einen Anwender liegt MOPS in vier verschiedenen Varianten vor, die hier kurz vorgestellt werden sollen.

Die **MOPS.lib** (object code libary): Der Anwender kann direkt auf die interne Modellstruktur, kurz IMR (internal model representation), bzw. die einzelnen Lösungsroutinen zugreifen. Dazu stehen eine C/FORTRAN-Schnittstelle und eine IMR-Schnittstelle zur Verfügung. Die MOPS.lib wird somit zum Teil des Anwenderprogramms. Sie steht in einer 32bit und in einer 64bit Version zu Verfügung. Mit der 64bit Version kann ein entschieden größerer Speicherblock angelegt werden und somit können größere Modelle gelöst werden.

Die **MOPS.dll** (dynamic link libary): Sie stellt Funktionen bereit, die aus verschiedenen Anwenderprogrammen wie z.B. FORTRAN, Visual Basic (.Net), C++, C# und Java aufgerufen werden können. Die MOPS.dll basiert auf der MOPS Libary.

Die **MOPS.exe** (portable executable): MOPS liegt als eine ausführbare Datei vor. Die erforderlichen Parametereinstellungen erfolgen mittels einer Textdatei, einem sogenannten Profile. Das zu lösende Problem, kann entweder aus dem MPS-Format oder aus dem Triplet-Format eingelesen werden. Textdateien werden des Weiteren für die LP-Basen und Lösungen des Baums, sowie für Statistiken und die Lösung des Problems verwendet.

MOPS Studio Das MOPS Studio ist eine benutzerfreundliche, interaktive Oberfläche zu AMPL[FoGK02] und zur MOPS.dll, d.h. Modelle können interaktiv in AMPL formuliert werden und per Mausklick optimiert werden. Eine optimale LP/IP-Lösung kann dann visualisiert werden.

ClipMOPS (MS Excel Add-In): ClipMOPS ist ein Excel-Add-in, das LP/IP-Modelle bis zu 250 Spalten und 400 Restriktionen im Tableau-Format in Excel mit Hilfe der MOPS-Dll löst.

Welche Möglichkeit auf MOPS zuzugreifen am besten ist, hängt von dem Anwenderprogramm ab bzw. von der Problemstellung, die mit MOPS gelöst werden soll.

Abbildung 7.1 soll die externen Zugriffsmöglichkeiten und den in diesem Zusammenhang relevanten Aufbau von MOPS verdeutlichen.

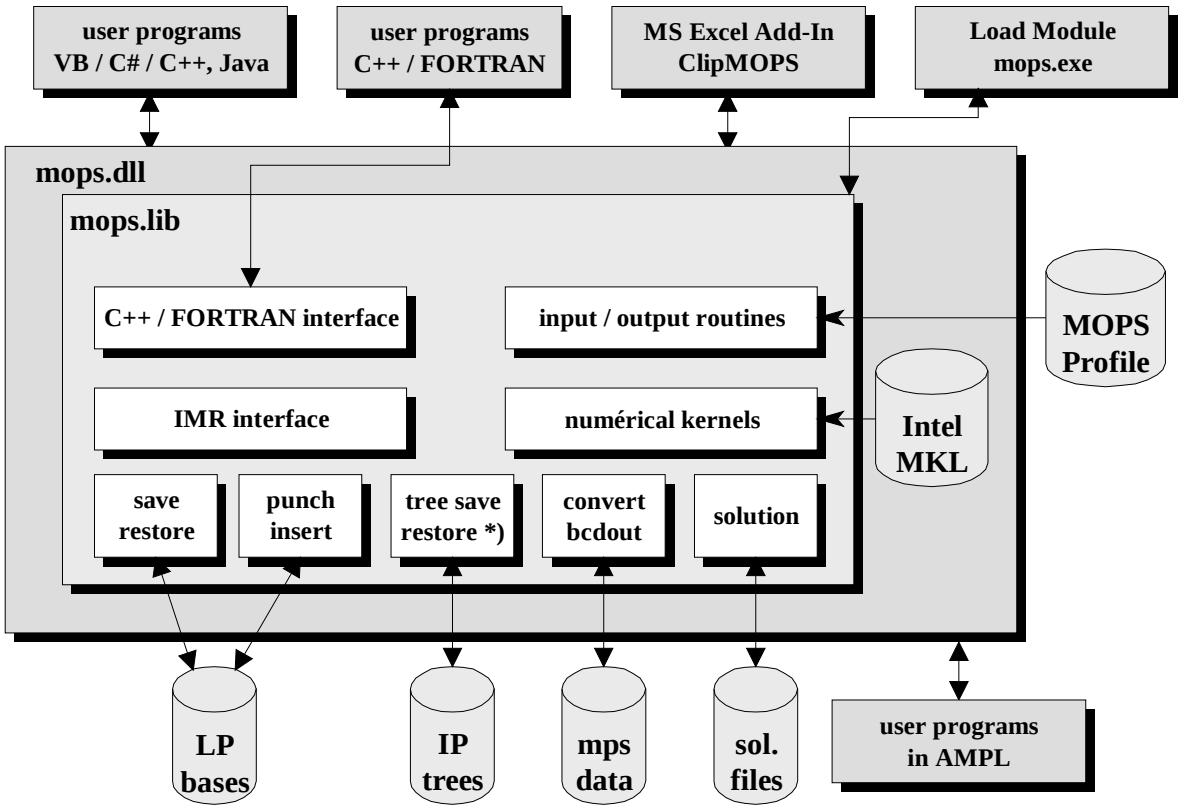

Abbildung 7.1: Externe Zugriffsmöglichkeiten auf MOPS

7.1.2 Interner Aufbau von MOPS

Die MOPS-internen Vorgänge zur Lösung eines Problems können in drei Phasen unterteilt werden. Dabei handelt es sich um eine Datenaufbereitungsphase, die LP-Lösungsphase und die IP-Lösungsphase. Handelt es sich bei dem zu lösenden Problem um ein Modell ohne ganzzahlige Variablen, ist das Modell schon nach der LP-Lösungsphase gelöst.

Im Rahmen der Datenaufbereitungsphase wird zunächst ein großer Speicherblock (b) angelegt (Allocate Memory). Dieser Speicherblock ist normalerweise 512MB groß, kann aber vom Anwender angepasst werden. Die einzelnen Arrays, die im Weiteren benötigt werden, sollen in diesem Speicherblock untergebracht werden. Nach dieser Vorbereitung werden die Modelldaten eingelesen. Sie können aus dem MPS- oder Triplet-Format eingelesen werden oder mittels Dll-Funktionen an MOPS übergeben werden. Das zu lösende Modell wird in MOPS interne Modellstruktur (IMR) umgewandelt. Damit ist die Datenaufbereitungsphase abge-

schlossen. Die benötigten Arrays sind angelegt und die Modelldaten liegen in gewünschter Form vor.

Es kann zur LP-Lösungsphase übergegangen werden. Diese beginnt mit dem LP-Preprocessing. Die Intensität des LP-Preprocessing hängt von den Modelleigenschaften und dem gewählten Lösungsverfahren ab. Handelt es sich beispielsweise um ein MIP-Modell wird das LP-Preprocessing nicht im vollen Maße ausgeführt, da dadurch das Supernode processing in seiner Effektivität beeinflusst werden würde.

Nach dem LP-Preprocessing wird das Modell standardmäßig skaliert.

Die eigentliche Lösung des linearen Problems kann wie bereits erwähnt, mittels dreier verschiedener Verfahren durchgeführt werden. Dem Primal Simplex Verfahren, dem Dual Simplex Verfahren oder dem Interior-Point Verfahren. Wird eines der beiden Simplex Verfahren verwendet oder auch beide, wird zunächst eine Ausgangsbasis gesucht (Crash-Verfahren). Kommt das Interior-Point Verfahren zur Anwendung, muss in der Regel anschließend während eines sogenannten Crossovers eine optimale Basis zu der optimalen Lösung gefunden werden. Eine optimale LP-Lösung des im LP-Preprocessing reduzierten Modells wird im Rahmen des Postsolves in eine optimale Lösung des Originalmodells transformiert.

Handelt es sich bei dem zu lösenden Problem um eines, das keine ganzzahligen Variablen beinhaltet, ist der Lösungsvorgang damit an dieser Stelle abgeschlossen.

Sind jedoch auch ganzzahlige Variablen Teil des Modells, muss zusätzlich noch die dritte Phase, die IP-Lösungsphase durchlaufen werden. Zu Beginn wird das Supernode processing durchgeführt, dabei kommen Schnittebnen und andere Techniken zum Einsatz, um den möglichen Lösungsraum für die IP-Lösung einzuschränken. Dem folgend wird mittels Heuristiken, versucht eine ganzzahlige Lösung zu finden, die dann als Obergrenze für die weitere Suche dienen kann. Abschließend kommt das Branch-and-Cut Verfahren zum Einsatz, durch welches die optimale Lösung ermittelt und bewiesen werden soll. Im Rahmen des Branch-and-Cut Verfahrens wird zur Lösung der auftretenden Teilprobleme immer wieder auf den Dualen Simplex Algorithmus zurückgegriffen. Auch ausgewählte Techniken des Supernode processings, sowie Heuristiken werden im Branch-and-Cut Verfahren wiederholt angewandt.

Der soeben beschriebene Ablauf der Lösungsermittlung wird anhand von Abbildung 7.2 verdeutlicht.

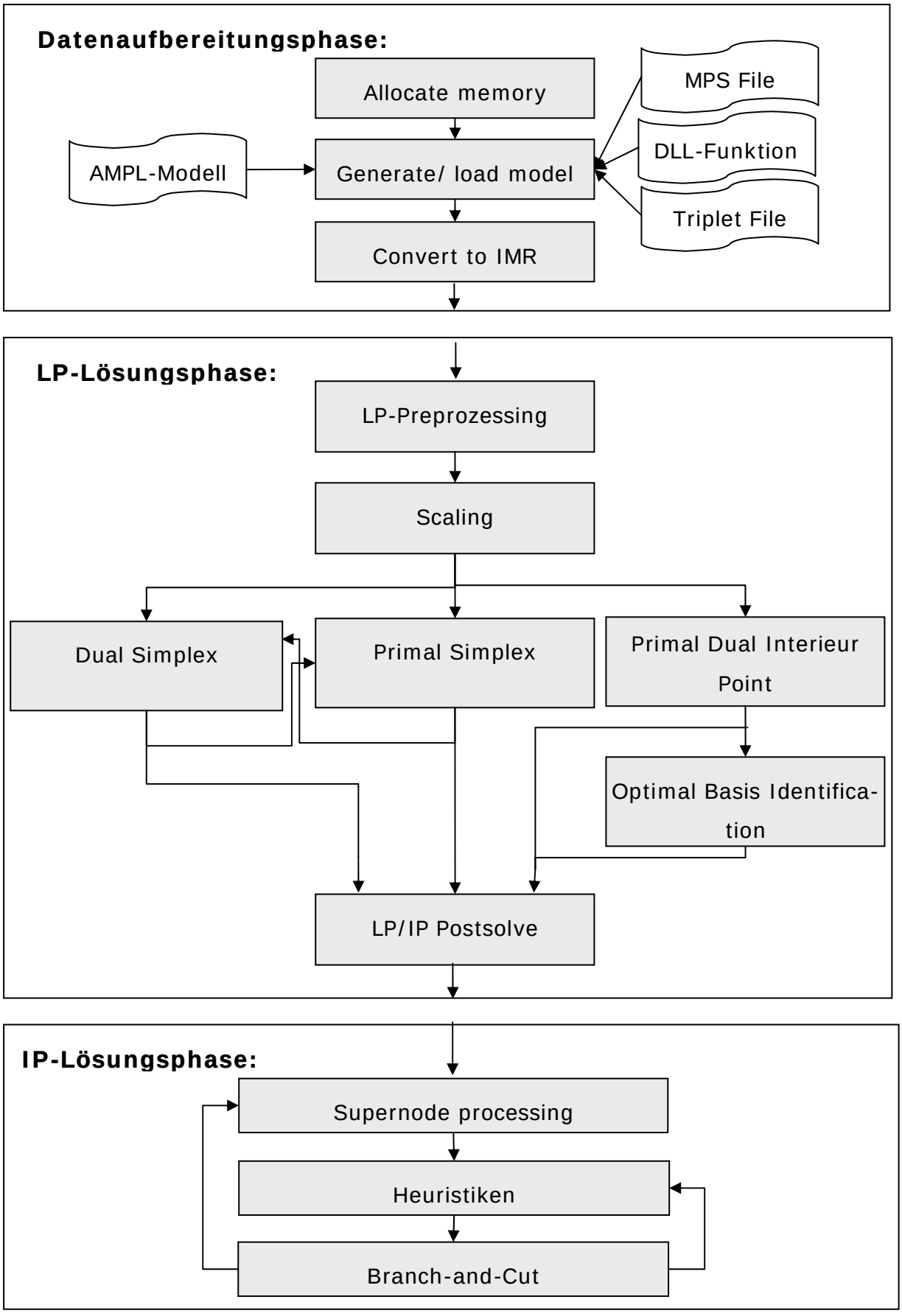

Abbildung 7.2: Interner Aufbau von MOPS

7.2 Der Branch-and-Cut Algorithmus

7.2.1 Strategienauswahl

Die Abbildung 7.4 verschafft einen Überblick über die Routinen, durch die der bereits in MOPS implementierte Branch-and-Bound Algorithmus zu einem Branch-and-Cut Algorithmus erweitert werden konnte.

xtreecut

Wird während des Branch-and-Cut Verfahrens eine LP-Optimierung durchgeführt, wird nachdem die Lösung bekannt ist, die Routine *xtreecut* aufgerufen. Diese Routine beinhaltet die in Kapitel 6.2.1 vorgestellten Strategien. Die einzelnen Strategien bestimmen, an welchen Knoten nach Cuts gesucht werden soll. Zu den Cuts die im Rahmen diese Branch-and-Cut Algorithmus abgeleitet werden zählen, die Clique, Implication, Cover und Gomory Cuts.

Die folgende Auflistung gibt einen Überblick über die möglichen Strategien und die dazugehörigen Parameter:

-	Cuts werden nur vor dem Branch-and-Cut Prozess gesucht
xbcstr1 = 1	Cuts werden an jedem Knoten gesucht (bis Cutspeicher voll ist).
xbcstr2 = 1	Cuts werden an jedem *k-ten* Knoten gesucht.
xbcstr3 = 1	Cuts werden bis zum *k-ten* Knoten gesucht.
xbcstr4 = 1	Cuts werden gesucht, wenn der relative Gap des aktuellen Knotens größer *k* ist.
xbcstr5 = 1	Cuts werden gesucht, wenn noch mindestens *k%* der Variablen, die am ersten Knoten fraktionell waren, einen fraktionellen Wert annehmen.
xbcstr6 = 1	Cuts werden gesucht, wenn die Differenz zwischen dem Zielfunktionswert eines Knotens und der globalen Untergrenze kleiner als *k%* des aktuellen globalen absoluten Gaps ist.
xbcstr7 = 1	Cuts werden gesucht, wenn die Differenz zwischen dem relativen Gap des aktuellen Knotens und dem globalen relativen Gap kleiner als *k* ist.

Abbildung 7.3: Strategien für den Branch-and-Cut Ansatz

Nachdem eine oder mehrere der Strategien ausgewählt wurden, muss ein Wert für *k* festgelegt werden. Jede Strategie braucht einen individuellen Wert für k. Die zum Einsatz kommenden Einflussfaktoren auf k sind:

- Der Anteil an Integervariablen am Modell
- Der Fortschritt der während des Supernode processings erzielt werden konnte.

Weitere Einflussfaktoren auf *k* ergeben sich erst innerhalb des Baumes. Um diese zu verwenden muss der vorher für *k* definierte Wert verändert werden. Die damit verbundene Reduzierung oder Erhöhung der Anzahl der Knoten, die ausgewählt werden, ist abhängig von der betreffenden Strategie. Damit alle Strategien mittels zweier Parametern angepasst werden können, werden sie in zwei Gruppen eingeteilt. Die eine Gruppe enthält die Strategien, die bei steigendem *k* an mehr Knoten nach Cuts suchen. Die zweite Gruppe enthält alle übrigen.

Der wohl wichtigste Einflussfaktor auf *k* innerhalb des Baumes ist die erfolglose Suche. Immer dann, wenn an einem Knoten erfolglos nach Cuts gesucht wird, wird eine Variable hoch gezählt. Übersteigt diese Variable einen bestimmten Wert, wird *k* so angepasst, dass die Anzahl der Knoten, an denen nach Cuts gesucht wird, reduziert wird.

Vorausgesetzt es handelt sich um einen Knoten, von dem Cuts abgeleitet werden sollen, werden die Routinen *xchclitr*, *xchimptr xcovtr* und *xgomotr* aufgerufen. Die in diesen Routinen angewandten Verfahren entsprechen im Wesentlichen denen des Supernode processings. Ein entscheidender Unterschied besteht darin, dass die Deskalierung innerhalb der Routinen vorgenommen wird. Schnittebenen werden von dem unskalierten Modell generiert, jedoch in skalierter Form an das Modell angehangen. Um die globale Gültigkeit der Cuts gewährleisten zu können, müssen fixierte Variablen innerhalb der Verfahren im Modell bleiben. Da während des Branch-and-Cut Verfahrens immer wieder nach Cuts gesucht wird, werden die einzelnen Routinen dahingehend angepasst, dass die Intensität der Suche verringert wird. Das lässt sich u. a. dadurch realisieren, dass nur noch eine Runde nach einer bestimmten Klasse von Cuts gesucht wird. Im Gegensatz dazu wird während des Supernode processing vor dem Branch-and-Cut Verfahren mehrere Runden hintereinander nach Cuts gesucht. Eine weiter Möglichkeit besteht darin, den erforderliche Grad an Verletzung den ein Cuts aufweisen muss, um ans Modell angehangen zu werden, zu erhöhen. Insgesamt dienen diese Anpassungen dazu die Anzahl der Cuts zu steuern.

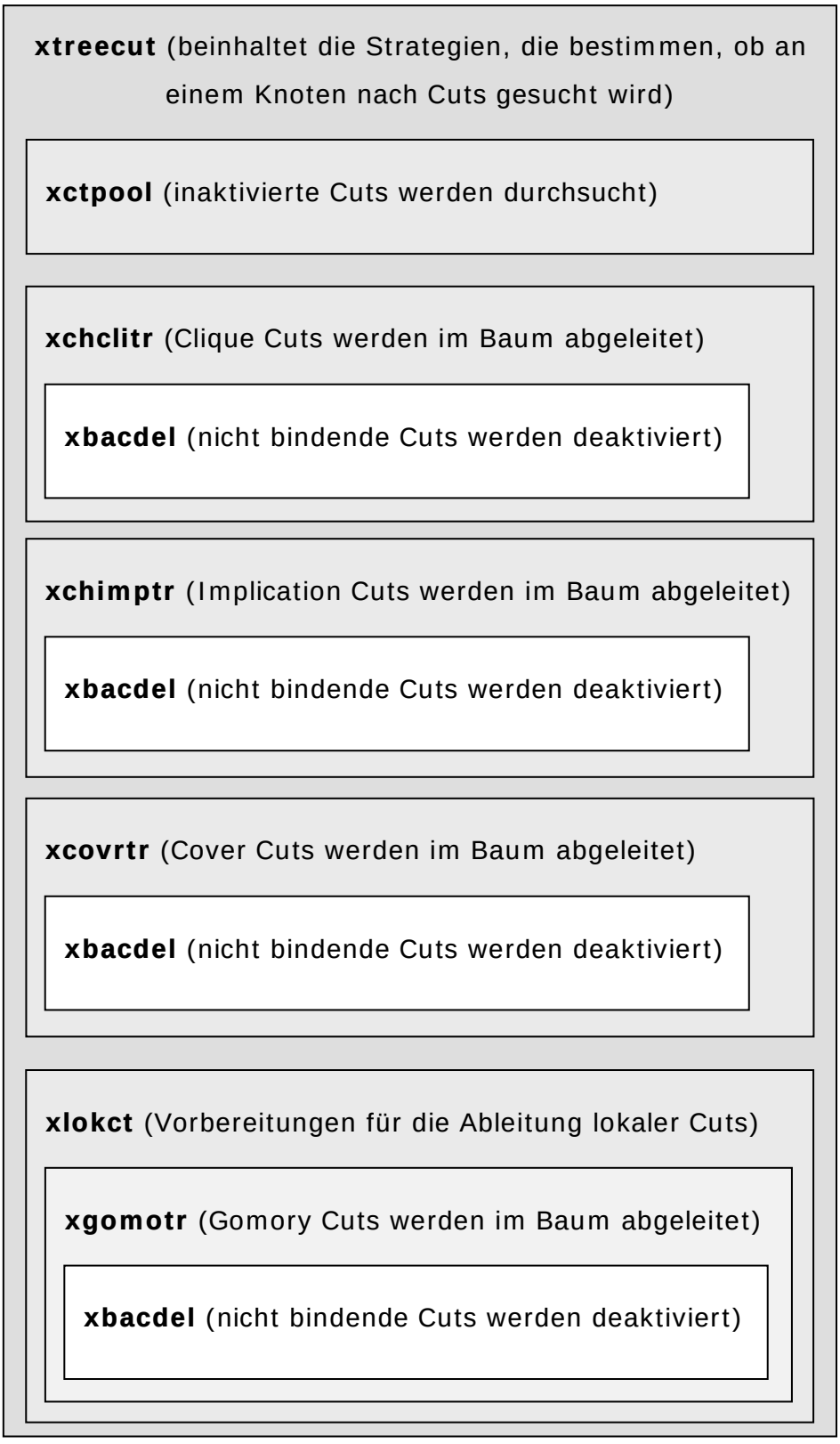

Abbildung 7.4: Routinen des Branch-and-Cut Ansatzes

Neben den Clique, Cover und Implikation Cuts sollen an ausgewählten Knoten auch lokale Gomory Cuts abgeleitet werden. Die dazu erforderlichen Vorbereitungen erfolgen in der Routine *xlokct*, die ebenfalls von der Routine *xtreecut* aufgerufen wird.

Wird in einer der Routinen ein Cut abgeleitet und an das Modell angehangen, kann die Routine *xbacdel* aufgerufen werden. Sie sorgt dafür, dass nicht bindende Schnittebenen inaktiviert werden.

Sobald eine bestimmte Anzahl von Schnittebenen inaktiviert wurde, werden, bevor nach neuen Cuts gesucht wird, in der Routine *xctpool* die alten inaktivierten Cuts durchsucht. Ist einer dieser Cuts verletzt, wird er reaktiviert.

7.2.2 Lokale Cuts

xlokct

Die Routine *xlokct* umfasst alle nötigen Vorbereitungen, um das Modell nach dem Ableiten von lokal gültigen Cuts wieder in seinen Originalzustand zurückversetzten zu können. Diese Routine kann für alle lokal gültigen Cuts angewandt werden; im Moment wird sie nur für Gomory Cuts benutzt.

Theoretisch sollten zwar Gomory Cuts zumindest für reine 0-1-Modell global gültig sein, praktisch ist davon leider nicht immer auszugehen. Die globale Gültigkeit der Gomory Cuts in reinen 0-1-Modellen basiert auf der Tatsache, dass alle Variablen die innerhalb des Baumes auf null oder eins fixiert wurden nicht Teil der Basis sind (vgl. Kapitel 6.6.2.2). Leider treten doch Fälle auf, in denen sich fixierte Variable in der Basis befinden. In einer solchen Situation kann eine Lösung mit dem gleichen Zielfunktionswert gefunden werden, in der die fixierte Variable nicht Teil der Basis ist. Dazu müssen weitere Iterationsschritte durchgeführt werden. Insgesamt wird das Verfahren aber dadurch zu aufwendig. Das hat zur Folge, dass Gomory Cuts nur als lokal gültige Cuts abgeleitet werden. Dazu wird die Routine *xgomotr* im Rahmen der lokalen Cutsuche *(xlokct)* aufgerufen. Es handelt sich dabei um eine auf den Gebrauch innerhalb des Baumes abgewandelte Version der Routine *xgomory* (vlg. Kapitel 7.3.2).

Tritt durch das kurzfristige Anhängen lokaler Cuts einer der folgenden drei Fälle ein, muss an diesem Knoten nicht weiter verzweigt werden:

- Das Modell mit Cuts wird unlösbar.
- Eine IP-Lösung wird gefunden.
- Der Zielfunktionswert am Knoten ist größer als die globale Obergrenze (Min.).

Anderenfalls wird der Zielfunktionswert, der sich ergibt, wenn die lokalen Cuts Teil des Modells sind gespeichert und der Knoten wird ohne diese Cuts in der Kandidatenliste gespeichert.

7.2.3 Aktivieren und Inaktivieren von Cuts

xbacdel

In dieser Routine können Cuts inaktivert werden. Dazu wird der Cut nicht richtig gelöscht und dennoch ist dieses Verfahren mit viel Aufwand verbunden, da der Cut sowohl in der zeilenweisen Struktur als auch aus der spaltenweisen Struktur des Modells inaktiviert werden muss.

Um die erforderlich Vorgehensweise deutlich zu machen soll hier zunächst die in MOPS verwendete Datenstruktur vorgestellt werden.

Das Modell liegt sowohl in der zeilenweisen Struktur vor,

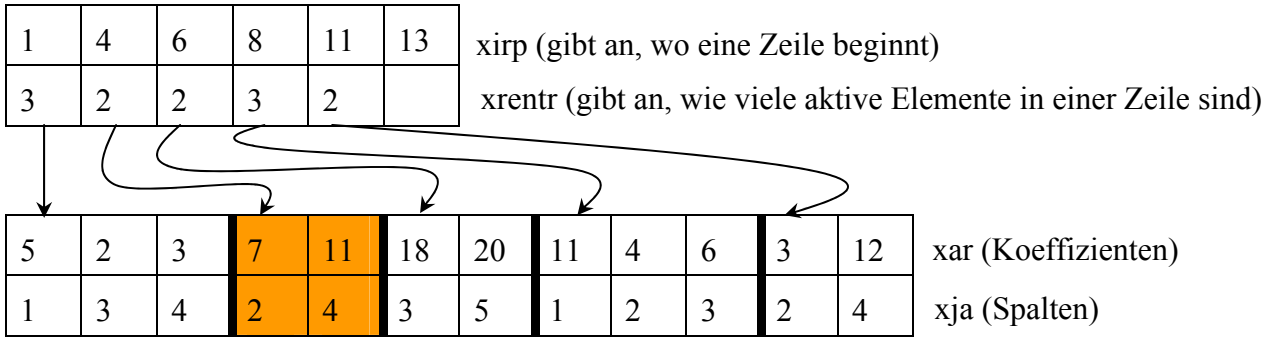

Abbildung 7.5: Zeilenweise Darstellung in MOPS

als auch in der spaltenweisen Struktur

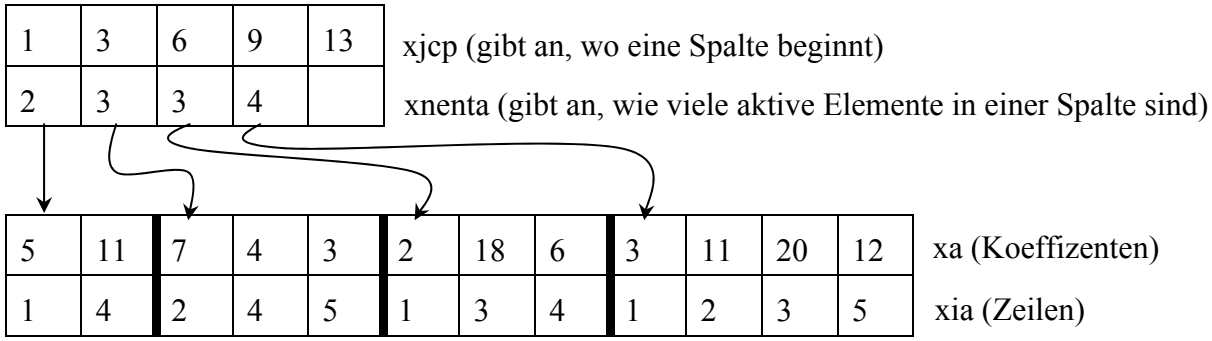

Abbildung 7.6: Spaltenweise Darstellung in MOPS

Die Inaktivierung einer Zeile muss in beiden Strukturen erfolgen. In der zeilenweisen Struktur wird die Zeile einfach als inaktiv markiert.

Für die spaltenweise Struktur hingegen ist der Vorgang erheblich aufwendiger. Die betreffende Zeile muss in den Spalten, in denen sie auftritt gesucht werden, um dann in den inaktiven Teil verschoben werden zu können.

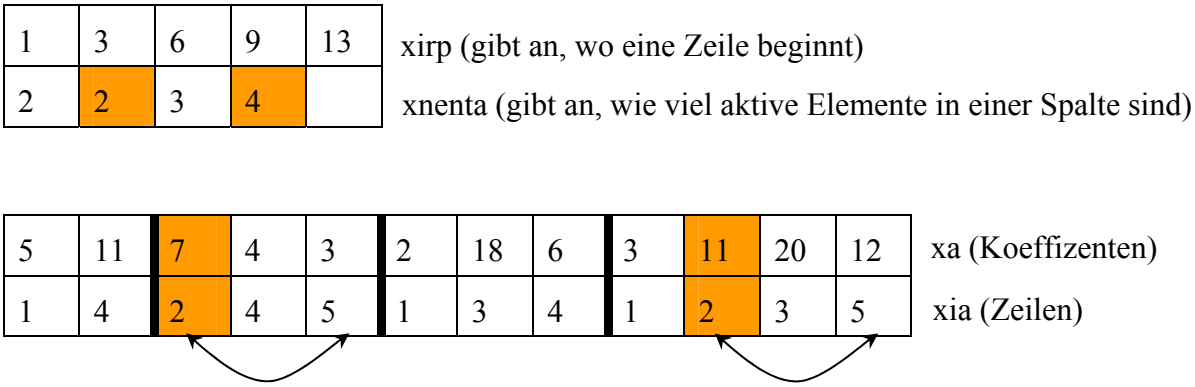

Abbildung 7.7: Verschiebung inaktiver Elemente

Diese Verschiebung erfolgt, indem das letzte aktive Element mit dem zu inaktivierenden vertauscht wird und die Anzahl der aktiven Elemente der betroffenen Spalte um eins reduziert wird (s. Abbildung 7.7)

Das Inaktivieren der Zeilen geschieht durch die Routine *xmipia*. Um auch die spaltenweise Struktur auf den aktuellen Stand zu bringen, muss die Routine *xcrimr* aufgerufen werden.

Die Routine *xcrimr* hat eine Komplexität von O(*m*,*n*,*nz*), wobei *m* die Anzahl der Zeilen, *n* die Anzahl der Spalten und *nz* die Anzahl der Nicht-Null-Elemente darstellt. Damit diese Routine möglichst selten aufgerufen wird, werden Schnittebenen nur dann inaktiviert, wenn vorher andere angefügt wurden und es ohnehin erforderlich ist, *xcrimr* aufzurufen.

In *xbacdel* wird bestimmt, wie oft eine Schnittebene nicht bindend sein muss, bevor sie inaktiviert wird. Dieser Wert hängt von der Anzahl der Nicht-Null-Elemente in dem betreffenden Cut ab.

xctpool

Wird die Routine *xctpool* aufgerufen, werden bereits abgeleitete Cuts daraufhin untersucht, ob sie die aktuelle LP-Relaxierung verletzen. Es handelt sich dabei genau um die Cuts, die vorher von der Routine *xbacdel* inaktiviert wurden. Wird ein Cut gefunden, der wieder von der aktuellen LP-Relaxierung verletzt wird, soll dieser reaktiviert werden. Um einen inaktivierten Cut zu reaktivieren, müssen genau die für die Routine *xbacdel* beschriebenen Vorgänge umgekehrt ausgeführt werden. In der zeilenweisen Struktur wird die Zeile einfach wieder als aktiv markiert. Für die spaltenweise Struktur wird die betreffende Zeile in den Spalten, in

denen sie auftritt, gesucht. Dabei muss in dem inaktiven Teil der Spalte gesucht werden, da die Zeile noch inaktiviert ist. Die Verschiebung in den aktiven Teil erfolgt, indem das erste inaktive Element mit dem zu aktivierenden vertauscht wird und die Anzahl der aktiven Elemente der betroffenen Spalte um eins erhöht wird. Das Aktivieren der Zeilen geschieht durch die Routine *xreact*. Um auch die spaltenweise Struktur auf den aktuellen Stand zu bringen, muss die Routine *xcrimr* aufgerufen werden.

xadobj

Soll die Zielfunktion als Restriktion an das Modell angehangen werde, wird die Routine *xadobj* aufgerufen. Die Untergrenze dieser neuen Restriktion ist der Zielfunktionswert des Vorgängerknotens. Innerhalb des Baumes kann es sinnvoll sein, die Zielfunktion anzuhängen, wenn lokale Cuts abgeleitet wurden oder Cuts inaktiviert wurden. So kann verhindert werden, dass ein Nachfolgerknoten bei einer Minimierung einen kleineren Wert hat als sein Vorgängerknoten.

7.3 Supernode processing

Große Fortschritte bei der Lösung gemischt-ganzzahliger Modelle konnten sowohl durch die Cover Cuts als auch durch die Gomory Cuts erzielt werden. Aus diesem Grund sollen an dieser Stelle diese beiden Techniken besonders hervorgehoben werden und im Detail vorgestellt werden.

7.3.1 Implementierungsaspekte von Cover Cuts

Abbildung 7.8 soll einleitend einen Überblick über die im Folgenden beschriebenen Routinen zum Auffinden von Cover Cuts geben.

xcovtr

xncver ist eine Rahmenroutine von der aus die Cover-Cut Suche gestartet und auch beendet wird. Die Restriktionen werden nacheinander betrachtet.

Im ersten Schritt wird geprüft, ob es sich wirklich um Ungleichungen handelt, da von Gleichungen keine Cover Cuts abgeleitet werden können. Dann werden alle Ungleichungen in ≤-Restriktionen umgewandelt und Variablen mit negativen Koeffizienten werden substituiert.

Für die Umwandlung in ≤-Restriktionen wird mit *(-1)* multipliziert. Für die Substitution von negativen Koeffizienten wird die Variable *x'= (1-x)* eingeführt.

Als letzter Schritt werden die Variablen, die keine Binärvariablen sind aus der Ungleichung entfernt. Damit sie nicht zu einer vorzeitigen Beschränkung führen, werden sie auf ihre Untergrenze gesetzt.

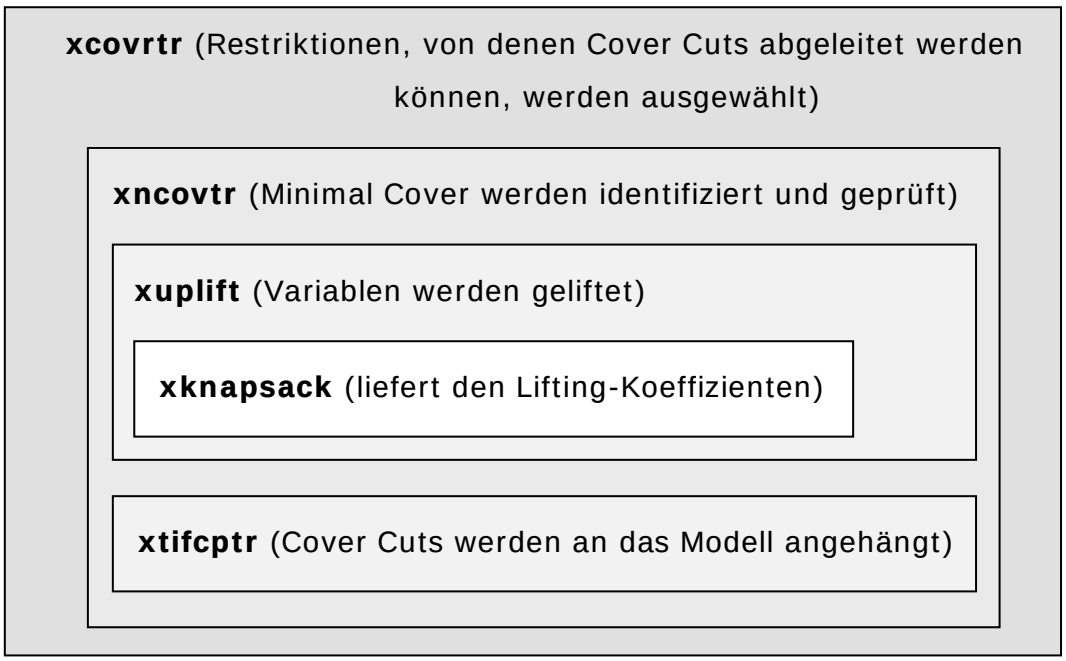

Abbildung 7.8: Routine zum Ableiten von Cover Cuts

Die Restriktionen liegen dann in gewünschter Form vor, so dass versucht werden kann, ein Cover Cut abzuleiten. Dazu wird die Routine *xncovtr* aufgerufen. Von dieser Routine wird ggf. ein neu generierter Cover Cut geliefert. Nachdem alle Nebenbedingungen betrachtet wurden und Cover Cuts an das Modell angehangen wurden, wird es erneut relaxiert gelöst. Konnte kein Cover Cut an das Modell angehangen werden, wird die Suche beendet. Anderenfalls werden erneut alle Ungleichungen nach verletzten Cover Cuts überprüft.

xncovtr

Zu Beginn dieser Routine wird die folgende Mengeneinteilung durchgeführt, wobei *N* die Menge der Variablen ist, die in der betrachteten Restriktion einen Koeffizienten ungleich null haben:

$U = \left\{ j \in N \middle| x_j^* = 1 \right\}$ die Variablen mit dem LP-Lösungswert eins,

$L = \left\{ j \in N \middle| x_j^* = 0 \right\}$ die Variablen mit dem LP-Lösungswert null

und $N \setminus (U \cup L)$ die Variablen mit einem fraktionellen LP-Lösungswert.

Dabei ist zu beachten, ob eine Variable vorher substituiert wurde oder nicht. Die Substituierung einer Variablen hat zur Folge, dass auch ihr LP-Wert umgedreht werden muss. Wird eine Variable in *U* aufgenommen, muss die rechte Seite um den Koeffizienten dieser Variablen verringert werden.

Gibt es mehr als ein Element in $N \setminus (U \cup L)$ werden diese absteigend sortiert. Diese sortierten Elemente der Menge $N \setminus (U \cup L)$ werden so lange in die Menge *K* aufgenommen, bis entweder keine Elemente mehr in $N \setminus (U \cup L)$ sind oder die Summe der Koeffizienten, die in *K* aufgenommen wurden, die aktualisierte rechte Seite übersteigt. Aus der so entstandene Menge *K* soll zusammen mit der Menge *U* das gesuchte Cover gebildet werden.

Dieses Cover wird im Weitern überprüft, ob es den für ein Cover charakteristischen Eigenschaften entspricht. Die Koeffizientensumme aller Variablen des Covers muss größer als die rechte Seite sein ($\sum_{j \in C} a_j > a_0$) und jeder beliebige Koeffizient muss subtrahiert werden können, damit die Summe der verbleibenden Koeffizienten kleiner oder gleich der rechten Seite wird. ($\sum_{j \in C} a_j - a_k \le a_0, \quad \forall k \in C$). Trifft das erste Kriterium nicht zu, stellt das eine Abbruchsbedingung dar. Trifft das zweite Kriterium nicht zu, muss nur die Variable, die gegen diese Bedingung verstößt aus dem Cover Cut herausgenommen werden. Kommt diese Variable ursprünglich aus der Menge *U*, muss ihr Originalkoeffizient zur rechten Seite addiert werden.

Zur Vorbereitung des Liftings werden die Mengen *F, L, C_2* und *C_1* gebildet.

$$L = \left\{ j \in N \middle| x_j^* = 0 \right\} \text{ und } F = N \setminus (C \cup L)$$

$$C_2 = U \text{ und } C_1 = C \setminus C_2 .$$

Wobei aus der Menge *C_1* das Ausgangscover $\sum_{j \in C_1} x_j \le |C_1| - 1$ gebildet wird.

Die Mengen *F* und *L* sollen up-geliftet werden, um möglicherweise die Ungleichung zu verschärfen. Die Menge *C_2* muss down-geliftet werden, um die Gültigkeit der Ungleichung sicherzustellen.

Die Reihenfolge in der Variablen geliftet werden, kann ausschlaggebend dafür sein, ob eine Variable Teil des Cover Cuts wird. Nur noch die fraktionellen Variablen, die der Menge *F*,

können zur Verletzung des Cover Cuts beitragen. Sie können ggf. die Verletzung auch erst hervorrufen. Folglich ist es sinnvoll, als Erstes diese zu liften. Es folgen die Variablen der Menge C_2 und abschließend die Variablen der Menge L. Sowohl für das Up-Lifting als auch für das Down-Lifting wird die Routine *xuplift* aufgerufen. Der entscheidende Unterschied zwischen dem Up- und dem Down-Lifting ist, dass beim Down-Lifting Elemente, die vorher entfernt wurden und beim Up-Lifting neue Elemente in die Ungleichung aufgenommen werden sollen. Praktisch stellt sich dieser Unterschied folgendermaßen dar. Während beim Up-liften der Koeffizient der betrachteten Variablen abgezogen werden muss, wird er beim Down-Lifting addiert. Der Lifting-Koeffizient einer up-gelifteten Variablen ergibt sich, indem die durch das Aufrufen von *xuplift* erhaltenen, maximierten Variablen von der rechten Seite des Cover Cuts abgezogen werden. Für den Lifting-Koeffizienten einer down-gelifteten Variablen muss die rechte Seite des Cover Cuts von den maximierten Variablen abgezogen werden. Des Weiteren muss im Rahmen des Down-Liftings die rechte Seite des Cover Cuts aktualisiert werden. Dazu wird der ermittelte Lifting-Koeffizient zur rechten Seite addiert.

Bevor der ermittelte Cover Cut an die Routine *xtifcptr* weitergegeben wird, um an das Modell angehängt zu werden, wird die Substitution der Variablen mit negativen Koeffizienten rückgängig gemacht.

xuplift

In *xuplift* werden die Vorbereitungen für eine Knapsack Optimierung getroffen, mittels derer ein Lifting-Koeffizient gefunden werden soll. Die von der Knapsack Optimierung benötigten Werte, sind die Lifting-Koeffizienten der bereits in das Ausgangscover aufgenommenen Variablen, die dazugehörigen Koeffizienten der Originalrestriktion und die Relation aus diesen beiden Werten. Die Koeffizienten werden daraufhin untersucht, ob sie größer als die rechte Seite sind. In diesem Fall können sie gleich aussortiert werden. Bevor es zur Knapsack Optimierung kommt, werden die Variablen in die für die Optimierung erforderliche Reihenfolge gebracht werden. Von der Routine *knapsack* wird der Zielfunktionswert über *xuplift* an *xncovtr* gegeben, wo mit Hilfe dieses Zielfunktionswertes dann der Lifting-Koeffizient der gerade betrachteten Variablen errechnet wird.

xknapsack

Die Subroutine *xknapsack* stammt aus [MaTo90]. Mit ihr lassen sich Knapsack Probleme der Art

$$z = max\left\{ \sum_{j\in J} p_j x_j \,\middle|\, \sum_{j\in J} w_j x_j \le c; x_j \in \{0,1\} \forall j \in J \right\} \text{ lösen.}$$

Dabei müssen die folgenden Bedingungen erfüllt sein:

$$|J| \ge 2$$

Es müssen mindestens 2 Elemente für den Knapsack zur Disposition stehen.

$$p_j, w_j \text{ und } c \in \mathbb{R}_+, \ \forall j \in J$$

Alle Zielfunktionskoeffizienten und Gewichte müssen jeweils positive reelle Zahlen sein.

$$w_j \le c \qquad \forall j \in J$$

Jedes einzelne Gewicht w_j muss kleiner sein als die Kapazität c.

$$\sum_{j\in J} w_j \ge c$$

Die Summe aller w_j muss größer sein als die Kapazität c.

$$p_j / w_j \ge p_{j+1} / w_{j+1} \forall j \in J$$

Mit steigendem Index der Elemente muss die Relation zwischen Nutzen und Gewicht abnehmen.

xtifcptr

In dieser Routine wird überprüft, ob der gefundene Cover Cut bereits existiert.

Da im Rahmen des Branch-and-Cut Ansatzes nur verletzte Cover Cuts abgeleitet werden, muss der neue Cover Cut nicht mit allen bisher generierten verglichen werden. Er muss allerdings mit den Cover Cuts verglichen werden, die genau wie er selbst von der gerade betrachteten Restriktion abgeleitet wurden.

Praktisch werden die Elemente der alten Cover Cuts mit denen des neu generierten Cover Cuts nacheinander verglichen. Sind alle Elemente identisch und nur die rechte Seite des neu gefundenen Cover Cuts ist kleiner, so wird lediglich diese aktualisiert. Sobald es einen Unterschied zwischen zwei Elementen gibt, ist ein neuer Cover Cut gefunden. Dieser wird dann an das Modell angehängt bzw. in die MOPS Speicherstruktur geschrieben. Diese Routine wird von *xncovtr* aufgerufen**.**

7.3.2 Implementierungsaspekte von Gomory Cuts

Analog zu den Abbildungen der vorangegangenen Kapitel soll auch Abbildung 7.9 zunächst einen Überblick über die Routinen, die zum Auffinden von Gomory Cuts angewandt werden, verschaffen.

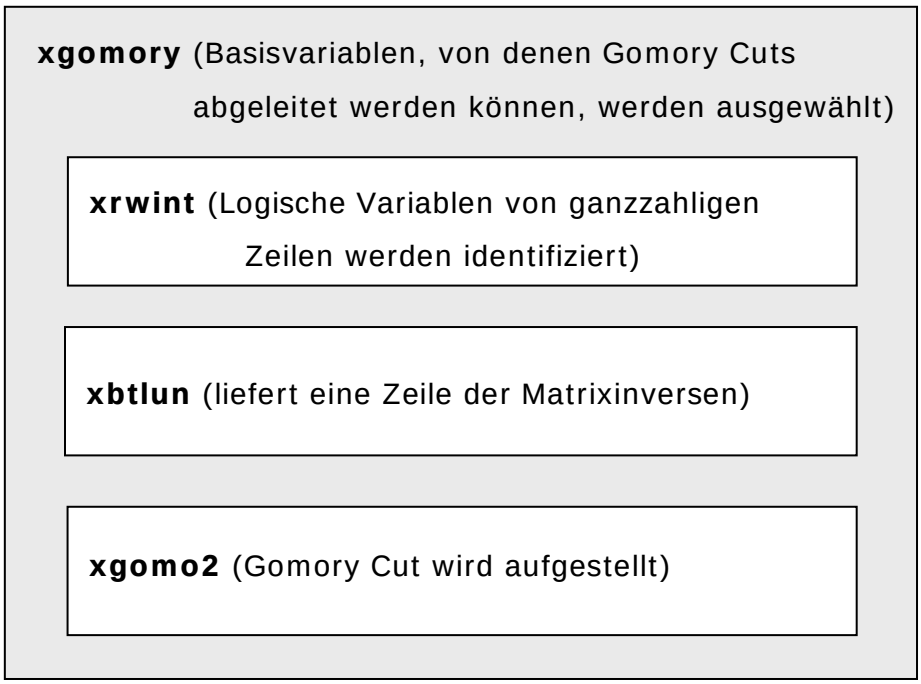

Abbildung 7.9: Routinen für das Auffinden von Gomory Cuts

xgomory

In der Routine *xgomory* werden Vorbereitungen getroffen damit Gomory Cuts abgeleitet werden können. Es wird eine Basisvariable ausgewählt und daraufhin überprüft, ob sie eine ganzzahlige Variable ist und ob sie in der aktuellen LP-Lösung einen fraktionellen Wert hat. Anschließend wird die Zeile der Matrixinversen, für die diese Variable in der Basis ist, ermittelt. Es sei A_B ist die *mxm* Matrix der Basisvariablen. Durch Bildung der inneren Produkte mit der Koeffizientenmatrix ergeben sich die Werte des Simplextableaus. Dabei ist, $\bar{a}_j = A_B^{-1} a_j$, wenn die Variable eine Strukturvariable ist, und $\bar{a}_j = A_B^{-1} e_i$, wenn die Variable eine logische Variable für die *i-te* Restriktion ist. Wurde eine Nicht-Basis-Variable an ihre Obergrenze gesetzt, muss das Vorzeichen, das ihr Wert im optimalen Simplextablau hat, umgedreht werden.

Die Werte der Zeile im Simplextablau werden an die Routine *xgomo2* übergeben. Diese Routine ermittelt einen Gomory Cut und gibt diesen an die Routine *xgomory* zurück.

Bevor dieser Cut an das Modell angehangen wird, werden einige Prüfungen bezüglich dessen Größe durchgeführt. Ein Gomory Cut kann aus verschiedenen Gründen abgelehnt werden. Ein Gomory Cut wird immer von der aktuellen LP-Lösung verletzt. Diese Verletzung kann aber auch sehr klein sein, was zur Ablehnung eines Gomory Cut führt. Der Grad der Verletzung muss einen bestimmten Wert (*xgovio)* übersteigen.

Das in Kapitel 4.7.5.1 beschriebene abgewandelte Markowitz Kriterium wird berechnet und überprüft. Ist davon auszugehen, dass ein Cut bei der nächsten Optimierung sehr viel Auffüllung an Nicht-Null-Elementen erzeugt, wird er abgelehnt.

Die Nicht-Null-Elemente aller Gomory Cuts insgesamt werden gezählt. Übersteigen sie den Wert *cutoff*, wird der letzte Cut abgelehnt und es werden keine weitern Gomory Cuts generiert. Der Wert cutoff wird am Anfang von *xgomory* berechnet. In diese Berechnung fließt sowohl die tatsächliche Anzahl der Nicht-Null-Elemente des Modells ein, als auch die Anzahl der Variablen des Modells. Alternativ kann der Wert auch vom Anwender beeinflusst werden.

Durch den Wert *xgommx* kann die maximale Anzahl an Gomory Cuts festgelegt werden. Ist dieser Wert einmal überschritten, wird die Suche nach Gomory Cuts nicht erneut aufgerufen.

Besteht ein Cut alle diese Kriterien, wird er an das Modell angefügt. Vorher werden aber noch alle anderen Basisvariablen daraufhin untersucht, ob ein Gomory Cut von ihrer Zeile des Simplextableaus abgeleitet werden kann.

xgomo2

In dieser Routine wird der Gomory Cut nach der Formel (4.5) abgleitet. Dazu werden die Variablen in ganzzahlige und nicht ganzzahlige unterschieden und mit den entsprechenden Koeffizienten versehen. Durch das Hinzufügen von Gomory Cuts neigen Probleme dazu numerisch instabil zu werden. Gomory Cuts sind i. d. R. sehr dicht besetzte Cuts, die außerdem noch von den z. T. sehr fraktionellen Werten des Simplextablaus abgeleitet werden. Die Problematik der numerischen Instabilität führte in der Vergangenheit dazu, dass Gomory Cuts nicht angewandt wurden, im Laufe der Zeit, durch Weiterentwicklung der LP-Software, konnte dieses Problem weitestgehend beseitigt werden. Trotzdem erfolgt eine Prüfung, ob die ermittelten Koeffizienten der Variablen, die in den Gomory Cut aufgenommen werden sollen, einen bestimmten Wert übersteigen. Nur dann werden sie Teil des Gomory Cuts, andernfalls werden sie auf ihre Obergrenze gesetzt, wenn sie einen positiven Koeffizienten haben, und auf ihre Untergrenze, wenn sie einen negativen Koeffizienten haben.

Sollten logische Variablen Teil des Gomory Cuts sein, werden diese durch Strukturvariablen ersetzt. Der neu generierte Cut wird an die Routine *xgomory* zurückgegeben.

xrwint

In dieser Hilfsroutine werden die logischen Variablen ermittelt, die zu einer Zeile gehören, in welcher nur ganzzahlige Variablen mit ganzzahligen Koeffizienten auftreten und auch die rechte Seite ganzzahlig ist. In diesem Fall muss auch die passende, logische Variable einen ganzzahligen Wert annehmen und wie eine ganzzahlige Variable behandelt werden. Diese Eigenschaft einer Variablen ist bei der Ermittlung eines Gomory Cuts relevant.