4 Algorithmen zur Lösung erweiterter Modellklassen

In diesem Kapitel werden Strategien und Heuristiken des Branch-and-Bound-Prozesses erläutert, um IP-Probleme u.a. mit den erweiterten Modellklassen zu lösen. Des Weiteren wird die Bound Reduction mit allen Anwendungsbereichen an einem Knoten beschrieben.

4.1 Bound Reduction

Die Bound Reduction einer Variablen schränkt den Wertebereich einer Variablen und daraus folgend den Lösungsraum des Modells ein, ohne dass dabei Integer-Lösungen abgeschnitten werden.

Die Bound Reduction wird mit Hilfe der minimalen Zeilensumme

$$L_i = \sum_{j \in P_i} a_{ij} l_j + \sum_{j \in N_i} a_{ij} u_j \tag{4-1}$$

und der maximalen Zeilensumme

$$U_i = \sum_{j \in P_i} a_{ij} u_j + \sum_{j \in N_i} a_{ij} l_j \tag{4-2}$$

mit $i \in I$ und den Teilmengen $P_i = \{j \mid j \in J, a_{ij} > 0\}$ und $N_i = \{j \mid j \in J, a_{ij} < 0\}$ durchgeführt. P_i umfasst alle Variablen mit einem positiven und N_i alle Variablen mit einem negativen Koeffizienten in der Zeile i.

Bei der Bound Reduction werden die Mengen P_i und N_i getrennt voneinander betrachtet. Damit eine Bound reduziert werden kann, müssen die jeweiligen zu betrachtenden Zeilensummen und –grenzen endlich sein. In den Formeln (4.3)-(4.6) werden die reduzierten Bounds l'_j und u'_j berechnet.

- für $j \in P_i$:
  - mit endlichem bu_i und L_i: $u'_j = \min\left\{l_j + \frac{(bu_i - L_i)}{a_{ij}}, u_j\right\}$ (4-3)
  - mit endlichem bl_i und U_i: $l'_j = \max\left\{u_j + \frac{(bl_i - U_i)}{a_{ij}}, l_j\right\}$ (4-4)
- für $j \in N_i$:
  - mit endlichem bu_i und L_i: $l'_j = \max\left\{u_j + \frac{(bu_i - L_i)}{a_{ij}}, l_j\right\}$ (4-5)
  - mit endlichem bl_i und U_i: $u'_j = \min\left\{l_j + \frac{(bl_i - U_i)}{a_{ij}}, u_j\right\}$ (4-6)

Der Term $Bound_j + \frac{|RHS_i - Zeilensumme_i|}{Koeffizient_{ij}}$ gibt an, welchen Grenzwert die Variable x_j annehmen darf, bevor die Restriktion i unzulässig wird. Liegt nun dieser Grenzwert innerhalb der Bounds der Variablen x_j, dann kann der Bereich der Variablen bis zu diesem Grenzwert eingeschränkt werden. Eine Überschreitung dieses Wertes würde zu einem unzulässigen Problem führen. Wenn sich in einer Zeile die Bound einer Variablen ändert, führt dies zu einer Veränderung der Zeilensummen aller Zeilen, in denen sich diese Variable befindet. Demzufolge kann eine Veränderung der Zeilensummen zu weiteren Reduktionen von Bounds führen.

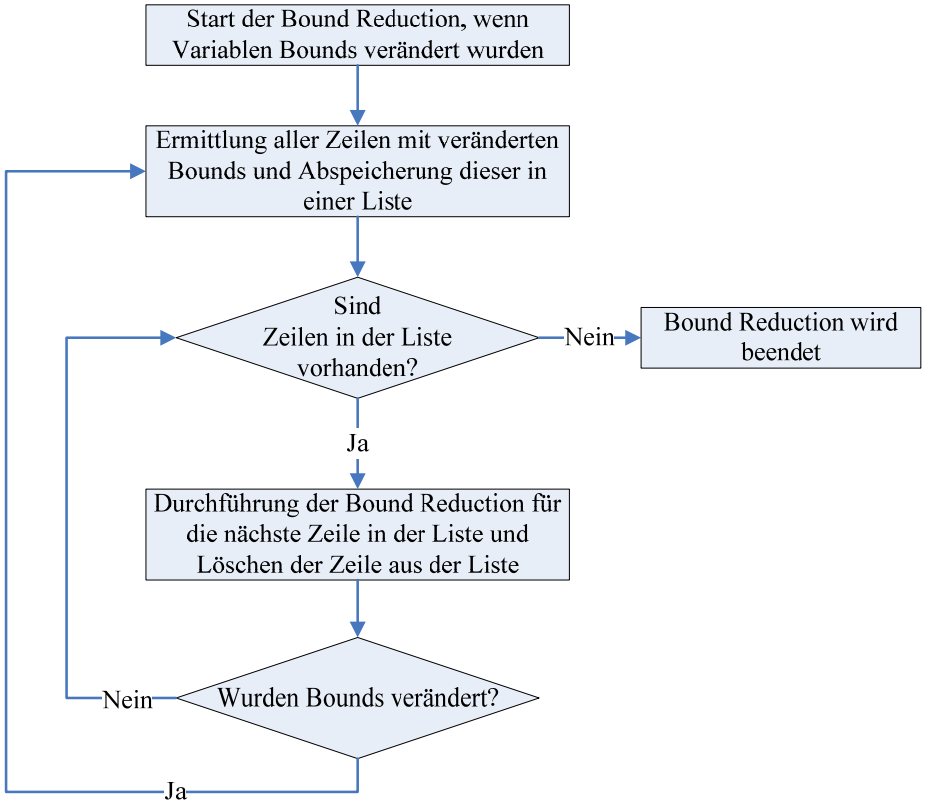

Abb. 4-1: Schematischer Ablauf Bound Reduction

Bei einer Integer-Variablen ist $l_j = \lceil l'_j \rceil$ und $u_j = \lfloor u'_j \rfloor$, da dieser Variablentyp per Definition nur ganze Zahlen annehmen kann.

Durch die Bound Reduction können Variablen fixiert werden, wenn eine Bound durch die Reduktion den Wert der anderen Bound annimmt. Eine Reduktion bei einer 01-Variablen führt immer zu einer Fixierung.

Die Bound Reduction erfolgt nur bei endlichen Zeilensummen. Beinhaltet allerdings eine Zeilensumme eine unendliche Bound, so kann nach J. Gondzio [Gon94] eine erweiterte Bound Reduction durchgeführt werden. Dabei wird für die Variable mit unendlicher Bound die Bound Reduction ausgeführt. Die Variable kann auch eine freie Variable sein, d.h. beide Bounds sind unendlich.

Um die reduzierte Bound zu berechnen, werden die Zeilensummen L'_i und U'_i ohne die jeweilige unendliche Bound berechnet. Bei der erweiterten Bound Reduction werden wie bei der einfachen Bound Reduction die Mengen P_i und N_i getrennt voneinander betrachtet.

In den Formeln (4-7)-(4-10) werden die reduzierten Bounds l'_j und u'_j für die Variable j mit unendlicher Bound berechnet.

- für $j \in P_i$:
  - mit endlichem bu_i und L'_i: $u'_j = \min\left\{\frac{(bu_i - L'_i)}{a_{ij}}, u_j\right\}$ (4-7)
  - mit endlichem bl_i und U'_i: $l'_j = \max\left\{\frac{(bl_i - U'_i)}{a_{ij}}, l_j\right\}$ (4-8)
- für $j \in N_i$:
  - mit endlichem bu_i und L'_i: $l'_j = \max\left\{\frac{(bu_i - L'_i)}{a_{ij}}, l_j\right\}$ (4-9)
  - mit endlichem bl_i und U'_i: $u'_j = \min\left\{\frac{(bl_i - U'_i)}{a_{ij}}, u_j\right\}$ (4-10)

Die Bound Reduction kann sowohl im Supernode Processing als auch während des Branch-and-Bounds angewandt werden. Im Folgenden sind die Anwendungen der Bound Reduction im Branch-and-Bound-Prozess dargestellt.

Anwendung der Bound Reduction nach der Verzweigung

Die Bound Reduction wird direkt nach der Verzweigung der Branchingvariablen ausgeführt. Durch die Veränderung des Wertebereichs wird die Branchingvariable eingeschränkt bzw. fixiert. Dadurch verändern sich die Zeilensummen der Zeilen, in denen diese Variable vorkommt. Somit kann diese Setzung weitere Fixierungen und Einschränkungen der Wertebereiche von Integer- aber auch von kontinuierlichen Variablen mit sich führen oder gar eine Unzulässigkeit hervorrufen. Mit Hilfe der Bound Reduction im Branch-and-Bound werden nach jedem Setzen der Branchingvariablen die gegebenenfalls zu fixierenden oder einzuschränkenden Variablen ermittelt und die Zulässigkeit der Einschränkung überprüft.

Anwendung der Bound Reduction im Probing

Das Probing wird standardmäßig im Supernode Processing vor dem Branch-and-Bound-Prozess eingesetzt. Es kann aber auch in Kombination mit der Bound Reduction nach der LP-Optimierung an jedem Knoten im Baum durchgeführt werden.

Beim Probing werden noch nicht fixierte Integer-Variablen kurzzeitig durch den auf- bzw. abgerundeten Wert der relaxierten LP-Lösung an dem Knoten eingeschränkt und bei Gleichheit der beiden Bounds fixiert. Jede Einschränkung des Wertebereichs einer Integer-Variablen führt zu einer Veränderung der Zeilensummen in den Restriktionen, in denen diese Integer-Variable vorkommt. Mit der Bound Reduction wird überprüft, ob durch die Einschränkung das relaxierte LP unzulässig wird. Wenn die Einschränkung zulässig ist, dann werden alle Änderungen rückgängig gemacht, und der zweite Branch wird durchgeführt. Bei einer Unzulässigkeit wird die Integer-Variable global gültig auf den zweiten Branch eingeschränkt. Alle durch die Bound Reduction ergebenen Einschränkungen gelten global für den Knoten und dessen Nachfolgeknoten. Führen diese Einschränkungen auch

zu einem unzulässigen relaxierten LP, dann ist der ganze Knoten unzulässig und kann aus der Knotenliste gelöscht werden.

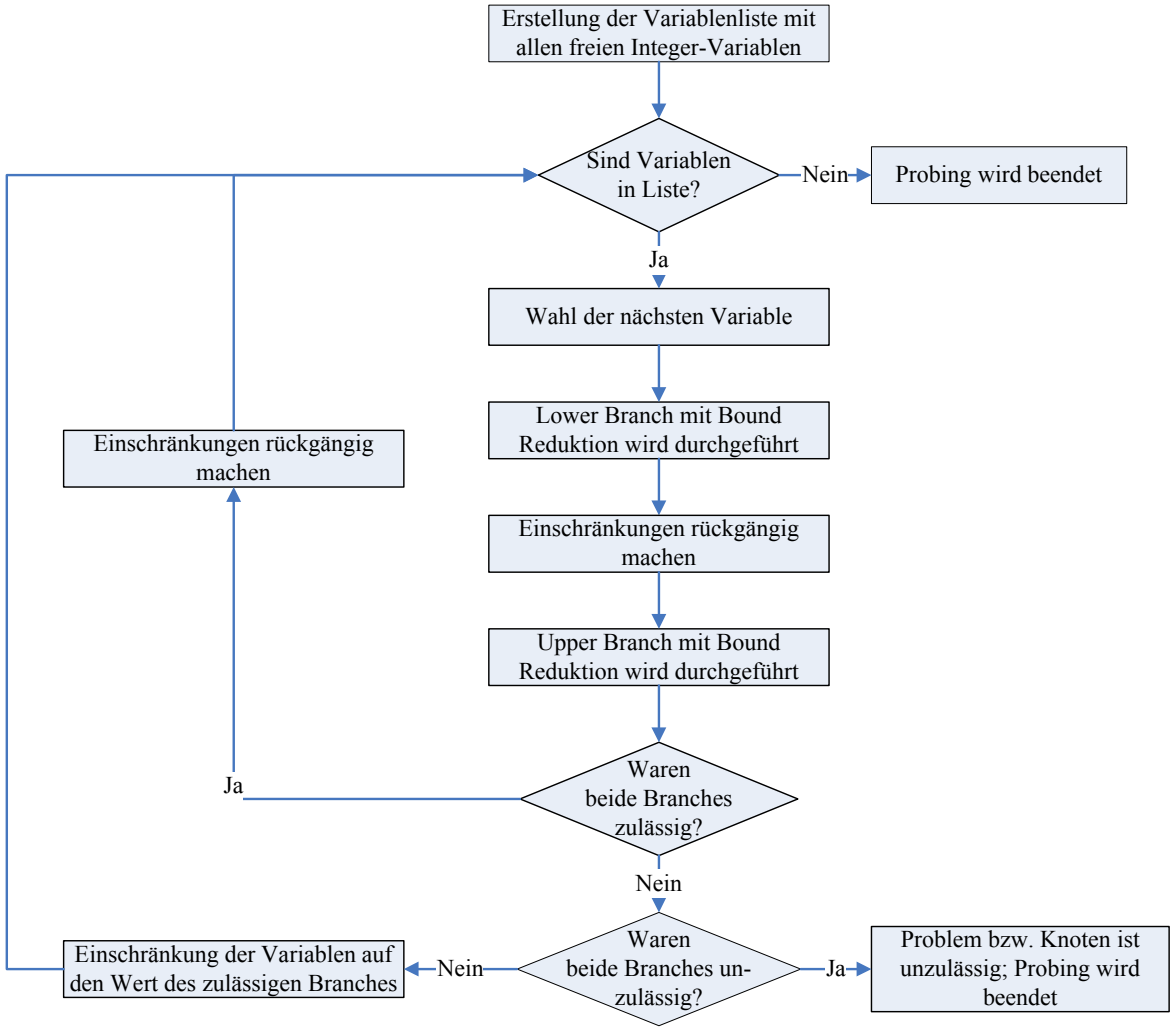

Abb. 4-2: Schematischer Ablauf Probing im Branch-and-Bound

Reduced Cost Fixing und Bound Reduction

Das Reduced Cost Fixing reduziert anhand der reduzierten Kosten die Bounds von Nichtbasis-Variablen. Diese Technik wird nach der LP-Optimierung aufgerufen, wenn bereits eine Integer-Lösung (zip) gefunden wurde bzw. eine obere Schranke für den IP-Zielfunktionswert existiert.

Wenn d_j die reduzierten Kosten der freien Integer-Nichtbasis-Variablen j mit $j \in J_I$ einer optimalen LP-Lösung mit einem Funktionswert von $z(LP)_i$ an einem Knoten i sind, dann ist die reduzierte Lower Bound

$$l_j' = \max\left(u_j - \left\lceil \frac{zip - z(LP)_i}{-d_j} \right\rceil, l_j \right) \tag{4-11}$$

mit $d_j<0$, d.h. Variable j liegt an der Upper Bound.

Die reduzierte Upper Bound ist

$$u_j' = \min\left(l_j + \left\lfloor \frac{zip - z(LP)_i}{d_j} \right\rfloor, u_j \right) \tag{4-12}$$

mit $d_j>0$, d.h. Variable j liegt an der Lower Bound.

Bei der Reduktion von Variablengrenzen durch das Reduced Cost Fixing verändern sich die Zeilensummen. Gegebenenfalls können weitere Bounds mit der normalen Bound Reduction reduziert werden. Das Reduced Cost Fixing wird solange aufgerufen, bis keine weiteren Bounds mehr reduziert werden können.

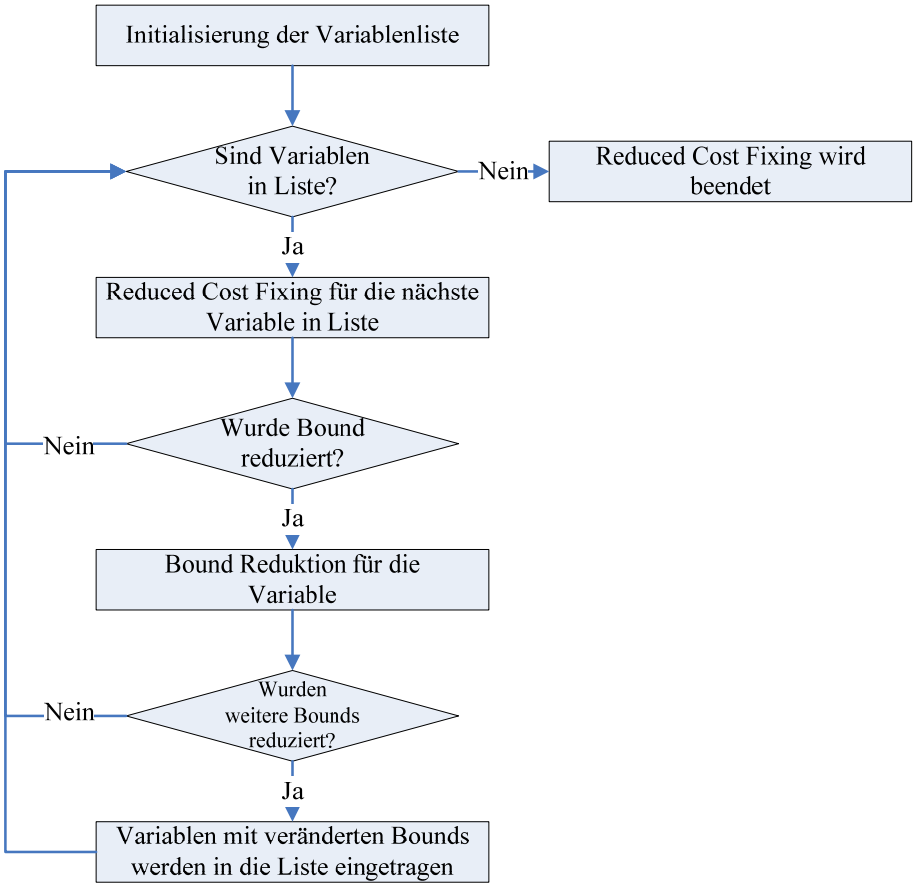

Abb. 4-3: Schematischer Ablauf Reduced Cost Fixing

4.2 Branch-and-Bound Algorithmus

Der Branch-and-Bound Algorithmus (Kapitel 2.1) zerlegt ein Problem in Teilprobleme, die wiederum zerlegt werden. So entsteht eine Baumstruktur, bei der die Teilprobleme durch Knoten dargestellt werden. Jede Zerlegung erfolgt die Verzweigung, d.h. dem Einschränken bzw. Fixieren einer fraktionellen Integer-Variablen ausgehend von dem Wert der relaxierten LP-Lösung des momentan betrachteten Knotens.

Ein Knoten wird weiter untersucht, wenn er nicht integer, zulässig und die Lösung des relaxierten LPs besser als die globale Obergrenze ist. Wenn bei einer Lösung keine Ganzzahligkeit vorliegt, dann muss mindestens eine Integer-Variable fraktionell sein.

Aus dieser Menge an fraktionellen Variablen J_{fn} eines Knotens n muss eine Variable $x_j = \lfloor x_j \rfloor + f_j$ mit $j \in J_{fn}$ und $0<f_j<1$ als Branchingvariable gewählt werden.

Da durch die Zerlegung eines Problems maximal zwei neue Teilprobleme entstehen, gibt es eine Menge NO an noch nicht betrachtenden Knoten. Aus dieser Menge muss ein geeigneter nächster Knoten gewählt werden.

Die Wahl einer Branchingvariablen und eines Knotens hat zwei Zielsetzungen:

- es soll eine gute Integer-Lösung gefunden werden, d.h. die globale Upper Bound des relaxierten LPs soll reduziert werden
- es soll die Optimalität der bisher besten Integer-Lösung bewiesen werden, d.h. die globale Lower Bound des relaxierten LPs soll angehoben werden

4.2.1 Wahl der Branchingvariablen

Es gibt eine Vielzahl von möglichen Auswahlstrategien, die u.a. in [Mar01], [LiSa99], [LaPo79] detaillierter beschrieben werden. Zu den klassischen Auswahlstrategien einer Branchingvariablen x_j zählen:

- Größte Fraktionalität:

$$j = \arg\left(\max_{k \in J_{fn}} \left(\min\left(f_k, (1 - f_k) \right) \right) \right) \tag{4-13}$$

- Größte Kosten:

$$j = \arg\left(\max_{k \in J_{fn}} c_k \right) \tag{4-14}$$

- Höchste Priorität:

$$j = \arg\left(\max_{k \in J_{fn}} p_k \right), \tag{4-15}$$

  wobei p_k mit $k \in J_I$ die vom Benutzer eingegebenen Prioritäten darstellen.

Da die Kosten, die Fraktionalitäten und auch eventuell die Prioritäten von fraktionellen Variablen die gleichen Werte annehmen können, tritt eine Branching-Strategie meist in Mischform mit einer anderen Strategie auf. In einer Patt-Situation wird die Branchingvariable nach einer anderen Strategie gewählt.

Beispiel: Die zwei fraktionellen Integer-Variablen x_1 mit $f_1=0,2$ und x_2 mit $f_2=0,3$ haben beide die gleichen Kosten von $c_1=c_2=100$. Bei einer Auswahlstrategie nach den größten Kosten entsteht eine Patt-Situation. Diese Patt-Situation wird durch eine weitere Strategie z.B. größte Fraktionalität aufgehoben. Demnach wäre die Variable x_2 die nächste Branchingvariable.

Neben diesen klassischen Strategien gibt es weitere Branching-Strategien, die im Folgenden genauer erläutert werden.

Pseudo Kosten

Die Pseudo Kosten zeigen die Wertminderung des Funktionswerts an, wenn eine fraktionelle Integer-Variable auf- bzw. abgerundet wird [GaRi77].

Ausgehend vom Elternknoten n werden zwei Nachfolgeknoten n+1 und n+2 entwickelt. Beim Knoten n+1 wird die fraktionelle Variable abgerundet und beim Knoten n+2 aufgerundet. Dann ergeben sich für die Variable x_j mit $j \in J_{fn}$ und einer Fraktionalität von f_j die Pseudo Kosten beim Abrunden von

$$PCL_j = \left| \frac{F_{n+1} - F_n}{f_j} \right| \tag{4-16}$$

und beim Aufrunden von

$$PCU_j = \left| \frac{F_{n+2} - F_n}{1 - f_j} \right|. \tag{4-17}$$

Die Schwierigkeit bei der Ermittlung der Pseudo-Kosten liegt darin, dass sowohl F_{n+1} als auch F_{n+2} am Knoten n noch nicht bekannt sind. Da die Pseudo-Kosten im Verlaufe des Baumes zwar nicht konstant bleiben, aber im Mittel ungefähr die gleiche Größe haben [LiSa99], wird bei der Wahl der Branchingvariablen auf bestehende Pseudo Kosten zurückgegriffen. Erst nach dem Branchen werden diese aktualisiert.

Die Branchingvariable x_j kann mit den Pseudo Kosten u.a. nach folgenden Strategien gewählt werden [LiSa99]:

- Kleinste Veränderung:

$$j = \arg\left\{ \min_{k \in J_{fn}} \left\{ \min\left\{ f_k * PCL_k, (1 - f_k) * PCU_k \right\} \right\} \right\} \tag{4-18}$$

- Maximale kleinste Veränderung:

$$j = \arg\left\{ \max_{k \in J_{fn}} \left\{ \min\left\{ f_k * PCL_k, (1 - f_k) * PCU_k \right\} \right\} \right\} \tag{4-19}$$

- Maximale bewertete Summe, wobei a≥0 und b≥0 sind und in der Regel a>b ist:

$$j = \arg\left\{ \max_{k \in J_{fn}} \left\{ a * \min\left\{ f_k * PCL_k, (1 - f_k) * PCU_k \right\} + b * \max\left\{ f_k * PCL_k, (1 - f_k) * PCU_k \right\} \right\} \right\} \tag{4-20}$$

Wie bei den anderen Strategien, kann in einer Patt-Situation eine zusätzliche Strategie die geeignete Branchingvariable wählen.

Am Anfang des Baums sind die Pseudo Kosten noch unbekannt und müssen demzufolge initialisiert werden. Es gibt drei Varianten der Initialisierung:

- *Setzen aller Pseudo Kosten auf Null:* Am Anfang des Baums findet keine Unterscheidung der fraktionellen Variablen statt. Gegebenenfalls muss eine weitere Strategie die Patt-Situation klären. Im Laufe des Baumes werden tendenziell Variablen gewählt, die vorher noch nicht verzweigt wurden.
- *Setzen aller Pseudo Kosten auf die Kosten der jeweiligen Variablen:* Wenn keine Beziehungen zwischen den Variablen bestehen würden, dann nehmen die Pseudo Kosten den Wert der Kosten an. Somit können die Kosten eine Tendenz der Pseudo Kosten wiedergeben. Durch die Beziehungen der Variablen untereinander können die Pseudo Kosten allerdings stark von den Kosten abweichen.

- *Trendberechnung durch teilweises Lösen des LPs:* Es wird eine bestimmte Anzahl an dualen Simplex-Iterationen für alle in Frage kommenden Integer-Variablen durchgeführt und mit dem daraus resultierenden Funktionswert die Pseudo Kosten berechnet. Diese Variante ist das genauste, aber auch, bei einer im Verhältnis großen Anzahl an Integer-Variablen, das zeitintensivste Verfahren.

Strong Branching

Die Idee des Strong Branching stammt von Applegate und wurde u.a. in [LiSa99] beschrieben.

Beim Strong Branching wird für eine Teilmenge der fraktionellen Variablen an einem Knoten getestet, ob eine Verzweigung nach diesen Variablen zu einer signifikanten Verschlechterung des Funktionswertes führen würde. Dazu werden nacheinander für jede Variable der Teilmenge ein Lower und ein Upper Branch durchgeführt. Das veränderte Modell wird jeweils für eine festgelegte Zahl an dualen Simplex-Iterationen gelöst. $z(LP)_{UBj}$ ist der Funktionswert, der sich beim Aufrunden und $z(LP)_{LBj}$ der Funktionswert, der sich beim Abrunden der fraktionellen Variablen x_j mit $j \in J_{fn}$ ergibt.

Bei dem Strong Branching müssen folgende Entscheidungen getroffen werden:

- *Bestimmung der Variablenmenge:* Für jedes Auf- bzw. Abrunden der Variablen muss beim Lösen des relaxierten LPs erst einmal eine duale Zulässigkeit des LPs erreicht werden, bevor dann die gegebene Zahl an dualen Simplex-Iterationen durchgeführt werden können. Dies ist sehr zeitintensiv, und somit wäre ein Strong Branching über alle fraktionellen Variablen x_j mit $j \in J_{fn}$ an einem Knoten $n \in NO$ sehr teuer. Demzufolge wird eine Teilmenge $J_{sbn} \subseteq J_{fn}$ gewählt, über die das Strong Branching durchgeführt werden soll. Nach folgenden Kriterien kann die Teilmenge erstellt werden:
  - es werden p% der fraktionellsten Variablen gewählt, wobei $0<p\leq 100$ ist.
  - mit $L = \max\{f_j : f_j \leq 0{,}5, j \in J_{fn}\}$ und $U = \min\{f_j : f_j \geq 0{,}5, j \in J_{fn}\}$ ist $J_{sbn} = \{j \in J_{fn} : 0{,}8L \leq f_j \leq U + 0{,}2(1-U)\}$ [LiSa99].
  - mit $fr_j = \begin{cases} \lceil \overline{x_j} \rceil - \overline{x_j} & \text{für } f_j > 0{,}5 \\ \overline{x_j} - \lfloor \overline{x_j} \rfloor & \text{für } f_j \leq 0{,}5 \end{cases}$ mit $j \in J_{fn}$ und $L = \max\{fr_j, j \in J_{fn}\}$ ist $J_{sbn} = \{j \in J_{fn} : 0{,}8L \leq fr_j\}$
- *Anzahl der dualen Simplexiterationen:* Je größer dieser Wert ist, desto genauer kann sich die Veränderung des Funktionswertes ablesen lassen, aber desto mehr Zeit braucht das Strong Branching. Je kleiner der Wert ist, desto schneller ist das Strong Branching, aber desto ungenauer die Tendenz des Funktionswertes. Somit ist die Anzahl der dualen Simplexiterationen ein Trade-Off zwischen Zeit und Genauigkeit. Da für jedes Modell zum Lösen des relaxierten LPs eine unterschiedliche Anzahl an Iterationen benötigt werden, ist es sinnvoll, die Anzahl nicht statisch festzulegen, sondern dynamisch durch das Modell zu berechnen.

  Eine Variante wäre *z.*B., die Anzahl durch eine Teilmenge der durchschnittlich für die relaxierten LPs benötigten Iterationen zu bestimmen.

- *Wahl der Branchingvariablen:* Die Auswahl der Branchingvariablen x_j ergibt sich aus den Zielfunktionswerten $z(LP)_{UBj}$ und $z(LP)_{LBj}$ und kann nach folgenden Kriterien durchgeführt werden:
  - Maximale Zielfunktionssumme:

$$j = \arg\left\{\max_{k \in J_{sbn}} \left\{z(LP)_{UBj} + z(LP)_{LBj}\right\}\right\} \tag{4-21}$$

  - Maximaler kleinster Zielfunktionswert:

$$j = \arg\left\{\max_{k \in J_{fn}} \left\{\min\left\{z(LP)_{UBj}, z(LP)_{LBj}\right\}\right\}\right\} \tag{4-22}$$

  - Maximale bewertete Summe, wobei $a \geq 0$ und $b \geq 0$ sind und in der Regel $a > b$ ist:

$$j = \arg\left\{\max_{k \in J_{fn}} \left\{a * \min\left\{z(LP)_{UBj}, z(LP)_{LBj}\right\} + b * \max\left\{z(LP)_{UBj}, z(LP)_{LBj}\right\}\right\}\right\} \tag{4-23}$$

  Demzufolge ist bei der gewählten Variablen die Tendenz einer Verschlechterung des Funktionswertes an den beiden Nachfolgeknoten am größten.

Tritt bei der Setzung einer fraktionellen Variablen eine Unzulässigkeit auf, dann kann die Variable auf den jeweils anderen Branch fixiert bzw. eingeschränkt werden. Sind beide Branches unzulässig, dann ist der ganze Knoten unzulässig.

Logical Branching

Zum Lösen eines Modells wird die Darstellungsweise des Grundmodells (1-2) in dem Kapitel 1.1 in eine interne Darstellung für den LP-Solver umgeschrieben. Jede Restriktion bekommt eine Schlupfvariable, die auch Logical Variable genannt wird und deren Bounds je nach Restriktionstyp endlich oder unendlich sind.

Beispiel:

Restriktion nach dem Grundmodell: $x_1 + 3x_2 - 4x_3 + x_4 \leq 2$

Restriktion in der internen Darstellung: $x_1 + 3x_2 - 4x_3 + x_4 + l = 0$, *wobei* $-2 \leq l \leq \infty$ *ist.*

Wenn nun alle Variablen und deren Koeffizienten in einer Restriktion Integer sind, dann muss bei einer Integer-Lösung auch die rechte Seite ganzzahlig sein, d.h. die Logical hat ganzzahlige Bounds und muss ganzzahlige Werte annehmen. Wenn nun in der Restriktion Integer-Variablen fraktionell sind, dann kann die Logical auch fraktionell sein. Wird nun nach dieser Logical verzweigt, dann wird die Restriktion weiter eingeschränkt und die Integer-Variablen der Restriktion können in der Summe auf einen ganzzahligen Wert gezwungen werden, was vielleicht zu einer Ganzzahligkeit jeder einzelnen Variablen führen kann.

Beispiel:

$x_1 + 3x_2 - 4x_3 + x_4 + l = 0$

Die Variable -2 ≤ l ≤ ∞ stellt die Logical dar. Die restlichen Variablen sind 01-Variablen mit x_1=0,4, x_2=0,3 und x_3=x_4=0. Demzufolge nimmt die Variable l den Wert -0,7 an. Da alle Variablen in der Restriktion 01-Variablen sind, muss die Logical in der optimalen Lösung einen ganzzahligen Wert annehmen. Wird nun nach der Logical verzweigt, entsteht folgende Einschränkung für die Restriktion:

1. Branch	*2. Branch*
$l \le -1$	$l \ge 0$
$1 \le x_1 + 3x_2 - 4x_3 + x_4 \le 2$	$x_1 + 3x_2 - 4x_3 + x_4 \le 0$

Die Auswahl der zu branchenden Logical ergibt sich aus deren Fraktionalität. Falls keine Logical fraktionell ist, wird eine andere Branching-Strategie gewählt.

Branchen nach Cliques

Das Konzept der Cliques wurde 1982 von E.L. Johnson und M.W. Padberg [JoPa83] entwickelt und wurde u.a. in [Sa1994], [HoPa91] und [SuSz94] beschrieben.

Cliques werden aus den Restriktionen, in denen 01-Variablen vorkommen, im Supernode Processing abgeleitet.

Unter der Annahme, dass alle Restriktionen als ≤-Restriktionen formuliert sind, ergeben sich die Mengen C, C^+ und C^- mit $C \subseteq J_{01}$, $C^+ = \{j \mid j \in C, a_j \ge 0\}$ und $C^- = \{j \mid j \in C, a_j \le 0\}$.

Eine Clique C wird durch vier Eigenschaften definiert [SuSz94]:

1. $\left(j \in C^+ \wedge x_j = 1\right) \Rightarrow \left(\left(\forall k \in C^+ - \{j\} \Rightarrow x_j = 0\right) \wedge \left(\forall k \in C^- \Rightarrow x_j = 1\right)\right)$
2. $\left(j \in C^- \wedge x_j = 0\right) \Rightarrow \left(\left(\forall k \in C^- - \{j\} \Rightarrow x_j = 1\right) \wedge \left(\forall k \in C^+ \Rightarrow x_j = 0\right)\right)$
3. C ist eine maximale Menge
4. $|C| \ge 2$

Demzufolge beschreibt eine Clique die logische Implikation, dass nur eine 01-Variable aus der Menge C^+ den Wert Eins oder eine 01-Variable aus der Menge C^- den Wert Null annehmen darf.

Diese Beziehungen lassen sich in einer Summe darstellen, wobei die Variablen aus der Menge C^- als Komplementäre dargestellt werden:

$$\sum_{j \in C^+} x_j + \sum_{j \in C^-} \overline{x_j} \le 1 \text{ mit } \overline{x}_j = 1 - x_j \text{ und j} \in C^- \qquad (4\text{-}24)$$

Beispiel:

$15x_1 + 12x_2 + 10x_3 + 8x_4 + 3x_5 \leq 21$*, wobei alle Variablen 01-Variablen sind.*

1. *Clique:* $C_1 = \{+1,+2,+3,+4\}$
2. *Clique:* $C_2 = \{+2,+3\}$

Aus diesen beiden Cliques können die Ungleichungen

$x_1 + x_2 + x_3 + x_4 \leq 1$

$x_2 + x_3 \leq 1$

formuliert werden.

Durch das Setzen der Komplementäre $\bar{x}_j = 1 - x_j$ mit j $\in$ C^- erhält die Clique die gleiche Struktur wie eine SOS. Wenn eine Clique fraktionelle 01-Variablen enthält, dann kann diese Clique verzweigt werden. Die Wahl des Branchingpunkts ergibt sich wie bei der SOS1 und SOS3 (siehe Kapital 3.2.1). Die Gewichtung erfolgt durch die Kosten der 01-Variablen oder der Reihenfolge der Elemente in der Clique.

Branchen mit den erweiterten Modellklassen

Jeder Variablentyp kann beim Verzweigen einen unterschiedlichen Einfluss auf die anderen Variablen haben. Somit kann eine direkte Setzung einer Variablen zu indirekten Fixierungen, z.B. durch die Bound Reduction, von weiteren Variablen führen. Je mehr Variablen durch eine Setzung fixiert werden, desto eingeschränkter ist der Lösungsraum und desto wahrscheinlicher ist es, eine Integer-Lösung zu finden. Demzufolge wird eine Reihenfolge der Variablentypen in Bezug auf den möglichen Einfluss bei der Verzweigung auf andere Variablen erstellt. Die Variablenauswahl wird in mehrere Ebenen unterteilt, welche in der Abb. 4-4 dargestellt sind. Die Variablenauswahl findet immer in der höchsten Ebene statt, in der es fraktionelle Variablen gibt.

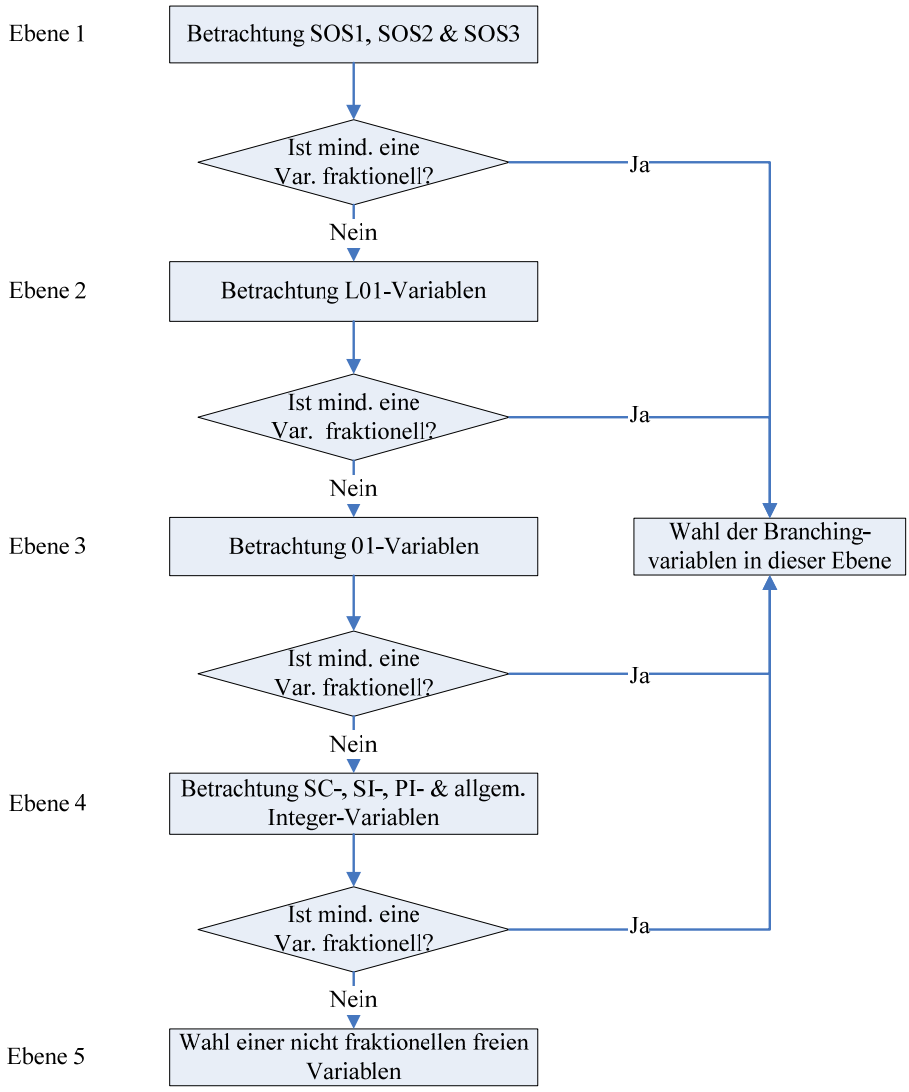

Abb. 4-4: Darstellung der Ebenen der erweiterten Modellklassen

1. SOS-Variablen: Durch das Branchen nach einer SOS werden meistens mehrere 01- oder kontinuierliche Variablen zwischen Null und Eins direkt fixiert, was weitere indirekte Fixierungen bzw. Einschränkungen mit sich führen kann.
2. L01-Variablen: Durch das Branchen nach L01-Variablen werden weitere L01-Variablen und die dazugehörigen kontinuierlichen Variablen direkt fixiert. Dies kann zu weiteren indirekten Fixierungen bzw. Einschränkungen führen.
3. 01-Variablen: Das direkte Fixieren einer 01-Variablen kann weiter indirekte Fixierungen bzw. Einschränkungen mit sich bringen.
4. SC-, SI-, PI- und allgemeine Integer-Variablen: Bei der Verzweigung dieser Variablen kommt es meistens nur zu einer Einschränkung des Wertebereichs und nicht zu einer Fixierung der Branchingvariablen. Demzufolge ist die Wahrscheinlichkeit im Vergleich zu einer direkten Fixierung geringer, dass weitere Variablen indirekt fixiert bzw. eingeschränkt werden können.
5. Freie Variable: Wenn alle ursprünglich fraktionellen Variablen, z.B. durch das Reduced Cost Fixing, fixiert wurden, dann wird die Branchingvariable aus den noch freien Integer-Variablen gewählt. Dabei werden auch die Ebenen 1-4 eingehalten, d.h. bei

z.B. freien 01- und allgemeinen Integer-Variablen wird immer eine 01-Variable als Branchingvariable gewählt.

4.2.2 Knotenauswahlstrategien

Das zweite Entscheidungsproblem des Branch-and-Bounds befasst sich mit der Knotenauswahl. Die Knotenauswahlstrategien legen fest, welcher noch offene Knoten aus der Menge NO als nächster Knoten gewählt werden soll.

In der Literatur werden die Knotenauswahlstrategien in statische und schätzungsbasierte Auswahlverfahren unterschieden [LiSa99]. Des Weiteren gibt es Mehr-Phasen-Strategien, die verschiedene Knotenstrategien miteinander verknüpfen.

4.2.2.1 Statische Auswahlstrategien

Die statischen Auswahlstrategien wählen den Knoten an konkret vorliegenden Werten, wie z.B. Zielfunktionswert oder Unzulässigkeit, aus. Diese Strategien sind nicht von einer Integer-Lösung abhängig. Im Folgenden werden drei statische Verfahren genauer erläutert.

Best-First-Verfahren

Zu den statischen Auswahlstrategien zählt das Best-First-Verfahren, welches die Auswahl aufgrund des Zielfunktionswertes $z(LP)_k$ eines Knotens k trifft. Dabei werden alle noch offenen Knoten der Menge NO betrachtet und der Knoten k mit dem besten Zielfunktionswert wird gewählt.

$$k = \arg\min_{l \in NO}\left(z(LP)_l\right) \tag{4-25}$$

Da immer der Knoten mit dem besten Zielfunktionswert gewählt wird, verbessert sich die Lower Bound des relaxierten LPs monoton. Wenn eine optimale Lösung des IPs mit einem Zielfunktionswert von z_{opt} gefunden ist, dann ist bei allen vorher betrachteten Knoten k $z(LP)_k \leq z_{opt}$. Demzufolge wurde kein überflüssiger Knoten betrachtet. Die Best-First-Strategie führt zu einer breiten Baumstruktur mit einer hohen Anzahl an offenen Knoten.

Ein Nachteil dieses Verfahrens ist, dass die Teilprobleme, die nacheinander gelöst werden, häufig sehr unterschiedliche Einschränkungen haben und somit zu längeren Lösungszeiten führen können.

Depth-First Strategie (LIFO)

Beim LIFO-Verfahren wird der Knoten, der zuletzt bearbeitet wurde, als nächster Knoten gewählt. Dabei wird immer nur einer der beiden Nachfolgeknoten entwickelt, während die restlichen Nachfolgeknoten erstmal zurückgestellt werden. Ist ein Knoten unzulässig, integer oder der Funktionswert schlechter als die bisher beste gefundene IP-Lösung, dann erfolgt ein Backtrack, d.h. es wird der am nächsten liegende noch offene Vorgängerknoten als nächster Knoten gewählt.

Dieses Verfahren zielt auf die Reduktion der Upper Bound durch schnelles Finden von neuen Integer-Lösungen, da diese zumeist in den tieferen Strukturen des Baums zu finden sind. Der Vorteil dieses Verfahrens ist, dass sich der Nachfolgeknoten vom Elternknoten nur durch eine weitere Einschränkung einer Integer-Variablen unterscheidet und somit die Lösungszeit des Teilproblems kürzer ist. Im Gegensatz zu den anderen Strategien müssen

bei diesem Verfahren nicht die Basen der Knoten gespeichert werden, da wie schon geschrieben, eine große Ähnlichkeit des Nachfolgeknotens zum Vorgänger besteht. Besonders zu den Anfängen der mathematischen Optimierung, als die Speicherkapazitäten noch gering waren, war dies ein wichtiger Vorteil. So benutzte auch Dakin [Dak65] und Little et. al. [Litetal63] dieses Verfahren. Dem geringen Speicherplatz und dem schnelleren Lösen steht allerdings ein extrem großer Baum gegenüber. So kann es vorkommen, dass ein Ast des Baums untersucht wird, der keine Verbesserungen bringt, aber viel Zeit vergeht, bevor dieser Ast wieder verlassen wird. Durch dieses Verfahren werden viele überflüssige Knoten gelöst, da durch eine gute globale Lower Bound viele Knoten hätten abgeschnitten werden können.

An einem Beispielmodell und den daraus resultierenden Suchbäumen werden die Unterschiede der LIFO- und der Best-First-Strategie verdeutlicht. Als Branching-Strategie wird die größte Fraktionalität gewählt.

Beispiel

$$
\begin{array}{lrcrcl}
\max & x_1 & + & 1{,}5x_2 & & \\
 & -1{,}5x_1 & + & 2x_2 & \leq & 5 \\
 & 8x_1 & + & 5x_2 & \leq & 18 \\
 & -2x_1 & + & 3x_2 & \leq & 2
\end{array}
$$

$x_1 \geq 0, x_2 \geq 0$ *und Integer*

Eine optimale LP-Lösung lautet: z(LP)=3,588, x_1=1,529 und x_2=1,294 und eine optimale IP-Lösung ist: z(LP) = 2,5, x_1=1 und x_2=1.

In Abb. 4-5 und Abb. 4-6 geben die Nummern an den Knoten die Reihenfolge der Bearbeitung wieder.

Der Suchbaum bei der Best-First-Strategie sieht wie folgt aus:

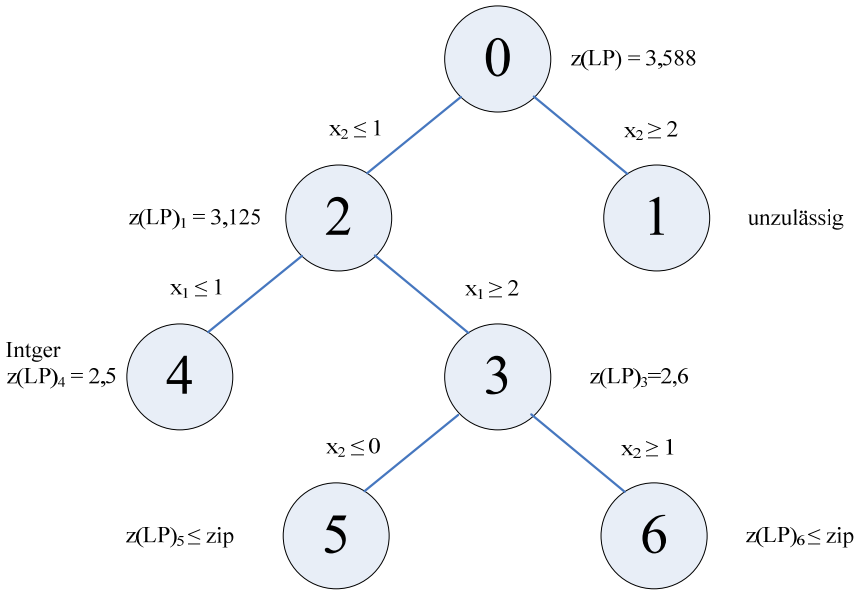

Abb. 4-5: Suchbaum Best-First

Der Suchbaum der LIFO-Strategie hat folgende Ausmaße:

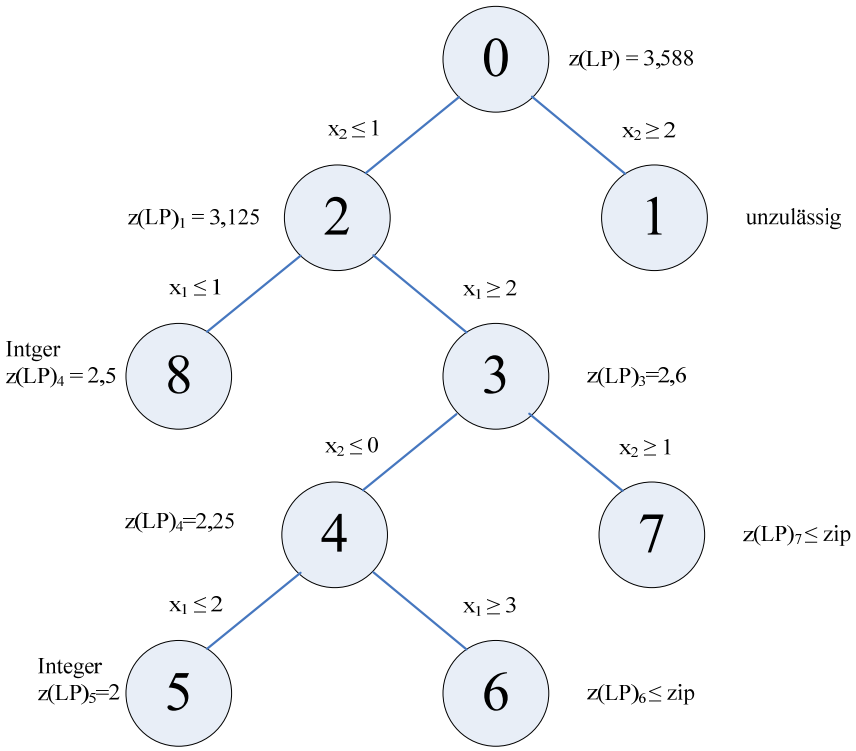

Abb. 4-6: Suchbaum LIFO

Anhand des Beispiels ist zu erkennen, dass bei der LIFO-Strategie überflüssige Knoten k mit $z(LP)_k \leq zip$ bearbeitet wurden (Knoten 5 und 6). Da sich diese optimale Integer-Lösung relativ weit oben im Baum befindet, wurde durch eine falsche Zweigwahl diese optimale IP-Lösung bei der LIFO-Strategie erst beim achten Knoten gefunden.

Geringste Unzulässigkeit

Für eine fraktionelle Lösung des relaxierten LPs an einem Knoten k kann die Unzulässigkeit

$$sif_k = \sum_{j \in J_{fk}} \min\left(f_j, 1 - f_j\right) \tag{4-26}$$

mit k ∈ NO bestimmt werden, indem für alle fraktionellen Variablen die minimale Fraktionalität aufaddiert wird. Der Knoten k mit

$$k = \arg\min_{l \in NO}\left(sif_l\right) \tag{4-27}$$

wird als nächster Knoten gewählt.

Je kleiner die Unzulässigkeit ist, desto weniger Variablen sind fraktionell oder desto weniger fraktionell sind die Integer-Variablen, d.h. die Lösung des relaxierten LPs kann in der Nähe einer Integer-Lösung liegen. Durch das Branchen nach einem Knoten mit geringster Unzulässigkeit besteht die Hoffnung, auf diese Integer-Lösung zu treffen.

4.2.2.2 Schätzungsbasierte Auswahlstrategien

Die statischen Verfahren geben keinen Aufschluss darüber, wie sich die Lösung verhalten könnte, wenn ein bestimmter Knoten im Branching-Prozess gewählt wird. Beispielsweise

gibt der Funktionswert an einem Knoten keine Auskunft darüber, wie sich der Funktionswert an den Nachfolgeknoten entwickelt. Somit ist der Funktionswert ein schlechter Indikator für den Nachfolgeknoten. Mit Hilfe des schätzungsbasierten Verfahrens wird an einem Knoten der Funktionswert der zu erwartenden besten Integer-Lösung für den Nachfolgeknoten geschätzt. Diese Schätzungen können u.a. nach dem Best Projection Prinzip von Hirst [Hir69] und Mitra [Mit73] oder dem Node Estimation Prinzip nach Bénichou et. al. [Béetal71] durchgeführt werden.

Diese Schätzungen geben nur eine ungefähre Richtung des Funktionswertes für den nachfolgenden Knoten an und sind somit keine gültige Lower oder Upper Bound für den Branch-and-Bound-Prozess.

Best Projection

Wenn sif_n die Summe der Unzulässigkeiten an dem Knoten n mit $n \in NO$ ist, sif_0 die Unzulässigkeit und $z(LP)_0$ den Funktionswert am Ausgangsknoten darstellen, dann ist die Schätzung

$$e(LP)_n = z(LP)_n + \left(\frac{zip - z(LP)_0}{sif_0} \right) * sif_n . \tag{4-28}$$

Der Knoten k mit

$$k = \arg\min_{n \in NO} e(LP)_n \tag{4-29}$$

wird als nächster Knoten gewählt.

Bei dem einfachen Best Projection-Kriterium

$$e(LP)_n = \left(\frac{zip - z(LP)_n}{sif_n} \right) \tag{4-30}$$

nach E.M.L. Beale [Bea70] mit $n \in NO$ ist der nächste Knoten

$$k = \arg\max_{n \in NO} e(LP)_n . \tag{4-31}$$

Demzufolge wird aus der Menge NO ein Knoten k gewählt, bei dem die Unzulässigkeit sif_k oder der Funktionswert $z(LP)_k$ dieses Knotens möglichst klein ist.

Wurde noch keine Integer-Lösung gefunden, dann wird für zip eine globale Upper Bound geschätzt.

Node Estimation

Die Schätzung der besten zu erwartenden IP-Lösung an einem Knoten mit Pseudo Kosten ist

$$e(LP)_k = z(LP)_k + \sum_j \min\left\{PCL_j * f_j, PCU_j * (1 - f_j)\right\}, \tag{4-32}$$

wobei $z(LP)_k$ der Funktionswert des aktuellen Knotens, PCL_j und PCU_j die Pseudokosten und f_j der fraktionelle Wert der Variablen x_j mit $j \in J_I$ sind und $k \in NO$ ist. Die Pseudo Kosten für die Estimation werden nach Kapital 4.2.1 berechnet und initialisiert.

Der Knoten k mit

$$k = \arg\min_{n \in NO}\left(e(LP)_n\right) \tag{4-33}$$

wird als nächster Knoten gewählt.

Bei jeder Neuberechnung der Pseudo Kosten kann sich der Wert für die Estimation verändern. Demzufolge muss dieser Wert angepasst werden. Wenn die Pseudo-Kosten auf Null oder auf die Kosten initialisiert werden, ist die Schätzung am Anfang des Baums noch nicht aussagekräftig, da die genauen Werte für die Pseudo Kosten fehlen.

4.2.2.3 Mehr-Phasen-Strategien

Unter den aufgeführten Knotenauswahlstrategien dominiert keine. Jede Strategie hat ihre Vor- und Nachteile. Durch das Verbinden einzelner Strategien wird versucht, die Vorteile der einzelnen Strategien auszuschöpfen und die Nachteile zu beseitigen. Die Grundidee der Zwei-Phasen- und der gemischten LIFO-Strategien wird in [LiSa99] beschrieben.

Gemischte LIFO-Strategie

Bei der LIFO-Strategie wurden Integer-Lösungen schnell gefunden, es wurde weniger Speicherplatz gebraucht und das Lösen erfolgt relativ schnell. Allerdings werden überflüssige Knoten bearbeitet und die Suche kann sich in einem erfolglosen Ast des Baums lange aufhalten. Um dies zu verhindern, wird die LIFO-Strategie mit anderen Knotenauswahlstrategien verknüpft. Dabei wechselt sich das LIFO-Verfahren mit einer anderen Strategie ab. Das LIFO-Verfahren wird nur an einem Knoten mit einem Gap

$$\frac{\left|z(LP) - zlb\right|}{\left|zip - zlb\right|} \leq gaplp \tag{4-34}$$

gestartet. Wenn noch keine Integer-Lösung gefunden wurde, dann ist der Gap immer erreicht. Das Starten und Beenden des LIFO-Verfahrens kann nach folgenden Kriterien erfolgen:

- *LIFO ohne Backtrack*: Das LIFO-Verfahren wird nach mindestens n Knoten im Branch-and-Bound gestartet. Ausgangsknoten ist der zuletzt bearbeitete Knoten. Das Verfahren wird beendet, wenn ein Knoten unzulässig, integer oder der Zielfunktionswert größer als die globale Upper Bound ist. Danach wird die andere Strategie für n Knoten ausgeführt.
- *LIFO mit Backtrack:* Das LIFO-Verfahren wird wie beim LIFO ohne Backtrack nach mindestens n Knoten im Branch-and-Bound gestartet. Ausgangsknoten ist der zuletzt bearbeitete Knoten. Das Verfahren wird nach maximal m Knoten beendet. Danach wird wieder zu der anderen Knotenauswahlstrategie gewechselt.
- *LIFO mit Backtrack und einem kritischen Wert:* Solange keine Integer-Lösung gefunden wurde, wird das LIFO-Verfahren ohne Backtrack angewandt. Vor dem Start des LIFOs wird ein kritischer Wert zcli berechnet. Dieser Wert kann wie folgt definiert werden:
  - Estimation des Ausgangsknotens k: zcli=$e(LP)_k$

– In Abhängigkeit vom Funktionswert des Ausgangsknotens k und der globalen Upper Bound zub mit 0<a<1:

$$zcli = z(LP)_k + (zub - z(LP)_k) * a \tag{4-35}$$

Das LIFO-Verfahren wird solange ausgeführt, bis der Funktionswert $z(LP)_k$ des zuletzt bearbeiteten Knotens k kleiner ist als zcli oder keine offenen Knoten mehr in dem Ast enthalten sind. Daraufhin wird mit einer anderen Knotenauswahlstrategie ein neuer Ausgangsknoten für das LIFO gewählt. Je größer zcli ist, desto länger wird das LIFO-Verfahren ausgeführt. Ist zcli relativ klein, kommt die andere Strategie mehr zum tragen.

Zwei-Phasen-Strategie

Bei einer Zwei-Phasen-Strategie werden zwei Knotenauswahlstrategien miteinander verbunden. Dies hat besonders dann den Vorteil, wenn in der Anfangsphase bei einer Strategie die Werte noch nicht aussagekräftig sind (siehe Estimation mit Pseudo Kosten) oder genaue Vergleichswerte noch nicht berechnet werden können (siehe zip bei Estimation nach Best Projection).

Es können folgende Zwei-Phasen-Strategien aufgestellt werden:

- *Geringste Unzulässigkeit und Estimation mit Best Projection:* Die Estimation kann nur genau berechnet werden, wenn eine Integer-Lösung gefunden wurde. Demnach wird bis zum Finden einer Integer-Lösung die Knotenstrategie Geringste Unzulässigkeit ausgeführt. Danach wird nach der besten Estimation entschieden.
- *Geringste Unzulässigkeit und Estimation mit Pseudo Kosten:* Bei der Initialisierung der Pseudo Kosten auf den Wert der Kosten oder auf Null, ist der Wert der Estimation meistens ungenau und kann zu falschen Entscheidungen führen. Demzufolge wird erst die Strategie Geringste Unzulässigkeit angewandt, bis eine Integer-Lösung gefunden wird. Danach wird mit der Estimation fortgefahren.
- *LIFO und Best-First-Strategie:* Eine LIFO-Strategie kann sehr schnell eine Integer-Lösung finden und verbessert so die globale Upper Bound. Eine Best-First-Strategie findet eher langsamer eine IP-Lösung, verbessert allerdings die globale Lower Bound. Demzufolge wird als erstes die LIFO-Strategie ausgeführt, bis eine Integer-Lösung gefunden wurde. Danach wird zur Best-First-Strategie gewechselt, die dann beweisen soll, dass die gefundene IP-Lösung optimal ist.
- *Eine Knotenauswahlstrategie und Best-First-Strategie:* Es wird eine beliebige Knotenauswahlstrategie ausgeführt. Nach einer bestimmten Knotenanzahl wird immer der Knoten mit dem besten Zielfunktionswert aus der Knotentabelle gewählt. Dadurch soll die globale Lower Bound angehoben werden.

Drei-Phasen-Strategie

Bei den Drei-Phasen-Strategien werden drei Knotenauswahlstrategien miteinander verknüpft. Im Gegensatz zu den Zwei-Phasen-Strategien werden die Drei-Phasen-Strategien durch eine Strategie erweitert, die dann zum Tragen kommt, wenn der Abstand der globalen Lower Bound zur globalen Upper Bound nur noch einen bestimmten Gap beträgt.

Als dritte Strategie kommen folgende Auswahlstrategien in Frage:

- *Best-First:* Durch dieses Verfahren wird die globale Lower Bound verbessert und somit der gap noch weiter verkleinert
- *„Percentage Error":* Der prozentuelle Fehler wurde von Forrest et. al. [FoHiTo74] und Beale [Bea79] eingeführt und ist

$$PE(LP)_n = 100 * \frac{zip - e(LP)_n}{z(LP)_n - zip} \tag{4-36}$$

für jeden zur Wahl stehenden Knoten n ∈ NO. Als nächster Knoten wird der Knoten k mit dem geringsten Fehler, d.h. mit

$$k = \arg\min_{n \in NO} PE(LP)_n, \tag{4-37}$$

gewählt.

4.3 Heuristiken

NP-harte Probleme sind in der Praxis schwer zu lösen. In manchen Fällen braucht ein IP-Solver sehr lange, um eine Integer-Lösung zu finden. Demzufolge sind Heuristiken sehr wichtig, da sie versuchen, möglichst schnell eine Integer-Lösung zu finden.

Bei einer Heuristik wird das Problem P durch Fixierungen oder zusätzliche Restriktionen in ein relaxiertes LP' verändert. Der Lösungsraum F(LP') ist eine Teilmenge von F(LP). Somit ist jede in der Heuristik gefundene Integer-Lösung auch für das Ausgangsproblem gültig. Da der Lösungsraum F(LP') eingeschränkter ist als F(LP), wird eine Integer-Lösung gegebenenfalls schneller gefunden. Mit einer in der Heuristik gefundenen Integer-Lösung verändert sich die globale Upper Bound von P und der Lösungsraum F(LP) wird verkleinert.

Heuristiken können entweder nach dem Supernode Processing oder während des Branch-and-Bounds als Local Search aufgerufen werden. Der einzige Zweck der Heuristiken im Branch-and-Bound ist das Finden einer besseren Integer-Lösung. Die Optimalität einer schon gefundenen Lösung wird mit den Heuristiken nicht bewiesen.

Da die mathematische Optimierung von ganzzahligen Problemen eine Großzahl von verschiedenen kombinatorischen Modellen umfasst, gibt es keine Heuristik, die für alle Probleme eine gute Lösung findet. Demzufolge gibt es eine Vielzahl von Heuristiken, die eingesetzt werden können. Dazu zählen die Relaxation-based Search Space Strategien (RSS) mit dem Spezialfall des Totalen Rundens, die nach dem Supernode Processing aufgerufen werden, und die Local Search, die eine Gruppe von Heuristiken umfasst, die sowohl vor als auch während des Branch-and-Bounds eingesetzt werden können.

In diesem Kapitel werden zunächst die Local Search Techniken beschrieben. Danach wird auf die Heuristiken vor und während des Branch-and-Bounds eingegangen.

4.3.1 Local Search

Die Local Search Heuristik sucht nach Integer-Lösungen im Umkreis von einer schon gefundenen Integer-Lösung oder der aktuellen LP-Lösung.

Zu den Local Search Heuristiken zählen u.a. die Relaxation Induced Neighborhood Search und das Local Branching.

Relaxation Induced Neighborhood Search (RINS)

Die ursprünglichen Ideen der Relaxation Induced Neighborhood Search gehen auf Danna et.al. [DaRoPa04] zurück und bezogen sich nur auf die Einsetzung der Local Search im Branch-and-Bound-Prozess. Bei dieser Heuristik wird die aktuelle Integer-Lösung mit der LP-Lösung an dem gerade betrachteten Knoten verglichen. Alle Integer-Variablen, die in beiden Lösungen den gleichen ganzzahligen Wert aufweisen, werden auf diesen Wert fixiert.

Für das Problem $P = \{x : Ax \le b\}$ ergibt sich ein IP-Lösungvektor z und ein LP-Lösungsvektor y. Die Menge $J_{yz} \subseteq J_I$ ist die Menge der Integer-Variablen, die in beiden Lösungen den gleichen ganzzahligen Wert annehmen. Dann verändert sich das Problem P in $P' = \{x : Ax \le b; x_j = z_j \forall\ j \in J_{yz}\}$.

Die Wahl dieser beiden Lösungen begründet sich darauf, dass die aktuelle Integer-Lösung wahrscheinlich nicht optimal ist, aber alle Ganzzahligkeitsbedingungen erfüllt und dass die aktuelle LP-Lösung zwar optimal ist und einen besseren Funktionswert hat, aber die Ganzzahligkeit verletzt wird. Somit erfüllen beide Lösungen jeweils eines der beiden Ziele. Durch das Fixieren der Variablen, wird sowohl in der Nachbarschaft der Integer-Lösung als auch in der Nachbarschaft der LP-Lösung gesucht. Wenn die eine Lösung qualitativ schlecht ist, dann hilft vielleicht die andere Lösung durch eine bessere Nachbarschaft. Durch das Fixieren wird eine ganzzahlige Teillösung gebildet. Für die restlichen Variablen, die die Ganzzahligkeit bzw. die Optimalität verhindern, wird im Branch-and-Bound-Verfahren versucht, eine komplette Lösung zu finden, die ganzzahlig und optimal ist.

Local Branching

Die ursprüngliche Idee für das Local Branching stammt von M. Fischetti und A. Lodi [FiLo03] und wurde von E. Danna [DaRoPa04] in der Anwendung verändert. In beiden Quellen wurde das Local Branching nur im Branch-and-Bound angewandt. Die folgende beschriebene Local Branching Heuristik nimmt die Grundidee von E. Danna auf.

Bei dem Local Branching wird der Lösungsraum durch eine zusätzliche Restriktion eingeschränkt. Die Local Branching Restriktion für ein reines oder gemischtes Binär-Modell

$$\sum_{j \in S_1} x_j + \sum_{j \in S_2} (1 - x_j) \le k \tag{4-38}$$

gibt an, dass maximal k ganzzahlige 01-Variablen aus einer IP-Lösung oder LP-Lösung ihren Wert verändern dürfen. Die Menge $S_1 = \{j \in J_{01} : x_j = 0\}$ umfasst alle 01-Variablen an der Lower Bound und $S_2 = \{j \in J_{01} : x_j = 1\}$ alle 01-Variablen an der Upper Bound. Die Einschränkung des Lösungsraums erfolgt somit „sanft“, da nicht explizit Variablen fixiert werden.

Durch das Lösen des veränderten Problems $P' = \left\{ x : Ax \le b; \sum_{j \in S_1} (1 - x_j) + \sum_{j \in S_2} x_j \le k \right\}$ wird in der Nachbarschaft der betrachteten Lösung nach einer besseren Integer-Lösung gesucht.

Bei Modellen mit allgemeinen Integer-Variablen wird die Local-Branching-Restriktion erweitert:

$$\sum_{j \in S_1} \frac{x_j - l_j}{u_j - l_j} + \sum_{j \in S_2} \frac{u_j - x_j}{u_j - l_j} \leq k \tag{4-39}$$

Die Menge $S_1 = \{ j \in J_I : x_j = l_j \}$ beinhaltet alle Integer-Variablen an der Lower Bound und $S_2 = \{ j \in J_I : x_j = u_j \}$ alle Integer-Variablen an der Upper Bound. Alle Integer-Variablen, die einen ganzzahligen Wert zwischen den Bounds haben, werden in diese Restriktion nicht aufgenommen. Die Restriktion (4-36) ist somit ein Spezialfall von (4-37).

4.3.2 Heuristiken vor dem Branch-and-Bound

Für jede in diesem Kapital beschriebene Heuristik vor dem Branch-and-Bound können folgende Eigenschaften gelten:

- *Fixierung der Nichtbasis-Variablen:* Bei den Heuristiken können ein prozentualer Teil der Nichtbasis-Variablen mit den im Betrag größten reduzierten Kosten auf die Bound, an der die Variable in der LP-Lösung nach dem Supernode Processing liegt, fixiert werden. Diese Fixierung findet vor der eigentlichen Heuristik statt.
- *Der zu rundende Variablentyp:* Entweder werden alle Integer-Variablen oder nur die 01-Variablen gerundet. Des Weiteren besteht die Möglichkeit, dass nur 01-Variablen nach dem Rundungsverfahren gerundet werden und alle anderen Integer-Variablen unabhängig von der Fraktionalität auf- bzw. abgerundet werden.
- *Rundungsintervalle:* Liegt die Fraktionalität f mit $0 \leq f < 1$ einer freien Integer-Variablen in den Rundungsintervallen [0,rlb] oder [rub,1], dann wird die Variable auf einen Integer-Wert gerundet.

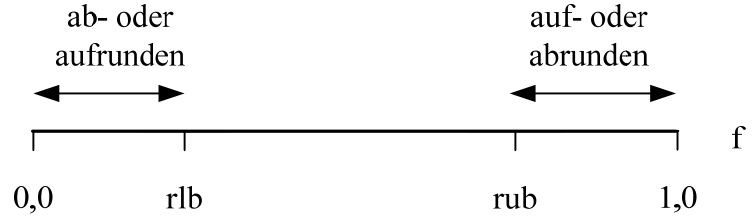

Abb. 4-7: Rundungsintervalle

  Wenn rlb<0 ist, wird an der Lower Bound und bei rub>1, wird an der Upper Bound nicht gerundet.
- *Rundungsrichtung:* Die Rundungsrichtung gibt an, in welche Richtung eine Integer-Variable mit einer Fraktionalität von f>0 gerundet werden soll. Zum einen gibt es das klassische Runden, wo bei $0 < f \leq$ rlb abgerundet und bei rub$\leq f < 1$ aufgerundet wird. Das entgegengesetzte Runden rundet bei $0 < f \leq$ rlb auf und bei rub$\leq f < 1$ ab. Integer-Variablen mit f=0 werden immer auf den Integer-Wert gesetzt, den sie in der Lösung annehmen.
- *Rundungshäufigkeit:* Die Rundungshäufigkeit gibt an, wie oft der Rundungsprozess durchgeführt werden soll, bevor der Branch-and-Bound-Prozess aufgerufen wird.

- *Veränderung der Rundungsparameter:* Können keine Integer-Variablen gerundet werden oder führt ein Rundungsdurchlauf zu einer Unzulässigkeit, dann können die Rundungsintervalle durch den Parameter rv angepasst werden.
- *Abbruchkriterien:* Die Heuristik kann nach einer gegebene Zeit, Knotenanzahl oder bei einem bestimmten Gap beendet werden.
- *Branching-Strategien:* In der Heuristik können andere Strategien angewandt werden als im Branch-and-Bound nach der Heuristik.

Da nicht für jede Heuristik alle aufgelisteten Eigenschaften gelten, kennzeichnet die Tabelle Tab. 4-1 für jede Heuristik die benötigten Eigenschaften durch jeweils ein Kreuz (X).

Tab. 4-1: Eigenschaften der Heuristiken vor dem Branch-and-Bound

	RSS	Totale Runden	RINS	Local Branching
Nichtbasis-Variablen	X	X	X	X
Variablentyp	X	-	X	X
Rundungsintervalle	X	X	X	-
Rundungsrichtung	X	-	-	-
Rundungshäufigkeit	X	-	-	-
Veränderung des Rundungsparameters	X	X	X	-
Abbruchkriterien	X	X	X	X
Strategien	X	-	X	X

Mit den festgelegten Eigenschaften einer Heuristik wird das Rundungsverfahren ausgeführt bzw. beim Local Branching eine Restriktion eingefügt. Werden keine Variablen fixiert, dann werden die Rundungsintervalle um rv vergrößert. Wurde durch das Runden eine Integer-Lösung mit einem Gap kleiner als der Heuristik Gap ermittelt, dann wird die Heuristik beendet. Ansonsten wird der eingeschränkte Lösungsraum mit dem Branch-and-Bound-Verfahren durchsucht.

Nach Beendigung des Branch-and-Bounds ist entweder ein Abbruchkriterium (Knotenlimit, Zeitlimit oder Gap) erreicht worden oder der eingeschränkte Lösungsraum war zu klein, so dass die Suche vorzeitig beendet wurde. Bei einem zu kleinen Lösungsraum werden die Setzungen rückgängig gemacht, die Rundungsintervalle jeweils um rv verkleinert und der Branch-and-Bound für die verbliebene Knotenanzahl oder das Zeitlimit gelöst. Dieser Vorgang wiederholt sich solange, bis ein Abbruchkriterium erreicht wurde.

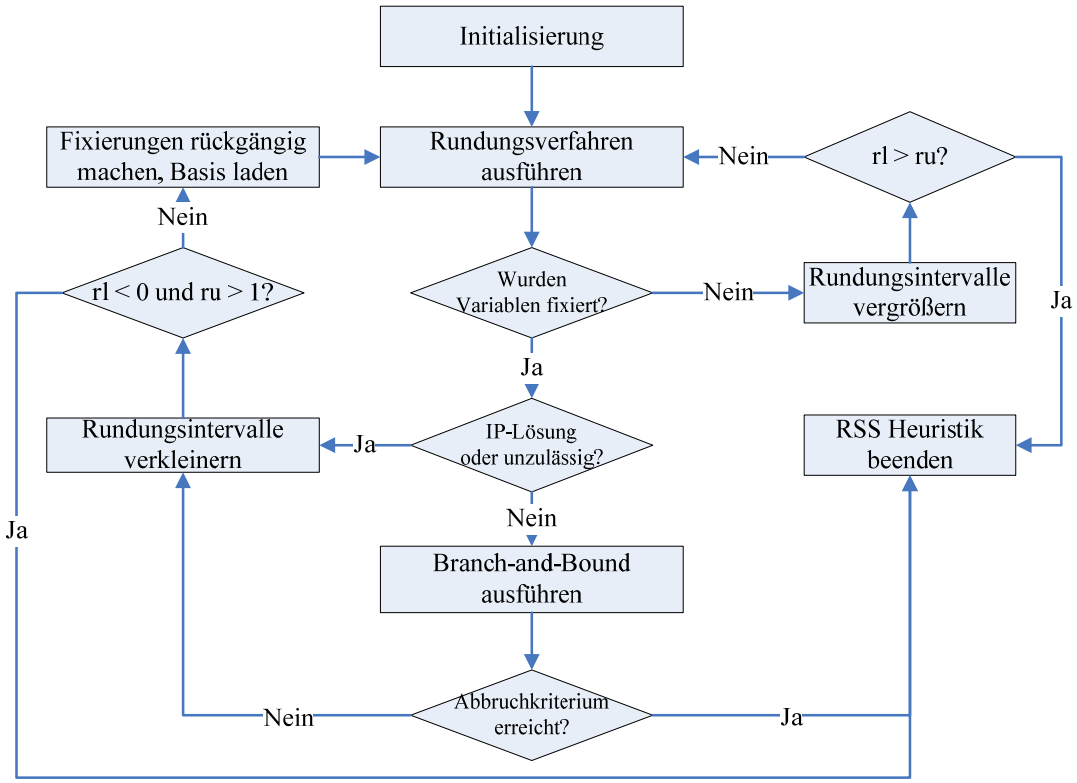

Abb. 4-8: Schematischer Ablauf Heuristik vor dem Branch-and-Bound

4.3.2.1 Relaxation-based Search Space Heuristik

Bei der Relaxation-based Search Space Heuristik (RSS Heuristik) werden Fixierungen ausgehend von der LP-Lösung nach dem Supernode-Processing (LP_{SNP}) durchgeführt. Die Grundidee der Heuristik stammt von P. Christophel [Chr05].

Bei dieser Heuristik werden freie Basis-Integer-Variablen x_j mit $j \in J_I$ und einer Fraktionalität von f_j in den Rundungsintervallen [0,rlb] und [rub,1] ab- bzw. aufgerundet.

Es können mehrere Rundungsvorgänge erfolgen, wobei solange gerundet und das relaxierte LP nach jedem Rundungsvorgang gelöst wird, bis

- eine Integer-Lösung gefunden wurde,
- das relaxierte Problem unzulässig ist,
- alle n Rundungsvorgänge absolviert sind oder
- keine weiteren Variablen fixiert werden können.

Bei einer Unzulässigkeit wird der letzte Rundungsvorgang aufgehoben, so dass die relaxierte LP-Lösung zulässig ist. Wenn eine Integer-Lösung gefunden wurde und der Heuristik-Gap noch nicht erreicht ist, werden die Rundungsintervalle um jeweils rv verkleinert und der Rundungsprozess erneut gestartet. Ansonsten wird der eingeschränkte Lösungsraum mit dem Branch-and-Bound-Prozess durchsucht.

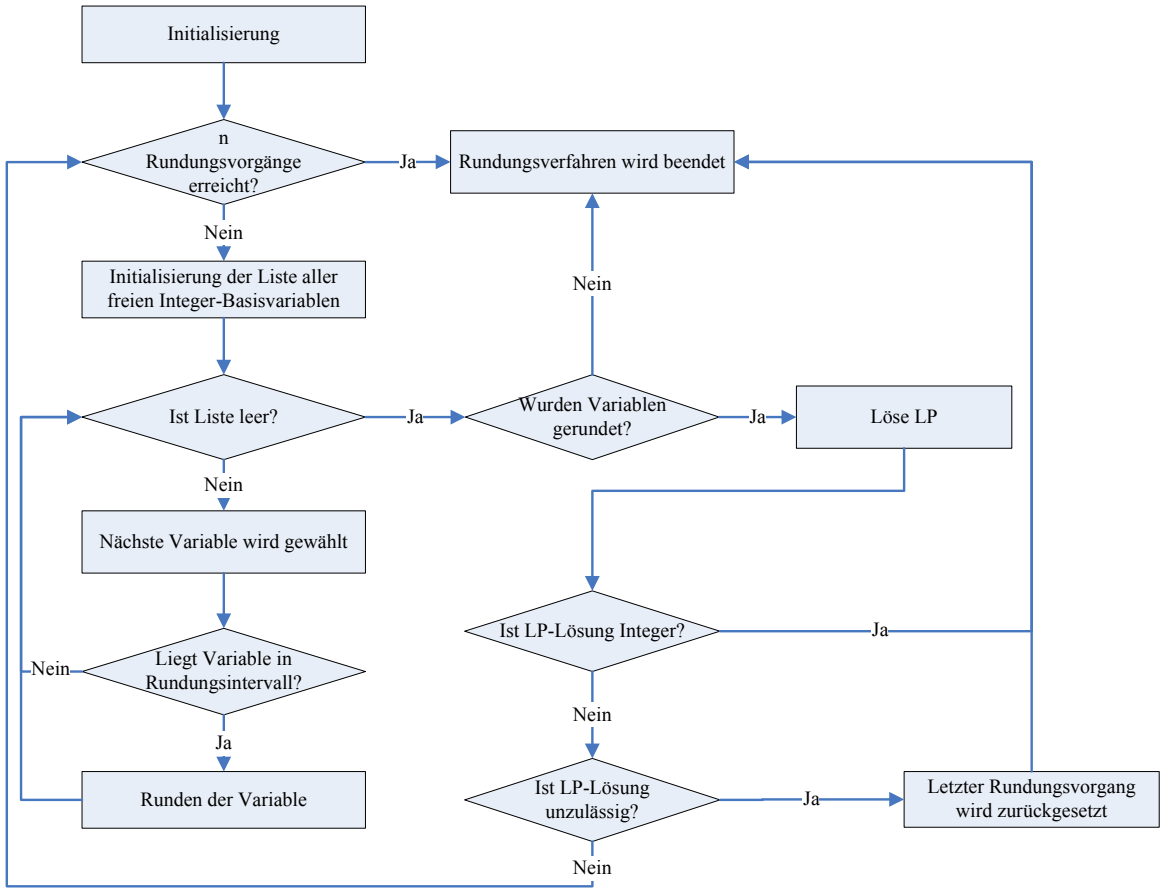

Abb. 4-9: Schematischer Ablauf RSS-Heuristik

4.3.2.2 Totales Runden

Ein Spezialfall der RSS Heuristik ist das Totale Runden. Bei dem Totalen Runden werden alle freien Integer-Variablen auf Basis der optimalen LP-Lösung fixiert, so dass beim Lösen des relaxierten LPs das Ergebnis entweder Integer oder unzulässig ist. Das Totale Runden setzt sich aus dem dynamischen Runden und der Gap-rounding Strategie zusammen. Die Gap-rounding Strategie wird nur aufgerufen, wenn beim dynamischen Runden der Heuristik-Gap nicht erreicht wurde.

Dynamisches Runden

Ausgehend von der LP-Lösung nach dem Supernode Processing LP_{SNP} werden alle freien fraktionellen Integer-Variablen der Menge J_I entsprechend der Rundungsintervalle gerundet und fixiert. Nach der Fixierung wird das relaxierte LP erneut gelöst. Falls die Lösung nicht Integer ist, wird eine Fixierung mit der neuen LP-Lösung durchgeführt. Können keine Variablen fixiert werden, dann werden die Rundungsintervalle durch den Parameter rv vergrößert. Dieser Vorgang wiederholt sich solange bis das LP integer oder unzulässig ist.

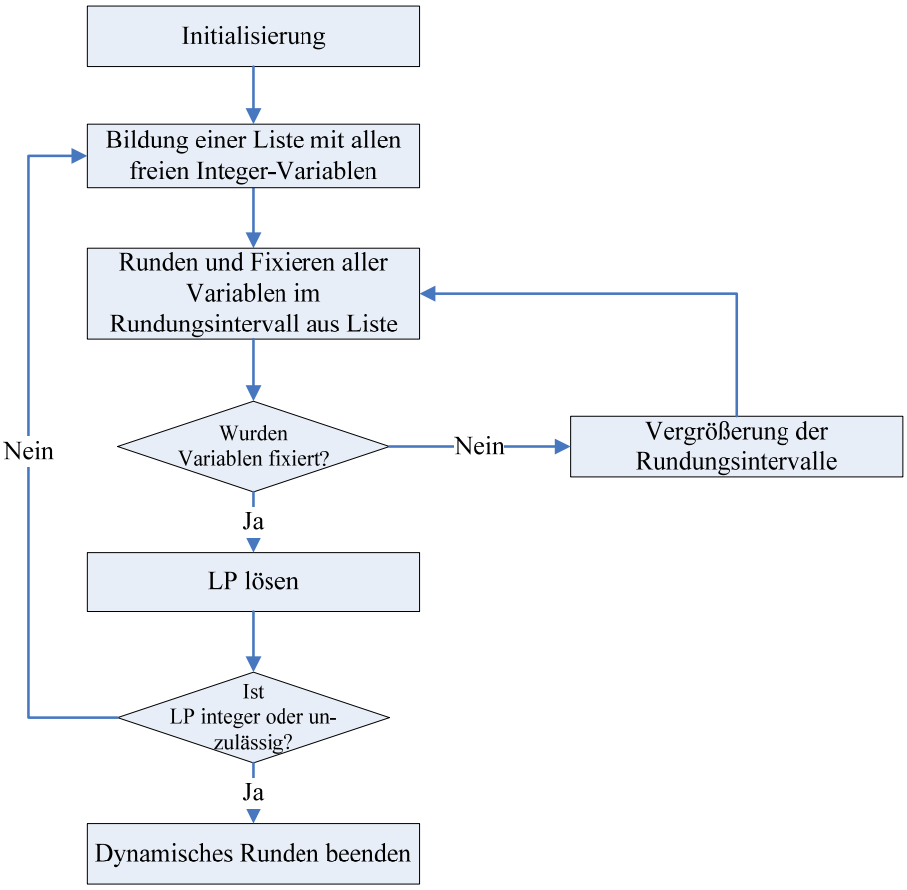

Abb. 4-10: Schematischer Ablauf Dynamisches Runden

Gap-rounding Strategie

Die ursprüngliche Idee dieser Strategie stammt von P. Christophel [Chr05]. Wenn der fraktionelle Wert f_j einer freien Integer-Variablen x_j mit $j \in J_I$ in einem der beiden Rundungsintervalle liegt, dann wird die Variable auf den nächsten ganzzahligen Wert gerundet. Liegt f_j nicht in den Intervallen, dann wird x_j mit positiven Kosten aufgerundet und mit negativen Kosten abgerundet.

$$\bar{x}_j = \begin{cases} \lfloor x_j \rfloor & \text{für } f_j \le rlb \vee \left(c_j \le 0 \wedge rlb < f_j < rub\right) \\ \lceil x_j \rceil & \text{für } f_j \ge rub \vee \left(c_j \ge 0 \wedge rlb < f_j < rub\right) \end{cases} \tag{4-40}$$

Das Runden nach den Kosten führt zu einem Anstieg des Zielfunktionswertes. Die relaxierte LP-Lösung wird weiter in den Lösungsraum verlagert. Wenn eine Variable zur nächsten Integer-Variable gerundet wird, dann verändert sich der Zielfunktionswert bei einer geringeren Fraktionalität kaum. Die LP-Lösung kann dadurch allerdings den Lösungsraum verlassen. Um eine gute Integer-Lösung durch das Runden zu erhalten, werden beide Strategien miteinander verknüpft.

Da die Wahrscheinlichkeit sehr gering ist, beim einmaligen Durchlaufen der Strategie eine Integer-Lösung zu erhalten, wird diese Strategie mehrmals durchlaufen. Dabei werden die Rundungsintervalle durch die Veränderung rv der Rundungsparameter bei jedem Durchlauf größer. Die anfänglich dominierende Rundungsstrategie nach den Kosten weicht langsam der Rundungsstrategie nach dem nächsten ganzzahligen Wert.

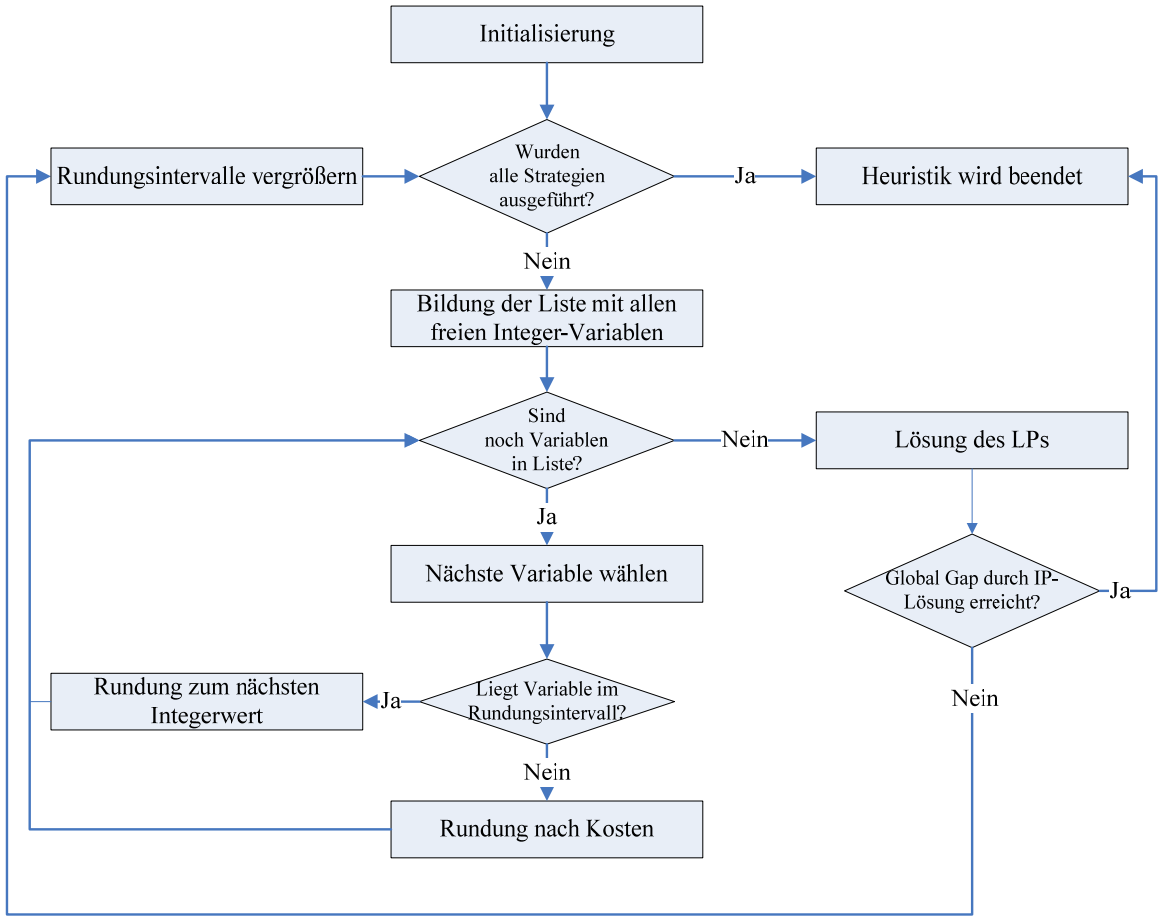

Abb. 4-11: Schematischer Ablauf Gap-rounding Strategie

4.3.2.3 RSS-Heuristik mit Totalem Runden

Innerhalb des Branch-and-Bounds einer RSS-Heuristik kann an jedem n-ten Knoten eine Rundungsheuristik aufgerufen werden. Dabei werden nach dem Prinzip des Totalen Rundens alle Integer-Variablen gerundet, so dass das Ergebnis entweder integer oder unzulässig ist. Beim Runden wird in Basis- und Nichtbasis-Variablen unterschieden. Zunächst werden die Basis-Variablen gerundet, wobei die Reihenfolge der zu rundenden Variablen aufsteigend nach der Fraktionalität gegeben ist. Die Bound Reduction überprüft die Zulässigkeit der Rundung. Falls keine Unzulässigkeit entstanden ist, werden die Nichtbasis-Variablen gerundet und danach das LP gelöst.

4.3.2.4 RINS vor dem Branch-and-Bound

Da die RINS-Strategie immer zwei Lösungen miteinander vergleicht, wird neben der LP-Lösung nach dem Supernode Processing LP_{SNP} eine weitere Lösung benötigt.

Es stehen zwei alternative Lösungen zur Auswahl:

- Als Vergleichslösung wird die originale LP-Lösung gewählt. In dieser Lösung werden nur die Integer-Variablen mit einem ganzzahligen Wert betrachtet. Der Nachteil dieser Variante ist, dass durch ein fehlendes Supernode Processing die beiden Lösungen gleich sind. In diesem Fall würden alle Variablen mit einem ganzzahligen Wert in der LP-Lösung fixiert werden.
- Es wird erst das Totale Runden durchgeführt. Sobald eine Integer-Lösung gefunden wird, startet die RINS-Heuristik. Der Nachteil dieser Variante ist, dass bei keiner gefundenen IP-Lösung in den Rundungsheuristiken die RINS-Heuristik nicht gestartet wird.

Die beiden Lösungen werden miteinander verglichen. Die Größe der Nachbarschaft ist von dem Rundungsparameter r mit $r = rlb = 1 - rub$ abhängig. x^I_j mit $j \in J_I$ gibt den Wert einer ganzzahligen Integer-Variablen j in der Integer-Lösung bzw. originalen LP-Lösung an. Wenn der Wert der Variablen x_j aus LP_{SNP} im Intervall $[x^I_j\text{-}r, x^I_j\text{+}r]$ liegt, dann wird die Variable auf den ganzzahligen Wert gerundet und fixiert. Je größer der Parameter r ist, desto mehr Variablen können fixiert werden und desto kleiner ist die Nachbarschaft.

Wird in der Heuristik eine neue Integer-Lösung gefunden, dann wird der Durchlauf beendet und ein neuer auf Basis der neuen Integer-Lösung und LP_{SNP} wird gestartet.

4.3.2.5 Local Branching vor dem Branch-and-Bound

Da die Local Branching Heuristik vor dem Branch-and-Bound-Prozess aufgerufen wird und es demzufolge noch keine IP-Lösung gibt, wird mit der Lösung LP_{SNP} die Local Branching Restriktion gebildet. Dabei werden nur die Variablen betrachtet, die einen ganzzahligen Wert in LP_{SNP} aufweisen.

Die Größe der Nachbarschaft ist von der Anzahl der Variablen in der Restriktion abhängig. Je geringer die Anzahl der ganzzahligen Integer-Variablen in der Restriktion, desto größer ist die Nachbarschaft, und je kleiner der Nachbarschaftsparameter k, desto kleiner ist sie.

Wurde der Heuristikdurchlauf vorzeitig beendet und wurde keine Integer-Lösung gefunden, dann wird der Lösungsraum durch $k' = k + \frac{k}{2}$ vergrößert. Wenn eine Integer-Lösung beim vorzeitigen Beenden gefunden wurde, dann wird die Local-Branching-Restriktion mit dieser Integer-Lösung und gleich bleibendem k neu gebildet.

4.3.3 Heuristiken während des Branch-and-Bounds

Eine Heuristik während des Branch-and-Bounds kann durch folgende Eigenschaften definiert werden:

- *Definition der Nachbarschaft:* Die Nachbarschaft gibt an, in welchem Bereich des Lösungsraums der Branch-and-Bound-Prozess angewandt werden soll und ist von der aktuellen LP und/oder der besten IP-Lösung abhängig.
- *Definition der Nachbarschaftsgröße:* Die Größe der Nachbarschaft gibt an, in welchem Umkreis um eine LP- bzw. IP-Lösung gesucht werden soll. Je größer die Nachbarschaft, desto größer der eingeschränkte Lösungsraum.
- *Definition des Heuristikstarts:* Jede Heuristik während des Branch-and-Bounds wird nach einem bestimmten Knotenlimit bei einem Knoten mit einem Gap von

$$\frac{|z(LP) - zlb|}{|zip - zlb|} \leq gaplp \tag{4-41}$$

gestartet. Wurde noch keine Integer-Lösung gefunden, dann ist der Gap immer erreicht.

- *Definition der Abbruchkriterien:* Die Suche in dem eingeschränkten Lösungsraum wird abgebrochen, wenn
  - der Lösungsraum vollständig untersucht wurde,
  - ein bestimmter Gap erreicht wurde,
  - ein Knotenlimit erreicht wurde oder
  - eine gegebene Anzahl an Integer-Lösungen gefunden wurde.

Die gesamte Heuristik während des Branch-and-Bounds wird beim Erreichen des Heuristik-Gaps beendet.

Die Nachbarschaft und die Nachbarschaftsgröße werden von den einzelnen Heuristiken definiert. Der Heuristikstart und die Abbruchkriterien sind für jede Heuristik, mit Ausnahme des Local Total Roundings, während des Branch-and-Bounds gleich.

Nachdem der eingeschränkte Lösungsraum bis zum Abbruchkriterium mit dem Branch-and-Bound durchsucht wurde, werden die Einschränkungen, wie z.B. Fixierungen durch die RINS-Heuristik, rückgängig gemacht oder die Local-Branching-Restriktion beim Local Branching inaktiviert. Wurde eine bessere Integer-Lösung gefunden, dann wird diese Lösung als neue globale Upper Bound benutzt. Der ursprüngliche Branch-and-Bound-Prozess wird weiter gelöst, bis die Kriterien für den Heuristikstart eintreten.

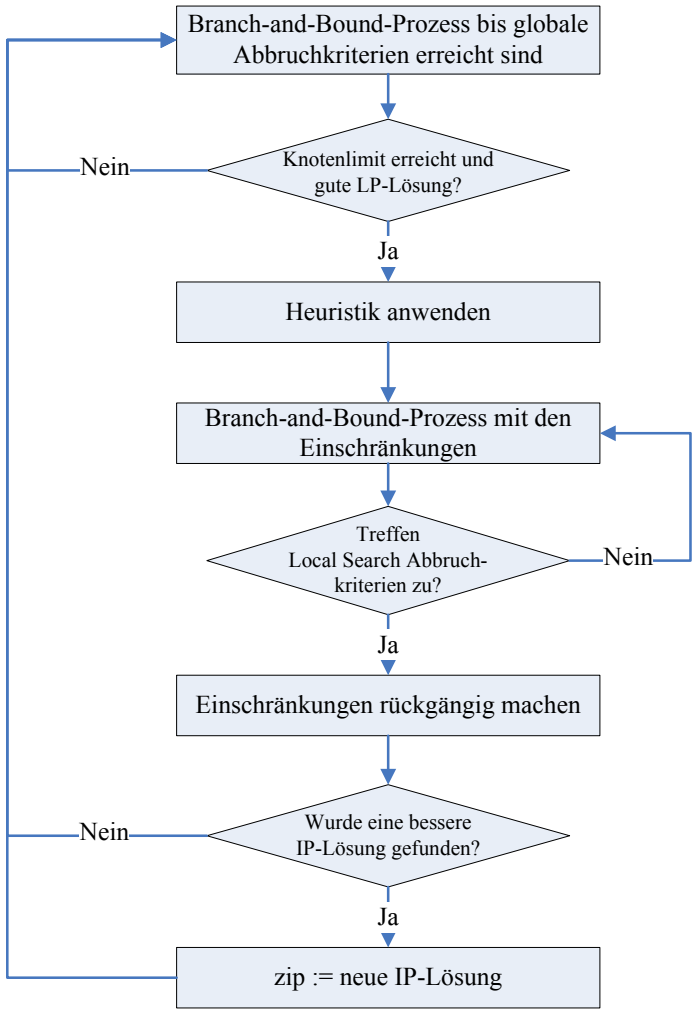

Abb. 4-12: Schematischer Ablauf Local Search während des Branch-and-Bounds

Im Folgenden werden die Nachbarschaften und die Nachbarschaftsgrößen der vier Local Search Heuristiken RINS, Local Branching, Local Rounding und Local Total Rounding definiert.

RINS während des Branch-and-Bound

Die Nachbarschaft und die Nachbarschaftsgröße bei der RINS-Heuristik sehen wie folgt aus:

- *Nachbarschaft:* Beim Einsetzen der RINS während des Branch-and-Bounds wird die aktuelle LP-Lösung mit der besten IP-Lösung verglichen. Falls noch keine IP-Lösung gefunden wurde, wird die LP-Lösung mit der LP-Lösung nach dem Supernode Processing verglichen. Alle Variablen, die in beiden Lösungen den gleichen ganzzahligen Wert aufweisen, werden auf diesen fixiert.
- *Nachbarschaftsgröße:* Die Größe der Nachbarschaft ergibt sich durch die Anzahl der fixierten Variablen. Je mehr Variablen fixiert werden, desto kleiner ist die Nachbarschaft. Bei der RINS Heuristik kann die Größe der Nachbarschaft von außen nicht gesteuert werden, da sich die Anzahl der fixierten Variablen aus den beiden Lösungen ergibt. In der Regel ist bei einer Nachbarschaftsbildung mit einer

Integer-Lösung die Nachbarschaft kleiner als bei dem Vergleich von zwei LP-Lösungen.

Local Branching während des Branch-and-Bound

Die Nachbarschaft und die Nachbarschaftsgröße werden beim Local Branching wie folgt gebildet:

- *Nachbarschaft*: Die Local Branching Restriktion wird mit der aktuelle LP-Lösung oder bei vorhandener IP-Lösung mit dieser gebildet. Alle Integer-Variablen, die in der Lösung einen ganzzahligen Wert haben, werden in die Restriktion aufgenommen. Mit einer IP-Lösung kann immer nur einmal eine Local-Branching-Restriktion gebildet werden. Wird keine weitere IP-Lösung gefunden, dann erfolgt die Bildung wieder mit der aktuellen LP-Lösung.
- *Nachbarschaftsgröße:* Die Größe der Nachbarschaft ist von der Anzahl der Variablen in der Local Branching Restriktion abhängig. Bei einer Integer-Lösung sind alle Integer-Variablen in der Restriktion. Somit ist die Nachbarschaft kleiner als bei einer LP-Lösung, wo nicht alle Integer-Variablen ganzzahlig sind. Die Nachbarschaft ist zusätzlich von dem Parameter k abhängig. Je größer der Parameter k ist, desto mehr Variablen können in der Restriktion ihren Wert wechseln und desto größer ist die Nachbarschaft.

Local Rounding

Die Nachbarschaft und die Nachbarschaftsgröße werden beim Local Rounding wie folgt definiert:

- *Nachbarschaft:* Beim Local Rounding werden ausgehend von der aktuellen LP-Lösung an einem Knoten alle freien Integer-Basis-Variablen in einem Abstand von rlb zu einem Integer-Wert auf diesen Wert gerundet und fixiert:

$$\bar{x}_j = \begin{cases} \lfloor x_j \rfloor & \text{für } f_j \le rlb \\ \lceil x_j \rceil & \text{für } f_j \ge 1 - rlb \end{cases} \quad \forall\ j \in J_I \tag{4-42}$$

- *Nachbarschaftsgröße:* Die Größe der Nachbarschaft ist abhängig von der Anzahl der fixierten Variablen. Je mehr Variablen fixiert sind, desto kleiner ist der zu durchsuchende Lösungsraum. Die Anzahl der zu fixierenden Variablen wird durch den Parameter rlb beeinflusst. Ist rlb relativ groß, dann werden viele Variablen fixiert und ist er gleich Null, dann werden nur die Basis-Variablen, die in der LP-Lösung einen Integer-Wert annehmen, fixiert. Die Anzahl ist aber auch von der Fraktionalität der LP-Lösung abhängig.

Local Total Rounding

Beim Local Total Rounding wird an bestimmten Knoten ein totales Runden durchgeführt. Die Rundungsstrategie ist die gleiche wie bei der Heuristik RSS mit Totalem Runden (siehe Kapitel 4.3.2.3). Falls durch das Runden eine Integer-Lösung gefunden wird, verbessert sich die globale Upper Bound und der Lösungsraum wird weiter eingeschränkt. Die Rundungen werden rückgängig gemacht und der Branch-and-Bound-Prozess wird fortgesetzt.