Bewertung und Analyse von Tearing Algorithmen

Diplomarbeit

von

Yuhan Yan
geboren am 25. Juni 1982 in Stadt Nanjing

12. August 2007

Betreuer:
Prof. Dr. H. Kleine Büning
Prof. Dr. W. Hauenschield

Universität Paderborn
Fakulität EIM
Institut für Informatik

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Lösung differential-algebraischer Gleichungssysteme (DAE-System) ist in vielen Bereichen wie Mathematik, Informatik und Physik ein interessantes Problem (Jus05). Viele praktische Beispiele lassen sich nicht mehr mit rein differentialen oder rein algebraischen Gleichungen modellieren. Hier bedarf es der Kopplung dieser beiden Gleichungsarten. Mit DAEs können beispielsweise ein einfaches mathematisches Pendel, ein elektronischer Schaltkreis oder ein mechanisches Mehrkörpersystem modelliert werden (Sch03). Da DAEs eine Kopplung von differentialen und algebraischen Gleichungen sind, können wir sie mit gewöhnlichen numerischen Methoden lösen. In ihrer praktischen Anwendung sind DAE-Systeme jedoch häufig sehr groß, wodurch ihre numerische Lösung komplex wird. Aus diesem Grund benötigen wir Methoden, mit denen DAE-Systeme vor ihrer numerischen Lösung symbolisch manipuliert werden. Das Tearing Verfahren ist eine der möglichen Techniken, lineare bzw. nichtlineare Gleichungssysteme symbolisch aufzubereiten. Wir können für Tearing das Wort Aufschneiden verwenden. Durch Tearing werden die Gleichungen bzw. Variablen eines Gleichungssystems nicht nur umsortiert, sondern zusätzlich aufgeschnitten, wenn wir eine kleine Menge von Variablen finden, durch die das gesamte System gekoppelt ist. Dadurch wird ein großes Gleichungssystem in mehrere kleinere Systeme zerlegt und die Leistung des numerischen Lösers verbessert.

1.1 Motivation

Es gibt viele Algorithmen bzw. Heuristiken für Tearing, die auf unterschiedlichen Theorien basieren. Ein Tearing Algorithmus kann durch Manipulation von Matrizen realisiert werden, wenn er die Struktur des untersuchten Systems betrachtet. Ein Tearing Algorithmus kann ebenso ein graphentheoretisches Verfahren sein wie beispielsweise der Tarjan Algorithmus (Tar72), welcher dazu dient, Gleichungssysteme gleichzeitig sowohl horizontal als auch vertikal zu sortieren. Tearing kann außerdem dazu verwendet werden, alge-

braisch gekoppelte Gleichungssysteme zu erkennen und zu isolieren. Es gibt keinen klaren Sieger unter den verschiedenen Tearing Algorithmen.

Ziel dieser Arbeit ist es, einen möglichst optimalen Tearing Algorithmus zu finden. Der Algorithmus von Cellier (CK06) ist eine der aktuellen Tearing Techniken und wird in der Modellierungssoftware Dymola (Elm04) eingesetzt. Um Gleichungssysteme effizient zu lösen, können wir zunächst die Größe des Systems reduzieren. Der Algorithmus von Cellier sortiert zuerst die Gleichungen und Variablen und findet die minimalen algebraischen Schleifen in einem System. Für das anschließende Finden von Variablen, die das Gleichungssystem koppeln, wird eine Heuristik verwendet. Diese Variablen sind die Tearing Variablen und werden vom numerischen Löser berechnet. Für das Auflösen der verbliebenen Variablen wird eine Reihenfolge bestimmt, damit wir sie mit Hilfe der bekannten Tearing Variablen durch einfache Substitution ausrechnen können. Das Ziel ist, die Anzahl von Tearing Variablen möglichst klein zu halten. Das Problem Tearing ist NP-vollständig. Folglich gibt es keinen deterministischen Algorithmus.

Die von Dymola verwendete Heuristik liefert jedoch nicht immer die optimale Lösung - die minimale Menge von Tearing Variablen. Aus diesem Grund wird in dieser Arbeit eine eigene Heuristik für den Cellier Algorithmus entwickelt, die eine möglichst optimale Lösung findet. Ein Tearing Algorithmus aus der Graphenthorie liefert ebenso wie der Cellier Algorithmus keine optimale Lösung. Sein Laufzeitverhalten ist jedoch besser. Er wird in dieser Arbeit modifiziert und zusätzlich mit eigenen Heuristiken ausgeführt, um eine bessere Lösung zu erhalten.

1.2 Struktur der Arbeit

Kapitel 2 gibt einen kurzen Überblick über allgemeine Lösungsverfahren für Gleichungssysteme. Anschließend werden in Kapitel 3 effektive Lösungen und wichtige Definitionen betrachtet. Den Schwerpunkt dieser Arbeit bildet das Tearing Verfahren. Nach einer kurzen Einführung werden Themen wie Historie, Grundidee und Komplexität von Tearing in Kapitel 4 dargestellt. Zusätzlich werden in diesem Kapitel bekannte Tearing Algorithmen präsentiert und analysiert. Dies beinhaltet auch die Darstellung von Vor- und Nachteilen. Für die Schwächen des jeweiligen Tearing Algorithmus werden in Kapitel 5 eigens entwickelte alternative Heuristiken bzw. Modifikationen vorgestellt. In Kapitel 6 werden sowohl ein selbst implementiertes Testprogramm zur Analyse verschiedener Tearing Verfahren als auch damit erzielte Ergebnisse präsentiert. Kapitel 7 besteht aus einer Zusammenfassung dieser Arbeit sowie einem Ausblick über zukünftige Forschung.

Kapitel 2

Allgemeine Lösungsverfahren für Gleichungssysteme

In diesem Kapitel werden die grundlegenden Lösungsverfahren für die Gleichungssysteme eingeführt. Die Gaußsche Eliminationsmethode wird für das Lösen eines linearen Gleichungssystems benutzt. Für nichtlineare Systeme ist das Newton Verfahren eine klassische Methode. Sie sind die Ausgangspunkte für das Finden weiterer effektiver Algorithmen.

2.1 Lineare und nichtlineare Gleichungssysteme

- **Lineares Gleichungssystem** besteht nur aus linearen Gleichungen, die mehrere Variablen enthalten. Wir können ein Gleichungssystem mit m Gleichungen und n Variablen immer in untere allgemeine Form bringen:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

  Wir können ein Gleichungssystem auch in Matrixform darstellen. Das ist in vielen Fällen sehr nützlich. Alle Koeffizienten a_{ij} werden in Koeffizientenmatrix A zusammengefasst.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Für die Variablen und die rechten Seiten des Gleichungssystems stellen wir jeweils einen Vektor dar.

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

Dann können wir ein lineares Gleichungssystem durch Matrix-Multiplikation darstellen:

$$A * x = b$$

- **Nichtlineares Gleichungssystem** enthält sowohl lineare Gleichungen als auch nichtlineare Gleichungen. Die nichtlinearen Probleme kommen sehr häufig im Bereich Mathematik und Physik vor.

  Sei eine stetige nichtlineare Funktion $f : R \longmapsto R$. Ihre Nullstellen werden als Lösung der Gleichung $f(x) = 0$ gesucht. Eine nichtlineare Gleichung kann z.B. eine Quadratische Gleichung: $ax^2 + bx + c = 0$ sein, auch die Sinus-Gleichungen z.B. $\sin x = 0$ ist nichtlinear. Nichtlineare Gleichungen besitzen keine Linearität, nämlich die Additivität und die Homogenität.

  1. **Additivität** $f(x + y) = f(x) + f(y)$
  2. **Homogenität** $f(\alpha x) = \alpha f(x)$

2.2 Allgemeine Lösungsverfahren

2.2.1 Gaußsche Eliminationsmethode

Zuerst sehen wir ein allgemeines Lösungsverfahren für lineare Gleichungssysteme. Sei ein Gleichungssystem mit n Gleichungen und n Variablen $Ax = b$ gegeben:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{m2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Im ersten Schritt sollen wir das Gleichungssystem in gestaffelte Form bringen. Das bedeutet, dass die Variablenanzahl in einer Gleichung wird von oben nach unten immer um eins verringert wird. Die Variablen werden immer wieder eliminiert, indem wir die Koeffizienten davon zu null bringen. Die erste Gleichung wird bei der Elimination nicht ändern. Die Variable x_1 in der zweiten Gleichung soll verschwinden. Und die Variablen x_1 und x_2 in Gleichung drei werden auch eliminiert, usw. Die letzte Gleichung enthält dann nur die Variable n. Die Elimination von Variablen kann durch Tauschen zwei Gleichungen oder durch Addition eine Zeile mit Vielfache einer anderen Gleichungen realisiert werden. Dieses Verfahren heißt die **Vorwärtssubstitution**. Als Beispiel sehen wir folgendes Gleichungssystem mit 3 Gleichungen und 3 Variablen.

$$\begin{aligned} 2x_1 + x_2 + x_3 &= 1 \\ -4x_1 - x_2 - x_3 &= 0 \\ 4x_1 + 3x_2 + 2x_3 &= 1 \end{aligned}$$

Die erste Gleichung wird nicht geändert. Sie wird mit 2 multipliziert und auf die zweite Gleichung addiert, dann von der dritten Gleichung subtrahiert. So verschwindet die Variablen x_1 in den zweiten und dritten Gleichungen.

$$\begin{aligned} 2x_1 + x_2 + x_3 &= 1 \\ x_2 + x_3 &= 2 \\ x_2 &= -1 \end{aligned}$$

Nun sollen wir die Variable x_2 von der dritten Gleichung eliminieren, indem wir die zweite Gleichung von der dritten subtrahieren. Dann bekommen wir die gestaffelte Form für das Gleichungssystem. Die **Vorwärtssubstitution** ist bisher fertig.

$$\begin{aligned} 2x_1 + x_2 + x_3 &= 1 \\ x_2 + x_3 &= 2 \\ -x_3 &= -3 \end{aligned}$$

Nach Vorwärtssubstitution enthält die letzte Gleichung nur die einzige Variable, und die vorletzte zwei, usw.. Wir können zuerst die letzte Gleichung auflösen, und dann die andere Gleichungen von unten nach oben durch Substitution auch auflösen. Dieses Verfahren heißt **Rückwärtssubstitution**. Im obigen Beispiel können wir durch die letzte Gleichung $-x_3 = -3$ die Variable $x_3 = 3$ bekommen. Dann setzen wir $x_3 = 3$ in der zweiten Gleichung und ergibt sich $x_2 = -1$. Und durch die Substitution von Variablen x_2 und x_3 in der ersten Gleichungen bekommen wir $x_1 = -1/2$. So ist das

Gleichungssystem aufgelöst.

$$x = \begin{pmatrix} x_1 = -1/2 \\ x_2 = -1 \\ x_3 = 3 \end{pmatrix}$$

Wir können die Gaußsche Eliminationsmethode auch durch LR-Zerlegung formal beschreiben. Eine reguläre Matrix A kann in das Produkt einer linken unteren Dreiecksmatrix L und einer rechten oberen Dreiecksmatrix R zerlegt werden. Die Umformungsinformation ist in diesen zwei Matrix L und R gespeichert. Die Diagonale von L soll zu 1 gebracht werden, damit die Umformung eindeutig zu sein. Deswegen brauchen wir für den allgemeinen Fall nach eine Permutationsmatrix P: $PA = LR$. Es gibt ein paar Algorithmus für die LR-Zerlegung. Für das Beispiel oben bekommen wir so eine Zerlegung.

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} * \begin{pmatrix} 2 & 1 & 1 \\ -4 & -1 & -1 \\ 4 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1/2 & 1/4 & 1 \end{pmatrix} * \begin{pmatrix} -4 & -1 & -1 \\ 0 & 2 & 1 \\ 0 & 0 & 1/4 \end{pmatrix}$$

Für die **Vorwärtssubstitution** haben wir nun: $Lz = b$. Wir können die z_i von oben nach unten ausrechnen:

$$z_i = \frac{1}{l_{ii}} \left(b_i - \sum_{k=1}^{i-1} l_{ik} \cdot z_k \right) \quad .$$

Die **Rückwärtssubstitution** ist durch $Rx = z$ realisiert. Die Variable x_i wird von unten nach oben berechnet:

$$x_i = \frac{1}{r_{ii}} \left(z_i - \sum_{k=i+1}^{n} r_{ik} \cdot x_k \right).$$

Die Gaußsche Elimination braucht n(n+1) Speicherplätze. Die Laufzeit für die LR-Zerlegung lautet $n^3/3$, für Vorwärtssubstitution

2.2.2 Newton Verfahren

Nun betrachten wir das nichtlineare Problem. Das Newton Verfahren ist ein häufig benutztes numerisches Lösungsverfahren für eine nichtlineare Gleichung oder ein nichtlineares Gleichungssystem.

2.2.2.1 Newton Verfahren für Gleichung

Wir können das Newton Verfahren für ein nichtlineare Gleichung $f(x) = 0$ benutzen, um den Näherungswert der Nullstellen dieser Gleichung zu finden. Die Abbildung 2.1 zeigt, wie Newton Verfahren funktioniert. Am Anfang

wird ein Ausgangspunkt x_0 ausgewählt. Dann wird die Tangente aus diesem Punkt bestimmt. Die Nullstelle von der Tangente ist die verbesserte Lösung für die Näherung der Nullstellen x^* von der originalen Gleichung. Diese Nullstelle wird dann als Ausgangspunkt für weitere Iterationen, bis eine bestimmte Grenze erreicht wird.

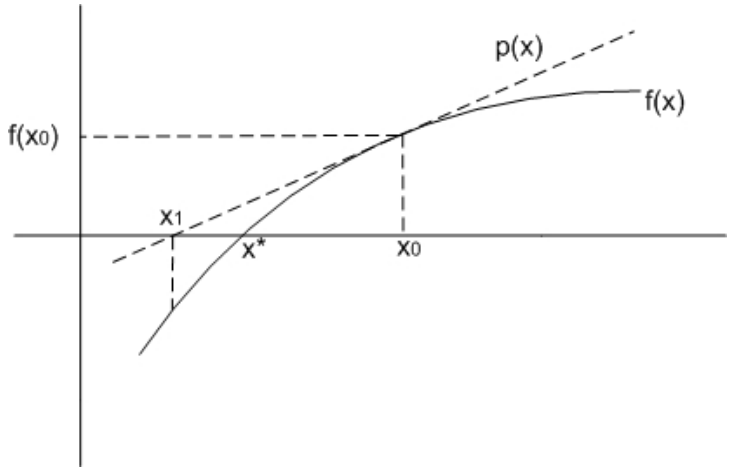

Abbildung 2.1: Newton Verfahren

Wenn die Gleichung $f(x) = 0$ differenzierbar ist, können wir die Tangente von der Gleichung mit dem Startpunkt x_0 als

$$p(x) = f(x_0) + f'(x_0)(x - x_0)$$

darstellen. Nach einem Iterationsschritt können wir die Nullstelle x_1 bestimmen ($f'(x_0) \neq 0$)

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad .$$

Die weiteren Nullstellen können wir durch die Form ($f'(x_k) \neq 0$)

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, k = 0, 1, 2, ...$$

finden.

Wir nehmen die Gleichung $f(x) = x^3 - 10$ als Beispiel. Daraus folgt $f'(x) = 3x^2$. Als Startpunkt wählen wir $x_0 = 2$. Die Nullstellen der Tangente

$$x_{k+1} = x_k - \frac{x_k^3 - 10}{3x_k^2}.$$

Wir starten das Newton Verfahren für drei Iterationen und bekommen die folgende Tabelle. Der Funktionswert $f(x_k)$ nähert sich immer Null.

k	x_k	$f(x_k)$
0	2	-2
1	2,1666667	0,1712962
2	2,1545036	9,59809e-4
3	2,1544346	3,07045e-8

Tabelle 2.1: Newton Verfahren

2.2.2.2 Newton Verfahren für Gleichungssystem

Ein nichtlineares Gleichungssystem ist eine Funktion $f : R^n \longmapsto R^n$, wenn die Anzahl der Gleichungen gleich die der Variablen. Wir können das Newton-Verfahren auch darauf anwenden. Dabei brauchen wir die sogenannte Jacobi-Matrix. Die Jacobi-Matrix ist die partielle Ableitung von $f(x)$:

$$J = \frac{\partial f}{\partial x} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}.$$

Danach folgt:

$$J(x_n)\Delta x_n = -f(x_n)$$

und

$$\Delta x_n = -(J(x_n))^{-1} f(x_n).$$

Nun können wir das Newton-Verfahren für das Gleichungssystem anwenden:

$$x_{n+1} = x_n + \Delta x_n = x_n - (J(x_n))^{-1} f(x_n).$$

Die nichtlinearen Gleichungen enthalten bei vielen Anwendungen nicht nur eine einzige Lösung. Das normale Newton-Verfahren liefert in diesem Fall nicht immer eine Lösung oder nicht die gewünschte Lösung. Dafür sind viele Methoden durch Modifikation von Newton-Verfahren entwickelt worden (Deu06).

Kapitel 3

Effektive Verfahren zum Lösen von Gleichungssysteme

3.1 Grundlegende Definitionen

Im letzten Kapitel haben wir bereits die bekannten Lösungsverfahren für lineare bzw. nichtlineare Gleichungssysteme eingeführt. Es gibt für dieses Problem auch effiziente Algorithmen. Zuerst werden hier einpaar grundlegende Begriffen (auch Datenstrukturen) dargestellt, die für Gleichungssysteme eine wichtige Rolle spielen.

3.1.1 Strukturinzidenzmatrix

Viele effiziente Algorithmen benötigen nicht nur die originalen Gleichungen sondern auch die strukturelle Information eines Gleichungssystems. Deshalb sollen wir die Informationen aus dem Gleichungssystem herausfinden und in einer speziellen Form speichern. Eine dieser speziellen Formen ist die Strukturinzidenzmatrix. Jede Zeile der Strukturinzidenzmatrix entspricht einer Gleichung aus dem betrachteten Gleichungssystem. Und jede Unbekannte wird durch eine Spalte präsentiert. Die Strukturinzidenzmatrix ist immer quadratisch, weil das Gleichungssystem hier immer gleich viele Gleichungen und Variablen enthält, um vollständig sein zu können. Ein nicht vollständiges Gleichungssystem kann nicht aufgelöst werden und wird hier nicht betrachtet. Das Element *[i,j]* aus der Strukturinzidenzmatrix entspricht der *Gleichung i* und *Variable j*. Wenn es den Wert 1 hat, bedeutet, dass die *Variable j* in der *Gleichung i* vorkommt, sonst wird der Matrixeintrag auf 0 gesetzt.

Sei ein Gleichungssystem gegeben:

$$\begin{aligned} f_1(V_1) &= 0 \\ f_2(V_3, V_4) &= 0 \\ f_3(V_5, V_6) &= 0 \\ f_4(V_7, V_8) &= 0 \\ f_5(V_9, V_{10}) &= 0 \\ f_6(V_1, V_3) &= 0 \\ f_7(V_3, V_5, V_7) &= 0 \\ f_8(V_5) &= 0 \\ f_9(V_2, V_4) &= 0 \\ f_{10}(V_4, V_6, V_{10}) &= 0 \end{aligned}$$

Die Abbildung 3.1 zeigt die Strukturinzidenzmatrix für das obere Beispiel.

Var. / GL.	1	2	3	4	5	6	7	8	9	10
1	1	0	0	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0
4	0	0	0	0	0	0	1	1	0	0
5	0	0	0	0	0	0	0	0	1	1
6	1	0	1	0	0	0	0	0	0	0
7	0	0	1	0	1	0	1	0	0	0
8	0	0	0	0	1	0	0	0	0	0
9	0	1	0	1	0	0	0	0	0	0
10	0	0	0	1	0	1	0	0	0	1

Abbildung 3.1: Strukturinzidenzmatrix

Durch horizontale und vertikale Sortierung können wir diese Matrix in die *lower-triangle* Form 3.2 bringen. Diese Sortierung entspricht der Permutation einer Matrix. Es gibt mehrere Algorithmen (BF03) (Ste65) für das Finden solcher Permutation. Die Gleichungen können nach der Reihenfolge der Zeilen in der Matrix nacheinander kausalisiert werden. Der Einträge auf Diagonal entspricht genau den Zuordnungen von Variablen zu Gleichungen im Gleichungssystem.

Var. / GL.	1	5	6	3	4	7	10	2	8	9
1	(1)	0	0	0	0	0	0	0	0	0
8	0	(1)	0	0	0	0	0	0	0	0
3	0	1	(1)	0	0	0	0	0	0	0
6	1	0	0	(1)	0	0	0	0	0	0
2	0	0	0	1	(1)	0	0	0	0	0
7	0	1	0	1	0	(1)	0	0	0	0
10	0	0	1	0	1	0	(1)	0	0	0
9	0	0	0	0	1	0	0	(1)	0	0
4	0	0	0	0	0	1	0	0	(1)	0
5	0	0	0	0	0	0	1	0	0	(1)

Abbildung 3.2: Matrix in lower-triangle Form

3.1.2 Strukturdigraph

Der Strukturdigraph stellt ebenfalls die struktuellen Informationen von Gleichungssystem dar. Er enthält im Prinzip die gleichen Informationen wie die Strukturinzidenzmatrix in einer anderer Darstellung. In vielen auf Graphentheorie basierenden Algorithmen wird diese Datenstruktur verwendet. Auf der linken Seite des Strukturdigraphen stehen die Gleichungen, auf rechten die Variablen. Und eine Linien zwischen einer Gleichung und einer Variablen bedeutet, dass diese Variable in der Gleichung vorkommt. Der Strukturdigraph wird beim *Tarjan Algorithmus* (Tar72) benutzt. Dieser Algorithmus wird im nächsten Kapitel genau beschrieben. Wir nehmen nochmal das Beispiel aus dem letzten Unterabschnitt. Die Abbildung 3.3 auf der nächsten Seite zeigt den entsprechenden Strukturdigraph.

3.1.3 Algebraische Schleife

Um ein DAE-System zu lösen, sollen wir für jede Variable bestimmen, durch welche Gleichung sie aufgelöst werden könnte. Das bedeutet, dass jede Variable genau zu einer Gleichung zugeordnet werden soll. Für die Variablenzuordnung gibt es mehrere Algorithmen, die in den nächsten zwei Kapiteln näher betrachtet werden. Wenn wir die Variablenzuordnung eines Gleichungssystems bestimmt haben, können wir das Strukturdiagramm davon erhalten. Es sei folgendes Gleichungssystem gegeben.

$$\begin{aligned} f_1(V_3, V_4) &= 0 \\ f_2(V_2) &= 0 \\ f_3(V_2, V_3, V_5) &= 0 \\ f_4(V_1, V_2) &= 0 \\ f_5(V_1, V_3, V_5) &= 0 \end{aligned}$$

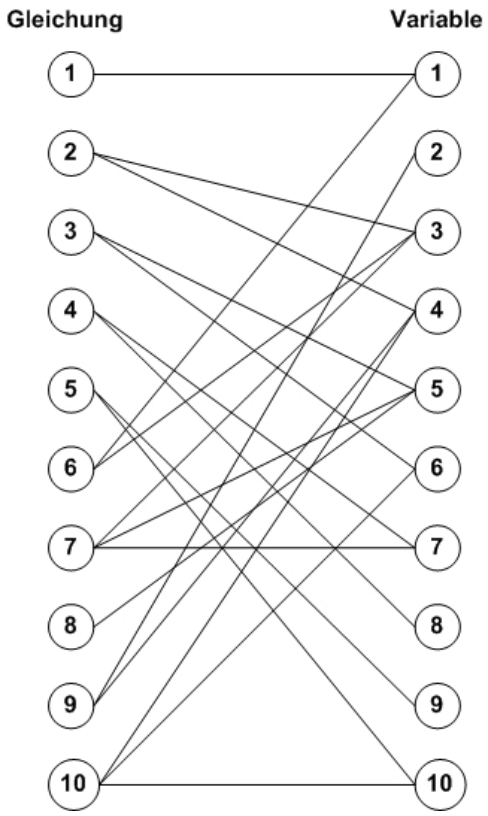

Abbildung 3.3: Strukturdigraph

Wir können nach bekannten Zuordnungsalgorithmen folgende Variablenzuordnung erlangen. Der genaue Ablauf der Zuordnung für dieses Beispiel sehen wir im Abschnitt 4.5.1.1.

VARIABLE	GLEICHUNG
V_1	f_4
V_2	f_2
V_3	f_5
V_4	f_1
V_5	f_3

Tabelle 3.1: Variablenzuordnung

Daraus können wir das Strukturdiagramm erstellen. Die Knoten im Diagramm entspricht den Gleichungen. Ein Kanten von f_i nach f_j mit bedeutet, dass die Gleichung f_j vor f_i aufgelöst werden soll, damit f_i das Ergebnis von f_j benutzen kann. Nun können wir das Diagramm 3.4 für das obere Beispiel herstellen.

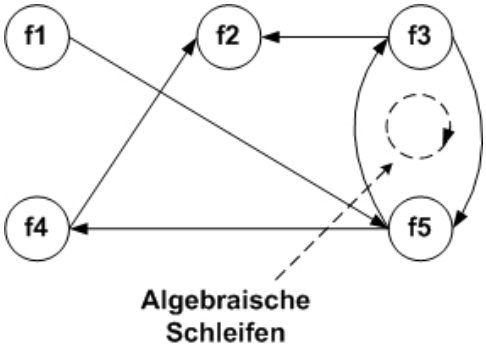

Abbildung 3.4: Algebraische Schleifen

Im oberen Diagramm können wir eine Schleife entdecken, nämlich die Gleichung 3,5,3. Die Gleichungen innerhalb einer Schleife können nicht nacheinander sondern müssen simultan gelöst werden. Wenn das Strukturdiagramm eines Gleichungssystems Schleifen enthält, können wir durch normale Sortierung keine Reihenfolge für das Lösen von Gleichungen bestimmen. Es ergibt sich folglich auch keine *lower-traingle* Form für die Strukturinzidenzmatrix. Aber wir können die Gleichungen innerhalb einer Schleife als einen zusammenhängenden Block in der Matrix darstellen. Auf dem Diagonal steht nun nicht nur die Variablen sondern auch die Blöcken, die nacheinander gelöst werden sollen. Die Abbildung 3.5 zeigt, wie eine Matrix in *block-lower-triangle(BLT)* 3.5 Form aussieht. Es gibt entsprechende Algorithmen für die Erzeugung der BLT-Matrix für eine normale Strukturinzidenzmatrix. Sie werden im Kapitel 4 genauer beschrieben.

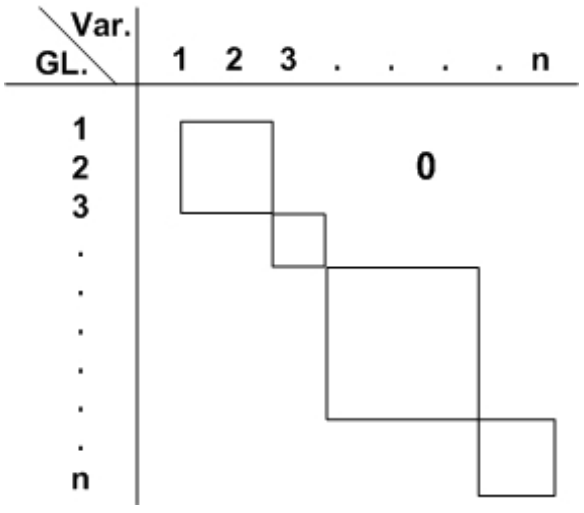

Abbildung 3.5: Block lower triangle Form

Die Gleichungen in einem Block sind miteinander zusammenhängend. Wir können das gesamte Gleichungssystem auflösen, indem wir die einzelnen Blöcken nacheinander auflösen. Das komplexe Problem ist in viele kleine Probleme zerlegt. Die Frage besteht jetzt darin, wie man die Gleichungen in einem Block lösen kann. Wir können hier natürlich die allgemeinen Algorithmen benutzen, z.B. Algorithmus von Gauß und Newton. Es gibt auch Algorithmen für diese speziellen Blöcke, mit denen die Gleichungen möglichst effizient gelöst werden können.

3.2 Lösungsverfahren

In diesem Abschnitt werden wir einige Algorithmen für das Auflösen eines Blocks nach der BLT-Zerlgung einführen. Die allgemeinen Algorithmen sind im letzten Kapitel bereits beschrieben. Hier werden nur einpaar effektive Verfahren, insbesondere die zwei symbolischen Verfahren Relaxation und Tearing, vorgestellt.

3.2.1 Relaxation

Relaxation (MEC95) ist eine symbolische Implementierung von Gaußsche Eliminationsmethode und kann nur auf lineare Gleichungssysteme angewendet werden. Wir präsentieren das Verfahren an folgendem physikalischen System mit vier Gleichungen und vier Variablen.

$$\begin{aligned} u - u_1 - u_2 &= 0 \\ u_1 - L_1 di_1 &= 0 \\ u_2 - L_2 \frac{di_2}{dt} &= 0 \\ di_1 - \frac{di_2}{dt} &= 0 \end{aligned}$$

Wir können es in Matrixform wie unten umwandeln. Die erste Gleichung enthält einen Konstante u. Wir bringen u nach rechter Seite der Gleichung und multiplizieren sie mit -1. Nach diesen Umformungsschritten ist die Gleichung 1:

$$u_1 + u_2 = u$$

Nun wird die Matrixform des Gleichungssystems erzeugt:

$$\begin{pmatrix} 1 & 0 & 0 & \boldsymbol{1} \\ 1 & -L_1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & -L_2 & 1 \end{pmatrix} * \begin{pmatrix} u_1 \\ di_1 \\ \frac{di_2}{dt} \\ u_2 \end{pmatrix} = \begin{pmatrix} u \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Unser Ziel ist, die Anzahl von Matrixeinträgen oberhalb der Diagonale, die nicht gleich Null sind, zu minimieren. In diesem Beispiel haben wir nur ein

solches Element. Das ist die Eins rechts oben in der Matrix. Die Einträge aus der Matrix sind alle ungleich Null. Nun können wir die Gaußsche Eliminationsmethode darauf anwenden. Zur Erinnerung:

$$A_{ij}^{(n+1)} = A_{ij}^{(n)} - A_{ik}^{(n)} {A_{kk}^{(n)}}^{-1} A_{kj}^{(n)}$$

$$b_{i}^{(n+1)} = b_{i}^{(n)} - A_{ik}^{(n)} {A_{kk}^{(n)}}^{-1} b_{k}^{(n)}$$

Nach Relaxationsalgorithmus eliminieren wir in jedem Schritt die erste Zeile und die erste Spalte. Der Index k in obiger Gleichung ist immer gleich 1. Der Eintrag auf der Position [i-1, j-1] nach n+1 Iterationen kann man mit folgenden Formeln beschreiben:

$$A_{ij}^{(n+1)} = A_{ij}^{(n)} - A_{i1}^{(n)} {A_{11}^{(n)}}^{-1} A_{1j}^{(n)}$$

$$b_{i}^{(n+1)} = b_{i}^{(n)} - A_{i1}^{(n)} {A_{11}^{(n)}}^{-1} b_{1}^{(n)}$$

Nach der n-ten Iteration gilt für der Eintrag [i, j] in Matrix A:

- Wenn der Eintrag auf der ersten Position in der i-ten Zeile in der Matrix A Null ist, ändert er nach der (n+1)-te Iteration nicht.
- Wenn der Eintrag auf der ersten Position in der j-ten Spalte in der Matrix A Null ist, ändert er nach der (n+1)-te Iteration nicht.

Nach der n-ten Iteration gilt für der Eintrag [i, j] in Matrix b:

- Wenn der Eintrag auf der ersten Position in der i-ten Zeile in der Matrix A Null ist, ändert er nach der (n+1)-te Iteration nicht.
- Wenn der Eintrag auf der ersten Position in der Matrix b Null ist, ändert er nach der (n+1)-te Iteration nicht.

Nun betrachten wir das obige Beispiel. Auf der zweiten Position in der ersten Spalte und auf der letzten Position der ersten Zeile der Matrix A sind die Einträge ungleich Null. Das erste Element in der Matrix b ist auch nicht Null. Deshalb soll neu erzeugte Matrix neue Einträge *c1* auf Position [2,4] in Matrix A und *c2* auf der zweiten Position in der Matrix b besitzen.

$$c1 = 0 - 1 * 1^{-1} * 1 = -1$$

$$c2 = 0 - 1 * 1^{-1} * (u) = -u$$

$$\begin{pmatrix} -L_1 & 0 & c_1 \\ 1 & -1 & 0 \\ 0 & -L_2 & 1 \end{pmatrix} * \begin{pmatrix} di_1 \\ \frac{di_2}{dt} \\ u_2 \end{pmatrix} = \begin{pmatrix} c_2 \\ 0 \\ 0 \end{pmatrix}$$

In der zweiten Iteration soll der Eintrag [2,3] in der Matrix A bzw. der Eintrag [2] in der Matrix b durch $c_3 = \frac{c_1}{L_1}$ bzw. $c_4 = \frac{c_2}{L_1}$ ersetzt werden.

$$\begin{pmatrix} -1 & c_3 \\ -L_2 & 1 \end{pmatrix} * \begin{pmatrix} \frac{di_2}{dt} \\ u_2 \end{pmatrix} = \begin{pmatrix} c_4 \\ 0 \end{pmatrix}$$

In der letzten Iteration ersetzen wir den Eintrag [2,2] der Matrix A durch $c_5 = 1 - L_2 c_3$ und den Eintrag [2] durch $c_6 = -L_2 c_4$. Dann bekommen wir die Gleichung:

$$(c_5) * (u_2) = (c_6)$$

Durch Substitution bekommen wir:

$$\begin{aligned} u_2 &= \frac{c_6}{c_5} \\ \frac{di_2}{dt} &= \frac{c_4 - c_3 u_2}{-1} \\ di_1 &= \frac{c_2 - c_1 u_2}{-L_1} \\ u_1 &= u - u_2 \end{aligned}$$

Wir haben alle nützlichen Informationen gesammelt und können einfach diese Gleichungen von oben nach unten lösen. Das originale System wird auch durch das folgende System ersetzt.

$$\begin{aligned} c_1 &= -1 \\ c_2 &= -u \\ c_3 &= \frac{c_1}{L_1} \\ c_4 &= \frac{c_2}{L_1} \\ c_5 &= 1 - L_2 c_3 \\ c_6 &= -L_2 c_4 \\ u_2 &= \frac{c_6}{c_5} \\ \frac{di_2}{dt} &= \frac{c_4 - c_3 u_2}{-1} \\ di_1 &= \frac{c_2 - c_1 u_2}{-L_1} \\ u_1 &= u - u_2 \end{aligned}$$

Der Relaxationsalgorithmus liefert bei manchen Anwendungen sehr elegante Lösung. Aber er kann nur auf lineares Problem angewendet werden. In vielen Fällen haben wir immer nichtlineare Systeme zu lösen. Deshalb brauchen wir einen anderen Algorithmus dafür. Der Tearing Algorithmus ist eine gute Auswahl dafür.

3.2.2 Tearing

Tearing ist eine bekannte Methode für das Auflösen von differential-algebraischen Gleichungen(DAE) und original von Kron eingeführt (Kro62). Sie ist in vielen physischen Modellen anwendbar und besonders nützlich in objektorientierter Systemen (EO94). Es gibt verschiedene Konzepte von Tearing. Die Grundidee ist, das System in viele kleinere Systeme zu zerlegen und die algebraische Schleifen im Systems durch Aufschneiden von Variablen zu brechen. Die gefundenen Tearing Variablen werden mit numerischen Verfahren gelöst und danach ins originale System zurückgesetzt. Weil die algebraische Schleife im System nicht mehr existiert und die einzelnen Blöcke nicht mehr mit einander zusammenhängend sind, können wir nach einer bestimmten Reihenfolge mit normalen Verfahren die restliche Gleichungen bzw. Blöcke auflösen 3.6.

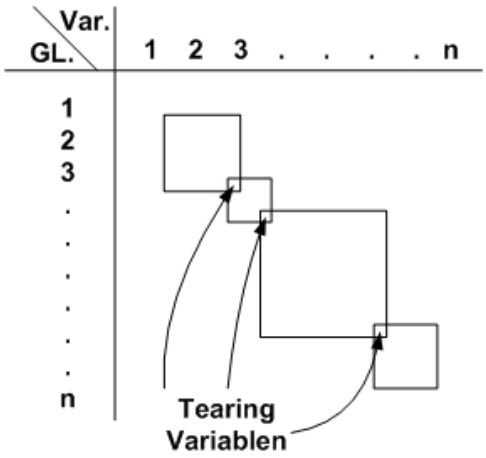

Abbildung 3.6: Tearing

Wir haben in den letzten Abschnitten zwei effizienten Algorithmen für das Lösen von algebraischen Schleifen gesehen. Der Relaxationsalgorithmus ist einfach zu realisieren. In manchen Anwendungen ist er auch sehr anspruchsvoll. Aber er kann nur auf lineare Systeme angewendet werden. Der Tearing Algorithmus ist mächtiger und kann auch auf nichtlineare Systeme anwenden. Aber er ist viel komplexer, z.B. das Auswählen von Tearing Variablen, numerische Lösung für Tearing Variable... In dieser Arbeit konzentrieren wir uns auf das Tearing Verfahren. Im nächsten Kapitel werden wir den dieses Verfahren näher anschauen.

Kapitel 4

Tearing Verfahren

4.1 Historie von Tearing

Kron hat im Jahr 1962 die Technik Tearing (Kro62) (Har63) entwickelt, um ein großes lineares System aus algebraischen Gleichungen in kleine Systeme aufzuschneiden. Die Lösungen von den kleinen Systemen bilden dann die Lösung für das originale Problem. Kron hat seinen Algorithmus auf elektronischem Netzwerk eingesetzt. Das Problem ist, wie und wo man das System aufschneiden soll. Kron braucht bei seiner Wahl sehr gute Kenntnisse von dem untersuchten System und auch viele physikalischen Verständnisse. Aber für die meisten großen Systeme ist es eine komplexe Arbeit, aus Kenntnissen von Systemen eine vernünftige Entscheidung beim Tearing zu treffen. Deshalb ist der Einsatz von seinem Algorithmus sehr beschränkt. Ein numerisches Tearing hat Rubin 1962 veröffentlicht (Rub62).

Donald v. Steward hat 1965 einen Tearing Algorithmus auf der strukturellen Ebene erfunden (Ste65). In seiner Methode braucht man die Struktur des untersuchten Systems in Matrixform darzustellen. Auch die Vorkommen von Variablen in einzelnen Gleichungen werden dabei analysiert. Er behauptet, dass Tearing auch für nichtlineare Systeme anwendbar ist.

Ollero und Amselem hat 1983 einen einfachen und robusten Algorithmus auf signal-flow Graph entwickelt (OA83). Der Tearing Algorithmus ist auf Graphentheorie basiert, um die passende Tearing Variablen zu finden.

Weil Tearing Problem NP-vollständig ist (CG97), ist es unmöglich, einen deterministischen Algorithmus dafür zu bauen. Approximationsalgorithmus ist ein guter Einsatz. Cellier hat in seinem Buch (CK06) einen solchen Tearing Algorithmus mit Heuristik auf Basis von Tarjan Algorithmus (Tar72) präsentiert. Der Algorithmus ist auch in Dymola (Elm04) für große Modelle eingesetzt.

4.2 Gleichungssysteme mit Tearing lösen

In diesem Abschnitt werden wir die Grundidee von Tearing durch ein kleines Beispiel verdeutlichen. Sei ein System mit folgenden zwei Gleichungen:

$$x_1 = f_1(x_2)$$

$$x_2 = f_2(x_1)$$

Wir können die Abhängigkeit zwischen den beiden Gleichungen durch den folgenden Graph 4.1 darstellen:

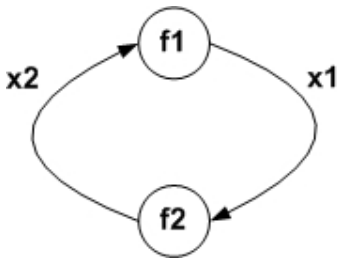

Abbildung 4.1: Grundidee von Tearing (a)

Wenn wir die erste Gleichung auflösen wollen, brauchen wir das Ergebnis von der zweiten Gleichung. Analog, für das Auflösen von der zweiten Gleichung ist das der ersten Gleichung notwendig. Offensichtlich gibt es hier eine algebraische Schleife. Wir haben nun zwei Möglichkeiten für das Aufschneiden, um diese Schleife zu brechen. Als Beispiel wählen wir die Kante x_2 aus. Das bedeutet, die Variable x_2 nehmen wir als Tearing Variable. Wir benutzen dabei ein numerisches Verfahren (z.B. Newton Verfahren) für die Approximation der Tearing Variable:

```
NEWx2 = INITx2
REPEAT
   x2 = NEWx2
   x1 = f1(x2)
   NEWx2 = f2(x1)
UNTIL converged(NEWx2 - x2)
```

Für die Iteration brauchen wir einen Startpunkt *INITx2*. Das Problem ist dann auf das Finden von Nullstellen der nichtlinearen Gleichung $x_2 - f_2(f_1(x_2)) = 0$ reduziert 4.2. Wenn wir x_2 gelöst haben, ist die Variable x_1 durch Substitution auch einfach zu lösen.

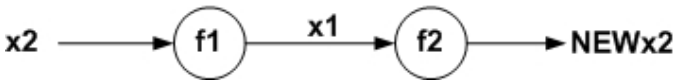

Abbildung 4.2: Grundidee von Tearing (b)

4.3 Klassifikation und Bewertung von Tearing

Es gibt mehrere Tearing Algorithmen. Sie können in drei Klassen verteilt werden (Mah90):

1. *Technik basiert auf Graphentheorie*: Tearing Verfahren in diesem Bereich benutzen viele Definitionen bzw. Algorithmen aus der Graphentheorie, z.B. Tiefensuche, Suchen nach Schleifen usw. (FFS06)

2. *Technik basiert auf Matrix*: Tearing Verfahren hier werden durch Matrixmanipulation realisiert.

3. *Implizites Enumeration*: In Tearing Verfahren aus dieser Kategorie werden meistens dynamische Programmierung und Branch-and-Bound Methode eingesetzt, um eine minimale Menge von Tearing Variablen zu finden.

Da ist eine Tabelle für die wichtigen Tearing Algorithmen. In der Tabelle

Algorithmus	Jahr	Klasse
Sargent and Westerberg	1964	1,3
Steward	1965	2
Lee and Rudd	1969	1
Christensen	1970	1
Upadhye and Grens	1972	2,3
Barkley and Motard	1972	1
Pho and Lapidus	1973	1,3
Cheung and Kuh	1974	2
Johns and Muller	1976	3
Motard and Westerberg	1981	3
Gundersen	1982	1
Ollero and Alselem	1983	1
Cellier	1990-	1,2

Tabelle 4.1: Wichitige Tearing Algorithmen

können wir sehen, dass es viele Varianten von Tearing gibt. Deshalb brauchen wir einige Kriterien, um einen Tearing Algorithmus zu bewerten.

- **Anzahl Tearing Variablen**
  Das erste Kriterium ist die Anzahl der von einem Algorithmus aus-

gewählten Tearing Variablen. Die ausgewählte Tearing Variablen werden dann durch numerische Verfahren gelöst. Das Verhalten des betrachten System hat die Komplexität diesem Schritt stark beeinflusst. Deshalb ist ein Algorithmus effizienter als ein anderer, wenn er wenigere Tearing Variablen braucht, um das ganze System aufzulösen. Die Anzahl der Tearing Variablen bestimmt auch die Größe der Blöcken im System. In einigen speziellen Fällen ist es auch nicht schlecht, wenn durch ein „schlechte“ Tearing viele kleinere Blöcke bekommen, damit ist die Berechnung des einzelnen Blocks viel vereinfacht.

- **Laufzeitverhalten**
  Wenn man einen Algorithmus bewertet, sollte man natürlich die Laufzeit davon betrachten. Erstens geht es hier um die Laufzeit für das Finden von Tearing Variablen und weitere Berechnungen nach dem Algorithmus. Zweitens sollte man auch die Laufzeit der Implementierung für verschiedene Algorithmen vergleichen.

- **Robustheit und Korrektheit**
  Ein guter Tearing Algorithmus sollte robust sein. Er sollte für jedes System immer eine Lösung finden, auch wenn das System sehr groß ist. Die gefundene Lösung sollte auch korrekt sein.

4.4 Allgemeines Laufzeitverhalten von Tearing

Tearing ist ein effizientes Lösungsverfahren für lineare und nichtlineare Gleichungssysteme. Es gibt verschiedene Varianten von Tearing. In diesem Abschnitt werden wir sehen, welche Laufzeit das Tearing Problem allgemein hat. Wenn das Tearing Problem nicht durch einen deterministischen Algorithmus realisiert kann, ist das Problem NP-vollständig. Hier benutzen wir die Technik *Restriktion* für das Beweisen der NP-Vollständigkeit des Tearing Problems (CG97). *Restriktion* für ein Problem P verlangt das Beweisen, dass das Problem P in einem speziellen Zustand ein bekanntes NP-vollständiges Problem P' enthalten kann.

Sei ein nicht singuläres Gleichungssystem mit n Gleichungen und n Variablen. Wir können für das gegebene System die Strukturinzidenzmatrix M bauen. Das Tearing Problem verwandelt sich nun in das Problem, wie man die Matrix M zur lower-traiangle Matrix zu umformatieren. Weil das System strukturell konsistent ist, ist die Strukturinzidenzmatrix damit auch nicht singulär. Deshalb können wir durch Permutation die Matrix M neu sortieren, damit alle Einträge auf der Diagonale von M ungleich 0 sind. Für $M = m_{ij}$ können wir einen Verbingsgraph G(J) definieren. G(J) hat n Knoten und eine Kante von Knoten i nach Knoten j existiert nur, wenn der Matrixeintrag $m_i j \neq 0$ und $i \neq j$ ist. Wenn wir alle Schleifen in diesem

Graph brechen, ist das Tearing fertig. Dafür gibt eine Definition: *essential-set*. Wenn man alle Knoten in *essential-set* aus dem Graph löschen, wird der Graph zyklenfrei. Ein *essential-set* mit minimaler Anzahl von Knoten heißt *minimum essential-set*. Das Tearing Problem kann durch Bestimmen von *minimum essential-set* im Verbindungsgraph gelöst werden. Das Problem ist NP-vollständig und von Karp (Kar72) bewiesen. Damit ist das Tearing Problem auch NP-vollstänig.

4.5 Bekannte Tearing Algorithmen

Wir haben die grundlegenden Kenntnisse von Tearing in letzten Abschnitten bereits vorgestellt. Nun führen wir einpaar bekannte Tearing Algorithmen ein. Als Beispiele werden nur die nicht singulären Probleme betrachtet (OB99b). Wenn ein System singulär ist, sollte man es zuerst mit entsprechendem Algorithmus (Pan88) in ein strukturell konsistentes System überführen.

4.5.1 Vorbereitung für Tearing

Wenn wir Tearing effizient durchführen wollen, sollen wir vor Tearing einige Vorbereitungen treffen (Ott99). Eine wichtige Technik ist die BLT-Zerlegung. Die BLT-Matrix haben wir schon im letzten Kapitel kurz beschrieben. Wenn wir vor Tearing die Strukturinzidenzmatrix eines Systems in BLT-Form bringen, können wir Tearing Algorithmus auf den einzelnen Blöcken anwenden, weil sie voneinander strukturell unabhängig sind. Das kann die Laufzeit von Tearing verbessern. Es gibt viele Algorithmen für BLT-Zerlegung. Sie basieren meistens auf Strukturdiagramm und Variablenzuordnung von einem System. Diese zwei Schritte sollen zuerst durchgeführt werden.

4.5.1.1 Variablenzuordnung

Als erstens wird es für jede Gleichung ermittelt, nach welcher Variablen sie aufgelöst werden muss. Die Variablenzuordnung spielt hier eine wichtige Rolle. Sie bestimmt die Abhängigkeit zwischen den einzelnen Gleichungen. Verschiedene Zuordnungen können das Tearing Ergebnis stark beeinflussen. Im Folgenden werden einige entsprechende Mechanismen vorgestellt.

Otter hat in (Ott99) die einfachste Variante für die Variablenzuordnung benutzt. Dieser rekursive Algorithmus (Pan88) arbeitet im Prinzip nach Tiefensuche mit Backtracking. Der Algorithmus ist wie unten als Pseudecode beschrieben.

```
assign(j) = 0, j = 1..n
for (alle Gleichungen i = 1..n)
```

```
    vMark(j) = false, für j = 1..n;
    eMark(j) = false, für j = 1..n;
    if not pathFound(i)
    return "Singulär";

function pathFound(i)
  eMark(i) = true;
  if(assign(j)= 0, für eine Variable j von Gleichung i)
  then success = true;
       assign(j) = i;
  else success = false;
       for(jede Variable j von Gleichung i, vMark(j) = false)
          vMark(j) = true;
          success = pathFound(assign(j));
          if success
          then  assign(j) = i;
return success;
```

Hier haben wir drei globale Variablen benutzt. Das int-Array *assign* speichert das Ergebnis von Variablenzuordnung. *assign(j)=i* bedeutet, dass die Variable j zur Gleichung i zugeordnet wird. Und als Initialisierung haben alle Variablen in *assign* den Wert 0. *eMark* benutzt man für die Markierung der singulären Modellteile, wenn nicht alle Variablen zugeordnet werden können. *vMark* wird für die Markierung der bereits untersuchten Variablen beim Backtracking benutzt. Die Funktion *pathFound(i)* sorgt für die Zuordnung einer Gleichung i. Zuerst wird gesucht, ob eine nicht zugeordnete Variable j in der Gleichung i gibt. Wenn es mindestens eine solche Variable gefunden ist, wird *assign(j)* auf i gesetzt. Sonst wird für jede in i vorkommende Variable j rekursiv untersucht, ob in der Gleichung *assign(j)* eine nicht zugeordnete Variable gibt. Wenn es am Ende für alle Gleichungen *success* zurückgegeben ist, terminiert der Algorithmus. Sonst sind die DAEs singulär.

Als Beispiel haben wir folgendes Gleichungssystem:

$$f_1(z_3, z_4) = 0, f_2(z_2) = 0$$

$$f_3(z_2, z_3, z_5) = 0, f_4(z_1, z_2) = 0$$

$$f_5(z_1, z_3, z_5) = 0$$

Nach dem Algorithmus oben können wir f_1 bis f_4 problemlos zuordnen und das Ergebnis von Abbildung 4.3(a) bekommen. Nun sollen wir eine Variable für f_5 auswählen. Aber alle Variablen, die $f5$ enthält, sind bereits zugeordnet. Die Zuordnung einer Variable in f_5 muss geändert werden, damit alle Gleichungen kausalisiert werden können. Zuerst nehmen wir z_1 raus. Die

Zuordnung wird zu f_5 geändert. Deshalb sollen wir für f_4 eine neue Variable finden, die ist z_2. Aber f_2 haben nur die einzelne Variable z_2 und lässt andere Zuordnung nicht zu. Das Backtracking schlägt fehl 4.3(b). Untersuchen wir der Reihe nach die Variable z_3. Die Gleichung f_1 hat noch eine Variable z_4 ohne Zuordnung. Deshalb werden z_4 zu f_1 und z_3 zu f_5 zugeordnet. Und damit ist das Gleichungssystem kausalisiert 4.3(c). Wir haben am Ende die Variablenzuordnung: $f_1 \leftarrow z_4$, $f_2 \leftarrow z_2$, $f_3 \leftarrow z_5$, $f_4 \leftarrow z_1$ und $f_5 \leftarrow z_3$.

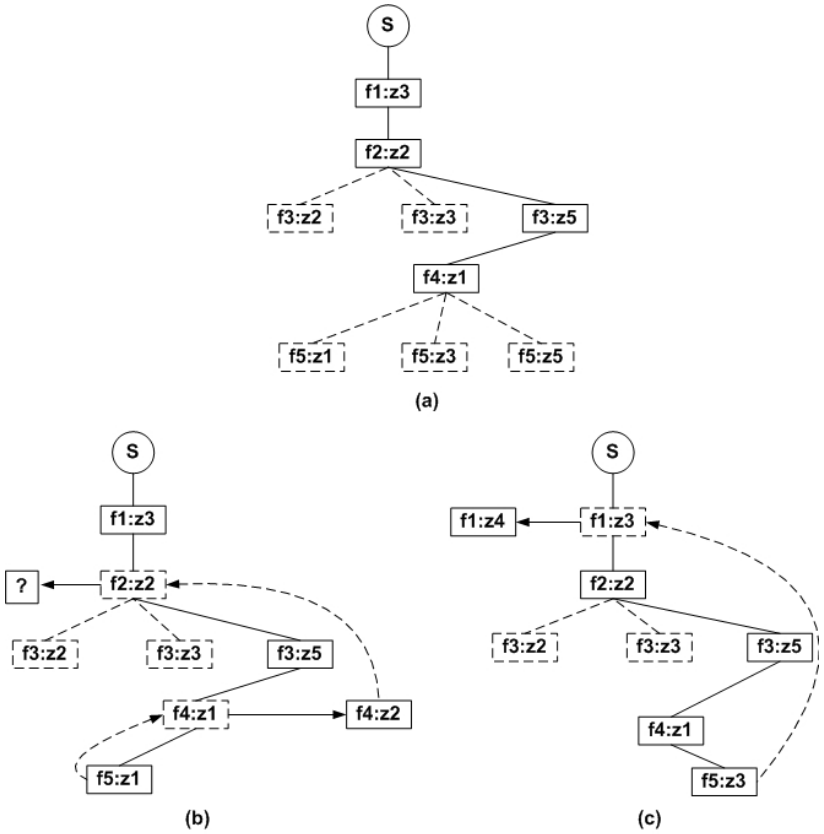

Abbildung 4.3: Tiefensuche mit Backtracking

Der obere Algorithmus führt die Tiefensuche nach der Reihenfolge von Gleichungen durch. Es gibt noch einen Algorithmus für Variablenzuordnung, der auf einen Tiefensuche nach Variablen basiert. Donald V. Steward hat diesen Algorithmus für das Finden von *Output-Set* eines Gleichungssystems eingesetzt (SW62). *Output-Set* hat hier die gleiche Definition von Variablenzuordnung. Als Datenstruktur braucht dieser Algorithmus die Strukturinzidenzmatrix von dem untersuchten Gleichungssystem. Wir nehmen nochmal das obere System als Beispiel. Die Abbildung 4.4(a) stellt die Strukturinzidenzmatrix dafür dar. Der Algorithmus ist wie unten in 3 Schritten beschrieben.

1. Wähle eine nicht zugeordnete Variable i aus. Suche in Spalte i, ob es eine noch nicht benutzte Gleichung j gibt. Wenn ja, wird i zu j zugeordnet und gehe in Schritt 2. Sonst gehe in Schritt 3.
2. Wenn alle Variablen zugeordnet sind, terminiert der Algorithmus. Sonst kehrt zu Schritt 1 zurück.
3. Lösche alle Markierungen von Variablen und Gleichungen. Nehme eine nicht untersuchte Kombination von Variable i und eine i enthaltende Gleichung j, die vorher für Variable m gesetzt ist, markiere i und j. Variable i wird nun zu Gleichung j zugeordnet. Wenn alle Gleichungen für i untersucht sind, ist das Gleichungssystem singulär.
   (a) Suche in Spalte m, ob eine unbenutzte Gleichung n gibt. Wenn ja, wird m zu n zugeordnet und gehe in Schritt 2.
   (b) Für jede m enthaltende Gleichung k: markiere m und k. Kehrt zu 3(a) zurück und untersuche die Variable, die vorher zu k zugeordnet ist.
   (c) Wenn in 3(b) keine solche Gleichung wie k gefunden ist, kehrt zu Schritt 3 zurück.

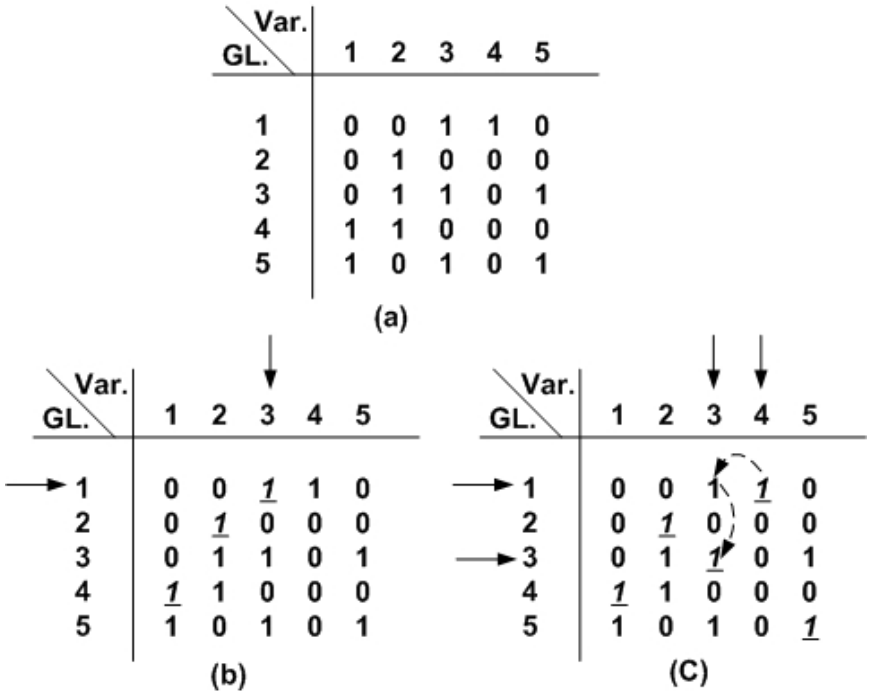

Abbildung 4.4: Variablenzuordnung mithilfe Strukturinzidenzmatrix

Für das Beispiel können wir die Variablen 1, 2 und 3 jeweils zu den Gleichungen 4, 2 und 1 zuordnen(Abbildung 4.4(b)). Dann ergibt sich ein Problem

mit der Variable 4. Sie kommt nur in Gleichung 1 vor. Deshalb setzen wir Gleichung 1 für sie. Und die Zuordnung für die Variable 3 soll dann zu Gleichung 3 geändert werden(Abbildung 4.4(c)). Damit bekommen wir die Variablenzuordnung: $f_1 \leftarrow z_4$, $f_2 \leftarrow z_2$, $f_3 \leftarrow z_3$, $f_4 \leftarrow z_1$ und $f_5 \leftarrow z_5$. Für das gleiche Problem haben wir mit zwei Algorithmen zwei verschiedene Ergebnisse bekommen.

4.5.1.2 Strukturdiagramm

Nachdem alle Variablen zugeordnet sind, können wir einen gerichteten Graph daraus erzeugen. Dieser Graph basiert auf die durch die Variablenzuordnung bestimmte Abhängigkeit der Gleichungen und heißt Strukturdiagramm. Jede Knoten in diesem Graph repräsentiert eine Gleichung f_i mit der zugeordneten Variablen V_j. Eine Kante $f_i : V_j \rightarrow f_m : V_n$ bedeutet, dass die Gleichung f_i für die Berechnung der Variable V_j das Kenntnis von der Variable V_n aus f_m braucht. Als Beispiel ist das folgende Gleichungssystem gegeben.

$$f_a(V_1, V_2, V_3, V_6) = 0, f_b(V_1, V_2) = 0$$
$$f_c(V_2, V_3) = 0, f_d(V_4, V_5) = 0$$
$$f_e(V_2, V_4, V_5) = 0, f_f(V_2, V_4, V_6) = 0$$

Durch den Zuordnungsalgorithmus aus dem letzten Abschnitt können wir folgende Tabelle bekommen: Nun können wir anhand der Zuordnung die

Variable	Gleichung
V_1	f_a
V_2	f_b
V_3	f_c
V_4	f_d
V_5	f_e
V_6	f_f

Tabelle 4.2: Neue Zuordnung

Abhängigkeit der Gleichungen bestimmen und entsprechenden Graph 4.5 erzeugen:

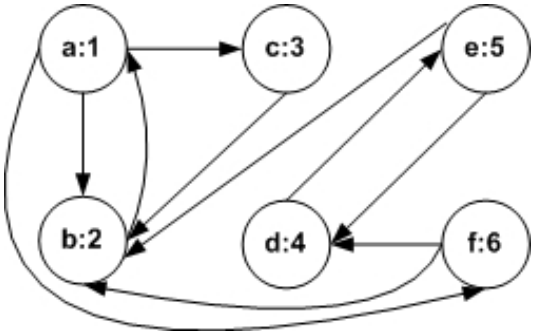

Abbildung 4.5: Gerichteter Graph nach Variablenzuordnung

Offensichtlich gibt es in dem erzeugten Graphen Schleifen. Die Variablen bzw. Gleichungen innerhalb einer Schleife kann nicht rekursiv berechnet werden. Sie sollen zusammen einen Diagonalblock in BLT-Matrix bilden und simultan berechnet werden. Um die Schleife zu brechen, brauchen wir den Tearing Algorithmus. Es gibt nun zwei Möglichkeiten, das System zu lösen. Wir können zuerst die Strukturmatrix in BLT-Form bringen, dann Tearing Algorithmus auf die einzelnen Blöcke anwenden. Und wir können auch BLT-Erzeugung und Tearing in einem Schritt schaffen. In den folgenden Abschnitten werden die zwei Vorgehensweise genau betrachtet.

4.5.1.3 BLT-Erzeugung

Es gibt viele Algorithmen, um eine normale Matrix in die Block-Lower-Triangle(BLT) Form zu bringen. Die Erzeugung kann z.B. auf Matrix-Ebene durch direkten Umtausch von Zeilen und Spalten durchgeführt werden. Wenn wir dabei Graphentheorie benutzt, können wir die einzelnen Blöcke in BLT-Matrix als die zusammenhängenden Komponenten in einem gerichteten Graph betrachten (BC96). Hier wird ein bekannter Algorithmus von Tarjan für die BLT-Zerlegung eingeführt.

Tarjan Algorithmus
Der Tarjan Algorithmus basiert auf Tiefensuche (Tar72) und ist ursprünglich für Herausfinden der zusammenhängenden Komponenten in einem gerichteten Graph eingesetzt. Wenn wir das Strukturdiagramm für ein Gleichungssystem erzeugt haben, können wir den Algorithmus darauf anwenden. Die gefundenen zusammenhängenden Komponenten stimmen mit der Blöcken einer BLT-Matrix überein. Jede zusammenhängende Komponente bildet einen Block ab. Auch die Nummerierung von Blöcken wird durch den Algorithmus bestimmt. Zuerst werden einige wichtige Definitionen aus der Graphentheorie dargestellt.

- **Tiefensuche(Depth first search, Backtracking)**

Die Tiefensuche ist eine uninformierte Suche, welche durch Expansion des jeweils ersten auftretenden Nachfolgeknotens im Graph nach und nach vom Startknoten aus weiter in die Tiefe sucht. Beim Durchsuchen des ganzen Graphen wird für jeden Knoten eine DFS-Nummer gegeben. Diese Nummer entspricht der Reihenfolge des Besuchs durch die Tiefensuche. Tiefensuche erzeugt am Ende aus dem originalen Graph einen DFS-Baum.

- **Stark zusammenhängend(strong connectivity)**
  Ein gerichteter Graph $G = (V, E)$ heißt zusammenhängend von einem Knoten v aus, falls es zu jedem Knoten w aus V einen gerichteten Weg in G gibt, mit v als Startknoten und w als Endknoten. G heißt stark zusammenhängend, falls G von jedem Knoten v aus V zusammenhängend ist.

- **Root von Komponenten**
  Sei C eine stark zusammenhängende Komponente aus dem Graph G. Die Knoten aus C bilden einen Unterbaum vom DFS-Baum von G. Und die Wurzel von diesem Unterbaum heißt Root von dieser stark zusammenhängenden Komponente C.

Der Tarjan Algorithmus benutzt die DFS Nummerierung für die Knoten. Alle Knoten bekommen während der Tiefensuche eine DFS-Nummer. Wenn ein Knoten v durch einen Kanten einen anderen Knoten oder Pfad w, der vor v besucht ist, erreicht, hat v eine kleinere DFS-Nummer als sich selbst erlangt. Wir sollen für alle Knoten die kleinste erreichbare DFS-Nummer festlegen. Diesen Wert definieren wir als *lowerlink*, kurz *low*. Am Anfang hat jeder Knoten immer den gleichen *lowerlink* Wert als die DFS-Nummer. Wenn man bei der Tiefensuche einen Knoten mit kleinerer DFS-Nummer erreicht, wird diesen Wert immer aktualisiert. Wir brauchen noch die Datenstruktur Stapel für das Bestimmen des Unterbaums. Bei der Tiefensuche wird jeder Knoten bei seinem Besuch in den Stapel gesetzt. Wenn wir einen Knoten v fertig mit der Tifensuche haben, bilden alle Knoten, die später in den Stapel gesetzt sind, einen Unterbaum mit dem Root v. Dieser Unterbaum ist genau eine stark zusammenhängende Komponente in diesem Graph.

Der Algorithmus kann als Diagramm in Abbildung 4.6 dargestellt werden.

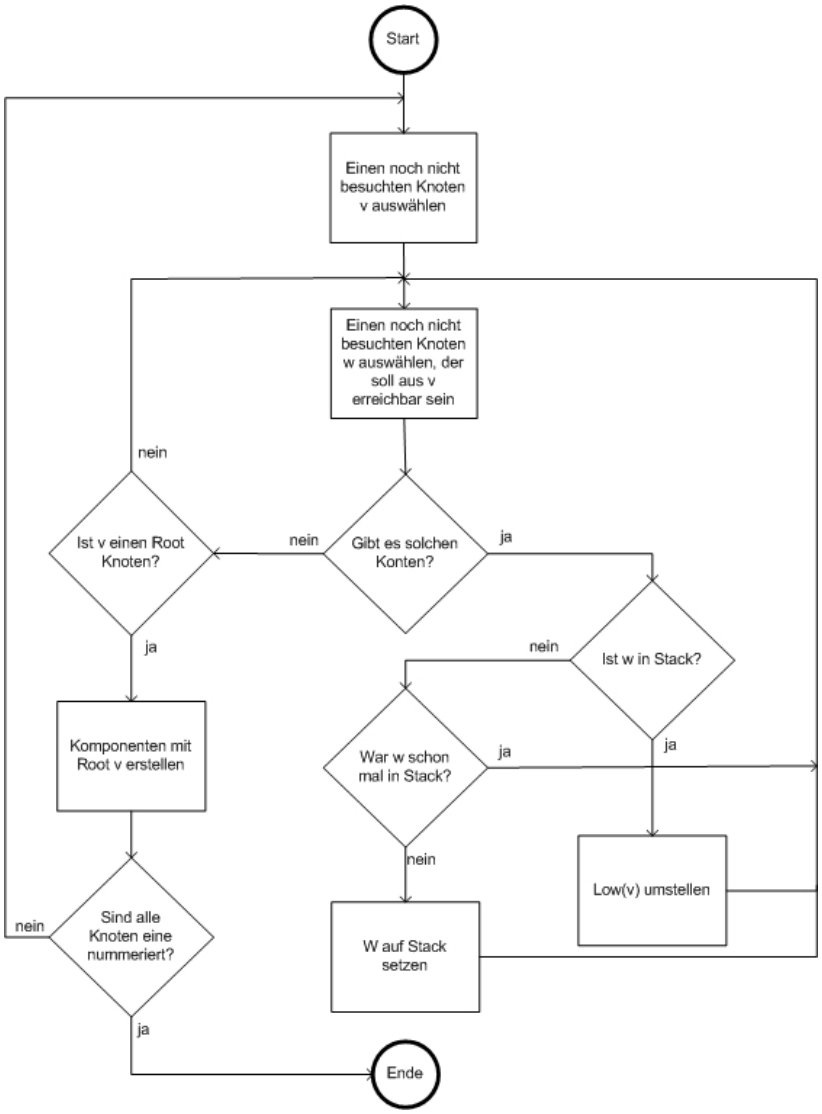

Abbildung 4.6: Tarjan Algorithmus

Wenn wir diesen Algorithmus auf das Strukturdiagramm anwenden, bekommen wir dann alle zusammenhängenden Komponenten aus dem Graph. Die Gleichungen aus einer Komponente bilden einen BLT-Block aus der Strukturinzidenzmatrix. Wir können dann nach der Reihenfolge von den gefundenen Komponenten die BLT-Matrix bauen.

4.5.2 Tearing Algorithmus von Cellier

In diesem Abschnitt wird das Tearing Verfahren von Cellier dargestellt werden. Das Verfahren basiert auf den Färbealgorithmus von Tarjan (Tar72). Als Datenstruktur braucht der Algorithmus den Strukturdigraph, den wir im letzten Kapitel schon beschrieben haben. Nach der Einführung von Tarjan Algorithmus wird erklärt, wie Tearing Variablen bei der Kausalisierung von DAEs ausgewählt werden können. Und am Ende werden noch einpaar Probleme dieses Algorithmus dargestellt.

4.5.2.1 Färbealgorithmus

Der Färbealgorithmus von Tarjan benutzt den Strukturdigraph für die Darstellung des Gleichungssystems. Wie im letzten Kapitel beschrieben stehen die Gleichungen auf der linken Seite des Graphen und Variablen auf rechten. Die Linie zwischen Gleichung i und Variable j bedeutet das Vorkommen von j in i.
Zu Beginn des Algorithmus sollen alle Kanten schwarz sein. Wenn wir einen Gleichung i für das Auflösen der Variable j ausgewählt haben, färben wir den Kanten zwischen i und j rot. In der hier verwendeten schwarzweißen Darstellung ersetzen wir die normalen Linien durch die gestrichelten. Wenn eine Variable j durch eine Gleichung aufgelöst wird, oder die Gleichung, die j enthält, für das Auflösen einer anderen Variablen eingesetzt ist, ist j daher bekannt. Und die am j endenden Linien werden blau gefärbt. Hier werden die Linien durch die strichpunktierten ersetzt. Eine kausale Gleichung hat genau eine rote (gestrichelte) Linie an sich. Eine akausale Gleichung besitzt nur schwarze(normale) und blaue (strichpunktierte) Linien. Analog haben die bekannte Variable genau eine rote(gestrichelte) Linien und die unbekannte nur schwarze (normale) und blaue (strichpunktierte). Jede Gleichung bzw. Variable darf nicht mehr als eine rote Linie besitzen.
Es sind zwei Regeln von dem Färbealgorithmus folgend gegeben:

1. **Für alle akausalen Gleichungen:** Wenn eine Gleichung nur mit einer schwarzen (normalen) Linie verbunden ist, färbe diese Linie rot (gestrichelt). Alle Linien, die mit der Variable, die am anderen Ende der roten Linien ist, wird blau (strichpunktiert) gefärbt. Die mit roter Linie verbundenen Gleichungen von diesem Typ werden nacheinander von 1 bis n nummeriert.

2. **Für alle unbekannte Variablen:** Wenn eine Variable nur mit einer schwarzen (normalen) Linie verbunden ist, färbe diese Linie rot (gestrichelt). Alle Linien, die mit der Gleichung, die am anderen Ende der roten Linien ist, wird blau (strichpunktiert) gefärbt. Die mit roter Linie verbundenen Gleichungen von diesem Typ werden nacheinander von n bis 1 nummeriert.

Für die Demonstration des Algorithmus nehmen wir das Beispiel aus dem letzten Abschnitt.

$$f_1(V_1) = 0, f_2(V_3, V_4) = 0$$
$$f_3(V_5, V_6) = 0, f_4(V_7, V_8) = 0$$
$$f_5(V_9, V_{10}) = 0, f_6(V_1, V_3) = 0$$
$$f_7(V_3, V_5, V_7) = 0, f_8(V_5) = 0$$
$$f_9(V_2, V_4) = 0, f_{10}(V_4, V_6, V_{10}) = 0$$

Der originale Strukturdigraph ist in Abbildung 3.3 abgebildet. Die Abbildung 4.7 zeigt den Strukturdigraph nach einer Färbeiteration. 5 Gleichungen aus dem Gleichungssystem werden kausalisiert. Die Gleichung Nr.1 und Nr. 8 werden durch Regel 1 kausalisiert, weil sie jeweils genau eine schwarze Linie besitzt. Und die Gleichung Nr. 4, Nr. 5 und Nr. 9 werden durch Regel 2 kausalisiert, weil die entsprechenden Variablen Nr. 8, Nr. 9 und Nr. 2 jeweils nur mit einer schwarzen Linie verbunden sind. Der Algorithmus wird weiter ausgeführt, bis alle Gleichungen kausalisiert werden oder er nicht mehr anwendbar ist. Der Algorithmus kann nicht angewendet werden, wenn das Gleichungssystem algebraische Schleifen enthalten oder singulär (OB99a) ist.

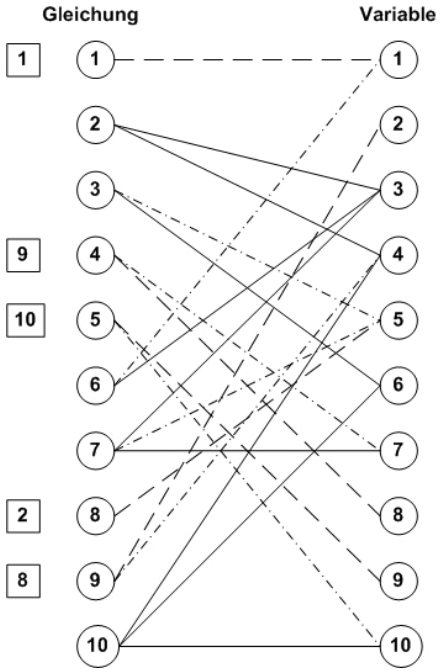

Abbildung 4.7: Strukturdigraph nach Teaing (partiell)

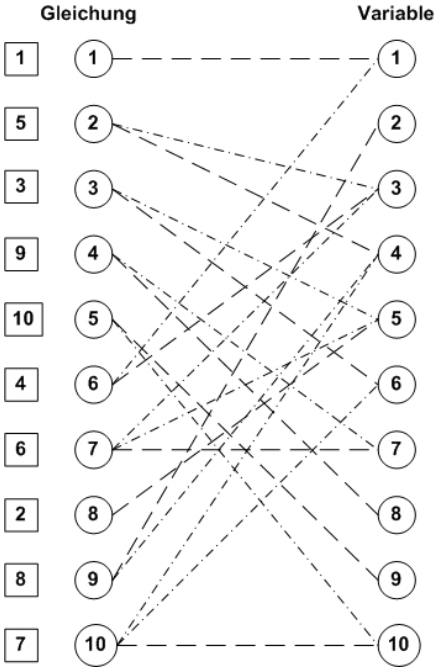

Abbildung 4.8: Strukturdigraph nach Tearing (komplett)

Wenn wir alle Gleichungen kausalisiert haben, bekommen wir die Strukturdigraph aus Abbildung 4.8. Jetzt können wir die Gleichungen nach die Nummerierung sortieren. Und die Variablen werden nach der Reihenfolge vom Auflösen auch sortiert. Die kausalisierten Gleichungen aus dem Beispiel werden wie folgend sortiert. Die daneben stehende Variable entspricht der Reihenfolge der Berechnung.

$$f_1(V_1) = 0 \Rightarrow V_1$$
$$f_8(V_5) = 0 \Rightarrow V_5$$
$$f_5(V_9), V_{10}) = 0 \Rightarrow V_9$$
$$f_4(V_7, V_8) = 0 \Rightarrow V_8$$
$$f_9(V_2, V_4) = 0 \Rightarrow V_2$$
$$f_3(V_5, V_6) = 0 \Rightarrow V_6$$
$$f_6(V_1, V_3) = 0 \Rightarrow V_3$$
$$f_10(V_4, V_6, V_{10}) = 0 \Rightarrow V_{10}$$
$$f_7(V_3, V_5, V_7) = 0 \Rightarrow V_7$$
$$f_2(V_3, V_4) = 0 \Rightarrow V_4$$

Wenn wir nun die Strukturinzidenzmatrix für das sortierte Gleichungssystem erzeugen, bekommen wir direkt eine lower-triangle Matrix.

Var. / GL.	1	5	6	3	4	7	10	2	8	9
1	1	0	0	0	0	0	0	0	0	0
8	0	1	0	0	0	0	0	0	0	0
3	0	1	1	0	0	0	0	0	0	0
6	1	0	0	1	0	0	0	0	0	0
2	0	0	0	1	1	0	0	0	0	0
7	0	1	0	1	0	1	0	0	0	0
10	0	0	1	0	1	0	1	0	0	0
9	0	0	0	0	1	0	0	1	0	0
4	0	0	0	0	0	1	0	0	1	0
5	0	0	0	0	0	0	1	0	0	1

Abbildung 4.9: Strukturinzidenzmatrix nach Färben

4.5.2.2 Tearing Algorithmus

Der Färbealgorithmus kann angewendet werden, wenn mindestens eine Gleichung oder Variable mit genau einer schwarzen Linie verbunden ist. Nun betrachten wir folgendes Beispiel:

$$f_1(V_1) = 0, f_2(V_5, V_6) = 0$$

$$f_3(V_7, V_8) = 0, f_4(V_9, V_{10}) = 0$$

$$f_5(V_3, V_4) = 0, f_6(V_2, V_6) = 0$$

$$f_7(V_6, V_8, V_{10}) = 0, f_8(V_1, V_5, V_9) = 0$$

$$f_9(V_7, V_9) = 0, f_{10}(V_3, V_5, V_7) = 0$$

Nach zwei Iterationen von Färbealgorithmus können wir die Strukturdigraph aus Abbildung 4.10 bekommen.

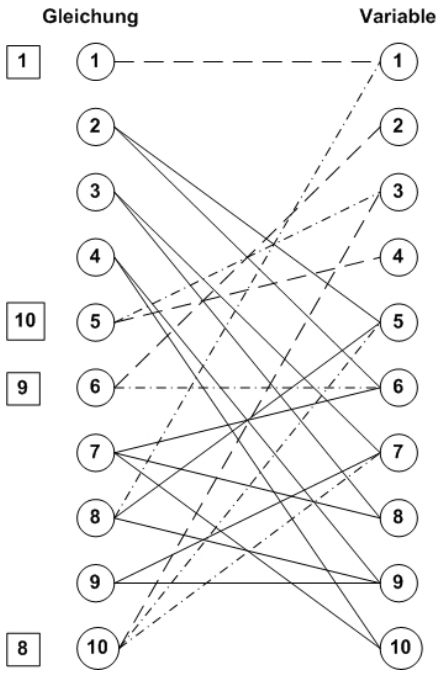

Abbildung 4.10: Strukturdigraph mit partieller Kausalisierung

Wir können in diesem Zustand den Färbealgorithmus nicht mehr benutzen, weil die Gleichungen bzw. Variablen jeweils mindestens zwei schwarzen Linien besitzen. Das bedeutet, dass die DAEs algebraische Schleifen enthalten. Um die Schleifen zu brechen und alle Gleichungen zu kausalisieren, verwenden wir Tearing Verfahren. Das Tearing Verfahren dient dazu, eine noch unbekannte Variable und eine diese Variable enthaltende Gleichung auszuwählen. Danach wird angenommen, dass die ausgewählte Variable durch die entsprechende Gleichung aufgelöst wird, damit die Färbealgorithmus fortgesetzt werden kann, weil die Anzahl von schwarzen Linien durch die Annahme reduziert ist. Das Problem besteht darin, wie man die Tearing Variable finden kann, damit man mit minimaler Anzahl von Tearing das gesamte Gleichungssystem zu kausalisieren. Dieses Problem ist NP-vollständig (CG97).

4.5.2.3 Heuristik von Cellier

Die Heuristik von Cellier (CK06) erzeugt immer eine kleine Anzahl von Tearing Variablen. In vielen Fällen erzeugt er sogar die optimale Lösung, d.h. die minimale Menge von Tearing Variablen. Der Vorteil dieser Heuristik ist, dass der Aufwand von Berechnung für die meisten Applikation quadratisch in die Größe des Gleichungssystems wächst (uC96). Die Heuristik kann in vier Schritten beschrieben werden.

1. Finde die Gleichungen, die mit den meisten schwarzen (normalen) Linien verbunden sind, heraus. Diese Gleichungen besitzen jeweils die größte Anzahl von noch unbekannten Variablen.

2. Finde in jeder gefundenen Gleichung die Variablen, die mit den meisten schwarzen (normalen) Linien verbunden sind, heraus. Diese Variablen kommen am häufigsten in den nicht verwendeten Gleichungen vor.

3. Bestimme für jede Variable aus dem 2. Schritt, wie viele weitere Gleichungen kausalisiert werden können, wenn diese Variable bekannt wäre.

4. Wähle die Variable als Tearing Variablen aus, durch die die meisten Gleichungen aufgelöst werden können.

Für das Beispiel aus Abbildung 4.10 besitzt die *Gleichung* 7 drei schwarze Linien, und die anderen nur zwei. Diese Gleichung enthält die *Variablen 6, 8 und 10*. Alle drei Variablen kommen jeweils zweimal in der unbenutzten Gleichung vor. Deshalb sollen wir für alle diese Variablen die Anzahl der dadurch kausalisierten Gleichungen bestimmen. Nach dem ersten Test merkt man, dass die *Variable 6* bereits erlaubt, sämtliche Gleichungen zu kausalisieren. Die Variable 6 wird dann als Tearing Variable ausgewählt. Die Verbindung zwischen *Gleichung* 7 und *Variable 6* wird rot (gestrichelt) gefärbt. Und die anderen Linien, die in die zwei Knoten enden, werden blau (strichpunktiert) gefärbt(Abbildung 4.11). Nach dem Auswählen der Tearing Variable sind neue Gleichungen(*Nr. 2*) bzw. Variablen(*Nr. 8 ,10*) mit genau einer schwarzen Linie aufgetaucht. Wir können nun den Färbealgorithmus fortsetzen. Nach kompletter Kausalisierung des Gleichungssystems können wir den Strukturdigraph aus Abbildung 4.12 bekommen.
Nachdem wir alle Gleichungen kausalisiert haben, können wir nun sie umsortieren, um die Strukturinzidenzmatrix in BLT Form umzuwandeln.

$$f_1(V_1) = 0 \Rightarrow V_1$$

$$f_7(V_6, V_8, V_{10}) = 0 \Rightarrow V_6 \hookleftarrow TearingVariable$$

$$f_2(V_5, V_6) = 0 \Rightarrow V_5$$

$$f_8(V_1, V_5, V_9) = 0 \Rightarrow V_9$$

$$f_9(V_7, V_9) = 0 \Rightarrow V_7$$

$$f_3(V_7, V_8) = 0 \Rightarrow V_8$$

$$f_4(V_9, V_{10}) = 0 \Rightarrow V_{10}$$

$$f_{10}(V_3, V_5, V_7) = 0 \Rightarrow V_3$$

$$f_6(V_2, V_6) = 0 \Rightarrow V_2$$

$$f_5(V_3, V_4) = 0 \Rightarrow V_4$$

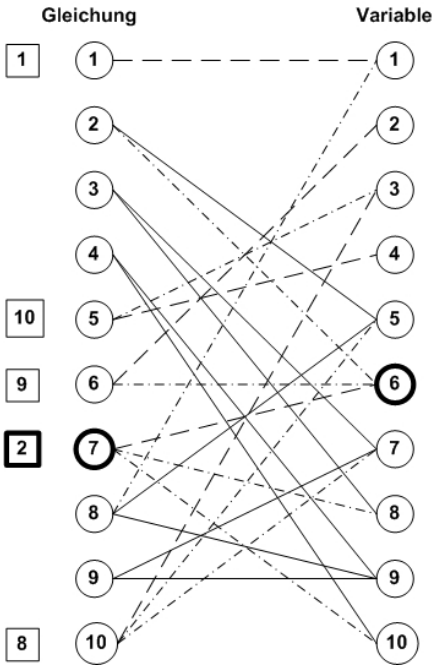

Abbildung 4.11: Auswahl von Tearing Variable

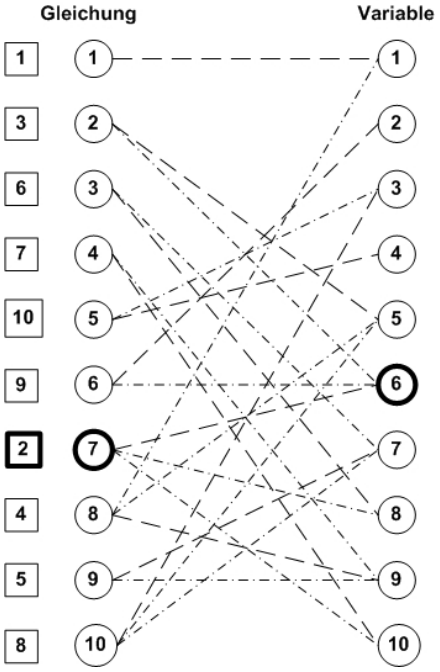

Abbildung 4.12: Kausalisierung nach Tearing

Daraus können wir die Strukturinzidenzmatrix in Abbildung 4.13 bekommen. Die Matrix ist in BLT(Block-lower-triangle) Form. Die Gleichung Nr. *1*, *10*, *6* und *5* bildet jeweils ein Block mit sich selbst. Die anderen Gleichungen(Nr. *7*, *2*, *8*, *9*, *3* und *4*) bilden einen großen Block. Die *Gleichung* 7 steht ganz oben in diesem Block. Sie haben zwei Variablen(Nr. *8* und *10*) rechts von der Diagonale der Matrix, weil die Tearing Variable zu ihr zugeordnet ist. Alle anderen Gleichungen dürfen nur Variablen auf oder links von der Diagonale enthalten. Die ausgewählte Tearing Variable werden dann durch numerisches Verfahren berechnet.

Var. / GL.	1	6	5	9	7	8	10	3	2	4
1	1	0	0	0	0	0	0	0	0	0
7	0	1	0	0	0	1	1	0	0	0
2	0	1	1	0	0	0	0	0	0	0
8	1	0	1	1	0	0	0	0	0	0
9	0	0	0	1	1	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0
4	0	0	0	1	0	0	1	0	0	0
10	0	0	1	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0	0	1	0
5	0	0	0	0	0	0	0	1	0	1

Abbildung 4.13: Durch Tearing erzeugte BLT-Matrix

4.5.2.4 Problem

Das Kernproblem von Cellier Tearing Verfahren ist die Erfindung einer möglichst optimaler Heuristik für die Tearing Variable und Gleichung, damit wir möglichst wenige Tearing Variablen für das gesamte System brauchen. Eine optimale Lösung zu finden, ist NP-vollständig. Eine gute Heuristik ist schwer zu finden. Die Heuristik liefert nicht immer die optimale Lösung, und in manchen Fällen führt sie sogar das System zu einem ungelösten Zustand. In diesem Zustand darf man die Färbealgorithmus und Tearing nicht mehr anwenden, obwohl das System nicht strukturell singulär ist. Die folgende Abbildung 4.14 verdeutlicht die Ursache des Problems.

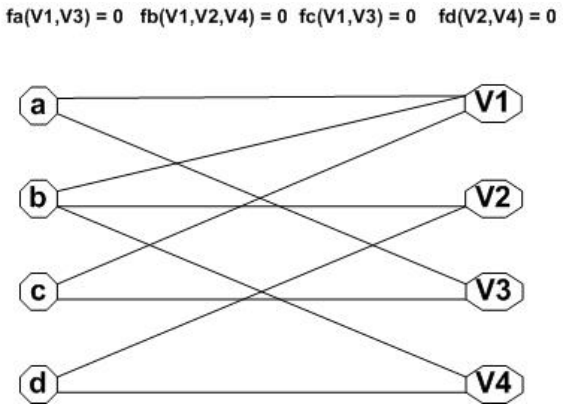

Abbildung 4.14: Problem beim Cellier Verfahren(a)

Hier haben wir ein kleines Gleichungssystem mit 4 Gleichungen und 4 Variablen. Jede Gleichung bzw. Variable ist mit mindestens 2 schwarzen Linien gebunden. Der Färbealgorithmus kann nicht mehr fortgesetzt werden. Hier verwenden wir die Standard Heuristik für Tearing. Die Gleichung f_b haben drei schwarze Linie an sich, d.h. sie enthält die meisten Variablen. Deshalb wählen wir sie als Anfangspunkt für Tearing aus. Die Gleichung f_b enthält die Variable V_1, und sie kommt auch in den meisten Gleichungen vor. Deshalb wählen wir sie als die Tearing Variable. Nach Anwenden des Färbealgorithmus kommen wir zu diesem Zustand 4.15. Die Gleichung f_c und Variable V_4 haben jeweils zwei blaue Linien an sich, aber keine schwarze mehr.

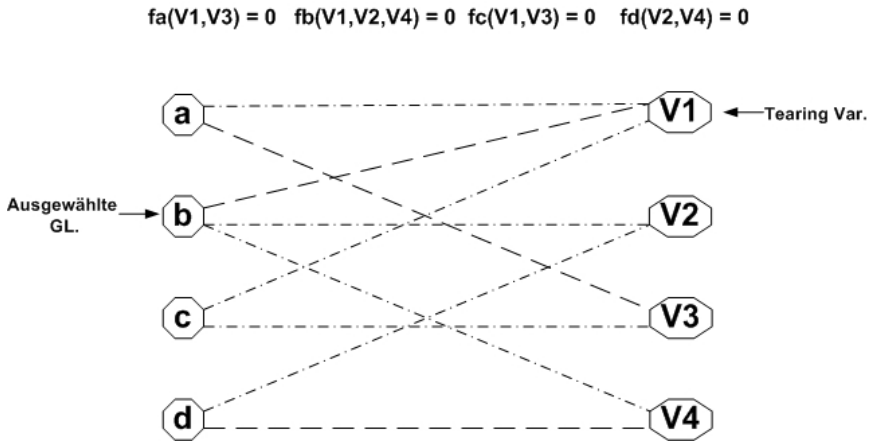

Abbildung 4.15: Problem beim Cellier Verfahren(b)

Normalerweise kann daraus geschlossen, dass dieses System singulär ist. Aber wir können durch zweimaligen Tearing die Gleichungen problemlos

kausalisieren 4.16. Die Gleichung f_a und f_c enthalten die gleichen Variablen V_1 und V_2. Aber nach Standard Heuristik wird die Variable V_1 zu Gleichung f_b zugeordnet. Deshalb muss entweder f_a oder f_c nach Färben nur blaue Linien haben. Das bedeutet, dass die Gleichung f_a und f_c nur in einem Block in BLT sein dürfen.

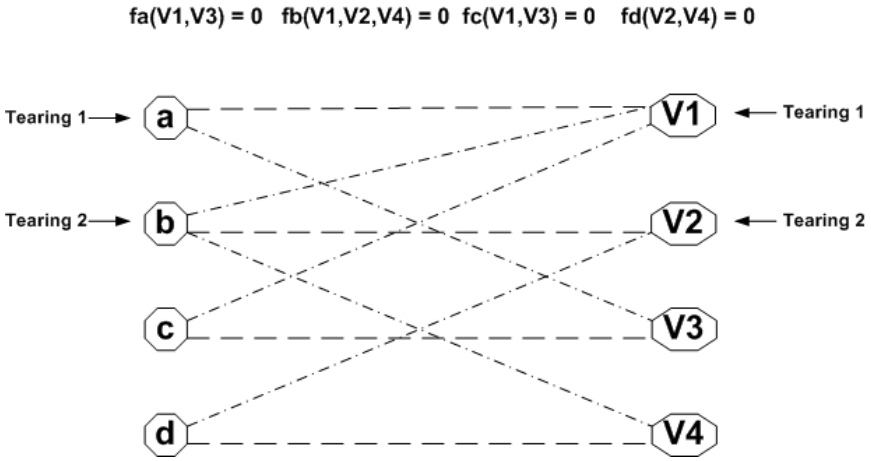

Abbildung 4.16: Problem beim Cellier Verfahren(c)

Allgemein: Wenn ein Gleichungssystem Gleichungen $f_1...f_n$ mit gleichen Variablen $V_1...V_m$ enthalten(mit $n \leq m$), dann dürfen die m Variablen nur durch $f_1...f_n$ aufgelöst werden, sonst führt es zu einem ungelöstem Zustand.

Welche Maßnahmen können wir treffen, wenn wir beim Cellier Verfahren solches Problem bekommen? Wir können das System in den Zustand vor Tearing bringen, und andere Tearing Variable nehmen(vielleicht mit andere Heuristik). Dazu brauchen wir für jeden Tearing Schritt ein Zwischenspeichern vom Systemzustand, weil wir nicht wissen, bei welchem Schritt einen nicht „richtige“ Tearing Variable genommen haben. Das Zurücksetzen kostet Zeit, und bei großem System ist das Zwischenspeichern auch sehr teuer. Wenn wir Tearing auf einen BLT-Block angewendet haben, können wir auch die alte Variablenzuordnung ändern, damit das Problem beseitigt werden kann. Wie man Variablen zu einer Gleichung zuordnet und der BLT-Block erzeugt, werden im nächsten Abschnitten genau beschrieben.

4.5.3 Der Ollero-Amselem Algorithmus

Das auf Färbealgorithmus basierende Tearing Verfahren ordnet die Variablen zu, während es die Tearing Variable findet. Dadurch können Probleme bei der Variablenzuordnung auftauchen, wie es im letzten Abschnitt beschrieben ist. Wenn wir nach der BLT-Zerlegung die Variablenzuordnung nicht mehr ändern, wird dieses Problem nicht mehr auftauchen. Nun führen

wir ein neues Verfahren - der Ollero-Amselem Algorithmus - für Tearing DAEs ein, die von Mah (Mah90) im Buch dargestellt hat. Die Variablenzuordnung ist die Grundlage für diesen Algorithmus. Das Finden von Tearing Variablen wird hier durch eine Technik aus Graphentheorie realisiert. Ollero-Amselem Algorithmus nutzt das Strukturdigraph als Eingabe aus und ist am Anfang im *Signal-Flow Graph* für das Finden minimaler *tear-set* eingesetzt.

4.5.3.1 Beschreibung von Algorithmus

Der Tearing Algorithmus hier ist ganz anders als bei Cellier Verfahren. Die Variablenzuordnung wird während Tearing nicht mehr ändern. Die Aufgabe hier ist, mit möglichst wenigen Variablen alle Schleifen im Strukturdigraph zu brechen. Es gibt dafür auch einen Algorithmen aus der Graphentheorie für das Finden des so genannten *essential-set* in einem gerichteten Graph, nämlich der *Ollero-Amselem Alorithmus* (OA83). Wenn ein gerichteten Graph Zyklen enthält, können wir mit diesem Algorithmus eine kleine Menge von Knoten raus finden. Wenn diese Knoten aus dem Graph weggenommen sind, wird dieser Graph zyklenfrei sein. In einem Strukturdigraph entspricht der einzelne Knoten immer einer Gleichung aus dem gesamten System. Wenn man die von *Ollero-Amselem Algorithmus* ausgewählten Gleichungen zuerst auflöst, kann man die weiteren Gleichungen aus dem System durch einfache Substitution auch auflösen. Deshalb ist das *essential-set* von dem Strukturdiagramm auch das *tear-set* von dem Gleichungssystem. Die dazu zugeordneten Variablen werden auch als Tearing Variablen ausgewählt.

Der *Ollero-Amselem Alorithmus* basiert auf die Verkürzung von Kanten durch Löschen eines Knotens im Graph. Es gibt vier Arten von der Vereinfachung 4.17.

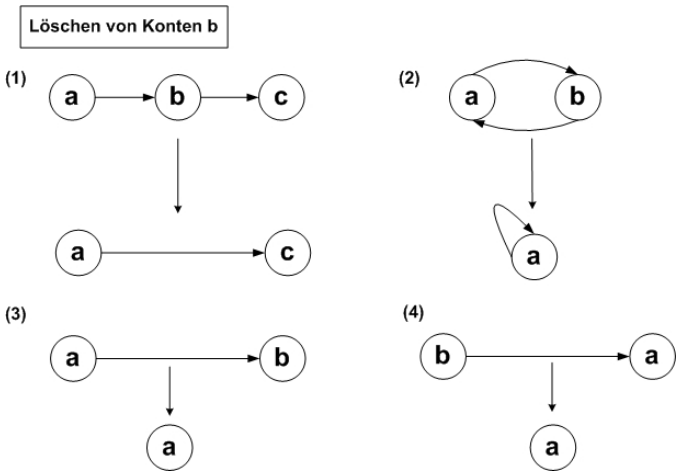

Abbildung 4.17: Vereinfachen Graph

Wir können die Knoten in zwei Mengen teilen. Die Knoten ohne Ein- oder Ausgang bilden die erste Menge. Diese Knoten befinden sich nicht in einer Schleife und kann beim Finden der Tearing Variablen vernachlässigt werden. Alle anderen Knoten sollen wir für Tearing testen. Nach Verkürzung von Kanten kann Knoten mit *self-Loop* erzeugt werden. Diese Knoten sollen als die Tearing Variablen ausgewählt werden. Der Algorithmus startet mit Knoten ohne Ein- oder Ausgang, dann Knoten mit Eingangs- oder Ausgangsgrad 1,2 usw. Wenn ein Knoten ausgewählt ist, sollen alle mögliche Verkürzung durchgeführt werden. Und dann wird der Knoten gelöscht. Wenn ein Knoten mit *self-Loop* gewählt ist, soll er zur *tear-set* eingefügt, und die Verkürzung wird nicht ausgeführt. Der Algorithmus terminiert, wenn alle Knoten aus dem Graph gelöscht sind. Folgend sind die Schritte des Algorithmus (Mah90):

1. Sei ES = 1, ES spezifiziert den Eingangs- oder Ausgangsgrad eines Knoten im Graph *G*.
2. Wähle einen Knoten v aus G aus.
   (a) Lösche v gelöscht und gehe zu Schritt 3, wenn v ohne Ein- oder Ausgang ist.
   (b) Setze v in die *tear-set*, lösche v und gehe zu Schritt 3, wenn v eine *self-Loop* enthält.
   (c) Lösche v und verkürze die Kanten, wenn die Eingangs- oder Ausgangsgrad von v gleich ES ist. Sonst gehe zu Schritt 4.
3. Algorithmus terminiert, wenn G keine Knoten mehr hat.
4. Zurück zu Schritt 2, bis alle Knoten für den aktuellen Wert von ES getestet sind.
5. Wenn mindestens einen Knoten von Schritt 2(b) gelöscht ist, wird ES auf 1 zurückgesetzt, sonst wird es um 1 erhöht und der Algorithmus wird auf Schritt 2 zurückgesetzt.

Als Beispiel nehmen wir ein Gleichungssystem aus letzten Abschnitten.

$$f_a(V_1, V_2, V_3, V_6) = 0, f_b(V_1, V_2) = 0$$

$$f_c(V_2, V_3) = 0, f_d(V_4, V_5) = 0$$

$$f_e(V_2, V_4, V_5) = 0, f_f(V_2, V_4, V_6) = 0$$

Das erste Bild aus der Abbildungen 4.18 zeigt den Strukturdigraph für dieses System. Jede Gleichung wird als einen Knoten abgebildet. Im ersten Schritt wird die Gleichung a betrachtet, weil die Eingangsgrad des Knotens gleich *ES*

= 1 ist. Der Knoten wird danach gelöscht. Es wird drei zusätzlichen Kanten in den Graph eingefügt (Bild 2), nämlich der Kanten $b \rightarrow b$, $b \rightarrow c$ und $b \rightarrow f$. Die Eingangs- bzw. Ausgangsgrad von Knoten b ist 3 bzw. 2. Sie sind nicht gleich wie ES-Wert. Deshalb werden wir weiter Knoten c , d und danach Knoten f betrachten (Bild 5). Weil wir alle Knoten einmal durchgesucht haben, sollen wir den Es-Wert um 1 erhöhen. Die Knoten b und e enthalten jeweils eine *self-Loop*. Die zwei Knoten werden dann in das *essential-set* eingefügt. Das bedeutet, dass es für das Auflösen des Gleichungssystems zwei Tearing Variablen brauchen, nämlich die Variablen 2 und 5, die zu der Gleichung b bzw. e zugeordnet sind.

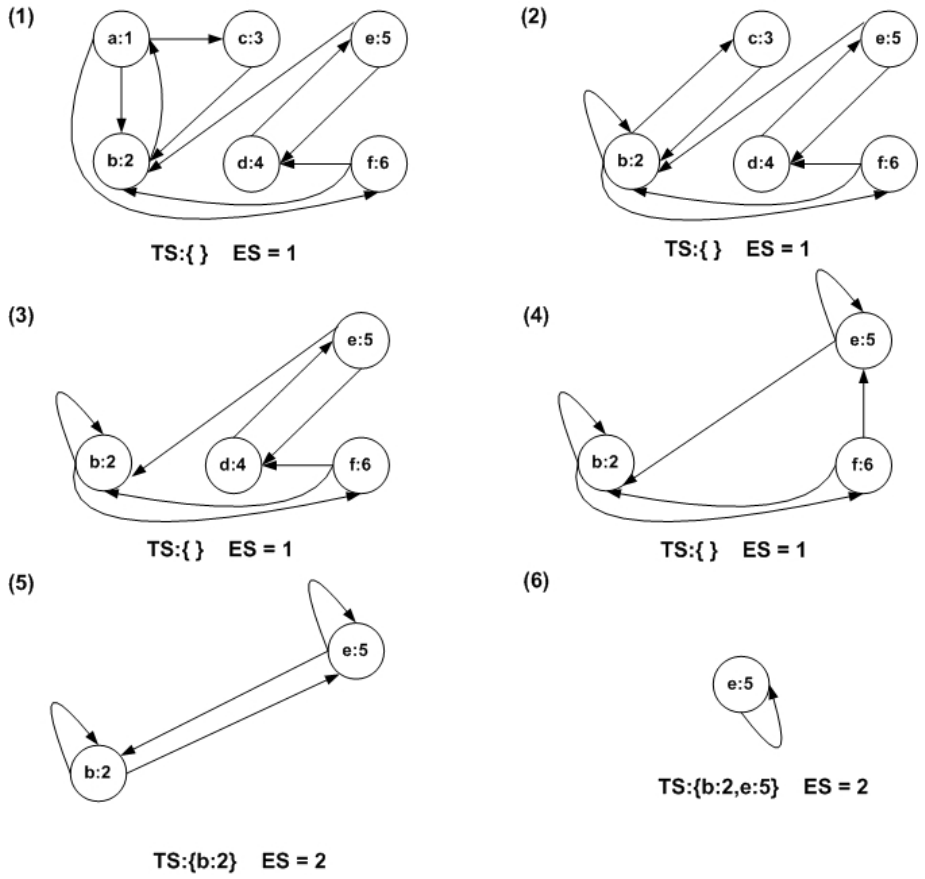

Abbildung 4.18: Ollero-Amselem Algorithmus

4.5.3.2 Problem

Der *Ollero-Amselem Algorithmus* ist einfach und robust. Aber es gibt auch folgende Probleme:

- Es gibt noch keinen Beweis, dass man mit diesem Algorithmus immer

das minimale *essential-set* von Knoten bekommt. Ollero und Amselem haben auch keine klare Behauptung darüber gegeben. Aber sie berichteten, dass ihr Algorithmus bei 5 veröffentlichen Beispielen immer die optimale Lösung berechnet.

- Der Ollero-Amselem Algorithmus findet nur das *essential-set* aus einem gerichteten Graph aus. Während des Algorithmus wird die Reihenfolge der auflösenden Gleichungen nicht bestimmt, wie bei Cellier Verfahren. Deshalb sollte man ihn modifizieren, damit er die Gleichungen auch vernünftig sortiert. Wir brauchen während des Algorithmus zusätzliche Informationen sammeln. Wenn ein Knoten ohne Ausgang beim Schritt 2(a) gefunden ist, bedeutet, dass die Gleichung in diesem Knoten die zugeordnete Variable selbst berechnet kann. Und solche Gleichungen sollen zu Beginn gelöst werden. Knoten ohne Eingang bedeutet, dass die durch diese Gleichung berechnetet Variable von keiner anderen Gleichung gebraucht wird. Solche Gleichungen können ganz am Ende gelöst werden. Immer wenn man eine Tearing Variable findet, soll der Graph aktualisiert, und die neue Knoten ohne Ein- oder Ausgang werden entsprechend behandelt.

- Mit diesem Algorithmus kann das Zuordnungsproblem von Cellier nicht mehr auftauchen. Weil die Variablenzuordnung wird während Tearing nicht geändert. Das bedeutet, dass das Verfahren für ein Gleichungssystem mit gleicher Variablenzuordnung immer gleiche Lösung findet, wenn sie nicht singulär ist. Aber sie findet nicht immer die beste Lösung für ein System. Das liegt an der Variablenzuordnung. Wie in den letzten Abschnitten beschrieben sind, bestimmt die Variablenzuordnung die Abhängigkeit zwischen den Gleichungen im System. Das Strukturdiagramm ist auch darauf basiert. Wir haben bis jetzt Variablen nur nach einer normalen Tiefensuche zugeordnet. Das kann nicht garantieren, dass dieses System nicht mit einer anderen Variablenzuordnung wenigere Tearing Variablen fürs Auflösen braucht. Dafür können wir entsprechende Heuristiken einbauen, damit die Variablen nicht „blind“ zugeordnet sind. Eine heuristische Suche ist hier sehr nützlich. Wie man dann Variablen zuordnet, sehen wir im nächsten Kapitel.

4.5.4 Der Steward Algorithmus

Im letzten Abschnitt haben wir der *Ollero-Amselem Algorithmus* eingeführt. Der Algorithmus findet während Tearing die Schleifen im Strukturdigraph und bricht sie mit entsprechenden Tearing Variablen. Wir können auch zuerst alle Schleifen aus dem Gleichungssystem herausfinden, und dann die Variablen mit meisten Vorkommen als Tearing Variablen wählen. Dafür gibt einen Algorithmus von Steward (SW62).

4.5.4.1 BLT Zerlegung von Donald v. Steward

Auf Basis von Kron's Artikel (Kro62) (Rot55) hat Donald v. Steward in 1965 eine Idee für Tearing herausgegeben (Ste65). Er hat in seinem Artikel zuerst einen Algorithmus für BLT-Zerlegung eingeführt, der auf reine Matrixumsortierung basiert. Als Beispiel ist die folgende Strukturmatrix 4.19 gegeben. Die Zeilen entspricht den Gleichungen, und Spalten den Variablen. Es gibt zwei Matrixeinträge: **x** und **o**. **x** auf Position (i, j) steht für das Vorkommen der Variable j in der Gleichung i. Und **o** auf Position (i, j) bedeutet, dass die Variable j zu der Gleichung i zugeordnet ist.

	1	2	3	4	5	6	7	8	9	10	11	12
1						x	x		o			
2			x	o							x	
3		x						x		o		
4								o				
5				x	x						o	
6			x									o
7	o							x				
8		o								x		
9					o	x						
10						o			x		x	
11	x			x			o					
12			o									x

Abbildung 4.19: BLT Erzeugung: Startpunkt

Bevor wir den Algorithmus anwenden, soll die Matrix zuerst mal umsortiert werden, damit alle zugeordneten Einträge auf der Diagonale stehen. Und alle **o**-Einträge sollen gelöscht werden. Dann bekommen wir für das Beispiel die Matrix in Abbildung 4.20. Jede Zeile in der Matrix haben zwei Attributen: *order* und *into*. *order* bedeutet die Reihenfolge von Blöcken. *into* bedeutet, zu welchem Block die Gleichung gehört. Jede Gleichung kann nur genau eine von den beiden Attributen enthalten.

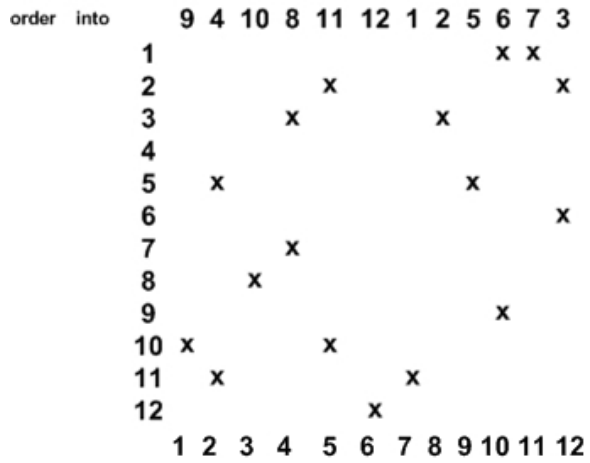

Abbildung 4.20: BLT Erzeugung: Vorbereitung

Nun können wir den Algorithmus betrachten. Das Diagramm für den Ablauf dieses Algorithmus wird in der Abbildung 4.21 gezeigt.

- Finde alle Zeile ohne Einträge raus, lösche sie und die entsprechende Spalte (hier entspricht das Löschen einer Zeile bzw. Spalte das Umtauschen aller Matrixeinträge aus der Zeile bzw. Spalte durch -). Dieser Schritt wird wiederholt, bis es keine solche Zeile gibt. Für die gefundenen Zeilen und Spalten geben wir jeweils eine Marke(hier **x**) und *order* Nummer. Die *order* wird immer hoch gezählt. Wenn alle Zeilen in diesem Schritt bereits gekennzeichnet sind, haben wir dann genug Informationen für die Blt-Erzeugung.

- Finde in der Matrix anhand der Einträge einen Pfad, bis eine Schleife auftaucht, z.B. *i-j-i*. Dann machen wir *union* und *collapse*.

  $union(i, j)$ besagt, dass es für jede **x** aus der Zeile j eine $+$ in die Zeile i auf entsprechende Spalte ergänzt werden soll, wenn es nicht auf der Diagonale $[i, i]$ steht. Für die Zeile m, die auf der Position $[m, j]$ **x** enthält, soll ein $+$ in die Position $[m, i]$ addiert werden. Die Zeile bzw. Spalte j wird dann gelöscht. Auf der Gleichungsebene können wir so verstehen: wir haben alle von Gleichung j benötigten Variablen in die Gleichung i addiert. Die Gleichung m braucht vorher die Information über zu j zugeordnete Variable. Weil wir j gelöscht haben, soll m zusätzliche Information über i haben, die alle Informationen von j geerbt hat. Bei einer gefundenen Schleife *i-j-i* ist $union(i, j)$ und $union(j, i)$ gleich, weil eine andere Schleife *j-i-j* enthält die Matrix auch. Hier machen immer $union(i, j)$, wenn $i < j$ ist.

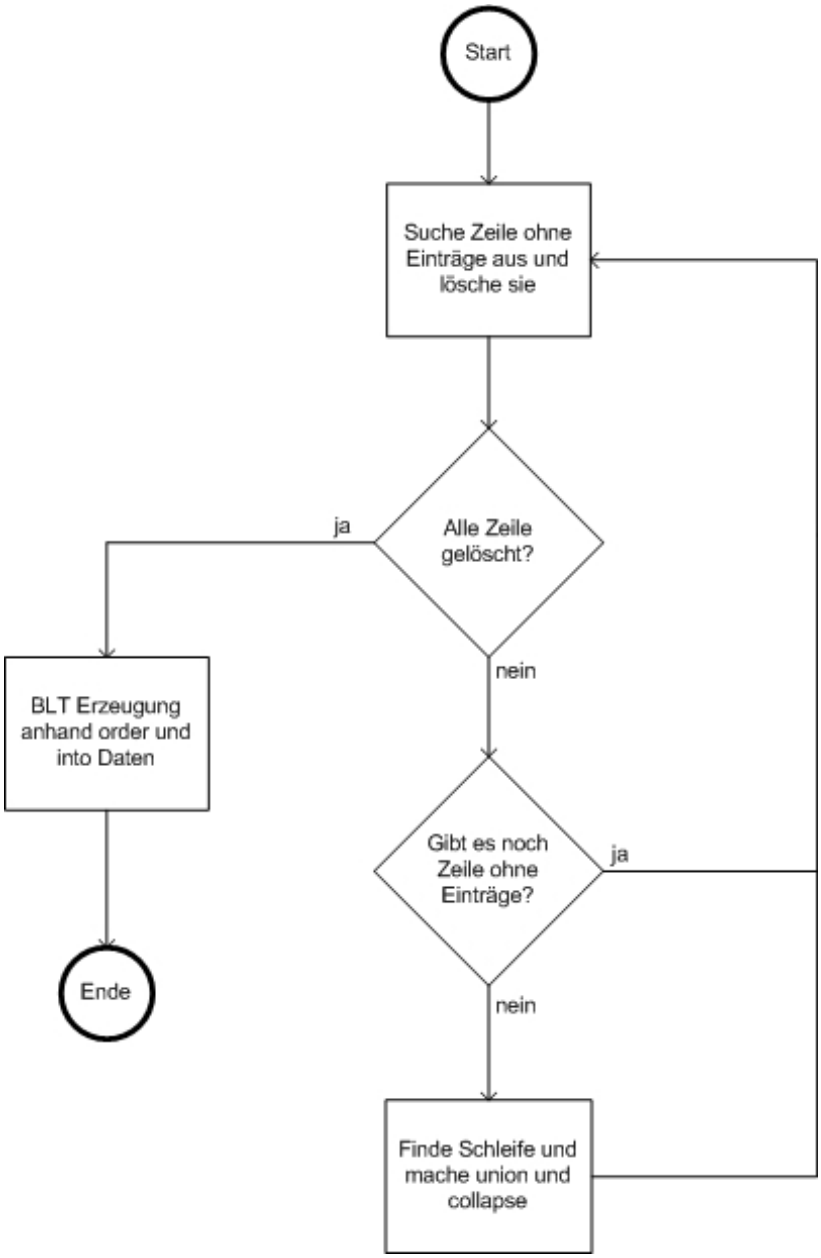

Abbildung 4.21: Donald v. Steward Algorithmus

collapse(i,j) bedeutet, dass die Gleichung j soll zum Block gehören, der mit der Gleichung i anfängt. Wir schreiben die Nummer i in die *into* Spalte von Zeile j.

- Durch *union* und *collapse* wird dann neue Matrix erzeugt. Sie kann neue Zeilen ohne **x** und + bringen. Sie bilden mit den Zeilen mit gleicher *into*-Nummer neue Blöcke.

Wir können nun diesen Algorithmus auf die Matrix aus der Abbildung 4.20 anwenden:
(1) Die Zeile 4 hat keinen Eintrag. Er wird gelöscht und bekommt die *order* 1.
(2) Durch das Löschen von Zeile 4 enthält die Zeile 7 auch keinen Eintrag mehr. Lösche sie und gebe sie die *order* 2.

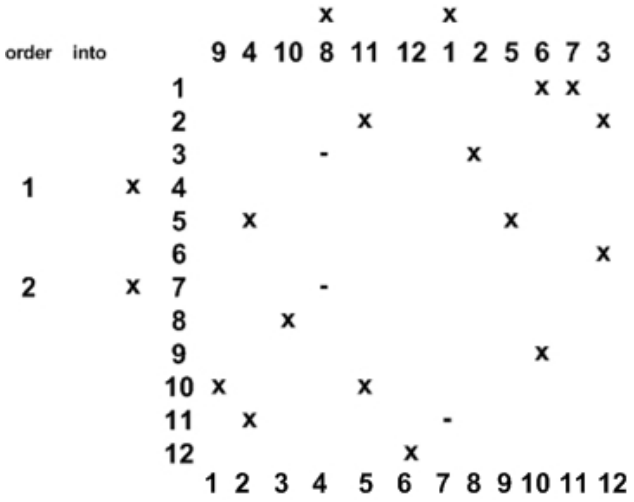

Abbildung 4.22: BLT Erzeugung: Löschen von Zeilen 4 und 7

(3) Es gibt keine Zeile ohne Einträge mehr. Wir beginnen Schleifensuche mit dem ersten Eintrag in Zeile 1. Das ist die Variable 6 auf Spalte 10. In Zeile 10 gibt es auch ein **x** auf der ersten Spalte. Deshalb haben wir eine Schleife 1-10-1 gefunden. Wir sollen nun $union(1, 10)$ 4.23 machen. Die Zeile 10 haben auf Spalte 5 ein **x**, deshalb schreiben wir ein + auf Position [1,5]. Die Zeile 9 haben ein **x** in der Spalte 10, so schreiben wir auch ein + in die ersten Spalte von ihr. Dann wird die Zeile 10 gelöscht. Durch $collapse(1, 10)$ wird die *into* Nummer von Zeile 10 auf 1 gesetzt.
(4) Schleife 1-5-2-5. $union(2, 5)$ und $collapse(2, 5)$.
(5) Schleife 1-2-9-1. $union(1, 2)$, $union(1, 9)$ und $collapse(1, 2)$, $collapse(1, 9)$.
(6) Schleife 1-11-1. $union(1, 11)$ und $collapse(1, 11)$.

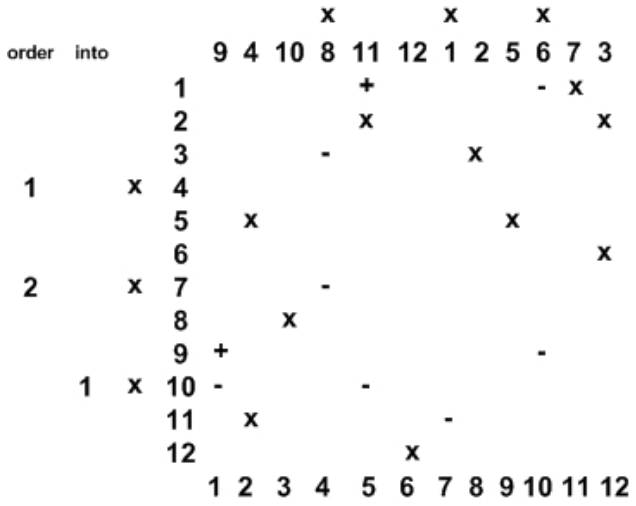

Abbildung 4.23: BLT Erzeugung: union(1,10)

(7) Schleife 1-12-6-12. $union(6, 12)$ und $collapse(6, 12)$.
(8) Zeile 6 ohne Einträge. *order* von 6 auf 3 setzen.
(9) Zeile 1 ohne Einträge. *order* von 1 auf 4 setzen.
(10) Schleife 3-8-3. $union(3, 8)$ und $collapse(3, 8)$.
(11) Zeile 3 ohne Einträge. *order* von 3 auf 5 setzen. So sind alle Zeilen von der Matrix gelöscht. Wir erreichen den Endzustand wie in Abbildung 4.24.

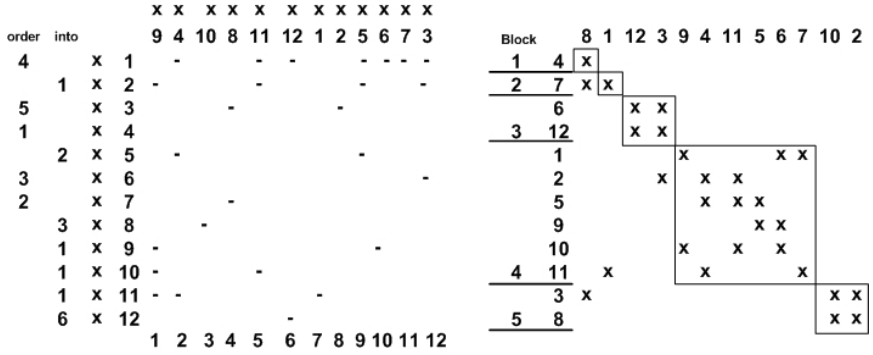

Abbildung 4.24: BLT Erzeugung: Ergebnis des Algorithmus

4.5.4.2 Beschreibung von Tearing

Nach der BLT-Zerlegung haben wir die Strukturinzidenzmatrix für das Gleichungssystem in BLT Form bekommen. Die einzelnen Blöcke, die aus mehr als zwei Gleichungen bestehen, enthalten jeweils algebraische Schleifen. Nun sollen wir für jeden Block die Tearing Variablen finden, um die Schleifen zu

brechen. Hier wird der Algorithmus von Donald v. Steward (Ste65) für das Beispiel aus der Abbildung 4.24(rechts) dargestellt.

Wir nehmen als Beispiel den Block 4 4.25 anfangs mit Gleichung 1 aus und verwenden den Algorithmus darauf. Die Einträge auf der Diagonale können vernachlässigt werden. Der Block besteht also aus der Gleichungen 1, 2, 5, 9, 10 und 11, zu denen die entsprechenden Variablen 9, 4, 11, 5, 6 und 7 zugeordnet sind.

	9	4	11	5	6	7
1					x	x
2			x			
5		x		x		
9					x	
10	x		x			
11		x				

Abbildung 4.25: Steward Tearing: Block 4

Zu Beginn des Algorithmus sollen alle Schleifen in diesem Block gefunden werden. Wir wählen zuerst die Gleichung 1 aus. Sie hat einen Eintrag mit in der Spalte mit der Variable Nummer 6, die zu der Gleichung 10 zugeordnet ist. Dann folgen wir den Weg 1-10 weiter und finden einen Eintrag von der Gleichung 10 in der Spalte mit Nummer 9, die von Gleichung 1 aufgelöst werden soll. Der Weg soll jetzt 1-10-1 sein. Und eine Schleife ist gefunden und wird als Schleife A gekennzeichnet. So können wir alle Schleifen in diesem Block herausfinden 4.26.

Nun bilden wir eine Matrix für die Schleifen(Abbildung 4.27 oben). Die Zeilen in der Matrix entsprechen den Schleifen und die Spalten den Gleichungen. Der Eintrag in der Matrix entspricht der Kombination von Gleichung und Variable, die für Tearing ausgewählt werden kann. Wenn wir eine Kombination auswählen, werden alle Schleifen, die diese Kombination enthalten, gebrochen. Der Eintrag m an Position [i, j] bedeutet, dass die Gleichung i in der Schleife j die Variable m für den Weg benutzt. Die jetzige Aufgabe ist, möglichst wenige Einträge aus der Matrix zu nutzen, um all Schleifen zu brechen.

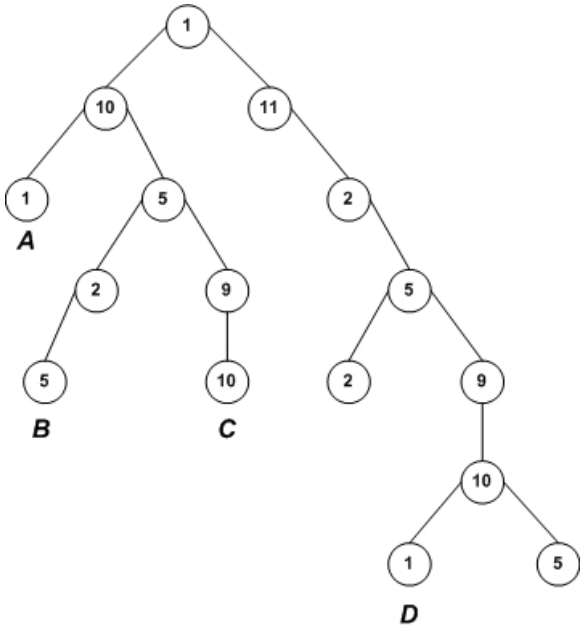

Abbildung 4.26: Steward Tearing: Schleifen im Block

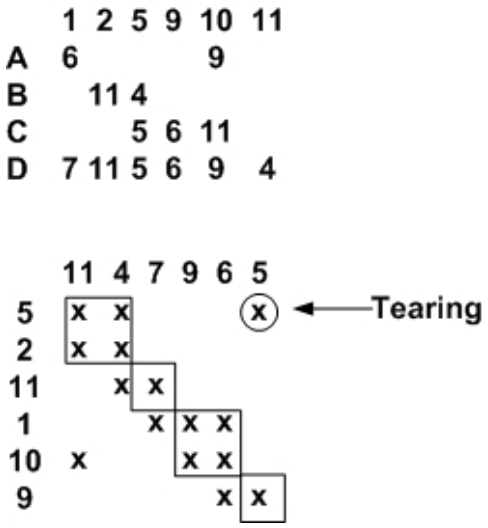

Abbildung 4.27: Steward Tearing: Schleifen Matrix und Tearing Variable

Die Variable 11 kommt zweimal in der Spalte 2 vor. Wenn wir Variable 11 von Gleichung 2 als Tearing Variable wählen, werden die Schleifen B und D gebrochen. Die Schleifen A und C bleiben nicht geändert. Weil die zwei Schleifen verschiedene Einträge in der Spalte 10 enthalten, kann man sie nicht durch ein Tearing brechen. Sie bilden zusammen wieder einen Block, der aus alle in der zwei Schleifen vorkommenden 4 Gleichung 1,5,9 und 10 bildet.

Es gibt auch andere Möglichkeit für das Tearing hier. Wir können z.B. die Variable 5 aus der Gleichung 5 als Tearing Variable festlegen. Dadurch werden die Schleifen C und D gebrochen. Die Schliefen A und B bleiben übrig. Die zwei haben keinen Eintrag in einer gleichen Spalte. Deshalb bilden die beiden jeweils einen Block, der aus 2 Gleichungen bildet. Wir können dann Tearing Variablen für die zwei kleineren Blöcke finden und das Gleichungssystem wird gelöst. In der Abbildung 4.27(unten) ist die Matrix, wenn wir so Tearing Variable wählen.

4.5.4.3 Problem

Steward hat in seinem Artikel auch erklärt, dass es keine gute Strategie gibt, wie wir unter den Variablen vernünftig auswählen, damit ein minimales *tearSet* erzeugt wird. Das Suchen nach alle Schleifen in einem Graph kostet auch Zeit, wenn das Problem hohe Komplexität enthält.

Kapitel 5

Eigene Tearing Heuristik

Im letzten Kapitel haben wir einige bekannten Tearing Algorithmen gesehen. Es gibt keine Behauptung, welcher Algorithmus der beste ist. Es gibt keinen klaren Sieger. Die NP-Vollständigkeit des Problems ist ein wichtiger Grund. Wir haben bei den Algorithmen auch Probleme gefunden. Deshalb werden wir in diesem Kapitel jeweils mögliche Lösung oder Verbesserungsidee dafür einführen.

5.1 Neue Heuristik für Cellier Verfahren

5.1.1 Problemdarstellung und Heuristik

Die Heuristik von Cellier findet nicht immer die optimale Lösung für Tearing. Hier meint die optimale Lösung, ein Gleichungssystem mit minimalen Tearing Variablen zu lösen. Folgend ist ein Gegenbeispiel gegeben.

$$f_1(V_1, V_2, V_3, V_6) = 0$$

$$f_2(V_1, V_2) = 0$$

$$f_3(V_2, V_3) = 0$$

$$f_4(V_4, V_5) = 0$$

$$f_5(V_3, V_4, V_5) = 0$$

$$f_6(V_1, V_4, V_6) = 0$$

Der entsprechende Strukturdigraph ist 5.1:

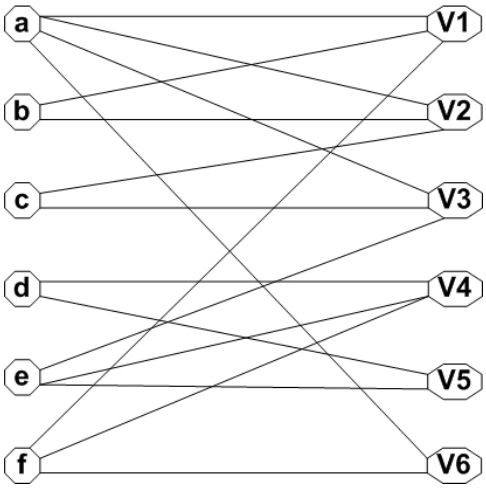

Abbildung 5.1: Strukturdigraph

Nach der Heuristik von Cellier wird die Gleichung a mit meisten schwarzen Linien für Tearing ausgewählt. Die Variablen V_1, V_2 und V_3 besitzen gleiche Anzahl von schwarzen Linien. Sie werden jeweils als Tearing Variable gewählt und getestet. Nach Färbealgorithmus bleiben die Variablen V_4 und V_5 immer nicht kausalisiert 5.2. Wir brauchen Tearing Verfahren nochmal zu starten.

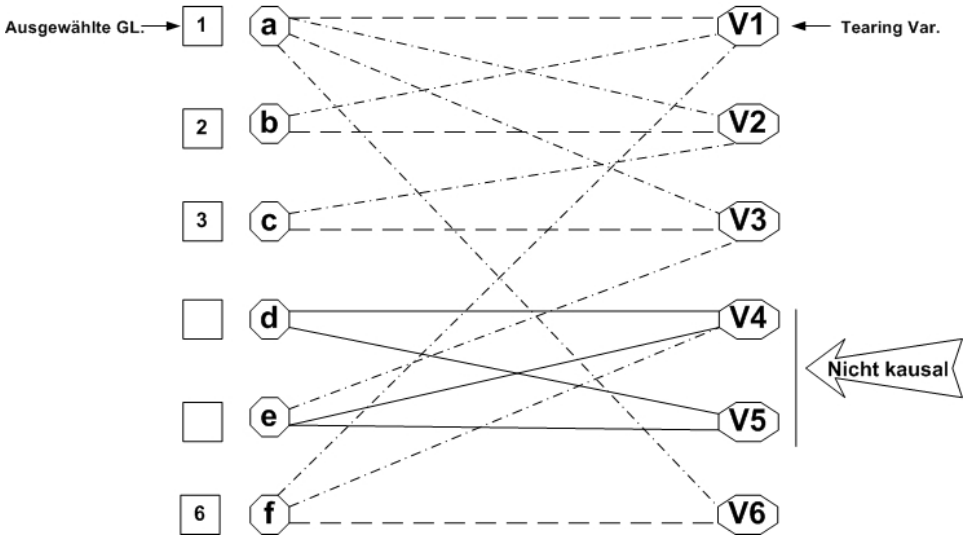

Abbildung 5.2: Färben für Tearing Variable V1

Wir können sehen, dass wir mit Cellier Heuristik mindestens zweimal Tearing starten sollen. Gibt es eine andere Wahl, damit wenigere Tearing Variablen gebraucht werden?

Von der Abbildung 5.3 und 5.4 können wir festlegen, dass nur eine Tearing Variable(z.B. V_4 aus Gleichung f oder $V3$ aus Gleichung e) für dieses Problem benutzt werden kann. Daraus folgt die folgende Überlegungen:

- Die Variablen mit wenigem Vorkommen kann nur durch einige „bestimmte“ Gleichungen aufgelöst werden. Das bedeutet, dass es nur wenige Möglichkeiten gibt, solche Variablen zu einer Gleichung zuzuordnen. Die Heuristik von Cellier nimmt immer die Gleichung bzw. Variablen mit meisten Vorkommen. Deshalb wird die Chance für das Auflösen einer Variable mit wenigem Vorkommen nach Tearing immer kleiner. Können wir diese Variablen zuerst betrachten?

- In Abbildung 5.3 und 5.4 werden zwei verschiedene Tearing Variablen aus zwei verschiedenen Gleichungen ausgewählt. Nach Färbealgorithmus ist die Gleichung, zu der eine bestimmte Variable zugeordnet ist, bleibt nicht geändert. Wir können schätzen, dass die Variablenzuordnung eines Gleichungssystems für das optimale Tearing statisch bestimmt sein kann.

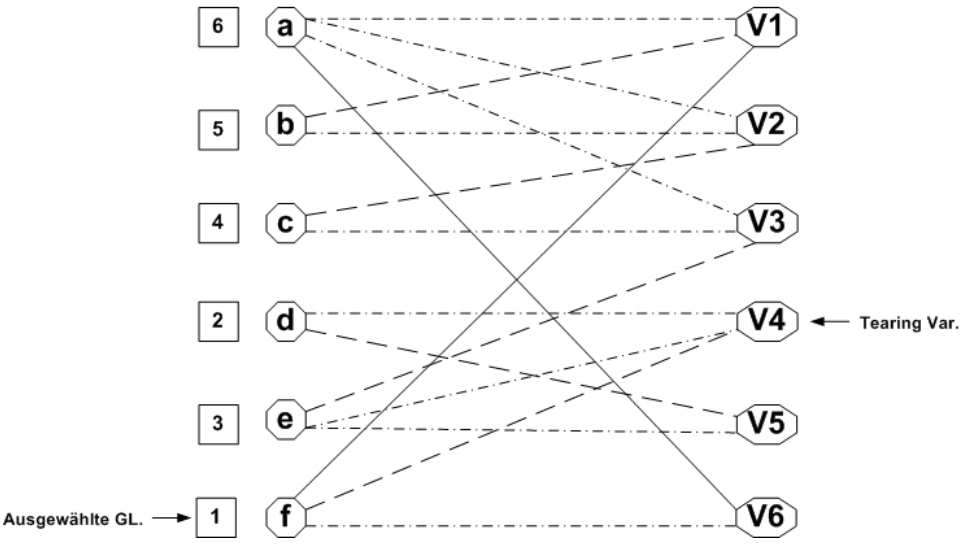

Abbildung 5.3: Neue Wahl von Tearing Variable(1)

Aus oberen Überlegungen folgt die folgende Heuristik:

1. Finde die Variablen, die am wenigsten vorkommen, heraus. Sie sind „schwer“ aufzulösen.

2. Finde die Gleichungen heraus, die schwer zu lösende Variablen enthalten. Und wähle die Variable v_i(es kann mehr als eine solche Variable

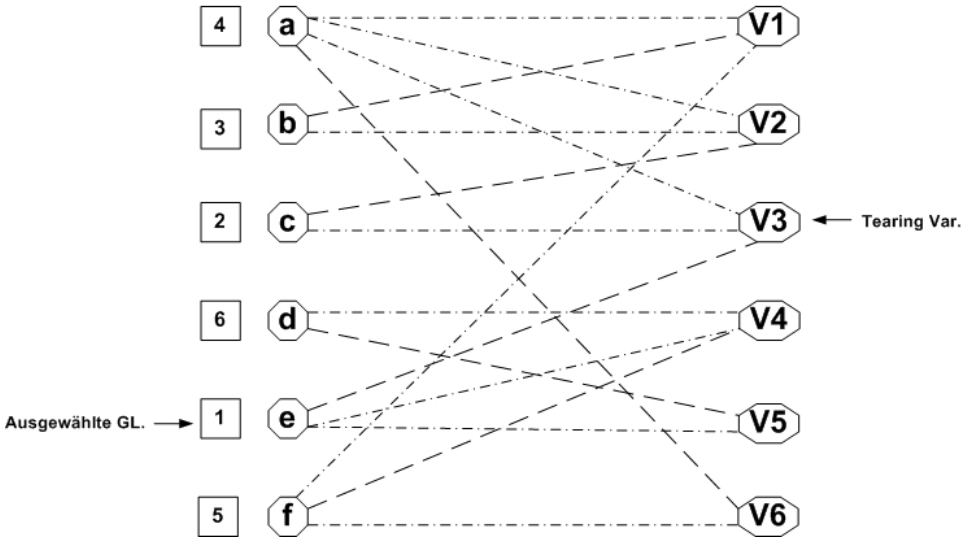

Abbildung 5.4: Neue Wahl von Tearing Varianble (2)

geben), die am häufigsten in diesen Gleichungen vorkommt, als Tearing Variable.

3. Teste alle Gleichungen $f_1 ... f_n$ mit v_i als Tearing Variable und $f_1 ... f_n$ als Ansgangsgleichung. Die Gleichung, durch die meisten anderen Gleichungen gelöst werden kann, wird als Ausgangsgleichung ausgewählt.

5.1.2 Vergleichung mit der alten Heuristik

Die neue Heuristik hat folgende Unterschiede im Vergleich zu der Heuristik von Cellier:

- Die Cellier Heuristik bevorzugt immer die Gleichung mit großer Auswahl von Variablen. Es ist angenommen, wenn diese Gleichung für Tearing ausgewählt worden ist, bekommen wir eine gute Gelegenheit, möglichst viele weitere Gleichungen aus System aufzulösen. In der oberen Heuristik betrachten wir zuerst die Variablen, die am wenigsten im System vorkommen, damit die Gleichungen, die solche Variablen enthalten, nicht alle zuvor zugeordnet sind.

- In Cellier Heuristik wird zuerst eine Gleichung, dann eine Variable aus dieser Gleichung für Tearing ausgewählt. Die neue Heuristik legt schon im ersten Schritt fest, welche Variable Tearing Variable ist. Erst danach wird weiter untersucht, welche Gleichung, die diese Variable enthält, soll für Tearing eingesetzt werden.

- Die neue Heuristik braucht in vielen Fällen mehr Aufwände für das

Finden einer optimalen Lösung für Tearing als die alte. Sie kann zwar bessre Lösung finden, aber auch längere Laufzeit dabei gebraucht wird.

Die beiden Heuristiken basieren auf gleiche Kenntnisse. Das Problem von Cellier Verfahren für Tearing, wie in Kapitel 4 erklärt ist, kann durch die neue Heuristik auch nicht gelöst werden. Wir brauchen noch weitere Maßnahmen dagegen ergreifen.

5.2 Modifikation von Ollero-Amselem Algorithmus

Damit wir eine vernünftige Sortierung nach Tearing bekommen, sollen wir den *Ollero-Amselem Algorithmus* ein bisschen modifizieren. Sei Graph G mit n Knoten gegeben. Setze G' = G.

1. Sei ES = 1, ES spezifiziert den Eingangs- oder Ausgangsgrad eines Knoten im Graph *G*.
2. Wähle ein Knoten v aus G aus.
   (a) Lösche v aus G und gehe zu Schritt 3, wenn v ohne Ausgang ist. Der Knoten bekommt eine Nummerierung von 1 bis n.
   (b) Lösche v aus G und gehe zu Schritt 3, wenn v ohne Eingang ist. Der Knoten bekommt eine Nummerierung von n bis 1.
   (c) Setze G' = G, wenn kein Knoten im letzten zwei Schritten(2a und 2b) gefunden ist.
   (d) Setze G = G' und v in die *tear-set*, lösche v aus G und gehe zu Schritt 2a, wenn v eine *self-Loop* enthält. Der Knoten bekommt eine Nummerierung von 1 bis n.
   (e) Lösche v und verkürze die Kanten aus G, wenn die Eingangs- oder Ausgangsgrad von v gleich ES ist. Sonst gehe zu Schritt 4.
3. Algorithmus terminiert, wenn G keine Knoten mehr hat.
4. Zurück zu Schritt 2, bis alle Knoten für den aktuellen Wert von ES getestet sind.
5. Wenn mindestens einen Knoten von Schritt 2d gelöscht ist, wird ES auf 1 zurückgesetzt, sonst wird es um 1 erhöht und der Algorithmus wird auf Schritt 2 zurückgesetzt.

Der modifizierte Algorithmus sortiert zuerst die Gleichung ohne Ein- oder Ausgang(2a und 2b). Wenn eine Tearing Variable gefunden ist(2d), wird sie gelöscht, dann wird es unter der gebliebenen Gleichungen nach dadurch kausalisierbaren Gleichungen gesucht.

5.3 Heuristik für Variablenzuordnung

Tearing Algorithmus wie Ollero-Amselem Algorithmus legt schon am Anfang die Variablenzuordnug fest. Die Zuordnung bestimmt die Abhängigkeit zwischen den Gleichungen. Die anschließende BLT-Zerlegung und Tearing basieren auch darauf. Das bedeutet, dass die Anzahl der Blöcke und Tearing Variablen auch davon beeinflusst sind. Im letzten Kapitel haben wir zwei Algorithmen für die Zuordnung gesehen. Die beiden basieren auf normale Tiefensuche. Es gibt keine Bewertung für das Auswählen von Gleichung und Variablen. Deshalb können wir hier passende Heuristik in die Suche einbauen, damit die Variablen nicht mehr „blind“ zugeordnet werden.

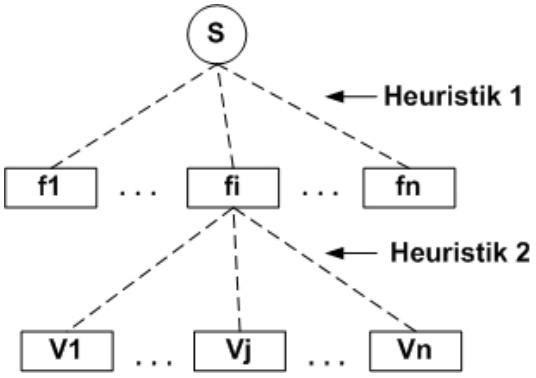

Abbildung 5.5: Heuristische Suche

Die Abbildung 5.5 zeigt, wo wir Heuristik brauchen. Wir brauchen zuerst mit *Heuristik 1* zu bestimmen, welche Gleichung wir als erste betrachten. Wenn wir eine Gleichung i ausgewählt haben, sollen wir mit *Heuristik 2* aus den Variablen in Gleichung i eine Variable j finden, der zu dieser Gleichung zugeordnet. Wir können einfache Strategie wie bei Cellier Verfahren benutzen, z.B. die Gleichung mit meisten bzw. wenigsten Variablen, die Variablen mit meisten bzw. wenigsten Vorkommen im System. Solche einfache Strategie bringt schon bessere Lösung als früher. Man sollte dabei beachten, dass mehrere Backtracking bei der Zuordnung gebraucht werden können, wenn man Heuristik einbaut.

Wir betrachten nochmal das Beispiel aus dem letzten Kapitel.

$$f_a(V_1, V_2, V_3, V_6) = 0, f_b(V_1, V_2) = 0$$

$$f_c(V_2, V_3) = 0, f_d(V_4, V_5) = 0$$

$$f_e(V_2, V_4, V_5) = 0, f_f(V_2, V_4, V_6) = 0$$

Diesmal ordnen wir zuerst die Variablen bzw. Gleichungen mit minimalem Vorkommen zu und bekommen das folgende Ergebnis. Die Abbildung 5.6

Variable	Gleichung
V_1	f_b
V_2	f_e
V_3	f_c
V_4	f_f
V_5	f_d
V_6	f_a

Tabelle 5.1: Zuordnung mit Heuristik

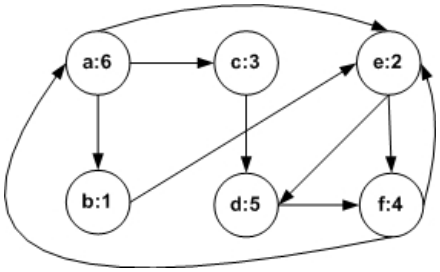

Abbildung 5.6: Neuer Strukturdigraph

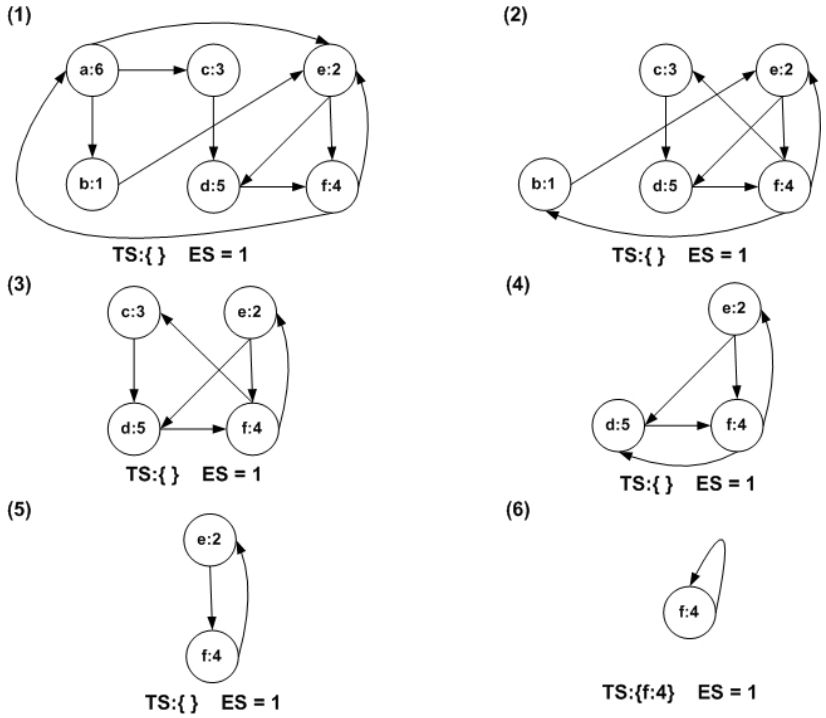

Abbildung 5.7: Ollero-Amselem Algorithmus

zeigt den neuen Strukturdigraph(im Vergleich zu Abbildung 4.5).Nach dem Algorithmus bekommen wir diesmal statt zwei Tearing Variablen nur eine(im Vergleich zu Abbildung 4.1). Wenn wir eine Bewertungsfunktion g und eine zulässige Schätzungsfunktion h für den Zwischenschritt während der Zuordnung finden, können wir die beiden kombinieren und eine optimale Lösung für die Zuordnung finden. Das ist die so genannte A* Suchstrategie. Die Funktion g bewertet alle durchgeführten Zuordnungen und die Funktion h schätzt, wie gut eine weitere Zuordnung ist. Eine zulässige heuristische Funktion ist noch nicht gefunden.

Kapitel 6

Implementierung und Testen

Für die Vergleichung von Tearing Methoden werden die beiden im Kapitel 4 dargestellten Algorithmen und im letzten Kapitel eingeführten neue Ideen jeweils mit C-Sprache implementiert. In diesem Kapitel werden wir die Implementierung und ein paar Testergebnisse betrachten.

6.1 Implementierung

6.1.1 Eingabeform

Für den Testen brauchen wir eine Format für Gleichungssysteme. Wir sollen für jedes zu testende Gleichungssystem die entsprechende Strukturinzidenzmatrix bilden, wie in Kapitel 3 eingeführt ist. Deshalb ist die eine eindeutige Form für Matrixdarstellung erforderlich. Hier benutzen wir ein einfaches Mechanismus, nämlich die *Matrix Market exchange* Format (BPR96). Sie definiert eine ASCII-Format Datei für eine beliebige Matrix, die sehr einfach zu repräsentiert oder in andere Form umgewandelt werden kann. Sie ist die primäre Format von *Matrix Market*(http://math.nist.gov/MatrixMarket/), eine visuelle Datenbank für Matrix im Internet.

Als Beispiel sehen wir eine 3x3 Matrix:

$$\begin{pmatrix} 2 & 0 & 1 \\ 0 & 0 & 2 \\ 1 & 3 & 5 \end{pmatrix}$$

Wir können sie in MM Format wie folgend darstellen:

```
3 3 6

1 1 2
3 1 1
3 2 3
```

```
1 3 1
2 3 2
3 3 5
```

Die erste Zeile *3 3 6* erklärt, dass eine 3x3 Matrix hier beschrieben wird, und sie besitzt 6 Einträge nicht gleich 0. Die weitere Zeile beschreibt den einzelnen Eintrag in der Matrix. Die Zeile *3 2 3* bedeutet, dass eine 3 auf Position [3 2] steht. Weil wir für unsere Aufgabe nur Strukturinzidenzmatrix brauchen, sind die Einträge immer 0 oder 1. Deshalb können wir hier auch nur die Position aufschreiben, wo eine 1 steht. Die Einträge sind nach Spalten sortiert.

6.1.2 Datenstruktur

Für die Implementierung ist die Datenstruktur sehr wichtig. Die C-Sprache bietet mit der Zeiger-Funktion eine gute Möglichkeit dafür. Wir brauchen am wichtigsten eine gute Datenstruktur für dünnbesetzte Matrix(sparse matrix), weil die zwei Tearing Verfahren mehrmals Matrix benutzt, z.B. Strukturinzidenzmatrix, Adjazentmatrix für Strukturdigraph... Diese Matrix ist meistens dünnbesetzt. Die Einträge in einer dünnbesetzten Matrix sind meistens Null. Wenn wir wie bei normaler Matrix alle Einträge speichern, werden viele Speicherplätze vergeudet, auch die Laufzeit wird dann verschlechtert. Deshalb ist hier ein normales zweidimensionales Array kein guter Kandidat. Wir brauchen eine praktische Lösung dafür.

Aus dem vorne erklärten Grund sollten wir für eine dünnbesetzte Matrix nur die Einträge speichern, die ungleich Null sind. Für Speichern eines Matrixeintrags brauchen wir nicht nur den Inhalt aufschreiben, auch die Position des Eintrags sollten wir nicht vergessen, nämlich die Zeilen- und Spaltennummer. Die Suche in einer Matrix ist besonders wichtig für unsere Aufgabe. Eine Suche in einer Matrix kann nach Reihenfolge von Zeilen oder Spalten durchgeführt werden. Damit ist die Position des ersten Eintrags in einer Zeile bzw. Spalte eine sehr nützliche Information. Die Position hier meint die Nummerierung in allen Matrixeinträgen, wenn wir sie zeilen- bzw. spaltenweise sortiert haben. Übrigens sollten wir eine dünnbesetzte Matrix zweimal speichern, einmal zeilenweise und einmal spaltenweise, damit die Suche in einer Matrix wesentlich erleichtert.

Als Beispiel bauen wir eine dünnbesetzte Matrix *mat* mit n Gleichungen, n Variablen und m Einträge. Sie ist ein Array mit acht Elementen. Die ersten vier Array-Elemente $mat[0] \dots mat[3]$ nutzen wir für das spaltenweise Speichern der Matrix, die letzten vier $mat[4] \dots mat[7]$ für das zeilenweise Speichern. In Abbildung 6.1 wird gezeigt, wie eine dünnbesetzte Matrix spaltenweise gespeichert wird. Die Elemente in $mat[0]$ sind die Zeilennummer von Matrixeinträgen, die in $mat[1]$ sind die Spaltennummer. $mat[2]$

speichert den Inhalt von Einträgen aus entsprechender Position in Matrix. Alle Einträge hier sind nach Spalten sortiert. Die Anzahl der Elemente in $mat[0] \dots mat[2]$ sind gleich die Anzahl der Matrixeinträge. $mat[3]$ speichert die Position des ersten Eintrags von jeder Spalte $P_1 \dots P_n$ und enthält n Elemente. Das zeilenweise Speichern ist analog. Nur die Sortierung von Einträgen ist nach Zeilen durchgeführt.

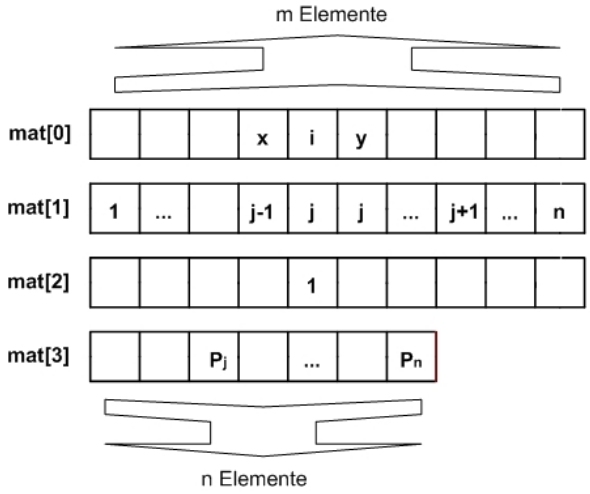

Abbildung 6.1: spaltenweise Speichern von dünnbesetzte Matrix

Für die Matrix aus dem letzten Abschnitt wird folgendes Array erzeugt.

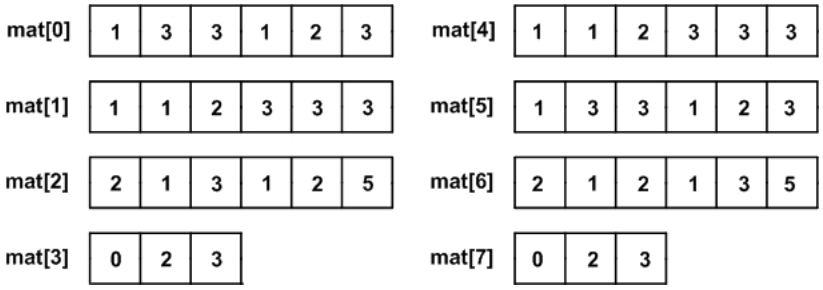

Abbildung 6.2: Verwaltung von Matrix

6.2 Testen

In diesem Abschnitt werden wir Testen von Tearing Algorithmen für einige große Probleme durchführen. Für das Testen benutzen wir Matrix aus *Matrix Market* als Strukturinzidenzmatirx von Gleichungssystemen. Beim Testen werden wir die in den letzten Kapiteln eingeführten Tearing Algorithmen betrachten. Anhand der Anzahl der gebrauchten Tearing Variablen

und der Laufzeit können wir einen Algorithmus beurteilen.

6.2.1 Testumgebung

Für den Testen benutzen wir ein Computer mit Mobile Intel(R) Pentium(R) 4 - M CPU 2.00GHz und 512MB RAM. Das Betriebssystem ist Microsoft Windows XP Professional Version 2002 Service Pack 2. Das C Programm enthalten die Implementierung für verschiedene Tearing Verfahren, inkl. dabei gebrauchten Algorithmen für Variablenzuordnung, BLT-Erzeugung, usw... Als Eingabe bekommt das Programm eine Strukturinzidenzmatrix eines Gleichungssystems in *Matrix Market Form*. Nachdem man ein Tearing Verfahren ausgewählt hat, wird das System analysiert und ggf. mit Tearing aufgelöst. Die dabei benutzten Tearing Variablen, erzeugte BLT-Blöcke und die Variablenzuordnung wird ausgegeben. Das Programm wird in einem normalen DOS-Shell aufgerufen und das Testergebnis ausgeben.

6.2.2 Testergebnis

Eine Strukturinzidenzmatrix eines Gleichungssystems ist normalerweise eine dünnbesetzte Matrix. Deshalb brauchen wir zum Testen eine Reihe von Sparse Matrix. Wir nehmen von Matrix Market die *Original Harwell sparse matrix test collection* von der *Harwell-Boeing Collection* in 1978. Das ist eine Ansammlung dünnbesetzten Matrix, die zuerst von Curtis and Reid bei Harwell eingeführt und später von Duff weiterentwickelt. Es gibt in der Ansammlung 36 Matrizen kommen aus verschieden Bereichen. Wir brauchen nur die quadratischen Matrizen zu testen. Und die Matrix, die ein System bestimmt, das beim Auflösen kein Tearing braucht, wird auch nicht betrachtet. Da ist eine Tabelle für die Eigenschaft der zu den testenden Matrizen. Die Matrizen besitzen Dimension von 32 bis 822 und Besetzungsgrad von 0,563% bis 12,30%.

Beispiel	Bereich	Dimension	Einträge	Dichte
ibm32	1971 IBM conference advertisement	32x32	126	12,30%
curtis54	stiff biochemical ODEs	54x54	291	9,98%
will57	Jacobian of emitter follower current switch circuit	57x57	281	8,65%
gent113	statistical application	113x113	655	5,13%
arc130	Laser problem	130x130	1282	7,59%
will199	stress-analysis	199x199	701	1,77%
str__200	Simplex method basis matrix	363x363	3068	2,33%
str__400	Simplex method basis matrix	363x363	3157	2,40%
str__600	Simplex method basis matrix	363x363	3279	2,49%
fs_541_1	ODE for atmospheric pollution problem	541x541	4285	1,46%
bp___200	Simplex method basis matrix	822x822	3802	0,563%
bp___400	Simplex method basis matrix	822x822	4028	0,596%
bp___600	Simplex method basis matrix	822x822	4172	0,617%
bp__1000	Simplex method basis matrix	822x822	4661	0,690%
bp__1200	Simplex method basis matrix	822x822	4726	0,699%
bp__1400	Simplex method basis matrix	822x822	4790	0,709%
bp__1600	Simplex method basis matrix	822x822	4841	0,716%

Tabelle 6.1: Eigenschaft von Matrizen

Zuerst sehen wir das Ergebnis von BLT-Zerlegung für die einzelne Matrix. Für diesen Schritt benutzen wir den Algorithmus aus dem Kapitel 4. Aus der Tabellen können wir sehnen, dass einige Matrizen - z.B. curtis54, fs_541_1 - einen „große“ Block enthalten, in dem die meisten Zeilen stehen. Das ist keinen guten Ausgangspunkt für Tearing. Diesen Blöcken erhalten nach Tearing meistens geschachtelten Tearing Variablen. Das bedeutet, dass wir für Auflösen einer Tearing Variable Kenntnisse von einer anderen brauchen. Das macht die numerische Lösung(z.B. Newton Verfahren) dafür schwer. Und die Matrix wie bp___200 bekommt nach der Zerlegung eine maximale Blockgröße von 40. Das ist viel kleiner als die originale Dimension der Matrix 822. Das hat das weitere Tearing erleichtert, weil die Matrix in viele „kleinere“ Blöcke zerlegt worden ist.

Beispiel	Anzahl Blöcke	Max. Blockgrösse	Blöcke zum Tearing (Blockgröße mind. 2)
ibm32	7	124	1
curtis54	1	54	1
will57	1	57	1
gent113	18	96	1
arc130	7	124	1
will199	10	188	3
str__200	234	130	1
str__400	203	142	5
str__600	142	213	3
fs_541_1	2	540	1
bp___200	646	40	20
bp___400	591	161	21
bp___600	537	216	18
bp__1000	463	207	22
bp__1200	447	220	22
bp__1400	403	372	15
bp__1600	450	217	20

Tabelle 6.2: BLT Information

Für die Bewertung von Tearing Algorithmen sollen wir die Anzahl der gebrauchten Tearing Variablen für Auflösen des gesamten Systems berücksichtigt. Die unten stehenden Tabelle zeigt, dass verschiedene Tearing Algorithmen bzw. Heuristik unterschiedliche Anzahl von Tearing Variablen brauchen. T1 ist der originale Ollero-Amselem Algorithmus. T2 und T3 sind zwei Varianten von modifizierten Ollero-Amselem Algorithmen mit neuen Heuristiken von mir. T2 betrachtet beim Tearing immer zuerst Gleichungen bzw. Variablen mit wenigen Vorkommen und T3 immer die mit meisten Vorkommen. T4 ist die Tearing Algorithmus von Cellier. T5 steht für Cellier Verfahren

mit meiner neuen Heuristik. In der Tabelle sind die Matrizen nach ihrer Dimension von oben nach unten sortiert. Wir können sehen, dass alle Tearing Algorithmen bei kleinen Systemen(Dimension ≤ 200) fast immer gleiche Anzahl Tearing Variablen brauchen. T5 ist in manchen Fällen besser, z.B. bei curtis54 und will199. Der maximale Abstand ist 17 bei Beispiel will199. Bei den drei Beispielen mit Dimension 363 gibt es unter T1 - T5 fast keine Unterschiede. Das Beispiel fs_541_1 kommt aus einem ODE System. Wir können sehen, dass der originale Ollero-Amselem und Cellier Algorithmus immer ein „neutrale“ Ergebnis bringen. Die unterschiedlichen Heuristiken von T2 und T3 bestimmen, dass T2 der schlechte und T3 die beste Auswahl für dieses Problem ist. T3 braucht fast 80 Tearing Variablen weniger als T2. Das ist schon ein großer Vorteil. Bei den 7 Beispielen gibt es keinen klaren Sieger. Das Cellier Verfahren (T4 und T5) brauchen meistens mehr Tearing Variablen als die anderen. Aber der Unterschied ist nicht groß.

Beispiel	Dimension	T1	T2	T3	T4	T5
ibm32	32	7	8	8	6	7
curtis54	54	32	32	30	31	27
will57	57	32	31	24	24	24
gent113	113	16	14	17	14	14
arc130	130	14	14	13	14	13
will199	199	38	38	37	26	21
str__200	363	36	36	34	35	49
str__400	363	45	44	46	45	51
str__600	363	66	65	66	64	67
fs_541_1	541	297	324	243	285	305
bp___200	822	48	50	46	47	50
bp___400	822	73	74	74	74	80
bp___600	822	92	90	91	90	99
bp__1000	822	122	118	123	130	120
bp__1200	822	130	131	131	138	133
bp__1400	822	132	127	132	140	134
bp__1600	822	125	130	127	134	140

Tabelle 6.3: Anzahl Tearing Variablen

Die Laufzeit eines Tearing Algorithmus spielt auch eine wichtige Rolle. Bei den kleinen Beispielen(Dimension $\leq$ 200) gibt es keinen großen Unterschied zwischen den Laufzeiten der verschiedenen Verfahren. Die Cellier Verfahren(T4 und T5) sind immer ein bisschen langsamer als die anderen. Wie im letzten Kapitel beschrieben ist, dass die neue Heuristik(T5) mehr berechnet, deshalb braucht sie längere Zeit als die originale(T4). Das originale Ollero-Amselem Algorithmus(T1) ist am schnellsten hier. Wenn man Heuristik darauf anbaut(T2, T3), dauert das Tearing auch länger.

Beispiel	T1	T2	T3	T4	T5
ibm32	0,001	0,010	0,010	0,001	0,029
curtis54	0,001	0,010	0,001	0,010	0,049
will57	0,011	0,001	0,001	0,018	0,029
gent113	0,011	0,010	0,001	0,001	0,039
arc130	0,010	0,033	0,030	0,015	0,239
will199	0,022	0,030	0,050	0,072	0,069
str__200	0,111	0,140	0,100	0,217	0,569
str__400	0,178	0,210	0,180	0,435	0,809
str__600	0,412	0,451	0,512	0,344	1,508
fs_541_1	0,256	0,351	1,515	72,333	22,546
bp___200	0,099	0,245	0,150	0,250	0,719
bp___400	0,229	0,051	0,291	0,471	1,128
bp___600	0,369	0,492	0,381	0,783	1,498
bp__1000	1,018	1,326	1,034	1,264	74,051
bp__1200	1,028	1,092	1,375	1,204	62,414
bp__1400	1,158	1,381	1,094	1,144	69,896
bp__1600	1,038	1,181	0,923	1,224	80,595

Tabelle 6.4: Laufzeit von Algorithmen

Kapitel 7

Zusammenfassung und Ausblick

In dieser Arbeit wurde die Tearing Technik für das effiziente Lösen von DAE-Systemen untersucht. Unser Ziel ist, eine möglichst optimale Lösung für Tearing zu finden. Das bedeutet, ein DAE-System mit möglichst wenige Tearing Variablen zu lösen. Deshalb haben wir einige bekannte Tearing Algorithmen betrachtet und analysiert, nämlich der Steward Algorithmus, der Cellier Algorithmus und der Ollero-Amselem Algorithmus. Der Steward Algorithmus basiert auf reine Umsortierung von Strukturinzidenzmatrix und besitzt eigene BLT-Zerlegung. Wir brauchen nach Tearing alle algebraischen Schleifen zu finden und sie mit passende Tearing Variablen zu brechen. Der Cellier Algorithmus basiert auf Strukturdigraph eines Gleichungssystems. Wir verwenden zuerst den Färbealgorithmus, bis keine Gleichung bzw. Variable gefärbt werden kann. Danach werden Tearing Variabeln durch eine Heuristik ausgewählt und als bekannt angenommen, damit der Färbealgoruthmus im gesamten System durchgeführt werden kann. Der Ollero-Amselem Algorithmus ist auch ein graphentheoretisches Verfahren. Wir bilden erst das Strukturdiagramm anhand der Variablenzuordnung. Mit Ollero-Amselem Algorithmus können wir ein Essential-Set von diesem Graph finden, dadurch alle Schleifen gebrochen werden. Die Knoten in Essential-Set sind genau die Tearing Variablen.

Die verschiedenen Tearing Algorithmen besitzen jeweils Vor- und Nachteile. Es gibt keinen klaren Sieger unter ihnen. Deshalb wurden im Rahmen dieser Arbeit auf Basis existierender Heuristiken neue entwickelt, mit dem Ziel, das minimale Tear-Set zu finden. Durch anschließenden Testen von verschiedenen Tearing Verfahren können wir einpaar Fazit bekommen. Der Cellier Algorithmus ist in manchen Beispielen viel langsamer als die andere. Der Ollero-Amselem Algorithmus hat immer das beste Laufzeitverhalten, auch wenn wir Heuristik in Variablenzuordnung eingefügt. Eine gute Heuristik

kann das Tearing Ergebniss wesentlich verbessern.

Die Testergebnisse haben gezeigt, dass die neue Heuristik für den Cellier Algorithmus für manche Probleme eine längere Laufzeit als die alte besitzt, da sie in vielen Fällen, abhängig vom zu Grunde liegenden Gleichungssystem, mehr als die alte Heuristik untersucht. Daraus ergibt sich die Überlegung, die alte und die neue Heuristik zu kombinieren und somit eine neue Strategie zu erzeugen. Die neue Heuristik betrachtet immer zuerst die Variablen mit der geringsten Anzahl an Vorkommen. Nach mehreren Iterationen des Färbealgorithmus ergeben sich viele Variablen mit gleichem Vorkommen. Deshalb werden sehr viele Gleichungen bzw. Variablen für Tearing getestet. Der weitere Einsatz dieser modifizierten Heuristik des Cellier Algorithmus verursacht die lange Laufzeit. Wir können jedoch ab einem „bestimmten“ Punkt die alte Heuristik verwenden. Die Bestimmung dieses Zeitpunkts, das sogenannte Fixpunkt Problem, kann in einer zukünktigen Arbeit untersucht werden.

Für den Steward Algorithmus brauchen wir zuerst alle Schleifen aus einem Strukturdiagramm raus zu finden und als Baumstruktur darstellen. Danach wird noch zusätzliche Strategie gebraucht, um „gute“ Tearing Variablen zu finden. Das ist schon ähnlich wie das Finden von Essential-Set bei dem Ollero-Amselem Algorithmus. Wir können auch die Heuristik davon benutzen.

Der Ollero-Amselem Algorithmus findet eine Essential-Set aus einem gerichteten Graph. Aber es ist nicht bewiesen, dass er immer die minimale Essential-Set findet. Das Problem, minimale Essential-Set zu finden, ist NP-vollständig. Die Variablenzuordnung eines DAE-Systems bestimmt die Abhängigkeit zwischen den Gleichungen. Das Strukturdiagramm basiert auf solche Abhängigkeit. Wir könnten die Tiefensuche für Variablenzuordnung so durchführen, damit sie nicht mehr „ziellos“ ist, wenn wir statt einer normalen Tiefensuche eine A^*-Suche durchführen. Wir könnten entsprechende Bewertung- und Schätzungsfunktionen einbauen. Ein Zwischenschritt in Zuordnung wird zuerst durch die Bewertungsfunktion analysiert, wie gut er für weitere Arbeit wäre. Durch die Schätzungsfunktion wird dann einen Vorschlag gemacht, wie wir die Variablen weiter zuordnen. Wegen der NP-Vollständigkeit können wir nur eine Approximation von der optimalen Lösung nehmen. Aber wenn diese Funktionen genug gut gebildet werden, könnte die Leistung des Ollero-Amselem Algorithmus noch verbessert werden.

Literaturverzeichnis

[BC96] Borutzky, W. ; Cellier, F. E.: Tearing bound graphs with dependent storage elements. In: *CESA '96, IMACS Multiconference*, 1996

[BF03] Bunus, P. ; Fritzson, P.: Semi-Automatic Fault Localization and Behavior Verification for Physical System Simulation Models. In: *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference*, 2003

[BPR96] Boisvert, Ronald F. ; Pozo, Roldan ; Remington, Karin A. ; Applied and Computational Mathematics Division (Hrsg.): *The Matrix Market Exchange Formats: Initial Design.* National Institute of Standards and Technology, Gaithersburg, MD 20899 USA: Applied and Computational Mathematics Division, 1996

[CG97] Carpanzano, E. ; Girelli, R.: The Tearing Problem: Definition, Algorithm and Application to Generate Efficient Computational Code from DAE Systems. In: *Proceedings of 2nd Mathmod Vienna, IMACS Symposium on Mathematical Modelling, Wien*, 1997

[CK06] Cellier, F. E. ; Kofman, E.: *Continuous System Simulation.* Springer Verlag, 2006. – ISBN 0–387–26102–8

[Deu06] Deuflhard, P.: *Newton Methods for Nonlinear Problems - Affine Invariance and Adaptive Algorithms.* Sringer Verlag, 2006

[Elm04] Elmqvist, H. ; DynaSim AB (Hrsg.): *Dymola - Dynamic Modeling Language, User's Manual, Version 6.0.* Research Park Ideon, Lund, Sweden: DynaSim AB, 2004

[EO94] Elmqvist, H. ; Otter, M.: Methods for Tearing Systems of Equations in Object-Oriented Modeling. In: *Proceedings ESM'94 European Simulation Multiconference*, 1994

[FFS06] Fritzsche, D. ; Frommer, A. ; Szyld, D.: Extensions of Certain Graph-based Algorithms for Preconditioning. (2006)

[Har63] Harrison, B. K.: A Discussion of Some Mathematical Techniques Used in Kron's Method of Tearing. In: *Journal of the Society for Industrial and Applied Mathematics* 11 (1963)

[Jus05] Juslin, K.: *A Companion Model Approach to Modelling and Simulation of Industrial Processes.* VTT Publications 574, 2005. – ISBN 0951–38–6660–2

[Kar72] Karp, R. M.: Reducibility among Combinatorial Problems. In: *Complexity of Computer Computations* (1972)

[Kro62] Kron, G.: Diakoptics - The Piecewise Solution of Large-Scale Systems. In: *MacDonald Co., London* (1962)

[Mah90] Mah, Richard S. H.: *Chemical Process Structures and Information Flows.* Butterworth Publishing, London, United Kingdom, 1990

[MEC95] M., Otter ; Elmqvist, H. ; Cellier, F.E.: Relaxing: A symbolic sparse matrix method exploiting the model structure in generating efficient simulation code. In: *Proc. Symp. Modelling, Analysis, and Simulation, CESA'96, IMACS MultiConference on Computational Engineering in Systems Applications, Lille, France, vol.1*, 1995

[OA83] Ollero, P. ; Amselem, C.: Decomposition algorithm for chemical process simulation. In: *Chem. Eng. Res. Des.* 61 (1983)

[OB99a] Otter, M. ; Bachmann, B.: Objektorientierte Modellierung Physikalischer Systeme, Teil 5: Singuläre Systeme. In: *Automatisierungstechnik* 47 (1999)

[OB99b] Otter, M. ; Bachmann, B.: Objektorientierte Modellierung Physikalischer Systeme, Teil 6: Strukturell inkonsistente DAEs. In: *Automatisierungstechnik* 47 (1999)

[Ott99] Otter, M.: Objektorientierte Modellierung Physikalischer Systeme, Teil 4: Transformationsalgorithmen. In: *Automatisierungstechnik* 47 (1999)

[Pan88] Pantelides, C.: The Consistent Initialization of Differential-Algebraic Systems. In: *SIAM Journal of Scientific and statistical Computing* (1988)

[Rot55] Roth, J. P.: The Validity of Kron's Method of Tearing. In: *Proc Natl Acad Sci U S A* (1955)

[Rub62] Rubin, D. I.: Generalized material balance. In: *CEP Symp. Ser. 58(37)* (1962)

[Sch03] Schulz, S.: Four Lectures on Differential-Algebraic Equations, Humboldt Universität zu Berlin. (2003)

[Ste65] Steward, D. V.: Partitioning and Tearing Systems of Equations. In: *Journal of the Society for Industrial and Applied Mathematics:Series B, Numerical Analysis* 2 (1965)

[SW62] Steward, D. V. ; Westerberg, A. W.: On an approach to techniques for the analysis of the structure of large systems of equations. In: *SIAM Rev.* 4 (1962)

[Tar72] Tarjan, R. E.: Depth First Search and Linear Graph Algorithms. In: *SIAM Journal of Computation* (1972)

[uC96] utzky, W. B. ; Cellier, F. E.: Tearing algebraic loops in bound graphs. In: *Trans. of SCS* 13 (1996)

Erklärung

Ich versichre, dass ich diese Arbeit selbstständig angefertigt habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Paderborn,12. August 2007

_____