Universität-Gesamthochschule Paderborn
Fachbereich 17 (Mathematik–Informatik)

Diplomarbeit

Graphenanalyse hydraulischer Schaltkreise zur Erkennung von hydraulischen Achsen und deren Kopplung

André Schulz

-

Paderborn, den 5. September 1997

Betreuer:
Prof. Dr. Hans Kleine Büning

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Paderborn, den 5. September 1997

Inhaltsverzeichnis

Kapitel 1

Überblick

1.1 Einleitung und Motivation

Mit der ständigen Weiterentwicklung in der technischen Welt wachsen auch die Anforderungen an Fachkräfte, die für die Arbeit in genau dieser Welt ausgebildet werden. Technische Anlagen werden durch den Fortschritt so komplex bzw. erfordern soviel Fachwissen, daß ein einzelner Ingenieur oft nicht in der Lage ist, kleine Fehler schnell zu diagnostizieren oder die Funktionalität von einem Teil einer solchen Anlage zu erkennen. Daher sind Werkzeuge, die diese Aufgaben unterstützen, eine große Hilfe, denn sie beschleunigen und vereinfachen viele wichtige Aufgaben wie Design, Simulation, Diagnose usw.

Sinn und Zweck dieser Arbeit ist, Methoden und Verfahren zu entwickeln, um eine solche komplexe Aufgabe aus dem Bereich „Design“ zu lösen, nämlich die Erkennung und Klassifikation von hydraulischen Achsen bzw. funktionellen Gruppen. Darüberhinaus möchten wir ein Verfahren zur Erkennung der Kopplung zwischen den hydraulischen Achsen einer Anlage angeben, um Information über mögliche Wechselwirkungen zwischen ihnen zu gewinnen.

Dadurch wird beispielsweise die Erkennung der Funktionsweise von bereits bestehenden Anlagen für jemanden, der die Anlage nicht kennt, einfacher und schneller. Ein anderes Einsatzgebiet liegt in der Didaktik: Auszubildende können mit Hilfe eines Systems, welches diese Verfahren anwendet, viel über Struktur und Verhalten von hydraulischen Anlagen lernen und sich dadurch den Einstieg in diesen Bereich des Maschinenbaus erheblich vereinfachen.

1.2 Nötige Vorbereitungen

Die im vorigen Abschnitt erwähnten Verfahren, die wir in dieser Arbeit entwickeln möchten, sind möglicherweise ohne weiteres nicht leicht zu verstehen, denn sie beruhen auf graphentheoretischen Überlegungen und den daraus abgeleiteten Algorithmen, und sie werden in einem speziellen Gebiet des Maschinenbaus eingesetzt, nämlich der Hydraulik. Also wird auf der einen Seite Wissen aus dem Bereich der Hydraulik notwendig sein und auf der anderen Seite Wissen aus der Graphentheorie – dabei ist der Anteil an graphentheoretischem Wissen erheblich umfangreicher als der Hydraulik-Anteil. Um daher den Einstieg in diese Thematik zu erleichtern, müssen wir zunächst einige Definitionen und Notationen aus den oben genannten Gebieten einführen, sowie die Vorgehensweise grob erklären.

1.2.1 Definitionen aus der Graphentheorie

Folgende Definitionen aus [15], [4] und [7] sind für die nächsten Kapitel von zentraler Bedeutung:

- Ein *Multigraph* G ist ein 3-Tupel (V, E, g), wobei V und E nichtleere, disjunkte, endliche Mengen sind, und $g : E \to 2^V$ eine Abbildung mit $2^V = \{ U \,|\, U \subseteq V, |U| = 2\}$ ist. V ist die Menge der Knoten, E ist die Menge der Kanten, und g ist die Inzidenzabbildung.

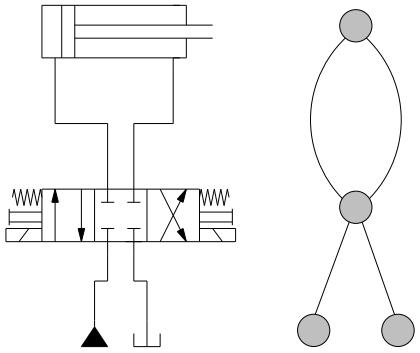

Abbildung 1.1: Eine hydraulische Schaltung und der zugehörige Multigraph

Informell beschrieben ist ein Multigraph ein Graph, der Knotenpaare besitzt, die durch mehrere Kanten verbunden sind. Diese Graphen sind für die Abstraktion des hydraulischen Schaltkreises gut geeignet, denn sie erlauben eine 1:1-Abbildung – in hydraulischen Anlagen können zwei Komponenten durch zwei verschiedene Kanten miteinander verbunden sein (siehe Abbildung 1.1).

- Ein zu einer hydraulischen Schaltung C gehörender hydraulischer Graph ist ein Multigraph $G_C = (V_C, E_C, g)$, dessen Elemente wie folgt definiert sind:

  1. V_C ist die Menge der Knoten, und es existiert eine bijektive Abbildung von der Menge der Nicht-Leitungen aus C in V_C.
  2. E_C ist die Menge der Kanten, und es existiert eine bijektive Abbildung von der Menge der Leitungen aus C in E_C.
  3. $g : E_C \to 2^{V_C}$ ist eine Funktion, die eine Kante $e \in E_C$ auf $(v_i, v_j) \in 2^{V_C}$ genau dann abbildet, wenn eine Leitung zwischen den mit v_i und v_j assoziierten Komponenten existiert und e mit dieser Leitung assoziert ist.

- Ein Graph $H = (V_H, E_H, g_H)$ heißt *Teilgraph* von $G = (V, E, g)$, falls $V_H \subseteq V$, $E_H \subseteq E$ und $g_H = g|E_H$. Ein Untergraph heißt *induzierter Untergraph* auf V_H, falls $E_H \subseteq E$ genau die Kanten beinhaltet, die mit den Knoten von V_H inzident sind. Für $T \subset V$ bezeichnet $G \setminus T$ den auf $V \setminus T$ induzierten Untergraphen. Abbildung 1.2 zeigt ein Beispiel für einen Graphen und einen induzierten Untergraphen.

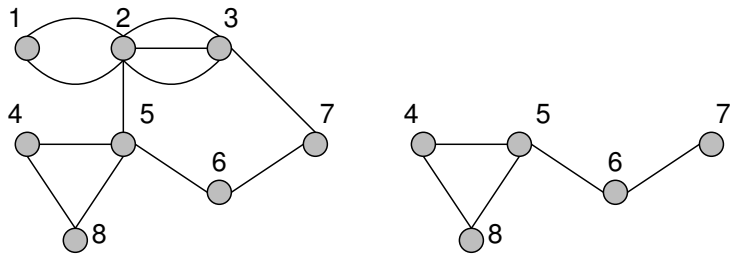

Abbildung 1.2: Ein Graph und induzierter Teilgraph auf $\{4, \ldots, 8\}$

- Ein Tupel $(e_1, \ldots, e_n)$ heißt *Kantenzug* („walk“), wenn $g(e_i) = \{v_{i_1}, v_i\}$, $v_i \in V$, $i = 1, \ldots, n$. Sind die v_i paarweise verschieden, dann heißt der Kantenzug *einfacher Weg* („path“). Abbildung 1.3 beschreibt zwei einfache Wege auf einem Graphen.

- Ein Graph G heißt *zusammenhängend*, wenn je zwei Knoten $v_i, v_j \in V$ verbindbar sind, d. h. es gibt einen Kantenzug mit Anfangsknoten v_i und Endknoten v_j. Falls G zusammenhängend ist, $G \setminus v$ aber nicht, so heißt v *Artikulationsknoten*. Falls G zusammenhängend ist, $G \setminus e$ für ein $e \in E$ aber nicht, so heißt e *Brücke* des Graphen. Die maximal zusammenhängenden Untergraphen von G heißen *Zusammenhangskomponenten*.

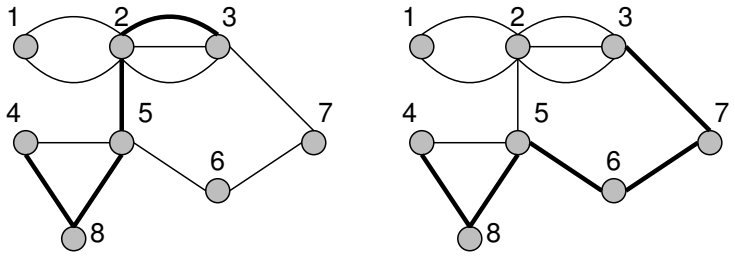

Abbildung 1.3: Einfache Wege auf einem Graphen

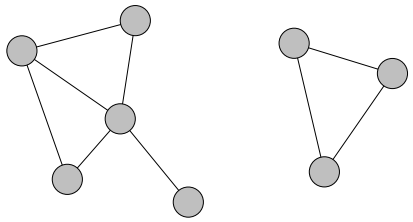

Abbildung 1.4: Ein Graph bestehend aus 2 Zusammenhangskomponenten

Besitzt ein Graph G keine Artikulationsknoten, so heißt er *zweifach zusammenhängend*, d. h. für jedes Knotenpaar existieren mindestens zwei disjunkte Pfade, die die Knoten miteinander verbinden. Ein zweifach zusammenhängender Teilgraph maximaler Größe eines Graphen heißt *Block*.

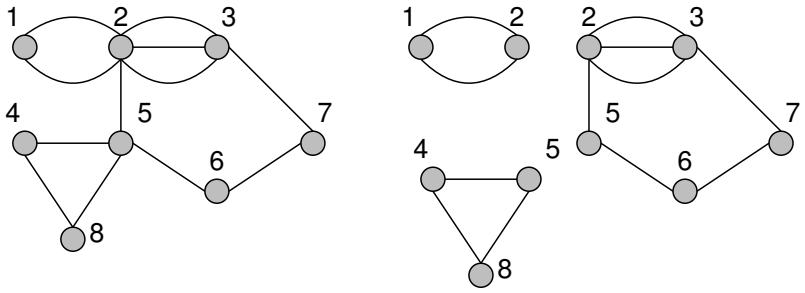

Abbildung 1.5: Ein Graph und seine Blöcke

- Für jeden Knoten $v \in V$ definiert man den *Grad* von v als die Anzahl der mit v inzidenten Kanten. Haben alle Knoten eines Graphen denselben Grad d, so ist der Graph *regulär* vom Grad d. Formelschreibweise: $deg(v)$.

Als weiterführende Literatur sei an dieser Stelle [3], [5] und [14] empfohlen, die zusätzliche Informationen sowie hierzu verwandte Algorithmen anbieten.

1.2.2 Definitionen aus dem Maschinenbau

Atomare Komponenten

Für die Untersuchung hydraulischer Anlagen benötigen wir eine Klassifizierung der dort auftretenden atomaren Komponenten – jede Komponente beinhaltet gewisse Informationen, besitzt Eigenschaften und erfüllt eine Aufgabe. Wir unterscheiden folgende Komponententypen, die in [15] eingeführt wurden:

- *Arbeitselemente* sind diejenige Elemente, die zur Kraftübertragung etwas beitragen. Hierzu zählen alle Arten von Zylindern und Motoren.
- *Kontrollelemente* sind für die Steuerung der Arbeitselemente zuständig, d. h. sie regeln den Fluß. Alle direktionalen Ventile gehören zu dieser Gruppe.
- *Versorgungselemente* liefern den nötigen Druck für alle anderen Elemente einer hydraulischen Schaltung. Diese Gruppe besteht nur aus Pumpen.
- *Hilfselemente* sind diejenige Elemente, die nicht in eine der obigen Klassen fallen, wie beispielsweise Speicher, Tanks, Schläuche usw.

Einige der Komponenten, die in den hydraulischen Schaltungen vorkommen können, können unter Umständen als Komponenten einer anderen Komponentenklasse fungieren. Dies ist dann der Fall, wenn solche Komponenten in einem bestimmten Kontext benutzt werden, wo sie dann eine andere Funktion erfüllen.

Hydraulische Achsen

Eine hydraulische Achse ist ein Teil einer Anlage, die eine bestimmte Funktion erfüllt. Sie besteht aus den oben beschriebenen atomaren Komponenten, wobei sie mindestens ein Arbeitselement und mindestens ein Versorgungselement besitzt. Verschiedene hydraulische Achsen können sich auch Komponenten miteinander teilen (siehe Abbildung 1.6).

Eine etwas allgemeinere Definition einer hydraulischen Achse liefert E. Vier in [17]:

> „Eine hydraulische Achse A repräsentiert und erfüllt gleichzeitig eine Teilfunktion f einer gesamten hydraulischen Anlage. A definiert die Verbindungen und das Zusammenspiel all jener Arbeits-, Kontroll- und Versorgungselemente, die zusammen der Funktion f entsprechen".

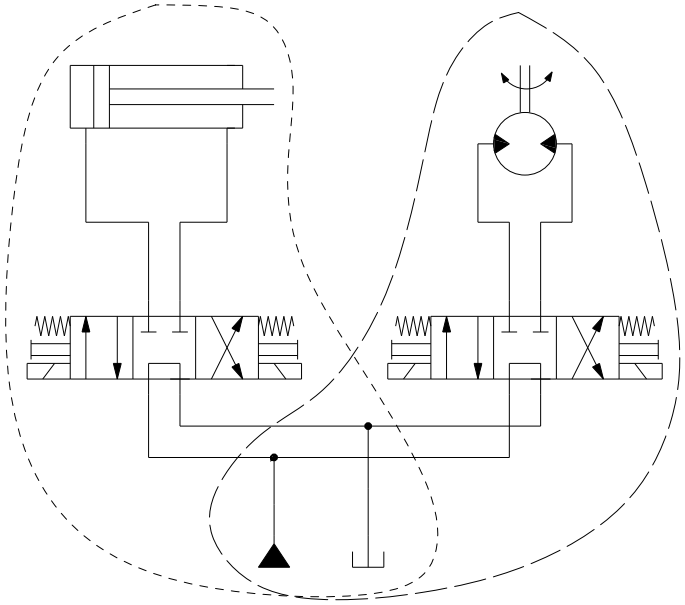

Abbildung 1.6: Achsen, die Komponenten miteinander teilen

Mit diesen Definitionen können wir eine hydraulische Anlage gemäß Abbildung 1.7 schematisch darstellen.

Wir möchten an dieser Stelle darauf hinweisen, daß eine gesamte hydraulische Anlage mindestens eine hydraulische Achse besitzen muß.

Kopplungsarten zwischen hydraulischen Achsen

Hydraulische Achsen bieten nur sehr grundlegende Funktionalitäten, so daß eine hydraulische Achse allein nicht in der Lage ist, eine schwierigere Aufgabe zu erfüllen. Erst wenn man mehrere, verschiedene Funktionalitäten bietende hydraulische Achsen miteinander verbindet, ist die Realisierung eines komplexeren Prozesses möglich.

Wir unterscheiden zwischen fünf verschiedenen Kopplungsarten, die in [15] definiert wurden. Sei dazu C eine hydraulische Schaltung, die zwei Teilschaltungen A und B enthält, wobei A und B zwei verschiedene hydraulische Achsen darstellen. Weiterhin seien $G_h(C) = (V_C, E_C, g_C)$, sowie $G_h(A) = (V_A, E_A, g_A)$ und $G_h(B) = (V_B, E_B, g_B)$ die zugehörigen hydraulischen Graphen.

Ebene 0 – keine Kopplung Ist $G_h(C)$ nicht zusammenhängend und sind $G_h(A)$ und $G_h(B)$ Untergraphen von verschiedenen Zusammenhangskomponenten aus G_h, so sind die hydraulischen Achsen A und B nicht gekoppelt.

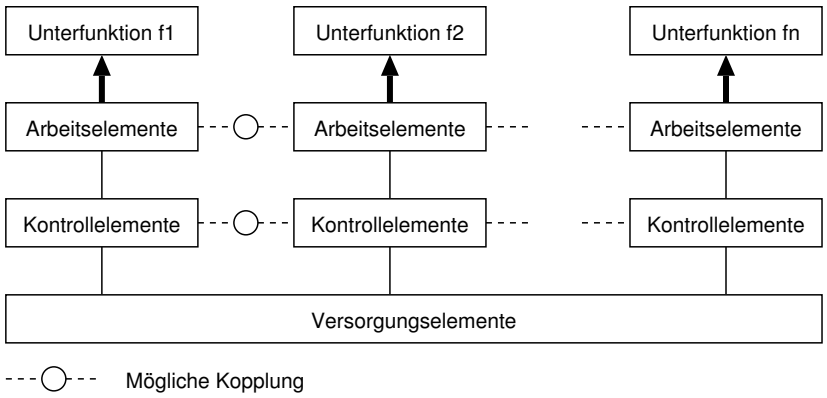

Abbildung 1.7: Abstrakte Sichtweise einer hydraulischen Anlage

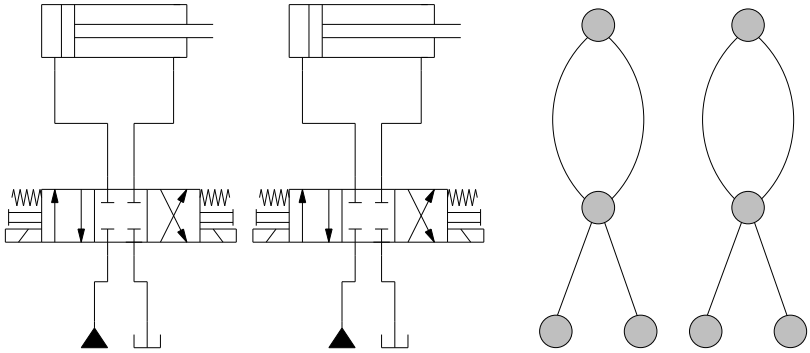

Abbildung 1.8: Nicht gekoppelte Achsen und der entsprechende abstrakte Graph

Zwei derartig gekoppelte Achsen besitzen also keinerlei physikalische Verbindung zueinander. Sie gehören daher zu unterschiedlichen Schaltungen, die getrennt untersucht werden können.

Ebene 1 – informationelle Kopplung Sei $\{e_1, \ldots, e_n\}$ die Menge der Kanten des Graphen G, deren Elemente jeweils mit einer Kontrollverbindung in der hydraulischen Schaltung C assoziert sind. Falls $G_{h'}(C) = (V_C, E_C - \{e_1, \ldots, e_n\}, g_C)$ nicht zusammenhängend ist und $G_h(A)$ und $G_h(B)$ zu unterschiedlichen Zusammenhangskomponenten in $G_{h'}$ gehören, so sind die hydraulischen Achsen A und B informationell gekoppelt.

Anzumerken wäre noch, daß die oben erwähnten Kontrollverbindungen auf unterschiedliche Art und Weise in hydraulischen Anlagen realisiert werden können. Die in Abbildung 1.9 dargestellten Kontrollverbindugen bestehen aus elektrischen Leitungen; solche Kontrollverbindungen können beispielsweise aber auch aus hydraulischen Leitungen bestehen.

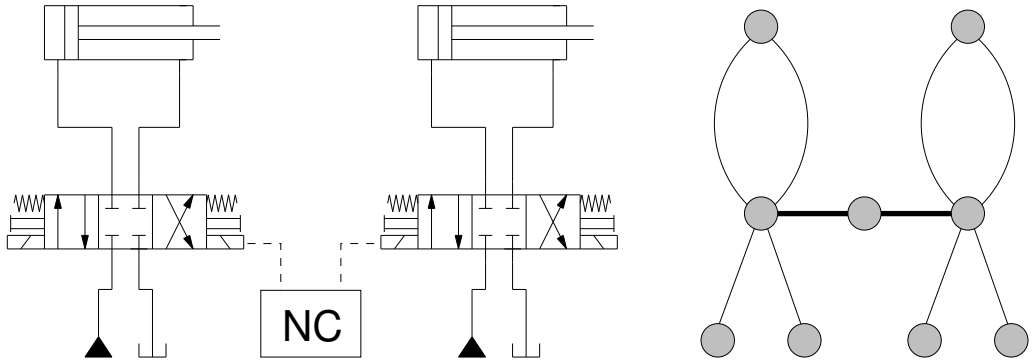

Abbildung 1.9: Informationell gekoppelte Achsen und der abstrakte Graph dazu

Ebene 2 – parallele Kopplung Sei $V_A^{'} := V_A - (V_A \cap V_B)$ und $V_B^{'} := V_B - (V_A \cap V_B)$ sowie $P_{w,s}$ die Menge aller Wege von einem Arbeitselement w zu dem Versorgungselement s, die über keine Kontrollkante gehen. Die hydraulischen Achsen A und B sind parallel gekoppelt, wenn $\exists v_X \in V_X^{'}, X \in \{A, B\}$ mit:

1. v_X ist mit einem Kontrollelement assoziert, und
2. $\forall p \in P_{w,s} \cap V_X^{'}$, $p = (v_1, \ldots, v_k)$, gilt: $\exists i \in \{1, \ldots, k\} : v_X = v_i$.

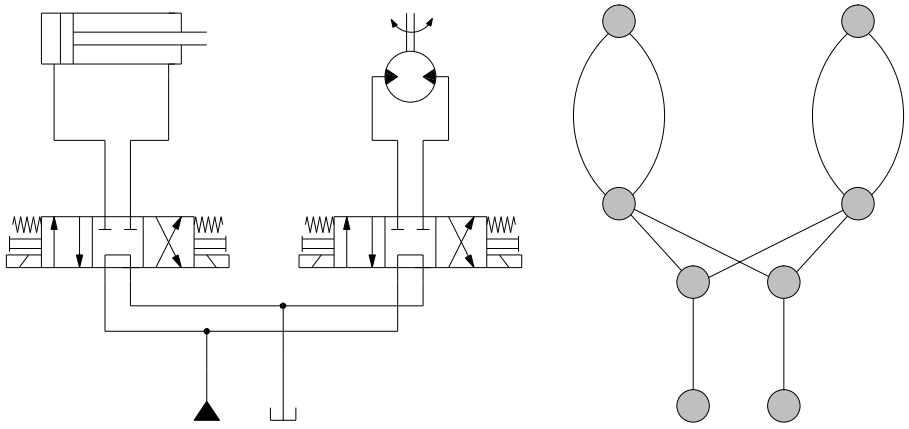

Abbildung 1.10: Parallel gekoppelte Achsen und der abstrakte Graph dazu

Aus Maschinenbausicht bedeutet die obige Definition, daß jede hydraulische Achse von ihrem eigenen Kontrollelement gesteuert wird, d. h. sie arbeiten unabhängig voneinander.

Ebene 3 – serielle Kopplung Sei $P_{w,s}$ die Menge aller Wege von einem Arbeitselement w zu dem Versorgungselement s, die über keine Kontrollkante gehen. Die hydraulischen Achsen A und B sind seriell gekoppelt, wenn mindestens eine der Achsen einen Weg $p \in P_{w,s}$ besitzt, der auch mindestens ein Kontrollelement der anderen Achse beinhaltet.

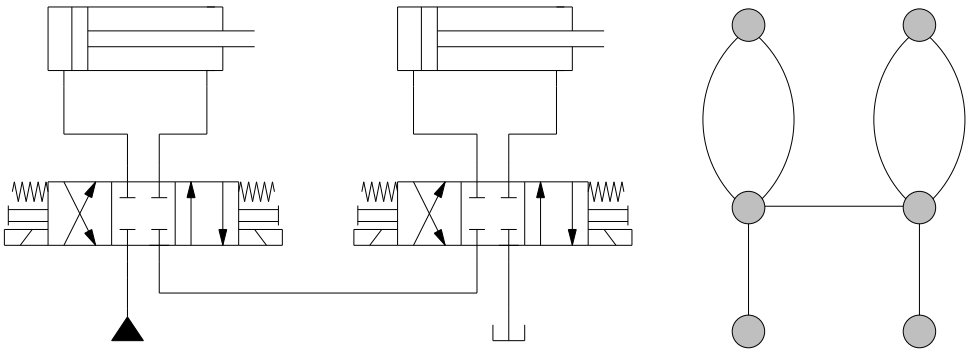

Abbildung 1.11: Seriell gekoppelte Achsen und der abstrakte Graph dazu

Eine serielle Kopplung impliziert eine Wechselwirkung zwischen den beteiligten Achsen, da mindestens eine dieser Achsen den Fluß, der alle Achsen versorgt, kontrolliert.

Ebene 4 – sequentielle Kopplung Sei $V_A' := V_A - (V_A \cap V_B)$ und $V_B' := V_B - (V_A \cap V_B)$ sowie P_{v_W,v_S} die Menge aller Wege von einem Arbeitselement v_W zu dem Versorgungselement v_S, die über keine Kontrollkante gehen. Die hydraulischen Achsen A und B sind sequentiell gekoppelt, wenn:

1. $\forall p \in P_{v_W,v_S} \cap V_X', X \in \{A, B\}$, $\exists v_B \in p$: v_B ist assoziert mit einer verhaltensändernden Komponente.
2. Die Achsen bestehend aus V_A' und V_B' sind nicht identisch, d. h. entweder die Mengen V_A' und V_B' sind nicht gleich, oder die Elemente aus diesen Mengen sind unterschiedlich angeordnet, d. h. sie unterscheiden sich in den Verbindungen zwischen den Komponenten.

Zwei sequentiell gekoppelte Achsen – auch eine *Folgeschaltung* genannt – wirken unterschiedlich, obwohl sie beide von derselben Kontrollkomponente gesteuert werden. Eine der Achsen besitzt immer Komponenten, die das Verhalten beeinflussen. Solche Komponenten bewirken beispielsweise eine zeitliche Verzögerung oder auch eine Druckminderung.

Die sequentielle Kopplung ist übrigens auch einen Sammelbegriff für alle Arten von Kopplungen, die nicht zu einer der oben definierten Kopplungsarten zugeordnet werden können.

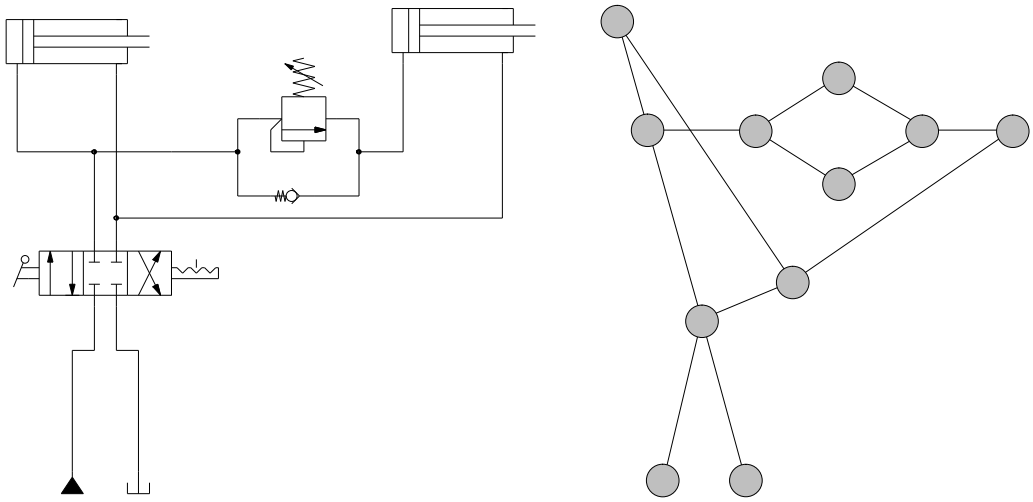

Abbildung 1.12: Sequentiell gekoppelte Achsen und der abstrakte Graph dazu

1.2.3 Allgemeine Verfahrensweise

Die in dieser Arbeit vorgestellten Verfahren sollen Bestandteil des Design- und Simulationssystems ArtDeco werden. Die in ArtDeco erstellten Schaltungen sollen in Graphen konvertiert werden, auf denen unsere Verfahren dann arbeiten werden. Da dieser Konvertierungsschritt eine Abstraktion einer Schaltung darstellt, verlieren wir spezielle Informationen bzw. Eigenschaften einzelner Teile, wie beispielsweise die Schaltstellung eines Ventils, das den Fluß steuert. Für unsere Zwecke aber enthalten unsere abstrahierten Graphen genug Information, so daß keine zusätzlichen Schritte notwendig sind.

Wir kommen nun zu den Hauptschritten unseres Verfahrens, die wir in den folgenden Unterabschnitten kurz beschreiben möchten.

Vorverarbeitung des Schaltkreisgraphen

Da hydraulische Anlagen, wie bereits in der Einleitung im Abschnitt 1.1 angedeutet, sehr komplex sein können, müssen wir damit rechnen, daß der Schaltkreisgraph ebenfalls von großer Komplexität sein kann, d. h. es sind sehr viele Knoten und Kanten vorhanden. Hinzu kommt die Tatsache, daß einige der zu lösenden Teilaufgaben relativ aufwendig sind, was eine hohe Laufzeit unseres Verfahrens bedeutet. Ein weiterer Grund für die Notwendigkeit einer Vorverarbeitung des Schaltkreisgraphen liegt darin, daß wir möglichst bereits existierende effiziente Algorithmen einsetzen möchten und nicht unbedingt neue Algorithmen für dieses spezielle Problem entwerfen wollen.

Aus diesen Gründen müssen wir den Schaltkreisgraphen in einem Vorverarbeitungsschritt vereinfachen, um eine zügigere Laufzeit unseres Verfahrens zu gewährleisten sowie bereits bekannte Algorithmen anwenden zu können. Im Kapitel 2 werden wir einige vereinfachende Techniken vorstellen sowie die für unser Verfahren am besten geeignete Technik näher erläutern.

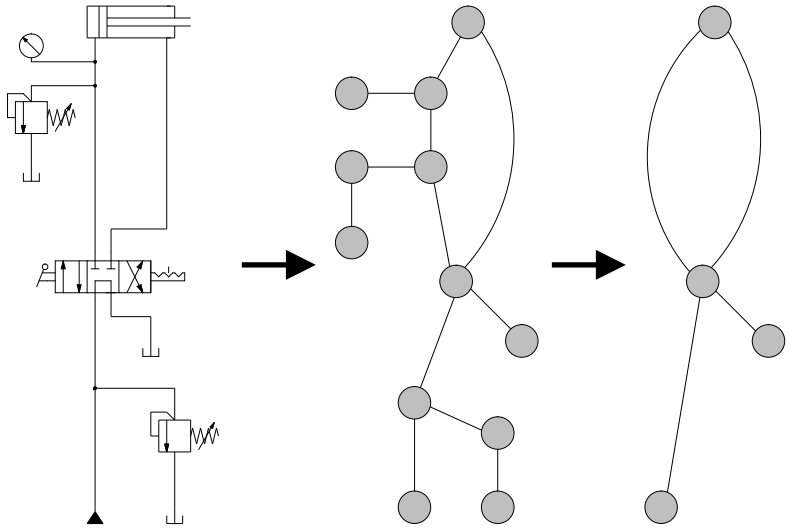

Abbildung 1.13: Beispiel für die Abstraktions- und Vereinfachungsschritte

Erkennung der hydraulischen Achsen

Um eine hydraulische Achse zu erkennen, brauchen wir uns nur an die Definition zu halten. Wir suchen also nach Gruppen von Komponenten, die zusammen eine bestimmte Funktionalität anbieten. Diese Gruppen bilden Teilgraphen des gesamten Schaltungsgraphen und besitzen mindestens ein Arbeitselement und mindestens ein Versorgungselement. Da die Arbeitselemente den von ihnen benötigten Druck von den Versorgungselementen erhalten, müssen diese innerhalb derselben Zusammenhangskomponente liegen und miteinander verbunden sein. Also gehören alle übrigen Komponenten des Teilgraphen, die auf den Wegen zwischen Arbeits- und Versorgungselementen liegen, zur selben hydraulischen Achse wie die Arbeits- und Versorgungselemente.

Folgende Lösungsvarianten bieten sich an:

1. Aus diesen ersten Überlegungen folgt, daß die Erkennung der hydraulischen Achsen mit Hilfe von Algorithmen zur Bestimmung von Zusammenhangskomponenten sowie zur Berechnung von kürzesten Wege realisiert werden kann, wobei man diese Algorithmen zum Teil verändern bzw. erweitern muß,

um bestimmte Situationen korrekt behandeln zu können sowie unnötige Berechnungen zu vermeiden. Auf diese Vorgehensweise wird im Kapitel 3 näher eingegangen.

2. Eine weitere Möglichkeit zur Lösung unserer Aufgabe bieten Graph-Einbettungen: Wir versuchen, eine *Schablone* einer hydraulischen Achse in unseren Schaltkreisgraphen einzubetten. Wir können anschließend alle von der Einbettung der Schablone berührten Knoten einer hydraulischen Achse zuordnen. Wir werden diese Methode im Kapitel 4 genauer beschreiben.

Bestimmung der Kopplungsart zwischen den hydraulischen Achsen

Sind alle hydraulischen Achsen einer Schaltung identifiziert worden, so ist die Kopplung zwischen ihnen bestimmbar. Ähnlich wie bei der Erkennung der hydraulischen Achsen, brauchen wir uns nur an die Definitionen der verschiedenen Kopplungsarten zu halten.

Die Erkennung von Kopplungen der Ebene 0 bzw. des Fehlens von irgendeiner Kopplung ist die leichteste Teilaufgabe der Bestimmung der Kopplung zwischen allen möglichen Achsenpaaren. Ein Achsenpaar ist auf keine Art und Weise miteinander gekoppelt, wenn die betreffenden Achsen sich in unterschiedlichen Zusammenhangskomponenten befinden. Da wir vor der Erkennung der hydraulischen Achsen bereits alle Zusammenhangskomponenten bestimmt haben, müssen wir an dieser Stelle lediglich die Komponenten beider Achsen aller möglichen Achsenpaare miteinander vergleichen.

Die Erkennung von Kopplungen der Ebene 1 bzw. informationellen Kopplungen hängt stark mit dem Vorhandensein von Kontrollkanten zusammen. Wie in der Definition dieser Kopplungsart müssen wir bei einer hypothetischen Entfernung einer solchen Kanten nur überprüfen, ob dadurch eine Zusammenhangskomponente weiter zerlegt würde.

Bei den letzten drei Kopplungsarten müssen wir Pfade miteinander vergleichen. Im Fall einer parallelen Kopplung besitzen die betreffenden Achsen jeweils eine eigene Kontrollkomponente; im Fall einer seriellen Kopplung besitzt mindestens eine der hydraulischen Achsen einen Pfad, der auch die Kontrollkomponente der jeweils anderen Achse mit einschließt; im letzten Fall, der sequentiellen Kopplung, unterscheiden sich die gekoppelten Achsen nur geringfügig – eine der Achsen besitzt verhaltensändernde Komponenten.

Analog zur Erkennung hydraulischer Achsen können wir die Bestimmung der Kopplungsart zwischen den hydraulischen Achsen wieder mit Hilfe von Algorithmen zur Bestimmung von Zusammenhangskomponenten und zur Pfadsuche

realisieren. Der Unterschied hierin liegt an der Lösung der verschiedenen Teilaufgaben: Die Erkennung der Kopplungen der Ebenen 0 und 1 – keine und informationelle Kopplung – bedarf nur an Informationen bezüglich der vorhandenen Zusammenhangskomponenten; die Erkennung der Kopplungen der Ebenen 2, 3 und 4 – parallele, serielle und sequentielle Kopplung – impliziert dagegen den Vergleich von Wegen, die von der Pfadsuche während der Erkennung der Achsen gefunden wurden.

Kapitel 2

Techniken zur Vorverarbeitung

In diesem Kapitel wollen wir einige Möglichkeiten zur Vereinfachung des Graphen etwas näher betrachten, wobei wir nur die für unsere Zwecke beste Vereinfachungstechnik in ausreichender Tiefe erläutern werden – sowohl was die theoretischen Grundlagen angeht, als auch die praktische Umsetzung. Weiterhin werden wir über einige Vorverarbeitungsschritte sprechen, die nicht zur Vereinfachung des Graphen beitragen, sondern von den Vereinfachungstechniken selbst benötigt werden.

2.1 Abstraktionsmethoden

In vielen Bereichen der Informatik sind effiziente Techniken zur Vereinfachung von komplexen Strukturen nötig, da die Menge und Größe der Daten oft eine effiziente Verarbeitung unmöglich macht. Ein Bereich, der stark auf solche Techniken angewiesen ist, ist die graphische Datenverarbeitung, denn hier muß man nicht nur die rechnerische Komplexität reduzieren, um eine Beschleunigung der Verarbeitung zu erreichen, sondern oft auch die visuelle Komplexität.

Ein einfaches Beispiel ist die Arbeit mit einem Visualierungs-Programm. Auf der einen Seite verwendet das System vereinfachende Techniken, um beispielsweise das Rendering eines Objekts zu beschleunigen – komplexe Objekte werden durch einfache überdeckt, damit die Berechnung des Schnittes der Lichtstrahlen mit den Objekten weniger Zeit in Anspruch nimmt. Auf der anderen Seite werden Maßnahmen ergriffen, um den visuellen Ballast für den Menschen zu reduzieren – Objekte werden zu Gruppen zusammengefaßt und als ein einzelnes Objekt dargestellt, oder sie werden aus dem Bild ganz entfernt.

Gerade weil in der graphischen Datenverarbeitung solche Techniken eine zentrale Rolle spielen, stammen die meisten Lösungen dazu aus diesem Bereich. Die

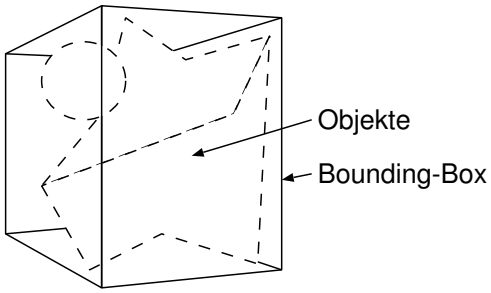

Abbildung 2.1: Komplexe Objekte und eine primitive Hülle

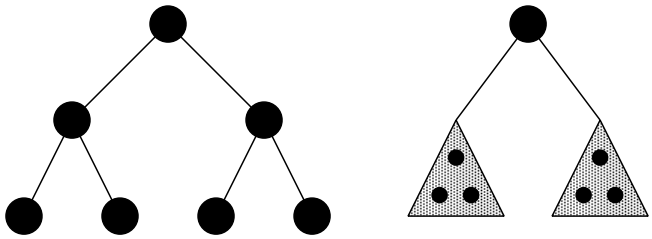

Abbildung 2.2: Ein Baum ohne und mit Meta-Knoten

allgemeine Vorgehensweise ist, aus einem gegebenen Graphen einen einfacheren, abstrakten Graphen abzuleiten. Dabei unterscheidet man folgende Abstraktionsmethoden wie in [6]:

- *Ghosting*: Irrelevante Knoten werden visuell gekennzeichnet, indem man sie mit einer anderen Farbe, etwa einer anderen Schattierung der Hintergrundsfarbe, markiert. Im Bereich der graphischen Oberflächen ist diese Technik weitverbreitet: irrelevante oder ungültige Objekte – beispielsweise inaktive Menüpunkte – werden *ausgegraut*, um dem Benutzer mitzuteilen, daß diese Funktionen nicht verfügbar sind. Diese Methode ist offensichtlich nur für die Reduzierung der visuellen Last für den Benutzer geeignet und nicht zur Beschleunigung von rechenintensiven Vorgängen.

- *Hiding*: Die Idee dieser Methode ist, alle irrelevanten Knoten und deren Kanten aus dem Graphen zu entfernen. Dieses Verfahren ist vorwiegend für die visuelle Vereinfachung von Graphen geeignet – in der Regel möchte oder darf man nicht einfach Knoten aus dem Graphen löschen, da dabei meistens Information verloren geht; aber für die Darstellung auf dem Bildschirm ist ein solcher Informationsverlust meist bedeutungslos und sogar willkommen.

- *Grouping*: Knoten, die zusammengehören, werden zusammengefaßt und als ein einziger Meta-Knoten dargestellt. Hierbei muß man zusätzlich definie-

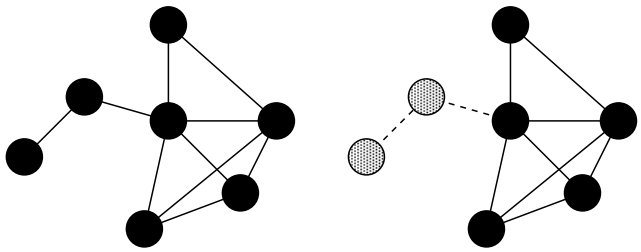

Abbildung 2.3: Ausblendung von irrelevanten Knoten

ren, wie diese Zusammengehörigkeit bestimmt wird. Vorstellbar wäre unter anderem die Gruppierung von benachbarten Knoten vom Grad zwei – damit würden wir Ketten eines Graphen auf einen einzelnen Knoten abbilden.

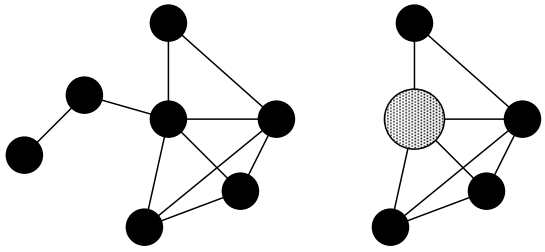

Abbildung 2.4: Gruppierung von Knoten

In den nächsten Abschnitten werden wir sehen, daß vor allem die letzte Abstraktionstechnik für unsere Zwecke geeignet ist, da sie viel an Flexibilität bietet, jedoch keinen Informationsverlust mit sich bringt. Darüberhinaus ist diese Abstraktionstechnik Bestandteil der Produktionen von Graph-Grammatiken, die wir im Abschnitt 2.3.3 behandeln werden.

2.2 Graph-Kontraktion und -Expandierung

Wie wir bereits gesehen haben, muß die Graph-Kontraktion vor der eigentlichen Berechnung der hydraulischen Achsen erfolgen, wodurch das Problem der Erkennung der hydraulischen Achsen erst lösbar gemacht wird. Dabei gilt, daß je effizienter und mächtiger wir die Graph-Kontraktion realisieren, desto einfacher können wir die Bestimmung der hydraulischen Achsen halten.

Da wir die Ergebnisse auf dem ursprünglichen Graphen haben möchten, müssen wir nach der Bestimmung der hydraulischen Achsen den komprimierten Graphen

wieder expandieren, um die von der Berechnung ausgeschlossenen Knoten ihren entsprechenden hydraulischen Achsen zuordnen zu können – aus diesem Grund scheidet die Abstraktionsmethode *Hiding* für die meisten Fälle aus. Erst dann, wenn alle hydraulischen Achsen vollständig bekannt sind, können wir mit der Bestimmung der Kopplung zwischen ihnen fortfahren.

Unser abstrakter Algorithmus sieht demnach folgendermaßen aus:

1. Graph-Kontraktion
2. Bestimmung der hydraulischen Achsen
3. Graph-Expandierung
4. Bestimmung der Kopplung zwischen den hydraulischen Achsen

Die Aufgabe der Vorverarbeitung besteht nicht nur aus der Verkleinerung des Problems, d. h. der Vereinfachung des Graphen bezüglich der Anzahl der Knoten und Kanten, sondern auch aus der Vereinfachung der Struktur des Graphen mit dem Zweck der Reduzierung der Anzahl von Sonderfällen, die zu behandeln sind. Einige Strukturen, die wir aus unserem Graphen entfernen möchten, sowie die Abstraktionsmethoden, die genau dies leisten, werden wir in den nachfolgenden Abschnitten kennenlernen. Zuvor möchten wir aber einige theoretische Grundlagen erarbeiten.

2.3 Theoretische Grundlagen der Vorverarbeitung

Die von unserem Verfahren zur Graph-Kontraktion verwendeten Routinen haben als Grundlage graphentheoretische Konzepte, die Transformationen auf Graphen bzw. Teilgraphen erklären. Es hat sich herausgestellt, daß alle zur Graph-Kontraktion benötigten Transformationen als Produktionsregeln einer einfachen *Graph-Grammatik* dargestellt werden können. Bevor wir aber das Konzept der Graph-Grammatiken näher betrachten können, müssen wir den Begriff der *Grapheinbettung* sowie das Konzept der *markierten Graphen* erläutern.

2.3.1 Graph-Einbettung

Eine Einbettung von einem Graphen $G_1 = (V_1, E_1)$ in einen Graphen $G_2 = (V_2, E_2)$ wird erklärt durch eine injektive Abbildung φ mit $\varphi : V_1 \to V_2$.

Im allgemeinen werden benachbarte Knoten aus G_1 nicht auf benachbarte Knoten aus G_2 abgebildet. Um eine Einbettung von G_1 nach G_2 vollständig zu beschreiben, muß daher noch für jede Kante $\{u, v\} \in E_1$ ein Weg $P_\varphi(u, v)$ angegeben werden, der in G_2 die Knoten $\varphi(u)$ und $\varphi(v)$ miteinander verbindet. Dieser Weg muß nicht unbedingt der kürzeste Weg zwischen $\varphi(u)$ und $\varphi(v)$ sein.

Ein Beispiel hierfür wäre die Einbettung des Shuffle-Exchange-Netzwerkes in das ungerichtete DeBruijn-Netzwerk aus [8]. Diese Netzwerke sind wie folgt definiert:

- Das Shuffle-Exchange-Netzwerk der Dimension k ist ein Graph $SE(k) = (V_k, E_k)$ mit
  - $V_k = \{0,1\}^k$ und
  - $E_k = \{\{a\alpha, \alpha a\} \mid \alpha \in \{0,1\}^{k-1}, a \in \{0,1\}\}$ (Shuffle-Kanten)
    $\cup\{\{\alpha a, \alpha \overline{a}\} \mid \alpha \in \{0,1\}^{k-1}, a \in \{0,1\}\}$ (Exchange-Kanten).

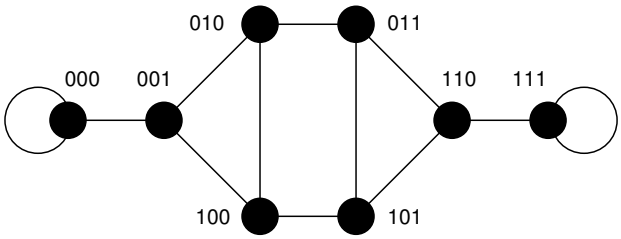

Abbildung 2.5: Das Shuffle-Exchange-Netzwerk der Dimension 3

- Das ungerichtete DeBruijn-Netzwerk der Dimension k ist ein Graph $DB(k)$, $DB(k) = (V_k, E_k)$, mit
  - $V_k = \{0,1\}^k$ und
  - $E_k = \{\{a\alpha, \alpha a\} \mid \alpha \in \{0,1\}^{k-1}, a \in \{0,1\}\}$ (Shuffle-Kanten)
    $\cup\{\{a\alpha, \alpha \overline{a}\} \mid \alpha \in \{0,1\}^{k-1}, a \in \{0,1\}\}$ (Shuffle-Exchange-Kanten).

Die Einbettung des Shuffle-Exchange-Netzwerkes in das ungerichtete DeBruijn-Netzwerk sieht folgendermaßen aus:

$$\varphi(u) = \begin{cases} u & , \text{ falls die Anzahl Einsen in u gerade ist} \\ s^{-1}(u) & , \text{ sonst} \end{cases}$$

Dabei ist $s^{-1}(\alpha a) = a\alpha$ mit $a \in \{0,1\}$, $\alpha \in \{0,1\}^{k-1}$ die Inverse der sogenannten Shuffle-Funktion oder einfach die Unshuffle-Funktion.

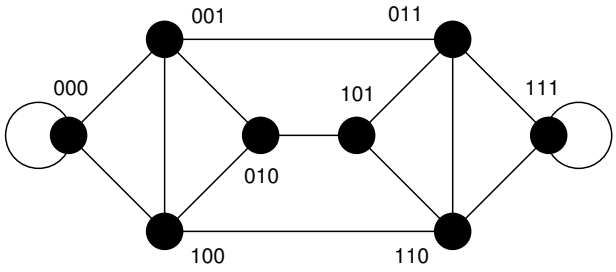

Abbildung 2.6: Das DeBruijn-Netzwerk der Dimension 3

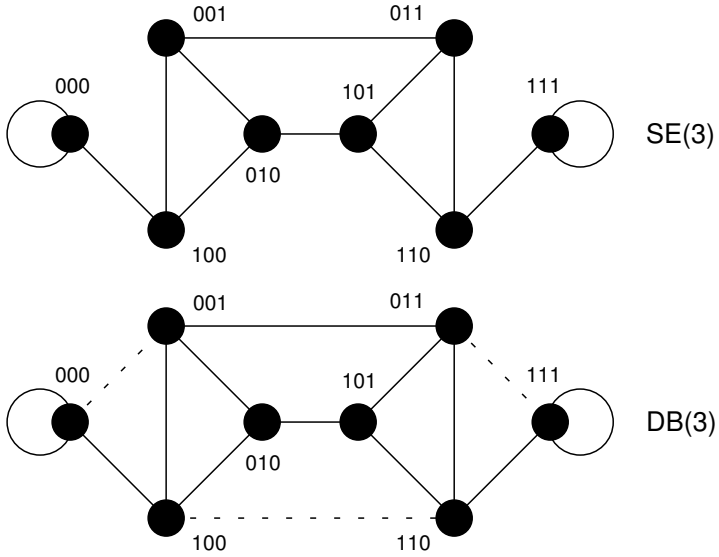

Abbildung 2.7: Einbettung des Shuffle-Exchange- in das DeBruijn-Netzwerk

Die Einbettung für dieses Beispiel ist trivial, denn das Shuffle-Exchange-Netzwerk ist Teilgraph des ungerichteten DeBruijn-Netzwerkes. Wäre dies nicht der Fall, so müßte mindestens eine Kante des Shuffle-Exchange-Netzwerkes auf einen Weg der Länge ≥ 2 abgebildet werden. Man kann anhand der Definitionen beider Netzwerke sich leicht überlegen, daß durch die Einbettung des ungerichteten DeBruijn-Netzwerkes in das Shuffle-Exchange-Netzwerk einige Kanten auf Wege der Länge 2 abgebildet werden, und zwar wird jede Shuffle-Exchange-Kante des DeBruijn-Netzwerkes auf einen Weg bestehend aus einer Shuffle- und einer Exchange-Kante des Shuffle-Exchange-Netzwerkes abgebildet.

Weiterführende Informationen zu diesem Thema findet der Leser in [1], [2], [11] und [18].

2.3.2 Markierte Graphen

Ein markierter Graph, auch *l-Graph* genannt (l steht hier für „labelled"), ist ein Graph mit markierten Knoten und Kanten. Wie in [9] definiert, ist ein markierter Graph über den Alphabeten Σ_V, Σ_E ein Tupel $D = (V, (\rho_a)_{a \in \Sigma_E}, \beta)$ mit

- V ist die endliche Menge der Knoten.
- ρ_a ist eine Relation auf V für beliebige $a \in \Sigma_E$, d. h. $\rho_a \subseteq V \times V$. Dabei ist a die Kantenmarkierung, und die Relation ρ_a beschreibt alle Kanten, die mit a markiert sind. Für unsere Zwecke setzen wir voraus, daß $\forall (u, v) \in \rho_a \Rightarrow (v, u) \in \rho_a$, d. h. die Relation ist reflexiv. Damit haben wir sichergestellt, daß unser Graph „ungerichtet" ist.
- $\beta : V \to \Sigma_V$ ist die Knotenmarkierungsfunktion.

Die Alphabete Σ_V und Σ_E dienen lediglich der Kennzeichnung der Knoten und Kanten.

Aus dieser Definition können wir folgern, daß es zwischen zwei Knoten u und v mehrere Kanten geben darf, sofern diese sich durch die Markierung unterscheiden. Allerdings ist es nicht möglich, daß zwei Knoten durch verschiedene Kanten, die gleich markiert sind, verbunden sind.

l-Graphen werden überwiegend zur Beschreibung von strukturellen Zusammenhängen im folgenden Sinne gebraucht (siehe auch [9]): Die Knoten des *l*-Graphen entsprechen den Unterstrukturen der Gesamtstruktur – sie sind mit der Unterstruktur selbst bzw. einer Kennzeichnung markiert, die die Unterstruktur charakterisiert. Wir unterscheiden außerdem zwischen Knotenmarkierung und Knotenbezeichnung: Erstere charakterisiert die durch den Knoten dargestellte Unterstruktur; letztere kennzeichnet eine Instanz eines solchen Knotens, da ein *l*-Graph mehrere Knoten, denen dieselbe Unterstruktur zugeordnet ist, besitzen darf. Also kann ein Knoten beispielsweise einen gesamten Teilgraphen repräsentieren – dies ist offensichtlich für die Abstraktion eines Graphen eine wichtige Eigenschaft bzw. Fähigkeit.

Ein passendes Analogon zum obigen Konzept der markierten und gekennzeichneten Knoten findet man bei Programmiersprachen: Dort hat man Datentypen und Variablen, die Instanzen solcher Datentypen sind.

2.3.3 Graph-Grammatiken

Graph-Grammatiken, auch Graph-Ersetzungssysteme genannt, sind eine Verallgemeinerung der Chomsky-Systeme über Zeichenketten. Analog zu den Gram-

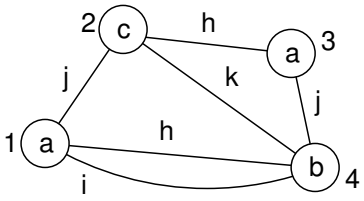

Abbildung 2.8: Ein l-Graph

matiken über einem Alphabet, besitzen auch Graph-Grammatiken Produktionsregeln, die genauso zu interpretieren sind wie ihre Pendants über Zeichenketten. Allgemein betrachtet besitzt jede Regel eine linke und eine rechte Seite; die linke Seite gibt an, was zu ersetzen ist, und die rechte Seite gibt an, was anstelle des Konstrukts der linken Seite in die gesamte Struktur eingesetzt wird. Für l-Graphen bedeutet dies, daß ein Teilgraph durch einen anderen Teilgraph ersetzt wird, wobei sich die nichttriviale Frage stellt, wie man den Teilgraphen der rechten Seite in den ursprünglichen Graphen einbettet. Also muß die Festlegung der Einbettung Bestandteil der Graph-Regel sein. Somit hat eine Graph-Regel, oder besser Graph-Produktion genannt, die Form $p = (d_l, d_r, \varphi)$, wobei d_l der linken Seite, d_r der rechten Seite und φ der Einbettungsfunktion entspricht.

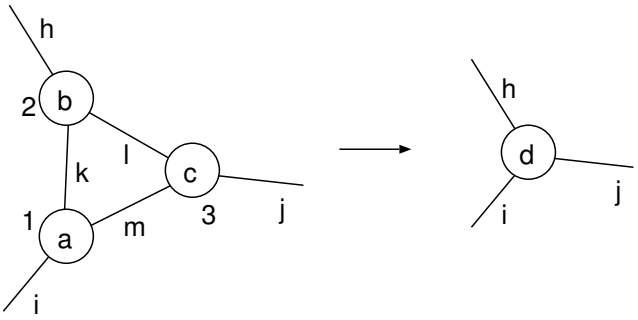

Abbildung 2.9: Eine Produktionsregel einer Graph-Grammatik

Eine Graph-Grammatik ist, wie oben angedeutet, eine Verallgemeinerung einer Chomsky-Grammatik. D. h., man kann eine Graph-Grammatik ähnlich wie zum Beispiel eine kontextfreie Grammatik definieren, also mit einem Startsymbol, einer Menge von Variablen, einer Menge von Terminalen und einer Menge von Produktionsregeln. Unser Ziel ist allerdings nicht das Ableiten eines Worts aus dem Startsymbol bzw. eines komplexen Graphen aus einem Startknoten, sondern die Kontraktion eines Graphen. Aus diesem Grund besteht eine für unsere Zwecke passende Graph-Grammatik ausschließlich aus der Menge von Produktionsregeln, wie wir sie oben beschrieben haben, zusammen mit dem zu bearbeitenden Graphen. Zusätzlich fordern wir, daß die rechte Seite jeder Produktionsregel echt

kleiner ist als die linke Seite, sofern es sich um eine Kontraktionsregel handelt.

Formal beschrieben ist eine für unsere Zwecke passende Graph-Grammatik ein 2-Tupel $GG = (G, P)$ mit:

- G ist der Ausgangsgraph.
- P ist die Menge der Produktionsregeln. P enthält zwei Arten von Produktionsregeln:
  1. Kontraktionsregel der Form $p = (d_l, d_r, \varphi)$ mit $|d_r| < |d_l|$.
  2. Regel zur Behandlung von Spezialfällen. Diese Regeln sind beispielsweise für die Ersetzung kritischer Komponenten zuständig.

[12], [10] und [16] liefern tiefergehende Einblicke in die Thematik der Graph-Grammatiken und gehen zum Teil auch auf mögliche Anwendungen ein.

2.4 Graph-Grammatik zur Graphabstraktion

Mit einer Graph-Grammatik hat man eine Sammlung von Vorschriften zur Transformation eines Graphen, wobei jede Vorschrift nur jeweils in einem bestimmten Kontext angewendet werden darf. Unser Ziel ist es, einen gegeben Schaltkreisgraphen in eine möglichst einfache Form zu bringen. Dies können wir mit Hilfe einer Graph-Grammatik erreichen, deren Produktionsregeln alle schwierig zu behandelnde Teilgraphen vereinfachen. Die in unserer Aufgabe auftretenden Fälle und die dazugehörigen Produktionsregeln, soweit sie benötigt werden bzw. effizient realisierbar sind, werden wir im folgenden kennenlernen.

Um unsere Produktionsregeln einfach und übersichtlich zu halten, werden wir im folgenden auf die Beschriftung von Kanten und die Kennzeichnung von Knoten verzichten. Da wir lediglich an den Knotentypen interessiert sind, werden wir nur die Knotenmarkierung angeben, um die Knoten nach ihren Typen unterscheiden zu können.

2.4.1 Löschung von irrelevanten Komponenten

Um die quantitative Komplexität unseres Schaltkreisgraphen zu reduzieren, können wir irrelevante Komponenten aus dem Graphen entfernen. In den meisten Fällen muß man aber den Kontext der betreffenden Komponente mitberücksichtigen, denn manche Komponenten haben mehrere Anwendungsmöglichkeiten – so kann beispielsweise ein Ventil, das wir als Kontrollelement definiert haben,

auch als ein Hilfselement fungieren. Nichtsdestotrotz existieren Fälle, in denen man eine Komponente unabhängig von ihrem Kontext entfernen darf, wie zum Beispiel ein Drosselventil.

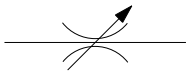

Abbildung 2.10: Ein Drosselventil

Wie wir in den nächsten Abschnitten sehen werden, wird dieser Fall bereits bei der Entfernung von Ketten mitberücksichtigt, so daß keine eigene Produktionsregel hierfür notwendig ist.

2.4.2 Ersetzung von kritischen Komponenten

Einige Komponenten erschweren unnötig die Bestimmung der hydraulischen Achsen, weil sie eine besondere Struktur des Graphen implizieren. Ein Beispiel dafür ist die Komponente *Antriebsaggregat*, die eine Pumpe und zwei Tanks in einem Knoten vereint. Dadurch wird die Pfadsuche (siehe Kapitel 3) dermaßen erschwert, daß ganze Teile des abstrakten Graphen nicht mehr erreicht werden. Als Ausweg aus dieser Situation bietet sich die Ersetzung solcher Knoten durch einfachere Knoten. Im obigen Fall würde man den Antriebsaggregat durch eine Pumpe und zwei Tanks ersetzen.

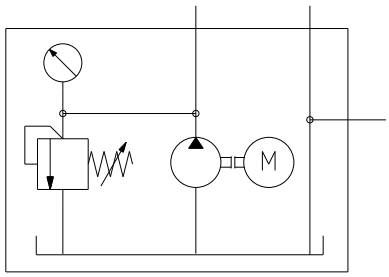

Abbildung 2.11: Ein Aggregat mit zwei Tankanschlüssen

Die zugehörige Produktionsregel der Graph-Grammatik ist in Abbildung 2.12 dargestellt.

Als eine Alternative könnte man auch nur die zu den Tanks gehörenden Kanten am Aggregat abtrennen und neue Tanks an den abgetrennten Kanten anhängen. In Abbildung 2.12 würde dann die Beschriftung P nicht mehr auftauchen, sondern nur noch A.

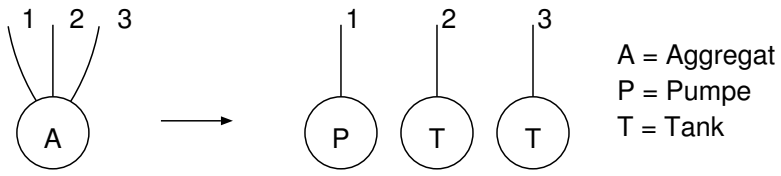

Abbildung 2.12: Substitution eines Aggregats durch andere Komponenten

2.4.3 Löschung von Kontrollkanten

Kontrollkanten tragen nichts zur Erkennung der hydraulischen Achsen bei – in einer hydraulischen Schaltung dienen sie lediglich der Steuerung von bestimmten Komponenten oder Teilschaltungen. Solche Steuerungsinformation geht bei der Abstraktion von der konkreten Schaltung auf jeden Fall verloren – es werden beispielsweise auch keine Schaltstellungen von direktionalen Ventilen berücksichtigt. Da unser abstrahierter Graph keinerlei dynamische Eigenschaften besitzt, sondern rein statischer Natur ist, können wir alle Kontrollkanten entfernen, da sie nur dynamische Bedeutung haben.

Die Entfernung von Kontrollkanten hat für den abstrakten Graphen eine wichtige Bedeutung, denn solche Kanten sind oft die einzige Verbindung zwischen verschiedenen Blöcken des Graphen, oder anders ausgedrückt, sie stellen Brücken des Graphen dar. Durch die Entfernung solcher Kanten erhalten wir oft mehrere voneinander völlig unabhängige Schaltungen, so daß wir die Lösung des Problems auf die Lösung mehrerer kleinerer Probleme zurückgeführt haben, ganz im Sinne des *Divide-and-Conquer*-Paradigmas.

Da die Identifizierung solcher Kanten in der Regel stark von ihrem Kontext abhängt, wäre eine Produktionsregel zur Erkennung und Löschung von Kontrollkanten mit einer großen Menge von Nebenbedingungen verbunden, was diesen eigentlich kleinen Schritt ungemein verkompliziert. Daher sehen wir davon ab, für diesen Fall eine komplexe Produktionsregel anzugeben, und lösen dieses Problem auf eine andere, bequemere Weise, soweit dies möglich ist.

Tatsächlich kann man einige Fälle sehr leicht finden, da manche Komponenten einen zusätzlichen Anschluß für eine Kontrollkante besitzen – man braucht also nur zu prüfen, ob der betreffende Anschluß besetzt ist. Eine Grammatik-Regel wäre für diesen speziellen Fall denkbar, aber um die Grammatik nicht unnötig aufzublähen, verzichten wir auf eine zusätzliche Regel. Solche Fälle sind bereits beim Aufbau des Graphen erkennbar und können daher dort bearbeitet werden.

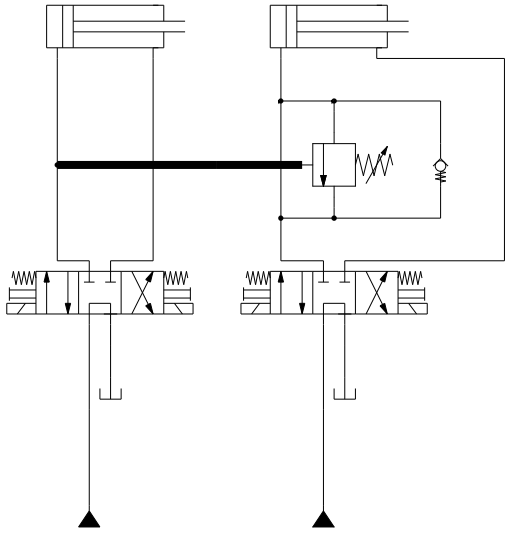

Abbildung 2.13: Zwei durch eine Kontrollkante verbundene hydraulische Achsen

2.4.4 Zusammenfassung von Tanks

Unsere Graphen enthalten oft eine große Zahl von Tanks und verursachen daher ebenfalls einen großen Aufwand, da für jedes Vorkommen eines Tanks der Pfadsuchalgorithmus von Ford angewendet werden muß (siehe hierzu Kapitel 3). Es ist aus diesem Grunde sehr wünschenswert, daß die Anzahl der Tanks auf das notwendige Minimum reduziert wird.

Da wir Tanks nicht beliebig aus dem Graphen entfernen dürfen, weil sie meistens doch in einem wichtigen Kontext auftreten, müssen wir sie irgendwie beibehalten und ihre Anzahl gleichzeitig reduzieren. Die einzige Möglichkeit, dies zu bewerkstelligen, liegt in der Zusammenfassung der Tanks, d. h. wir verwenden einen Meta-Tank, der stellvertretend für alle Tanks in den Graphen eingeführt wird.

Ein kleines Problem kann aber durch die Zusammenfassung der Tanks entstehen: Ist die hydraulische Schaltung in mehrere Zusammenhangskomponenten aufgeteilt, so werden durch diese Transformation alle Zusammenhangskomponenten miteinander verbunden, so daß wir bei der Erkennung der hydraulischen Achsen falsche Ergebnisse erhalten. Wir müssen daher diesen Schritt für jede Zusammenhangskomponente getrennt durchführen.

Da eine graphische Darstellung dieser Produktionsregel wegen der benötigten Kontextinformationen – gemeint sind hier die Daten über den Zusammenhangskomponenten – recht schwierig ist, müssen wir uns mit dieser informellen Beschreibung zufrieden geben.

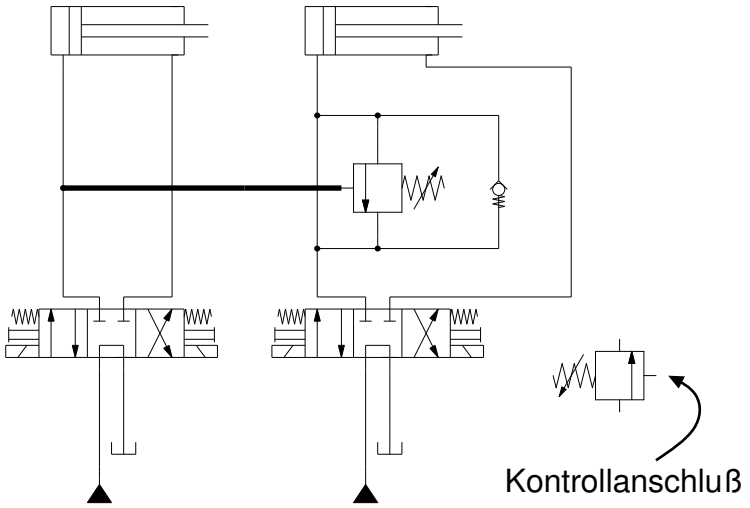

Abbildung 2.14: Eine Komponente mit Kontrollanschluß in einer Schaltung

2.4.5 Erkennung und Löschung von Ketten

Eine Kette ist ein Teilgraph, dessen Knoten höchstens Grad zwei haben. Leider können wir nicht beliebige Ketten entfernen, da manche Knoten für die Schaltung eine essentielle Rolle spielen, wie zum Beispiel Arbeitselemente oder etwa eine Pumpe. Das heißt, daß wir nicht nur die Ketten finden, sondern auch den Typ eines jeden Knotens in der Kette untersuchen müssen, bevor wir irgendwelche Transformationen durchführen können.

Da verschiedene Typen von Komponenten unterschiedlich behandelt werden, unterscheiden wir zwischen einigen Kettenarten, die wir in den folgenden Abschnitten näher erläutern werden.

Tote Zweige

Tote Zweige sind Ketten, die einen Knoten vom Grad eins besitzen. Solche Ketten bestehen in den meisten Fällen aus Hilfselementen, die für die Bestimmung der hydraulischen Achsen nicht maßgeblich sind. Allerdings gibt es Fälle, in denen eine Entfernung der Kette zu weiteren Komplikationen führt, so daß man eine Sonderbehandlung durchführen muß.

Ein Beispiel für einen solchen Fall ist eine Kette, die den einzigen Tank der Schaltung beinhaltet – da für die Bestimmung der hydraulischen Achsen mindestens ein Tank vorhanden sein muß (siehe Kapitel 3), darf man eine solche Kette nicht entfernen. Ein weiteres Beispiel für eine nicht zu entfernende Kette ist ein toter Zweig, der ein Arbeitselement oder eine Pumpe beinhaltet. Solche Ketten

darf man zwar nicht vollständig entfernen, aber man kann sie verkürzen, falls sie mehrere benachbarte Hilfselemente besitzen.

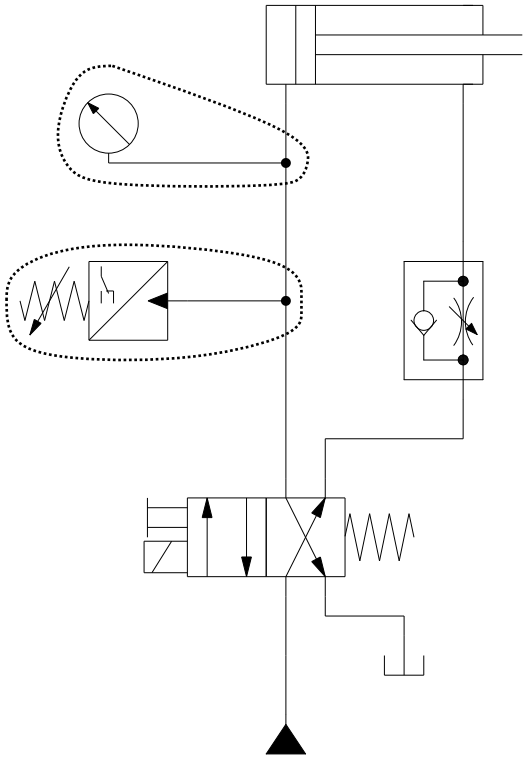

Abbildung 2.15: Eine Schaltung mit toten Zweigen

Die Produktionsregel für die Entfernung kann unterschiedlich aussehen: entweder verwendet man eine einzige einfache Regel, die eine Kette dann inkrementell verkürzt, oder man benutzt eine variable Regel, die eine Kette beliebiger Länge entfernt. Beide Varianten sähen wie in Abbildung 2.16 aus.

Ketten nicht essentieller Komponenten

Ketten, die keine Elemente von essentieller Bedeutung für die später anzuwendenden Verfahren besitzen, dürfen ebenfalls entfernt werden. Sind irgendwo in der Kette dagegen wichtige Elemente, so wird sie nur kürzer (analog zu den toten Zweigen) durch Anwendung der Ersetzungsregeln.

Hier haben wir, wie in dem vorigen Abschnitt, zwei Varianten der Produktionsregeln – eine einfache Regel für den wiederholten Gebrauch und eine variable Regel (siehe Abbildung 2.17).

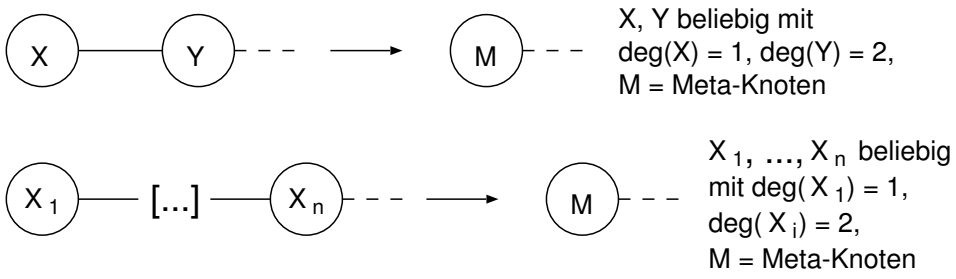

Abbildung 2.16: Produktionsregeln zur Entfernung von toten Zweigen

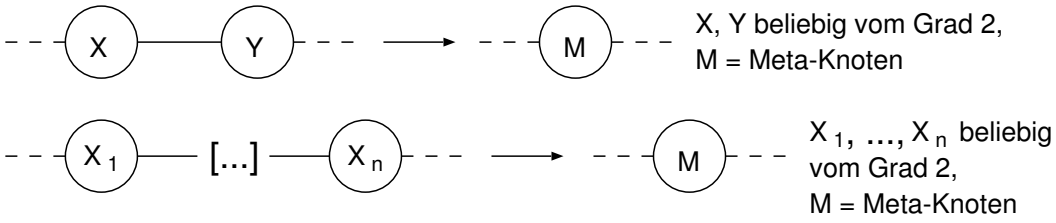

Abbildung 2.17: Regeln zur Entfernung von Ketten nicht essentieller Elementen

Ketten gleichartiger Komponenten (Arbeitselemente)

Wir haben in dem vorigen Abschnitt angegeben, daß Ketten, die einen Arbeitselement oder eine Pumpe beinhalten, nicht vollständig entfernt werden dürfen. Sind in einer Kette mehrere benachbarte Arbeitselemente vorhanden, so darf man sie zu einem „Meta-Arbeitselement“ umwandeln – diese Transformation verursacht keinen Informationsverlust und kann daher immer durchgeführt werden (mit Hilfe der letzten Produktionsregel).

Diese Vereinfachung können wir deswegen vornehmen, weil benachbarte Arbeitselemente zu einer und derselben hydraulischen Achse gehören – eine Zusammenfassung dieser Komponenten hat also keinerlei negative Konsequenzen.

2.4.6 Erkennung und Löschung von Schleifen

Eine hydraulische Schaltung kann Schleifen unterschiedlichen Typs enthalten. Schleifen, die Arbeitselemente weder beinhalten noch kontrollieren, dürfen entfernt werden, da sie keine besondere Bedeutung für die Achsenbestimmung haben. Darüberhinaus ist eine schleifenfreie Schaltung besonders effizient zu untersuchen, da die Bestimmung der kürzesten Wege relativ trivial wird. Allerdings sei darauf hingewiesen, daß eine hydraulische Schaltung fast immer Schleifen besitzt, die nicht entfernt werden dürfen.

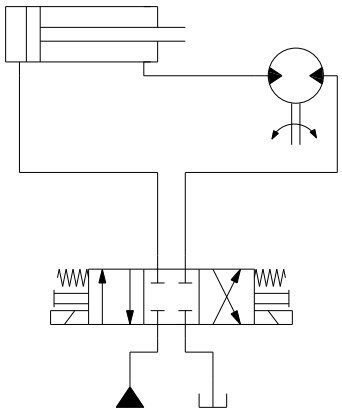

Abbildung 2.18: Hydraulische Achse mit zwei Arbeitselementen in einer Kette

Um die Komplexität dieser Ersetzung so klein wie möglich zu halten, betrachten wir ausschließlich den Fall einer Schleife, in der zwei Komponenten parallel geschaltet sind. Diese Einschränkung der Regel ist zulässig, wenn der Graph keine Ketten mehr beinhaltet, was man durch Anwendung der Regel zur Löschung von Ketten erreicht.

Alternativ zur obigen Regel kann man eine noch einfachere Regel angeben, wenn man alle nicht essentiellen Komponenten vom Grad zwei entfernt, d. h. man löscht alle Ketten der Länge eins. Dies ist mit der Regel zur Entfernung von Knoten vom Grad ≤ 2 aus dem nächsten Abschnitt leicht machbar. Eine kettenfreie Schleife würde dann aus zwei Verbindungskomponenten bestehen, die über zwei verschiedene Kanten miteinander verbunden sind. Die entsprechende Graph-Regel würde wie in Abbildung 2.21 aussehen.

Oft enthalten Schleifen mehr als zwei Verbindungskomponenten, so daß die obige Regel nicht anwendbar wäre. Durch wiederholte Anwendung der Regel zur Verschmelzung benachbarter Verbindungskomponenten (siehe nächsten Abschnitt) werden Verbindungskomponenten solange miteinander verschmolzen, bis nur zwei übrig geblieben sind. Der letzte Schritt kann dann durch Anwendung der obigen Regel realisiert werden oder durch eine weitere Anwendung der Regel zur Verschmelzung von Verbindungskomponenten.

Anzumerken wäre noch, daß Schleifen auch verschachtelt auftreten können. In diesem Fall müssen wir die Regeln zur Ketten- und zur Schleifenentfernung mehrmals anwenden.

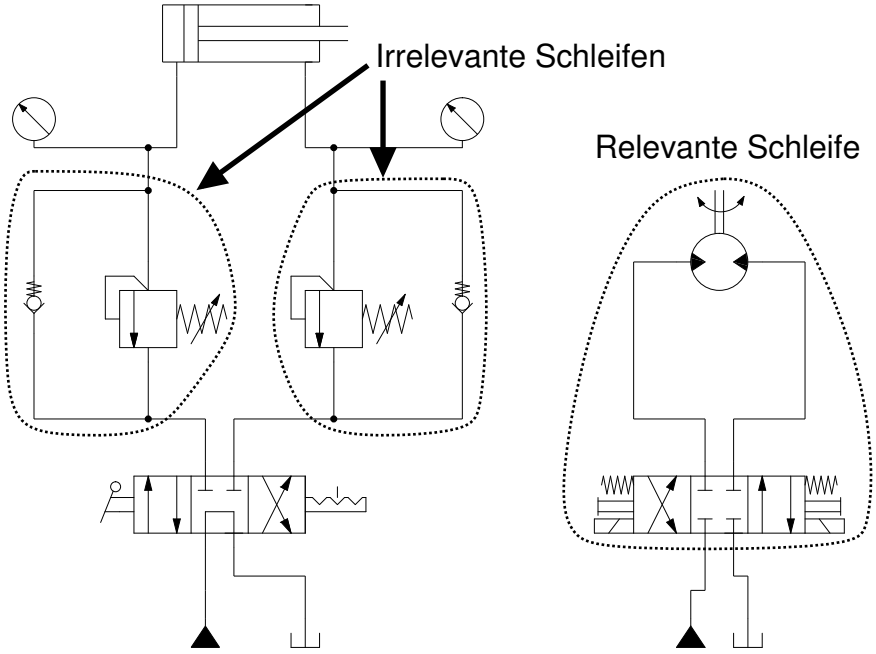

Abbildung 2.19: Entfernbare und nicht entfernbare Schleifen

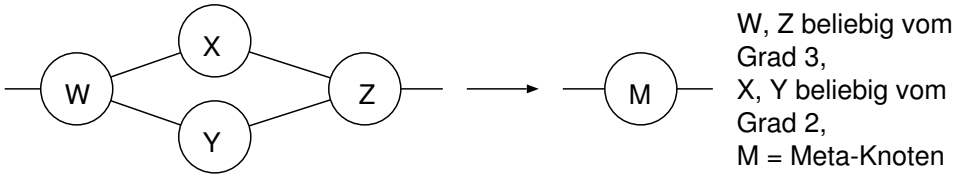

Abbildung 2.20: Produktionsregel zur Ersetzung einer Schleife

2.4.7 Weitere Komprimierungsregeln

Entfernung von Knoten vom Grad ≤ 2

Sind alle Ketten und toten Zweige komprimiert worden, so kann man den übriggebliebenen Knoten in einem letzten Schritt ganz entfernen. Hat man beispielsweise einen toten Zweig komprimiert, so besteht der Zweig aus einem einzigen Knoten vom Grad eins. Dieser Knoten kann dann in einem einfachen Schritt ganz entfernt werden – sind alle solche Knoten entfernt worden, so ist der Graph frei von toten Zweigen, was die Achsenbestimmung ebenfalls vereinfacht. Bei den übrigen Ketten wird die Entfernung des letzten Knotens keine nennenswerte strukturelle Vereinfachung bewirken, allerdings erzielt man dadurch eine Reduzierung der Anzahl der Knoten, was wiederum eine Beschleunigung bzw. eine kürzere Laufzeit für die Achsenbestimmung bedeutet. Im Übrigen ist ein solcher Schritt Voraus-

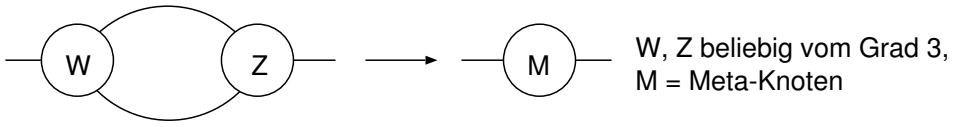

Abbildung 2.21: Produktionsregel zur Ersetzung einer kleinsten Schleife

setzung für die Anwendbarkeit der zweiten Regel zur Schleifenentfernung.

Man beachte, daß die Regeln zur Erkennung und Löschung von Ketten durch die Regel von Abbildung 2.22 simuliert werden kann, aber daß das Umgekehrte nicht gilt.

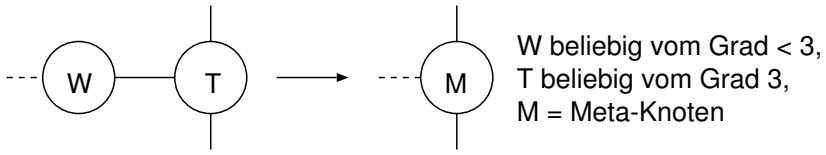

Abbildung 2.22: Produktionsregel zur Entfernung von Knoten vom Grad ≤ 2

Für die spätere Bestimmung der hydraulischen Achsen werden alle gelöschten Knoten mit einem noch vorhandenen Knoten, dem sogenannten *Verankerungsknoten* assoziiert, damit wir die entfernten Knoten nach der Expandierung einer Achse zuordnen können. Bei Knoten vom Grad 2 stellt sich die Frage, mit welchem Nachbarknoten sie verankert werden – es kann leider der Fall eintreten, daß je nach Wahl des Verankerungsknotens unser entfernter Knoten einer falschen hydraulischen Achse zugeordnet wird.

Wir müssen daher bei einem solchen Schritt die Nachbarknoten genauer betrachten, um den richtigen Verankerungsknoten zu wählen. Da jede hydraulische Schaltung Komponenten unterschiedlicher Bedeutung beinhaltet, ist es naheliegend, daß wir den zu entfernenden Knoten dem „wichtigeren“ Nachbarknoten zuordnen, d. h., die Nachbarknoten eines zu entfernenden Knoten besitzen einen gewissen Grad an „Anziehungskraft“ (*Attraktion*). Der Einfachheit halber definieren wir die Attraktionskraft eines Knotens durch den Komponententyp (siehe Abschnitt 1.2.2), zu dem er gehört. Das bedeutet, daß wir vier Attraktionsklassen haben, die folgendermaßen geordnet werden – dabei sei ψ der Maß der Anziehungskraft, die von einer Komponentenklasse ausgeübt wird:

$$\psi(Hilfselement) < \psi(Kontrollelement) < \psi(Versorgungselement) < \psi(Arbeitselement)$$

Es gibt jedoch eine Schwierigkeit bei dieser Verfahrensweise: Liegt ein zu entfernender Knoten zwischen zwei Knoten derselben Komponentenklasse, so können

wir keine eindeutige Wahl des Verankerungsknotens treffen. Dies ist oft der Fall bei komplexen Schaltungen, die viele Verzweigungen enthalten: Eine zu entfernende Komponente liegt zwischen zwei Verbindungskomponenten (siehe Abbildung 2.23). In einem solchen Fall müssen wir auf die Entfernung des Knotens verzichten, es sei denn, wir untersuchen die Kette weiter – dies wäre zwar möglich, würde aber unter Umständen einen Aufwand von $O(|V| + |E|)$ bzw. $O(n)$ für jeden solchen Fall hervorbringen, da wir schlimmstenfalls den gesamten Graphen durchsuchen müßten.

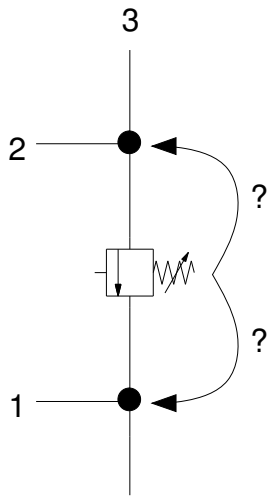

Abbildung 2.23: Ein entfernbares Element zwischen Verbindungskomponenten

Der Verzicht auf die Entfernung solcher Knoten hat aber weitere Konsequenzen – wir können beispielsweise die zweite Grammatik-Regel zur Entfernung von Schleifen nicht mehr anwenden. Wir müssen daher die erste Regel anwenden, die einen geringfügig höheren Aufwand mit sich bringt.

Verschmelzung benachbarter Verbindungskomponenten

Bei der Regel zur Erkennung und Löschung von Ketten gleichartiger Elemente haben wir uns auf Knoten vom Grad zwei beschränkt. In der Tat können wir sogar Knoten von einem höheren Grad miteinander verschmelzen, wobei der so entstandene Knoten nun einen noch höheren Grad besitzt. Dabei zielen wir auf die Verschmelzung von Verbindungskomponenten, in ArtDeco „Triconnections“ genannt, ab. Diese Knoten haben den Grad drei, und die Verschmelzung zweier solcher Knoten resultiert in einem Knoten vom Grad vier. Lassen wir die Hintereinanderausführung dieses Schrittes zu, so erhalten wir bei der Verschmelzung zweier Knoten v_1, v_2 mit $grad(v_1) = h$ und $grad(v_2) = j$ einen Knoten v_3 mit

$grad(v_3) = h + j - 2$, wenn die Knoten nur über eine einzige Kante miteinander verbunden sind.

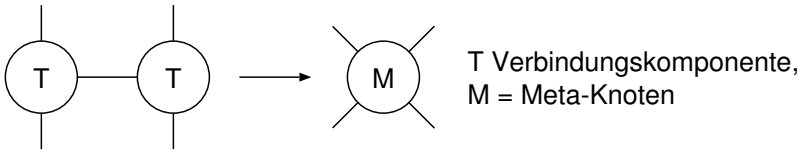

Abbildung 2.24: Regel zur Verschmelzung von Verbindungskomponenten

Man beachte die Tatsache, daß dieser Schritt auch der Vereinfachung von Schleifen dient – siehe dazu Bild 2.25.

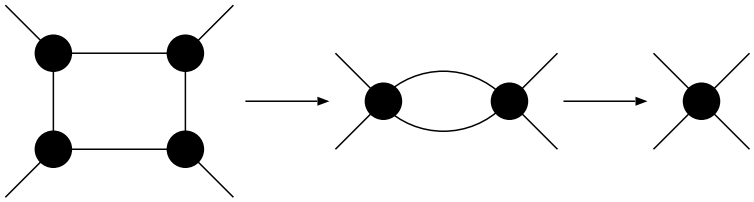

Abbildung 2.25: Entfernung von Schleifen mittels der letzten Regel

Leider liefert die Anwendung dieser Regel nicht immer das gewünschte Ergebnis. Man kann sich leicht ein Beispiel überlegen, bei welchem eine Anwendung dieser Regel die spätere Suche nach den hydraulischen Achsen erschwert – man betrachte zwei benachbarte Triconnections, die zu unterschiedlichen hydraulischen Achsen gehören. Die Verschmelzung zweier solcher Triconnections würde schließlich eine falsche Achsenzuordnung für eine der Triconnections bedeuten. Daher ist die Anwendung dieser Regel mit äußerster Vorsicht durchzuführen, da wir zum Zeitpunkt der Graph-Kontraktion keinerlei Information zur Achsenzuordnung einzelner Elemente haben. Abbildung 2.26 zeigt eine hydraulische Schaltung, bei der die Anwendung dieser Regel falsche Ergebnisse liefern würde.

Obwohl die Anwendung der obigen Regel relativ riskant ist, gibt es Fälle, in denen diese Regel bedenklos benutzt werden kann. Zwei benachbarte Verbindungskomponenten, die über zwei Kanten miteinander verbunden sind, ist beispielsweise ein solcher Fall. Eine weitere Möglichkeit der Anwendung besteht bei Schleifen von solchen Verbindungskomponenten (siehe Abbildung 2.25). Für solche Fälle, in denen immer eine Art Schleife vorhanden ist, kann diese Regel problemlos angewendet werden. Die einzige Hürde, die dann noch zu überwinden ist, ist die Erkennung einer solchen Schleife. Diese Aufgabe kann aber leicht mit einem Algorithmus zur Bestimmung von Blöcken gelöst werden (siehe Abschnitt 2.5.1).

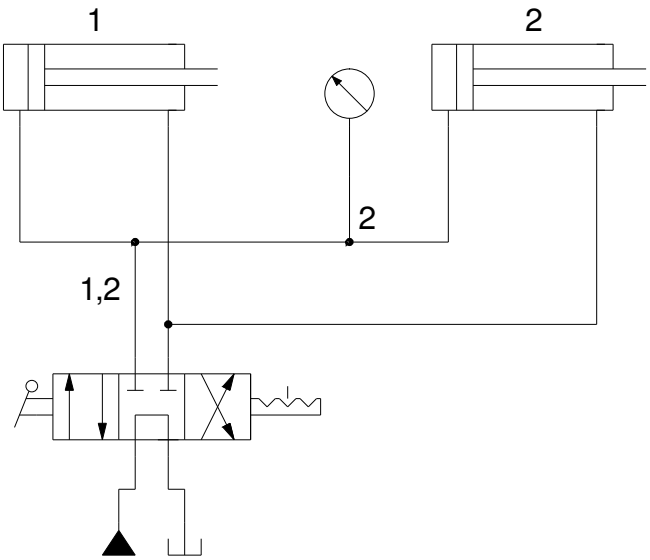

Abbildung 2.26: Benachbarte Verbindungselementen aus verschiedenen Achsen

2.4.8 Reihenfolge der Regelanwendungen

Eine beliebige sequentielle Anwendung der oben beschriebenen Regeln mag eine nicht zu verachtende Vereinfachung des Schaltkreisgraphen bedeuten, nichtsdestotrotz ist in den meisten Fällen die Reihenfolge der Anwendungen sehr wichtig, da man zum Teil kleinere Graphen erhalten kann. Man muß unter Umständen sogar eine wiederholte Anwendung von bestimmten Regeln in Kauf nehmen – man denke beispielsweise an verschachtelten Schleifen: hat man die „innere“ Schleife komprimiert bzw. entfernt, so muß man die so entstandene Kette vorher löschen, bevor man die „äußere“ Schleife in Angriff nehmen kann.

Eine weitere Möglichkeit der Anwendungsstrategie wäre die Feuerung der Regeln, bis keine weitere Anwendung möglich ist – unsere Graph-Kontraktion wäre dann mit einem *Produktionsregelsystem* vergleichbar. Um die Graph-Kontraktion so zu realisieren, müßte man den Regeln Prioritäten zuordnen, und der Reihe nach die Anwendbarkeit der Regeln überprüfen und gegebenenfalls eine Regel feuern. Da aber die Prüfung der Anwendbarkeit in unserem Fall bereits sehr kostspielig ist, müssen wir uns auf eine möglichst einfache Reihenfolge der Regelanwendungen beschränken.

Eine vernünftige Reihenfolge der Regelanwendungen wäre demnach:

1. Ersetzung aller kritischen Komponenten
2. Löschung aller Kontrollkanten

3. Zusammenfassung von Tanks
4. Erkennung und Löschung von Ketten gleichartiger Komponenten
5. Erkennung und Löschung von toten Zweigen
6. Erkennung und Löschung von Ketten irrelevanter Elementen
7. Entfernung von Knoten vom Grad ≤ 2
8. Verschmelzung benachbarter Verbindungskomponenten
9. Erkennung und Löschung von Schleifen. Bei Feuerung dieser Regel wiederhole Verfahren ab Schritt 6.

Abschließend sei an dieser Stelle erwähnt, daß die bisher beschriebene Graph-Kontraktion einen Aufwand von $O(n)$ besitzt. Dies folgt aus der Tatsache, daß jeder Vereinfachungsschritt aus einer linearen Suche durch die Knotenliste besteht, mit gegebenenfalls anschließender Entfernung von Knoten. Auch die Wiederholung der letzten Schritte ist nur bei verschachtelten Schleifen notwendig – da dies aber eine Seltenheit darstellt und, wenn es doch vorkommt, die Verschachtelungstiefe sehr gering ist, ändert das Vorhandensein von verschachtelten Schleifen unseren Aufwand nur geringfügig. Daher ergibt sich im Normalfall für die Graph-Komprimierung ein linearer Gesamtaufwand.

2.5 Weitere Vorverarbeitungsschritte

In diesem Abschnitt möchten wir einige Hilfsschritte erläutern, die unsere Regeln zur Graph-Kontraktion und -Expandierung vereinfachen bzw. erst ermöglichen sollen. Diese Schritte dienen lediglich der Informationsgewinnung und Beschleunigung späterer Schritte und führen keine strukturellen Transformationen auf dem Graphen durch.

2.5.1 Bestimmung der Blöcke

Die Bestimmung der Blöcke des abstrakten Graphen ist ein notwendiger Schritt der Vorverarbeitung. Wir haben im Abschnitt 2.4.7 gesehen, daß die Verschmelzung von benachbarten Verbindungskomponenten sehr gefährlich ist, es sei denn die Verbindungskomponenten bilden eine Schleife. Um diese Regel anwenden zu können, ist es notwendig, daß wir solche Schleifen identifizieren können. Das ist natürlich mit einem Algorithmus zur Bestimmung von Blöcken machbar.

Wir dürfen aber nicht außer Acht lassen, daß wir mit abstrahierten hydraulischen Schaltungen arbeiten, die immer etliche Schleifen beinhalten. Viele von diesen Schleifen aber sollen nicht entfernt werden (siehe Abschnitt 2.4.6), da sie essentielle Komponenten beinhalten. Wir müssen daher den Algorithmus zur Bestimmung der Blöcke leicht verändern, damit wir ausschließlich die entfernbaren Schleifen erkennen. Dies ist ohne weiteres leicht realisierbar, wenn wir vor dem Besuch eines Knotens überprüfen, ob es sich dabei um einen essentiellen Knoten handelt oder nicht. Alternativ dazu kann man auch die Kanten solcher essentieller Komponenten einfach deaktivieren, so daß der Block-Algorithmus sie gar nicht erst besucht.

Wünschenswert wäre ebenfalls Information bezüglich der Verschachtelung der Blöcke, um die Schleifenentfernung gezielt durchführen zu können. Da der Algorithmus zur Bestimmung der Blöcke leider nur „globale“ Informationen liefert, müssen wir ihn auch noch derart ändern, daß er Informationen bezüglich der Blöcke innerhalb bestimmter Teilgraphen liefert. Dies erreichen wir dadurch, daß wir den Algorithmus auf einen vorher bestimmten Block beschränken, aus dem eine Kante entfernt wurde – als Ergebnis erhalten wir alle Blöcke des nicht zweifach zusammenhängenden Teilgraphen. Dieses Verfahren müssen wir gegebenenfalls desöfteren wiederholen, um alle Schleifen zu erkennen.

Auf die Erkennung verschachtelter Blöcke kann man verzichten, wenn keine bestimmte Reihenfolge bei der Komprimierung notwendig ist. Ist dies der Fall, so kann eine Schleife samt allen inneren Schleifen entfernt werden.

Da der Algorithmus eine einfache Erweiterung der Tiefensuche ist, verursacht er einen Aufwand von $O(|E|)$. Durch die wiederholte Anwendung für die Bestimmung von verschachtelten Blöcken erhalten wir jeweils denselben Aufwand nochmals. Da die Bestimmung der Blöcke auf dem unkomprimierten Graphen stattfinden muß, haben wir höchstens den vierfachen Aufwand, denn der Graph besitzt zu diesem Zeitpunkt nur Knoten vom Grad ≤ 4. Also benötigt dieser Schritt insgesamt linearen Aufwand.

Der Algorithmus zur Bestimmung der Blöcke eines Graphen kann in [7] und [13] nachgeschlagen werden, sollten nähere Informationen zur Arbeitsweise und Aufwand benötigt werden.

2.5.2 Kantensortierung

Die Kantensortierung ist ebenfalls ein notwendiger Schritt der Vorverarbeitung und dient der Festlegung der Reihenfolge der Schritte zur Graph-Kontraktion und -Expandierung. Im Abschnitt 3.2 werden wir sehen, daß die Erkennung identischer hydraulischer Achsen sehr problematisch ist. Wir müssen zum Teil während der Graph-Expandierung wiederherzustellende Knoten vergleichen, um

unterschiedliche hydraulische Achsen als solche zu erkennen und von der weiteren Nachverarbeitung auszuschließen.

Die Sortierung der Kanten gewährleistet, daß gleiche Teilschaltungen auf dieselbe Art und Weise komprimiert werden. Da die Kantenlisten aller Knoten nach einem bestimmten Kriterium – beispielweise dem Komponentennamen der benachbarten Knoten – sortiert sind, können wir davon ausgehen, daß bestimmte Komponenten vor anderen wegkomprimiert werden. Dadurch erreichen wir eine sortierte Komprimierung, die eine spätere Bearbeitung während der Expandierung vereinfacht.

Da die Sortierung der Kantenlisten aller Knoten vor der Graph-Kontraktion stattfindet, haben wir einen festen Aufwand für jede Liste, denn alle Knoten haben höchstens vier Kanten – also ist auch das Sortierverfahren irrelevant, und diese Aufgabe kann mit einem beliebigen Sortieralgorithmus erledigt werden. Insgesamt haben wir dann einen Aufwand von $O(|V|)$ für diesen Schritt.

2.5.3 Verwendung von Hilfslisten

Für die Abbildung eines realen Graphen auf Datenstrukturen existieren zwei klassische Varianten: Entweder man verwendet eine Adjazenzmatrix, um den Graphen zu beschreiben, oder man greift auf Adjazenzlisten zurück.

In unserem Fall haben unsere hydraulische Schaltungen viele Knoten, die nur mit wenigen anderen verbunden sind – also ist die zugehörige Adjazenmatrix relativ dünn besetzt. Hinzu kommt die Tatsache, daß unsere Knoten relativ viel Information beinhalten müssen und daher viel Speicherplatz benötigen. Also würde die Verwendung einer Adjazenzmatrix eine enorme Speicherplatzverschwendung mit sich bringen.

Daher müssen wir auf Adjazenzlisten zurückgreifen. Solche verkettete Listen sind im Gegensatz zu den Adjazenzmatrizen viel platzsparender und erlauben eine kompakte und flexible Speicherung der Knoten- und Kanten-Informationen. Leider haben Adjazenzlisten den großen Nachteil, daß man keinen direkten Zugriff auf beliebige Knoten hat, wie es bei Adjazenzmatrizen der Fall ist. Um einen bestimmten Knoten zu erreichen, müssen wir unter Umständen die ganze Adjazenzliste durchsuchen. In unserem konkreten Fall haben wir den Vorteil, daß der Grad des Graphen auf vier beschränkt ist, d. h. die Länge der Kantenlisten ist ebenfalls auf vier beschränkt. Also bleibt der effiziente Zugriff auf bestimmte Knoten des Graphen weiterhin problematisch.

Abhilfe aus diesem Problem schaffen Strukturen wie Hash-Tabellen. Hash-Tabellen sind, wie die Adjazenzmatrizen, auf eine bestimmte Große beschränkt, die man bei der Initialisierung festlegen kann, und sie dienen dem schnelleren Zugriff

auf bestimmte Daten in einer komplexen bzw. großen Struktur. Der Zugriff auf die Daten läuft über Schlüsseln, die man als Argument der Hash-Funktion benutzt.

Daher greifen wir auch hier wieder auf verkettete Listen zurück und verwalten eine kleine Anzahl an Listen spezieller Komponenten, und erreichen damit eine ähnliche Wirkung wie bei Hash-Tabellen. Wir setzen diese Hilfslisten ein für jede wichtige Klasse von Komponenten, wie zum Beispiel die Arbeitselemente. Auf diese Weise können wir gezielt alle Komponenten einer bestimmten Art bearbeiten, ohne den gesamten Graphen durchsuchen zu müssen. Daher können einige Vereinfachungsschritte – wie die Verschmelzung benachbarter Arbeitselemente, um nur ein Beispiel zu nennen – erheblich schneller durchgeführt werden. Die einzige zusätzliche Arbeit, die man mit Hash-Tabellen oder -Listen hat, ist der Aufbau der Liste. Bei uns ist aber der Aufbau der Hilfslisten ein Nebenprodukt der Einlese-Routine, so daß dieser Schritt keinen nennenswerten Aufwand mit sich bringt.

Allerdings ist Vorsicht bei der Verwendung solcher speziellen Listen geboten, denn bei jeder Änderung des ursprünglichen Graphen müssen wir gegebenenfalls auch die Hilfslisten aktualisieren. Vergißt man eine solche Aktualisierung vorzunehmen, so ist der Zustand der Datenstrukturen inkonsistent und die Algorithmen arbeiten nicht korrekt bzw. gar nicht.

Kapitel 3

Erkennung der Achsen durch Pfadsuche

In diesem Kapitel wollen wir ein Verfahren angeben, mit dem die hydraulischen Achsen einer hydraulischen Schaltung identifiziert werden können. Wir haben im Abschnitt 1.2.3 gesehen, daß man, wenn der Graph einen bestimmten Einfachheitsgrad hat, die Achsen mittels Pfadsuche bestimmen kann. Wir wollen also diesen Lösungsweg etwas näher untersuchen und anschließend der Frage nach identischen hydraulischen Achsen nachgehen. Schließlich wollen wir auf die noch ungelösten Probleme und die davon betroffenen Fälle eingehen.

Für alle nachfolgenden Abschnitte sei der zu untersuchende Graph bereits komprimiert.

3.1 Pfadsuche im Schaltkreisgraphen

Wir haben im Abschnitt 1.2.2 hydraulische Achsen genauer definiert, und zwar sind hydraulische Achsen funktionelle Gruppen einer hydraulischen Schaltung, die mindestens ein Arbeitselement und mindestens ein Versorgungselement besitzen. Darüberhinaus sind die Arbeitselemente einer Achse mit allen Versorgungselementen derselben Achse über Wege verbunden, und alle auf diesen Wegen liegenden Komponenten gehören ebenfalls zur selben Achse. Da offensichtlich die Begriffe „Wege“ und „Pfade“ sehr eng mit dem Begriff der „hydraulischen Achse“ zusammenhängen, ist es naheliegend, daß wir die Aufgabe der Erkennung hydraulischer Achsen durch Pfadsuche lösen.

Zu unserem Glück existieren bereits einige sehr effiziente Pfadsuchalgorithmen, so daß wir das Rad nicht neu erfinden müssen. In [7] und [13] werden die bekanntesten Pfadsuchalgorithmen vorgestellt, sowie deren Aufwand und Korrektheit gezeigt. Einige dieser bekannten Algorithmen sind:

- *Dijkstra's Algorithmus*: Dieser Algorithmus ist dafür entwickelt worden, den kürzesten Weg zwischen zwei Knoten eines Graphen mit gewichteten Kanten zu bestimmen (siehe [7]). Unser Graph besitzt zwar keine gewichteten Kanten, aber das ist an dieser Stelle vernachlässigbar. Der Algorithmus von Dijkstra verursacht einen Aufwand von $O(|V|^2)$, der laut [7] allerdings auf $O(|E| \cdot log(|V|))$ reduziert werden kann, wenn man Heaps verwendet.

- *Floyd's Algorithmus*: Dieser Algorithmus berechnet induktiv alle Wege zwischen allen Knoten. Die Idee hinter dem Algorithmus von Floyd besteht aus der Aufteilung des Problems in Teilprobleme, die, wenn sie gelöst worden sind, die Lösung zum eigentlichen Problem liefern. Der Algorithmus verfolgt wegen seines induktiven Vorgehens das Programmierparadigma des Dynamischen Programmierens. Daher hat er eine Laufzeit von $O(|V|^3)$ – und zusätzlich dazu verursacht er einen Platzaufwand von mindestens $O(|V|^2)$ laut [7].

- *Ford's Algorithmus*: Der Algorithms von Ford ähnelt dem Algorithmus von Floyd in seiner Arbeitsweise: hier wird das Problem ebenfalls in kleinere Teilprobleme aufgeteilt, allerdings geschieht das implizit. Der Algorithmus baut beginnend von einem Knoten v einen Baum kürzester Wege zu allen anderen Knoten und verursacht dabei eine Laufzeit von $O(|V| \cdot |E|)$ bzw. $O(n^2)$.

Wenn wir unsere Aufgabe und die Definition einer hydraulischen Achse näher betrachten, stellen wir fest, daß wir lediglich *alle* Pfade, die *alle* Arbeitselemente mit *allen* Versorgungselementen verbinden, untersuchen müssen. Daher müssen wir unser Problem in verschiedenen Durchläufen lösen. D. h., wir betrachten zunächst alle Pfade, die alle Arbeitselemente mit einem bestimmten Versorgungselement verbinden, bevor wir ein anderes Versorgungselement in die Betrachtung aufnehmen. Diese Teilaufgabe können wir sehr leicht mit Floyd's oder Ford's Algorithmus lösen – da Floyd's Algorithmus einen hohen Platzaufwand besitzt, werden wir Ford's Algorithmus verwenden.

Um also den gesamten Graphen zu untersuchen, berechnen wir alle Pfade von einem Versorgungselement zu allen Arbeitselementen – dies tun wir wiederholt für jedes weitere Versorgungselement. Der Aufruf von Ford's Algorithmus produziert einen gerichteten Baum, dessen Kanten alle in Richtung des Versorgungselements zeigen. Diese Nachfolgerbeziehung können wir dazu nutzen, alle auf einem solchen Pfad liegenden Komponenten einer bestimmten Achse zuzuordnen.

Da unsere Schaltungen Multigraphen sind, müssen wir von den Arbeitselementen aus zwei verschiedene Wege zu einem Versorgungselement finden. Dazu müssen wir den Algorithmus von Ford erneut anwenden, wobei wir zuvor eine Kante eines jeden Arbeitselements vorläufig „ausschalten“ müssen, damit der Algorithmus

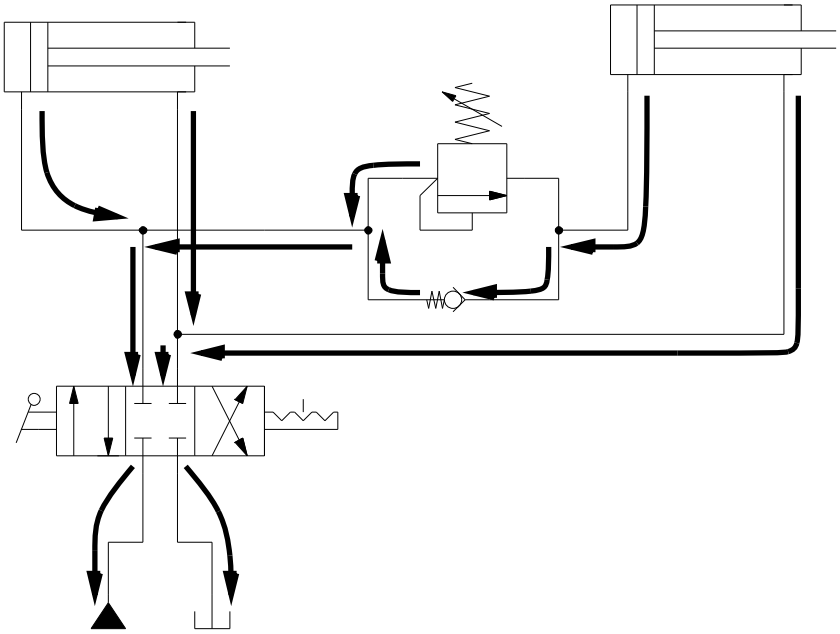

Abbildung 3.1: Eine Schaltung mit Nachfolgerinformation

den zweiten Weg finden kann. Anschließend verfolgen wir beide Wege, um die betroffenen Knoten einer Achse zuzuordnen.

Ein Detail, welches wir bisher nicht beachtet haben, ist die Tatsache, daß in einer hydraulischen Schaltung der von einem Versorgungselement produzierte Druck irgendwo aus der Schaltung fließen muß. Diese Aufgabe wird ausschließlich von Tanks erledigt. Wenn nun eine bestimmte Achse eingeschaltet ist, so sind nicht nur alle Komponenten auf dem Weg zwischen Arbeitselement und Versorgungselement eingeschaltet, sondern auch alle Komponenten zwischen Arbeitselement und Tank. Aus diesem Grund müssen wir diese Wege zwischen Arbeitselementen und Tanks ebenfalls finden, um die dort liegenden Komponenten der richtigen Achse zuordnen zu können. Dazu wenden wir den Algorithmus von Ford erneut an, wobei er diesmal alle Pfade zwischen den Arbeitselementen und einem Tank bestimmen soll.

Es ergibt sich dann folgender Algorithmus zur Bestimmung aller Achsen:

Algorithmus zur Bestimmung der hydraulischen Achsen

- **Input**: Komprimierter Graph G, Hilfslisten für Versorgunselemente und Hilfslisten für Tanks.
- **Output**: Komprimierter Graph G', dessen Knoten Achseninformationen enthalten.

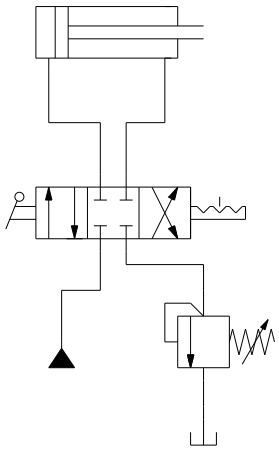

Abbildung 3.2: Eine Schaltung mit nicht-trivialem Tank-Zweig

1. Für jedes Versorgungselement v tue:

   (a) Berechne alle Pfade beginnend von v zu den Arbeitselementen mit dem Algorithmus von Ford

   (b) Ordne alle Knoten auf den Pfaden einer Achse zu

   (c) Schalte eine Kante aller Arbeitselemente aus

   (d) Berechne erneut alle Pfade beginnend von v zu den Arbeitselementen mit dem Algorithmus von Ford

   (e) Ordne alle Knoten auf den Pfaden einer Achse zu

2. Für jeden Tank t tue:

   (a) Berechne alle Pfade beginnend von t zu den Arbeitselementen mit dem Algorithmus von Ford

   (b) Ordne alle Knoten auf den Pfaden einer Achse zu

   (c) Schalte eine Kante aller Arbeitselemente aus

   (d) Berechne erneut alle Pfade beginnend von t zu den Arbeitselementen mit dem Algorithmus von Ford

   (e) Ordne alle Knoten auf den Pfaden einer Achse zu

Wie man leicht erkennt, ist der obige Algorithmus sehr zeitintenisv, da der Algorithmus von Ford mehrfach benutzt wird und die gefundenen Pfade anschließend noch durchsucht werden müssen. Wir erhalten folgende Aufwände für die einzelnen Schritte des obigen Algorithmus:

1. $O(|V|)$ Schleifendurchläufe
   (a) $O(|V| \cdot |E|)$ Aufwand für Ford's Algorithmus
   (b) $O(|V| \cdot |E|)$ Aufwand für die Untersuchung aller Pfade
   (c) $O(|V|)$ Aufwand für die Ausschaltung der Kanten
   (d) $O(|V| \cdot |E|)$ Aufwand für Ford's Algorithmus
   (e) $O(|V| \cdot |E|)$ Aufwand für die Untersuchung aller Pfade

   Gesamtaufwand der Schleife: $O(|V| \cdot (|V| \cdot |E|))$

2. $O(|V|)$ Schleifendurchläufe
   (a) $O(|V| \cdot |E|)$ Aufwand für Ford's Algorithmus
   (b) $O(|V| \cdot |E|)$ Aufwand für die Untersuchung aller Pfade
   (c) $O(|V|)$ Aufwand für die Ausschaltung der Kanten
   (d) $O(|V| \cdot |E|)$ Aufwand für Ford's Algorithmus
   (e) $O(|V| \cdot |E|)$ Aufwand für die Untersuchung aller Pfade

   Gesamtaufwand der Schleife: $O(|V| \cdot (|V| \cdot |E|))$

Insgesamt ergibt sich ein asymptotischer Laufzeitaufwand von $O(|V| \cdot (|V| \cdot |E|))$ bzw. $O(n^3)$ für den obigen Algorithmus zur Bestimmung der hydraulischen Achsen.

Nun könnte man sich die Frage stellen, was wäre, wenn wir den Algorithmus von Dijkstra benutzt hätten. Diese Frage läßt sich dadurch beantworten, daß wir den Algorithmus von Ford einfach durch den Algorithmus von Dijkstra ersetzen bzw. versuchen, das gleiche Ergebnis mit dem Algorithmus von Dijkstra zu erzielen.

Eine wiederholte Anwendung von Dijkstras Algorithmus auf alle Arbeitselemente des Graphen würde eine Laufzeit von $O(|V| \cdot (|E| \cdot log(|V|)))$ bzw. $O(n^2 \cdot log(n))$ bedeuten. Diese naive Simulation hat offensichtlich eine schlechtere Laufzeit als Ford's Algorithmus, welcher die Wege zu allen Arbeitselementen in quadratischer Zeit bestimmt.

Durch eine Änderung des Algorithmus von Dijkstra könnte man dieselben Ergebnisse produzieren wie der Algorithmus von Ford. Man müßte den Algorithmus von Dijkstra dazu zwingen, alle Knoten des Graphen in die Berechnung miteinzubeziehen – diese Variante des Algorithmus von Dijkstra wird in [4] detailliert behandelt. Eine solche Änderung würde dasselbe Laufzeitverhalten hervorbringen wie bei Ford's Algorithmus. Benutzt man allerdings einen Heap anstatt einer Liste in der Implementierung von Dijkstra's Algorithmus, so kann man doch ein besseres Laufzeitverhalten erwarten: $O(|E| \cdot log(|V|))$.

Leider besitzt der oben angegebene Algorithmus zur Bestimmung der hydraulischen Achsen noch Schritte, die einen Aufwand von $O(|V| \cdot |E|)$ in Anspruch nehmen, so daß der Gesamtaufwand unverändert bei $O(|V| \cdot (|V| \cdot |E|))$ bzw. $O(n^3)$ bleibt.

3.2 Nachverarbeitung

Als letzte zur Bestimmung der hydraulischen Achsen gehörende Teilaufgabe verbleibt die Erkennung der identischen Achsen. Diese Aufgabe ist relativ schwer zu lösen, da man auf dem unkomprimierten Graphen arbeiten muß – nichtsdestotrotz versuchen wir auch hierfür eine akzeptable Lösung zu finden.

Zunächst betrachten wir die Definition von identischen Achsen. Identische Achsen sind in unserem Fall hydraulische Achsen, die man nicht voneinander unterscheiden kann: Sie besitzen die gleichen Komponenten, die gleiche Struktur und verrichten dieselbe Arbeit. Weil solche Achsen sich wie eine einzige Achse verhalten, dürfen sie nicht von verschiedenen Kontrollelementen gesteuert werden, d. h. identische Achsen haben neben Kontrollelementen einige bis viele Komponenten gemeinsam und liegen daher offensichtlich in derselben Zusammenhangskomponente.

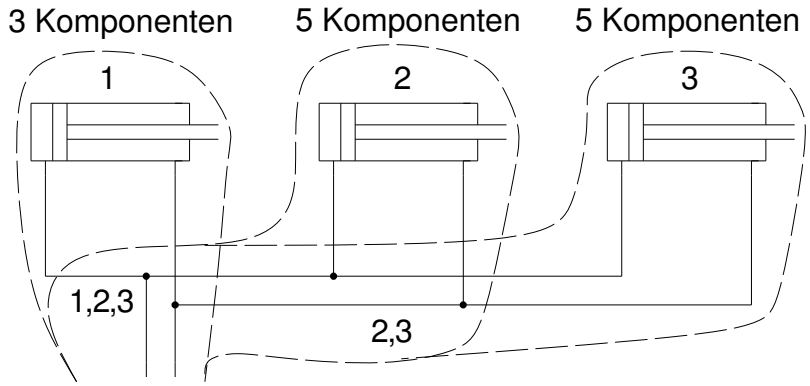

Abbildung 3.3: Identische Achsen mit unterschiedlicher Anzahl Komponenten

Gerade die obige Definition identischer Achsen stellt eine große Hürde dar, denn wir müssen nicht nur die Komponenten beider Achsen vergleichen, sondern auch noch deren Struktur. D. h. wir müssen bei einem Vergleich zweier Achsen die Position einer jeden Komponente einer Achse mit der Position der entsprechenden Komponente der anderen Achse vergleichen.

Dabei müssen wir flexibel sein, wenn wir Verbindungskomponenten betrachten, denn sie können in unterschiedlicher Anzahl auftreten (siehe Abbildung 3.3). Sol-

che Verbindungskomponenten haben aber eine bestimmte Eigenschaft: Sie verbinden Achsen miteinander, d. h. sie treten nur dort auf, wo zwei Achsen ineinander übergehen und stellen somit die ersten gemeinsamen Komponenten dar.

Aus diesen ersten Überlegungen können wir bereits einige Probleme erkennen:

- Der Vergleich der Komponenten kann nur im unkomprimierten Graphen geschehen, da wir alle Komponenten überprüfen müssen.
- Zwei Achsen haben unter Umständen eine unterschiedliche Anzahl von Komponenten, sind aber dennoch identisch.
- Der Strukturvergleich impliziert eine Untersuchung der Nachbarschaft eines jeden Knoten, was den Aufwand beträchtlich erhöht. Dabei müssen wir auch Strukturen unterschiedlicher Topologien, die das gleiche Verhalten verursachen, als identisch befinden können.

Um das erste Problem zu beseitigen expandieren wir unseren Graphen, damit wir wieder alle Knoten zur Verfügung haben. Hieraus entstehen aber wiederum weitere Schwierigkeiten: Auf der einen Seite haben wir, je nach Aufbau der hydraulischen Achsen, komplexe Strukturen wie tote Zweige oder sogar Schleifen, was eine einfache Bearbeitung erschwert; auf der anderen Seite verlieren wir alle Informationen bezüglich der Nachfolgerbeziehungen, die wir während der Kontraktion gewonnen haben, so daß eine gezielte und „gerichtete“ Untersuchung nicht mehr möglich ist.

Das zweite Problem stellt keine aufwendige Aufgabe dar, denn wir können die betroffenen Verbindungskomponenten leicht erkennen: Wir müssen nur auf die ihnen zugewiesenen Achsennummern achten. Auf diese Art und Weise können wir eine gesonderte Behandlung beginnen, sobald eine solche Komponente gefunden wurde.

Eines der oben erwähnten Probleme können wir zum Glück schon jetzt eliminieren: tote Zweige. Da tote Zweige eine Art paralleler Schaltung darstellen, haben sie auf alle erreichbaren Achsen Einfluß. Daher können wir bei der Erkennung identischer Achsen auf die Untersuchung toter Zweige verzichten, da beide Achsen von toten Zweigen gleichermaßen betroffen werden.

Das dritte Problem stellt für uns das größte Hindernis dar, denn nach der Expandierung des Graphen verlieren wir die einfache Struktur des komprimierten Graphen, so daß jetzt mehr Arbeit zu leisten ist.

Es bieten sich hier verschiedene Möglichkeiten, um diese Aufgabe bei diesen Voraussetzungen zu lösen. Zwei davon werden wir in den nächsten Unterabschnitten behandeln und näher untersuchen. Zunächst aber wollen wir die Möglichkeit einer Voruntersuchung in Betracht ziehen.

3.2.1 Voruntersuchung der hydraulischen Achsen

Im komprimierten Graphen besitzen alle hydraulischen Achsen eine relativ einfache Form. In diesem Zustand sind sie leicht zu untersuchen, daher ist es naheliegend, daß man vor der Expandierung bereits unterschiedliche Achsen vom späteren Vergleich ausschließt. Dadurch reduziert sich die Anzahl der später zu untersuchenden Fälle.

Ein weiterer Grund für eine Untersuchung zu diesem Zeitpunkt beruht auf der Tatsache, daß wir für den komprimierten Graphen noch Informationen bezüglich der Nachfolgerbeziehungen zwischen den Knoten besitzen. Also ist ein einfacher, gerichteter Vergleich zweier Achsen möglich.

Daher vergleichen wir die Achsen noch im komprimierten Zustand, indem wir beginnend bei den Arbeitselementen die Nachfolgerbeziehung ausnutzen, um alle Komponenten der Achsen schrittweise miteinander zu vergleichen. Da identische Achsen auch im komprimierten Zustand identisch sind, können wir jede Diskrepanz beim Vergleich zweier Achsen als ein Zeichen für deren Ungleichheit deuten. In den meisten Fällen werden wir hiermit einen großen Teil der möglichen Kombinationen von Achsenpaaren als nicht identisch befinden und damit einiges an Zeit während der späteren Untersuchungen sparen.

Wir fragen uns, bevor wir mit den genaueren Untersuchungen fortfahren, wieviel diese Voruntersuchung eigentlich an Aufwand kostet. Wenn wir annehmen, daß unser Graph k Achsen besitzt, so sind es $\binom{k}{2} = \frac{k(k-1)}{2} \in O(k^2)$ mögliche Achsenpaare, die untersucht werden müssen. Für den Vergleich eines solchen Paares benötigen wir lineare Zeit, da wir die von den Nachfolgerbeziehungen definierten Pfade einfach durchgehen müssen. Also haben wir im schlimmsten Fall einen Aufwand von $O(|V|^2 \cdot |E|) \in O(n^3)$ für die Voruntersuchung.

3.2.2 Schrittweise Expandierung

Die Idee, auf der die schrittweise Expandierung basiert, ist die folgende: Wir fügen wegkomprimierte Knoten vereinzelt wieder in unseren Graphen ein, so daß wir immer nur zwei einzufügende Knoten miteinander vergleichen müssen.

Dieser schrittweise Vergleich setzt allerdings voraus, daß alle Knoten in einer vordefinierten Ordnung komprimiert wurden. Im Abschnitt 2.5.2 haben wir den Vorverarbeitungschritt der Kantensortierung behandelt, mit dem wir eine sortierte Kontraktion erreichen. Da die entfernten Knoten auf einem Stapel liegen, werden sie in der umgekehrten Reihenfolge, also ebenfalls sortiert, wieder in den Graphen eingefügt.

Leider ist es so, daß zwei auf dem Stapel aufeinander liegende Knoten nicht unbedingt miteinander vergleichbar sind, da sie unter Umständen nicht vom selben

Typ sind – das kommt daher, daß bei der Kontraktion sich die Eigenschaften mancher Knoten ändern, wie zum Beispiel der Inhalt der Kantenliste: Bei Entfernung eines Knotens vom Grad zwei wird dieser Knoten aus den Kantenlisten seiner Nachbarn gelöscht, und diese fügen jeweils eine Kante zum anderen Nachbarn in ihre eigenen Kantelisten ein, was meistens die Sortierung durcheinanderbringt. Auch eine nachträgliche Sortierung dieser Kantenlisten dürfte keine Abhilfe schaffen, denn das eigentliche Problem stammt aus der Struktur des Graphen. Hinzu kommt, daß eine Sortierung nach jeder Operation notwendig wäre, was den Aufwand leider doch erhöht, wenn auch nur geringfügig.

Ein weiteres Problem hat man, sobald zwei möglicherweise identische Achsen mehrere Komponenten eines Typs besitzen. Unter Umständen werden dann identische Knoten gefunden, die aber nicht in vergleichbaren Kontexten liegen, d. h. sie gehören unter Umständen zu unterschiedlichen Teilen der Achsen, so daß keine konkrete Aussage gemacht werden kann. Man muß in einem solchen Fall diese Knoten vorläufig in eine Liste oder auf einen Stapel ablegen, bis man den passenden Knoten in einem späteren Expandierungsschritt zur Verfügung hat.

Daher ist die Verwirklichung dieser Strategie mit einem erhöhten Speicherplatzverbrauch verbunden. Besitzt der Schaltungsgraph h Achsen, so sind auch h Listen bzw. Stapel zu verwalten, die alle zusammen aber nicht mehr Platz in Anspruch nehmen als der gesamte Graph. Der Zeitaufwand bei diesem Verfahren ist dagegen angenehm klein, da man beim Einfügen eines Knotens nur seine Nachbarn und gegebenenfalls die Knoten in der entsprechenden Liste kontrollieren muß. Daher kommt es bei jedem Expandierungsschritt zu einem Mehraufwand von $O(deg(v) + |E|) \in O(|E|)$ für einen Knoten v. D. h. für die Expandierung als ganzes, daß sie schlimmstenfalls um den Faktor $O(|E|)$ bzw. $O(n)$ langsamer wird.

3.2.3 Rekursive Dursuchung

Sollten die bisherigen Maßnahmen versagen, so bleibt einem nichts anderes übrig, als möglicherweise identische Achsen auf dem expandierten Graphen miteinander zu vergleichen. Wir haben aber bereits erkannt, daß wir keinerlei gültige Informationen zur Navigation im Graphen besitzen, da bei der Expandierung alle Nachfolgerbeziehungen verloren gegangen sind. Wir müssen also die hydraulischen Achsen ohne diese Hilfskonstrukte durchsuchen.

Wir können allerdings ähnlich wie mit den Nachfolgerbeziehungen vorgehen, indem wir die Knoten einer hydraulischen Achse einfach der Reihe nach besuchen. Dazu müssen wir uns lediglich merken, woher wir gerade kommen, d. h. welchen Knoten wir gerade verlassen haben, damit wir nicht wieder den bereits besuchten Weg durchgehen. Ist eine Achse schleifenfrei, so können wir einfach und ohne

Mehraufwand die Knoten hintereinander besuchen und landen zwangsläufig bei einer Komponente, die auch einer anderen Achse gehört – damit wäre die Arbeit erledigt.

Sind in unseren hydraulischen Achsen Schleifen oder tote Zweige – also irgendwelche achseninterne Verzweigungen – enthalten, so müssen wir mehrere Wege gleichzeitig betrachten. Genau an dieser Stelle kommt die Rekursion ins Spiel: Liegt ein solcher Fall vor, so fangen wir unsere Prozedur beginnend bei dem aktuellen Knoten noch einmal an und lassen sie den Zweig überprüfen. Mit anderen Worten führen wir eine eingeschränkte Tiefensuche durch.

Eine kleine Schwierigkeit kann allerdings bei dieser Vorgehensweise auftreten: Wenn in einer Verzweigung die beiden Nachfolger vom gleichen Komponententyp sind, so können wir nicht entscheiden, welchen Zweig zu folgen ist (siehe dazu Abbildung 3.4). Die einzige Lösung zu diesem Problem besteht daraus, die Nachfolger der Nachfolger zu vergleichen, um den richtigen Weg zu finden. Wir müssen aber möglicherweise diesen Lookahead-Schritt mehrmals wiederholen, was den Aufwand erhöht.

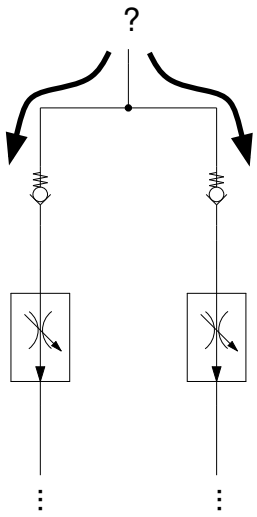

Abbildung 3.4: Entscheidungsproblem: Welcher Zweig soll verfolgt werden?

Im Gegensatz zur Variante der schrittweisen Expandierung verbraucht die rekursive Durchsuchung keinen zusätzlichen Speicherplatz. Auch was der Zeitaufwand angeht ist dieses Verfahren nicht allzu schlecht: Bei schlimmstenfalls $O(|V|^2)$ bzw. $O(n^2)$ zu untersuchenden Achsenpaare wäre höchstens $O(|V|^2 \cdot |E|)$ bzw. $O(n^3)$ Zeit nötig, falls keine entartete Fälle auftreten, wo viele Lookahead-Schritte notwendig sind. Angenommen, alle Achsen wären so entartet, dann würden wir $O(n^4)$ Zeit für die rekursive Durchsuchung benötigen. Da solche Fälle äußerst unwahrscheinlich sind, können wir davon ausgehen, daß diese Methode die Aufgabe in

höchstens kubische Zeit erledigt.

3.3 Mögliche Verbesserungen

In diesem Abschnitt wollen wir kurz auf einige mögliche Verbesserungen, die allerdings nicht ohne weiteres realisierbar sind, eingehen und einen Ansatz für eine Verwirklichung angeben.

3.3.1 Entfernung redundanter Komponenten

Am Anfang dieses Kapitels haben wir einen Algorithmus zur Bestimmung der hydraulischen Achsen angegeben. Dieser Algorithmus besteht aus zwei Schleifen, die jeweils für jedes Vorkommen eines Versorgungselements bzw. eines Tanks durchlaufen werden. Einige dieser Komponenten sind für die Erkennung der Achsen irrelevant, da sie keine neue Informationen liefern. Ein Beispiel für eine solche Komponente ist eine Pumpe, die unmittelbar in der Nähe einer anderen Pumpe liegt (siehe Abbildung 3.5).

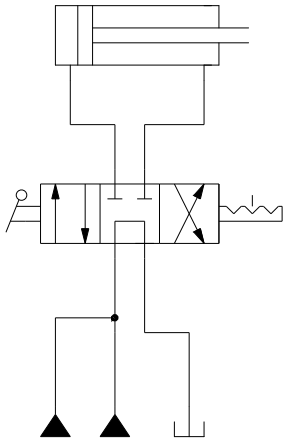

Abbildung 3.5: Eine Schaltung mit einer redundanten Pumpe

Solche Fälle sind aber schwer zu erkennen, da man dafür wiederum die Information aus dem oben genannten Algorithmus benötigt. Eine Entfernung solcher Knoten wäre aber sehr wünschenswert, da man dann die Schleifen des Algorithmus nicht so oft durchlaufen müßte, was in einer kürzeren Laufzeit resultiert.

Bei den Tanks haben wir etwas mehr Glück – unsere Graph-Grammatik ist in der Lage, einige Vorkommen dieser Komponente aus dem Graphen zu entfernen.

Nichtsdestotrotz können manche Tanks unserer Graph-Grammatik „etwas vormachen“, so daß sie leider nicht entfernt werden (siehe Abbildung 3.6). Das liegt daran, daß manche Produktionsregel in bestimmten Fällen nicht greifen darf, da sonst zuviel aus dem Graphen entfernt wird. Leider können Tanks, die sich in einem solchen Kontext befinden, trotzdem für die Berechnung irrelevant sein.

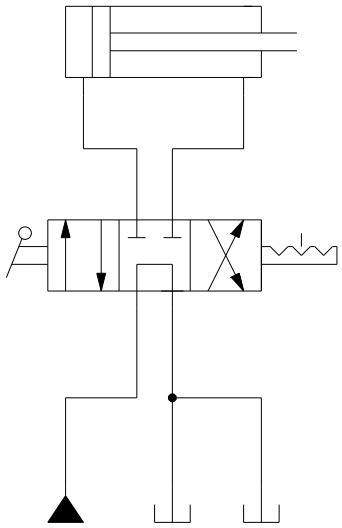

Abbildung 3.6: Eine Schaltung mit einem redundanten Tank

In beiden oben beschriebenen Fällen überschreiten die Lagen der Komponenten die „Reichweite“ der Grammatik-Regeln, so daß solche Fälle leider nicht erkannt werden. Zur Abhilfe müßte man die Produktionsregeln der Graph-Grammatik erweitern, so daß sie nicht mehr dermaßen lokal beschränkt, sondern etwas globaler sind – aber dann würden sie schon langsam in den Bereich der Schablonen-Einbettungen kommen (siehe Kapitel 4).

3.3.2 Erhaltung der Nachfolgerinformation

Bei der Erkennung identischer Achsen haben wir leider auf die Nachfolgerinformation, die wir bei der Pfadsuche gewonnen hatten, verzichten müssen. Diese Information wäre beispielsweise für die rekursive Durchsuchung sehr wertvoll, denn uns wären einige komplizierte Vergleiche erspart geblieben. Auch für manche entartete Achse, bei der wir viele Lookahead-Schritte hätten machen müssen, wären diese Informationen vorteilhaft – damit würden wir fast immer auf Lookahead-Schritte verzichten können. Lookahead-Schritte wären dann nur noch für Schleifen mit gleich langen „Hälften“ notwendig; für alle anderen Schleifen und alle toten Zweige würden uns die Nachfolgerinformationen ausreichen.

Leider gibt es keine einfache Lösung zu diesem Problem. Um diese Nachfolgerinformationen auf dem unkomprimierten Graphen zu haben, hätten wir leider den Algorithmus von Ford zur Pfadsuche einige Male durchführen müssen, was einen zu großen Zeitaufwand bedeuten würde, da der Graph nun unkomprimiert ist.

Eine andere Alternative dazu wäre eine bei der Expandierung schrittweise Ergänzung der Nachfolgerinformation. Leider ist auch in diesem Fall der Aufwand groß, da man beim Einfügen eines Knotens alle seine Nachbarn näher betrachten und gegebenenfalls Nachfolgerinformation ändern muß – die Expandierung würde um mindestens einen Faktor $O(|E|)$ bzw. $O(n)$ verlangsamt. Hinzu kommt die Tatsache, daß für Komponenten eines toten Zweiges keine Nachfolgerinformation vorhanden ist – nur die Nachfolgerinformation von Komponenten, die einen Vorgänger und einen Nachfolger besitzen, kann wiederhergestellt werden.

Diese Variante ist aber der oben beschriebenen *Brute-Force*-Methode vorzuziehen, da der Aufwand hierfür ein wenig erträglicher ist – allerdings sollte man diese Variante nicht zusammen mit der Methode der schrittweisen Expandierung verwenden, da wir den Aufwand für die Expandierung zu sehr erhöhen würden.

3.4 Ungelöste Probleme

Obwohl unsere Algorithmen, zumindest in der Theorie, bereits eine gute Arbeit leisten, sind sie nicht in der Lage, alle Probleme der Erkennung von hydraulischen Achsen zu lösen. Wir wollen im folgenden einige dieser noch ungelösten Probleme betrachten und auf mögliche Lösungen hinweisen.

3.4.1 Einstellbare Komponenten

Eine hydraulische Schaltung besitzt in der Regel mehrere Möglichkeiten zur Einstellung: Man kann die Schaltstellung der Ventile ändern oder auch die Eigenschaften bestimmter Komponenten. Solche Komponenten besitzen beispielsweise einen einstellbaren Widerstand.

In der Regel stellen solche Komponenten keine größere Probleme dar, aber bei der Erkennung identischer Achsen können sie jedoch den Vorgang stören bzw. erschweren. Angenommen, eine hydraulische Schaltung besitzt zwei identische Achsen, die wiederum jeweils mindestens eine solche Komponente besitzen. Sind diese einstellbaren Komponenten gleich eingestellt, so sind die Achsen identisch; sind sie aber unterschiedlich eingestellt, so sind die Achsen verschieden, auch wenn die Komponenteneinstellungen nur minimal voneinander abweichen.

Glücklicherweise ist die Behandlung dieser Fälle mit keinem nennenswerten Mehraufwand verbunden. Die konkreten Einstellwerte solcher Komponenten können

ohne weiteres beim Einlesen der Schaltungsdatei abgefragt werden, und beim Vergleichen zweier identischer Achsen ist lediglich eine weitere Abfrage nötig.

Der einzige Aufwand, den man bei diesen Komponenten hat, ist, daß man für jede solche Komponente eine gesonderte Behandlung einführen muß. Ideal wäre eine Zusammenfassung solcher Komponenten unter einer Gruppe bzw. das Hinzufügen einer abstrakten Eigenschaft „Einstellwert“ für die Datenstrukturen unserer Knoten.

3.4.2 Nicht trivial erkennbare Kontrollkanten

Wie wir im Abschnitt 2.4.3 gesehen haben, sind Kontrollkanten für unsere Berechnungen irrelevant und können daher entfernt werden. Leider mußten wir auch feststellen, daß in vielen Fällen die Erkennung solcher Kanten extrem kompliziert ist, da diese Eigenschaft stark vom Kontext der Kante abhängt. Daher haben wir uns auf einfach zu erkennende Fälle beschränkt.

Wünschenswert wäre aber, daß wir alle solchen Fälle erkennen würden, da die Entfernung solcher Kanten nicht nur einen quantitativen sondern auch einen qualitativen Vorteil hat: Wir haben nicht nur weniger Kanten, sondern wir vereinfachen gegebenenfalls auch unsere Aufgabe – man denke hierbei an den Wegfall von zu untersuchenden Wegen oder an die vollständige Teilung des Graphen in mehrere Zusammenhangskomponenten.

Eine mögliche Lösung wäre eine Erweiterung der Graph-Regeln derart, daß sie in der Lage wären, diesen Kontext mitzuberücksichtigen. Eine solche Erweiterung würde aber unsere Graph-Regel zur Schablonen-Einbettung transformieren und dadurch den Aufwand der Vorverarbeitung nicht unerheblich vergrößern (siehe Kapitel 4).

Eine weitere Möglichkeit zur Lösung dieser Aufgabe stellen Datenbanken dar. Es wäre vorstellbar, daß wir eine möglichst einfache Datenbank mit den Beschreibungen solcher Fälle verwenden, um gezielt Kontrollkanten zu finden. Leider kann auch dieser Lösungsweg ineffizient sein, wenn eine Datenbankanfrage mit großer Häufigkeit einen Plattenzugriff bedeuten würde. Heutige Datenbank-Systeme gleichen dieses Problem aus, indem sie Strategien wie Caching verwenden, so daß sich der Einsatz von Datenbanken lohnt.

Kapitel 4

Schablonen-Einbettung

In diesem Kapitel wollen wir eine Alternative zum Verfahren zur Bestimmung der hydraulischen Achsen mittels Pfadsuche angeben. Im Abschnitt 1.2.3 haben wir zusätzlich zur Pfadsuche auch noch die Möglichkeit der Lösung unseres Problems mittels Einbettungen erwähnt.

Da dieser Alternativweg unserer Meinung nach mindestens so komplex ist wie der Lösungsweg der Pfadsuche, werden wir keine konkreten Verfahren angeben, sondern nur das Prinzip etwas näher erläutern und die damit verbundenen Schwierigkeiten beschreiben.

4.1 Struktursuche im Schaltkreisgraphen

Wie wir im Abschnitt 2.3.1 gesehen haben, ist eine Einbettung nichts anderes als die Abbildung der Knoten eines Graphen auf die Knoten eines anderen Graphen, wobei Kanten des ursprünglichen Graphen gegebenenfalls auf Wege im Bild-Graphen abgebildet werden müssen. Um das Problem der Bestimmung der hydraulischen Achsen lösen zu können, müssen wir einen Schritt weiter gehen und den Begriff der *Schablonen-Einbettung* präziser definieren:

- Eine *Schablonen-Einbettung* ist die Einbettung φ eines abstrakten Graphen T – hier *Schablone* („Template“) genannt – in einen konkreten Graphen G. Dabei ist $\varphi(v) = w$ für $v \in T, w \in G$ genau dann, wenn $\psi(v) = \psi(w)$ für eine *Eignungsfunktion* ψ gilt.

Informell beschrieben bedeutet die obige Definition, daß eine Schablonen-Einbettung eine übliche Einbettung ist, die einige zusätzliche Nebenbedingungen erfüllen muß. In unserem Fall müßte die Schablonen-Einbettung die Knoten der Schablone

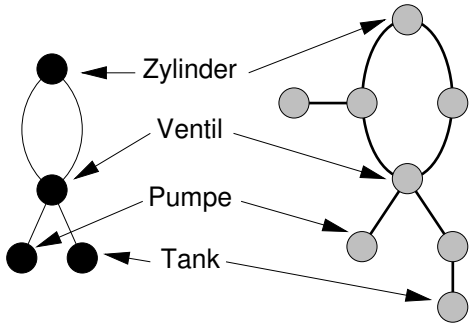

Abbildung 4.1: Eine Schablone und ein Graph

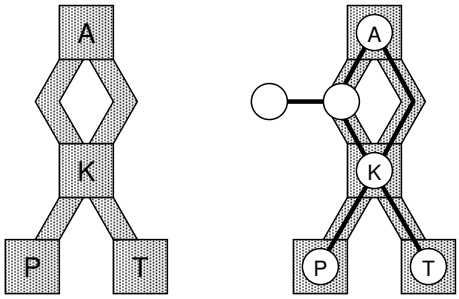

Abbildung 4.2: Eine Schablone und ihre Einbettung in einen Graphen

auf Knoten des Graphen derselben Komponentenklasse abbilden. Abbildung 4.1 zeigt eine Schablone und einen passenden Graphen.

Wie wir aus der Abbildung 4.2 entnehmen können, muß der Graph vor der Einbettung der Schablone komprimiert worden sein, denn Konstrukte wie beispielsweise tote Zweige erschweren oder verhindern sogar eine korrekte Erkennung der hydraulischen Achse.

Desweiteren können wir erkennen, daß eine Schablonen-Einbettung eine etwas andere Natur besitzt als die Einbettungen, die wir für die Produktionsregeln unserer Graph-Grammatik aus Abschnitt 2.4 verwendet haben. Dort haben wir Einbettungen benutzt, um meistens kleine Teilgraphen zu erkennen – diese Einbettungen sind von Natur aus lokal. Die Schablonen-Einbettung umfaßt dagegen den gesamten Graphen, da die Knoten der Schablone prinzipiell überall im Graphen eingebettet werden können – gerade diese globale Natur der Schablonen-Einbettung zeugt auf der einen Seite von der Mächtigkeit dieses Konzepts, auf der anderen Seite aber impliziert diese Eigenschaft einen großen, möglicherweise unbezahlbaren Aufwand.

Ist der Graph komprimiert, so liefert eine Schablonen-Einbettung eine hydraulische Achse der Schaltung. Folglich muß eine Einbettung der Schablone für jede

hydraulische Achse bzw. für jedes nach der Kontraktion noch vorhandene Arbeitselement neu gefunden werden.

Bevor wir der Frage nach der zu erwartenden Komplexität nachgehen, schauen wir uns die Erkennung einer hydraulischen Achse mit Hilfe einer Schablonen-Einbettung etwas näher an. Daraufhin werden wir einige Aspekte der Aufwandsabschätzung besser nachvollziehen können. Zunächst aber wollen wir den Prozeß einer Einbettung genauer betrachten und schrittweise verfeinern.

Angenommen, wir wollen einen Graph $G_1 = (V_1, E_1)$ in einen anderen Graphen $G_2 = (V_2, E_2)$ einbetten, wobei wir zunächst keine Bedingungen an die Einbettung stellen wollen, d. h., wir führen eine ganz gewöhnliche Einbettung durch. Damit die Einbettung korrekt erfolgt, müssen wir sicherstellen, daß alle Knoten des Graphen G_1 auf Knoten des Graphen G_2 abgebildet werden – dieser Schritt ist trivial. Weiterhin müssen wir gewährleisten, daß alle in G_1 durch eine Kante verbundenen Knoten in G_2 auch irgendwie miteinander verbunden sind: entweder eine Kante aus G_1 wird auf eine Kante aus G_2 abgebildet, oder man findet einen einfachen Weg, der die Knoten miteinander verbindet. Offensichtlich müssen Knoten aus G_1 auf Knoten einer und derselben Zusammenhangskomponente von G_2 abgebildet werden. Es reicht allerdings nicht, zu wissen, daß ein Weg existiert, der zwei Knoten miteinander verbindet – wir wollen schließlich alle von der Einbettung berührten Knoten von G_2 kennen. Daher müssen wir einen solchen Weg finden, beispielsweise mit Tiefen- oder Breitensuche. Also sind folgende Schritte für eine primitive Einbettung notwendig:

1. Bestimmung der Zusammenhangskomponenten: linearer Aufwand. Dieser Schritt wird allerdings bei der Vorverarbeitung bereits durchgeführt, so daß wir den Aufwand hier vernachlässigen können.

2. Zuordnung der Knoten: linearer Aufwand. Hierfür braucht man lediglich die Knotenlisten beider Graphen durchzugehen.

3. Pfadsuche : linearer Aufwand für jedes Knotenpaar, wenn wir Breiten- oder Tiefensuche verwenden. Hierfür würden wir dann schlimmstenfalls kubische Laufzeit benötigen, falls der Graph G_1 eine Clique bilden würde. In [7] wird erläutert, wie man den Aufwand auf $O(|V_2| \cdot log|V_2| + |E_2|)$ reduzieren kann, in dem man den Algorithmus zur Bestimmung des minimalen Spannbaumes von Prim unter Zuhilfenahme von Fibonacci-Heaps verwendet.

Insgesamt ergibt sich für eine einfache Einbettung ein Aufwand von $O(|V_2| \cdot log|V_2| + |E_2|)$ bzw. $O(n \cdot log(n))$. Eine solche Einbettung hat allerdings eine beliebige Struktur und ist deswegen unbrauchbar. Man kann zum Beispiel alle Knoten von G_1 auf einen einzigen Knoten von G_2 abbilden und dabei alle oben erwähnten Bedingungen erfüllen (außer der Injektivität der Einbettung) – die so

gewonnene Einbettung ist aber völlig bedeutungslos. Nebenbei sei auch erwähnt, daß wir auch hier auf Algorithmen zur Pfadsuche angewiesen sind, so daß unser Verfahren bestenfalls eine Art Mischung von Schablonen-Einbettung und Pfadsuche darstellt.

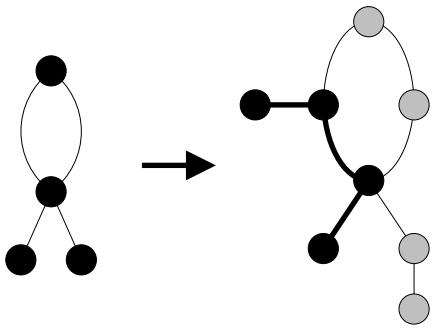

Abbildung 4.3: Eine primitive Einbettung

Nun fordern wir für die Einbettung, daß alle Knoten von G_1 auf Knoten desselben Grades in G_2 abgebildet werden. Dadurch erhalten wir eine Einbettung, die unsere Anforderungen an die Struktur etwas besser erfüllt. Um diese Bedingung zu erfüllen, müssen wir für jeden Knoten aus G_1 einen passenden Knoten aus G_2 finden. Diese Suche kann unter Umständen bedeuten, daß für jeden Knoten aus G_1 jeweils der ganze Graph G_2 durchsucht werden muß. Wir erhalten daher für die naiv durchgeführte Zuordnung der Knoten unter der Bedingung der Gradgleichheit einen quadratischen Aufwand. Dieser Aufwand kann etwas verbessert werden, wenn man an dieser Stelle Hilfslisten von Knoten für jeden Grad benutzt – leider kann man hier keine konkreten Aussagen zur Aufwandsminderung machen, da im schlimmsten Fall (alle Knoten haben denselben Grad) die Verwendung von Hilfslisten keine Vorteile bringt. Es ergibt sich also im Durchschnitt für diesen Schritt ein linearer Aufwand. Im übrigen verursachen alle anderen Schritte denselben Aufwand wie die primitive, bedingungsfreie Einbettung.

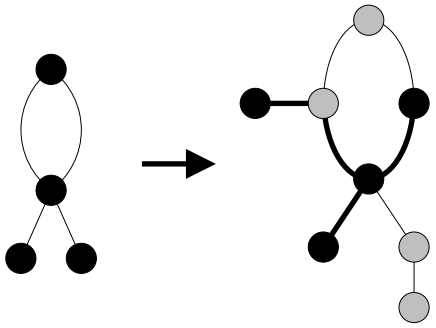

Abbildung 4.4: Eine Einbettung mit Gradbedingung

In der letzten Einbettungsvariante hatten wir gefordert, daß Knoten vom Grad d auf Knoten vom Grad d abgebildet werden, aber nichts bezüglich der Rolle der Kanten im Hinblick auf die Verbindungswege gesagt. Es ist also möglich, daß obwohl ein Knoten vier Kanten hat, drei oder weniger Kanten für die Verbindung zu anderen Knoten benutzt werden – also wird mindestens eine Kante mehrfach benutzt. Daher schränken wir unsere Einbettung weiter ein, in dem wir nun fordern, daß alle Kanten eines Knotens benutzt werden, und zwar wird jede Kante als erste Kante des Verbindungsweges zu seinem „Nachbar“ verwendet. Auf diese Weise werden Schleifen in der Schablone auch als Schleifen auf dem Bildgraphen erscheinen.

Die oben erwähnte Forderung hat allerdings Konsequenzen, denn wir können nicht mehr die Verbindungswege mit Hilfe des Algorithmus zur Bestimmung eines minimalen Spannbaumes finden. Der Algorithmus von Prim liefert immer einen beliebigen Weg zwischen zwei Knoten und nimmt keine Rücksicht auf die Struktur der Schablone. Die Forderung nach der Verwendung aller Kanten ist aber eine notwendige Bedingung für die Anpassung der Einbettung an die Struktur der Schablone.

Wir müssen also das Problem der Pfadsuche neu lösen, und zwar so, daß die lokalen Anforderungen an die Verbindungswege erfüllt werden. Es bieten sich hierfür zwei Möglichkeiten an: entweder man löst diese Aufgabe komplett mit einem anderen Algorithmus, oder man versucht den minimalen Spannbaum so zu erweitern, daß die obige Bedingung erfüllt wird.

Die erste Variante scheint auf den ersten Blick komplizierter zu sein, aber wenn man in Betracht zieht, daß die vom Algorithmus von Prim gelieferten Wege zum Teil sehr umständlich sind (siehe dazu Abbildung 4.5), da manche nützliche Kante nicht benutzt wird, so steigt die Motivation für die Suche nach einem neuen, besseren Lösungsweg.

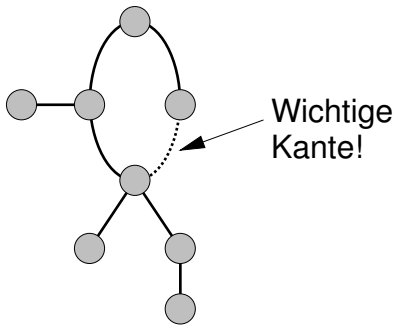

Abbildung 4.5: Ein Spannbaum, der eine wichtige Kante nicht berücksichtigt

Hat man alle Knoten von G_1 auf Knoten von G_2 abgebildet, so braucht man nun nur noch die Verbindungswege für die nicht trivial einbettbaren Kanten zu be-

stimmen. Diese Aufgabe kann man beispielsweise elegant mit dem Pfadsuchalgorithmus von Dijkstra (siehe [4] und [7]) lösen, welcher den kürzesten Weg zwischen zwei Knoten eines Graphen findet, falls ein solcher Weg existiert. Um unsere Bedingung der Berücksichtigung aller Kanten zu erfüllen, müssen wir gegebenenfalls eine oder mehrere Kanten des Anfangsknotens vorläufig „ausschalten“, um den Algorithmus von Dijkstra zu zwingen, eine bestimmte Kante in die Lösung aufzunehmen. Alternativ kann man auch den Algorithmus leicht verändern, damit er bei einer bestimmten Kante mit der Pfadsuche beginnt – diese Möglichkeit ist offensichtlich besser als die vorläufige Ausschaltung aller anderen Kanten. Da der Algorithmus von Dijkstra eine Komplexität von $O(|E_2| \cdot log|V_2|) \subset O(n \cdot log(n))$ hat, und wir unter Umständen Wege für alle Knotenpaare suchen müssen, erhalten wir einen Gesamtaufwand von $O(|V_1|^2 \cdot (|E_2| \cdot log|V_2|))$ bzw. $O(n^3 \cdot log(n))$.

Der Vorteil der zweiten Variante ist, daß die Berechnung des minimalen Spannbaumes einige Verbindungswege bereits mit einschließt, so daß nur noch wenige Wege zu suchen sind. Wir müssen nämlich alle eingebetteten Knoten daraufhin untersuchen, ob alle Kanten auch für die Wege verwendet wurden. Dies ist leicht zu überprüfen, wenn wir alle Verbindungswege gespeichert haben. Ist eine Kante nicht benutzt worden, so müssen wir den „falschen“ Verbindungsweg finden (unter allen $deg(v)$ Wegen ausgehend vom Knoten v), löschen und anschließend einen neuen finden. Die Suche nach einem neuen Weg realisieren wir wieder mit dem Algorithmus von Dijkstra. Insgesamt erhalten wir einen Aufwand von $O(|V_2| \cdot log|V_2| + |E_2| + k^2 \cdot (|E_2| \cdot log|V_2|))$ mit $k \leq |V_1|$ bzw. $O(n \cdot log(n) + n + n^2 \cdot (n \cdot log(n))) = O(n^3 \cdot log(n))$. Also sind beide Alternativen äquivalent bezüglich des asymptotischen Aufwands – die erste Variante dürfte aber bessere Ergebnisse liefern, da dort nur die kürzesten Wege gefunden und verwendet werden.

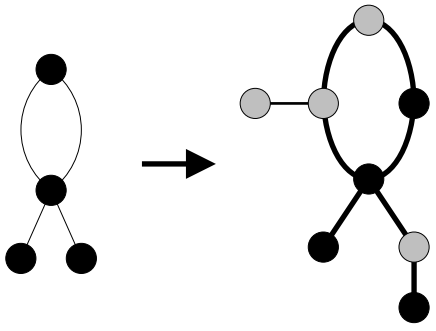

Abbildung 4.6: Eine Einbettung mit Kantenausnutzungsbedingung

Nun sind wir fast am Ziel und brauchen nur noch die Knotentypen zu berücksichtigen; daher fordern wir jetzt, daß Knoten der Schablone ausschließlich auf Knoten derselben Komponentenklasse abgebildet werden. Dadurch stellen wir

sicher, daß alle von der Einbettung berührten Knoten auch die gewünschten Eigenschaften besitzen. Darüberhinaus entspricht die Struktur der Einbettung auch der Struktur einer einfachen hydraulischen Achse.

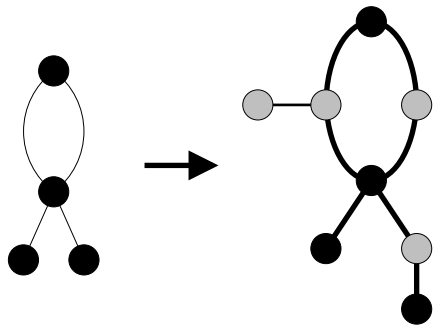

Abbildung 4.7: Eine Einbettung mit Komponententypbedingung

Da wir bei der Einbettung mit Gradbedingung bereits Knoten durch ihre Eigenschaften – in dem Fall war es der Grad – ausgewählt haben, verursacht die Überprüfung einer weiteren Eigenschaft keine nennenswerten Mehrkosten, so daß der Aufwand für die Einbettung mit Komponententypbedingung nicht größer wird. Damit hätten wir unser Ziel erreicht und eine hydraulische Achse mit einer Schablonen-Einbettung gefunden.

Eine letzte Aufgabe muß allerdings noch erledigt werden: Alle von der Einbettung berührten Knoten müssen ihren hydraulischen Achsen zugeordnet werden. Dafür müssen wir alle Verbindungswege durchgehen und alle Knoten, die auf diesen Wegen liegen, der aktuellen hydraulischen Achse zuordnen. Unsere Schablone G_1 besitzt $|E_1|$ Kanten, und aus diesem Grund haben wir höchstens $|E_1|$ Verbindungswege, die wir bearbeiten müssen. Da jeder dieser Verbindungswege im schlimmsten Fall eine Länge von höchstens $|E_2|$ besitzen kann, haben wir einen Aufwand von $O(|E_1| \cdot |E_2|)$ bzw. $O(n^2)$ im allerschlimmsten Fall. Wenn wir nun annehmen, daß der Graph G_2 $k, k \leq |V_2|$, hydraulische Achsen bzw. zu untersuchende Arbeitselemente hat, so kommen wir auf einen Gesamtaufwand von $O(k \cdot n^2)$ bzw. $O(n^3)$.

Also besitzt das oben genannte Verfahren einen asymptotischen Aufwand von $O(n^3 \cdot log(n))$. Da unsere hydraulischen Schaltungen einen maximalen Grad von vier besitzen, können wir annehmen, daß der reale Aufwand etwas kleiner ist. Allerdings ist nach der Kontraktion des Graphen der Grad nicht mehr beschränkt, so daß wir keine genaueren Aussagen über die Aufwandsminderung machen können, obwohl der Graph dann meistens viel weniger Knoten und nur einzelne Knoten mit höherem Grad besitzt.

4.2 Nachverarbeitung

Die Erkennung der identischen hydraulischen Achsen ist ein weiteres Problem, das noch zu lösen bleibt. Wie wir im Abschnitt 3.2 gesehen haben, benötigen wir genaue Informationen über alle Knoten der zu vergleichenden Achsen, so daß eine Überprüfung auf identische Achsen auf dem komprimierten Graphen unmöglich ist.

In diesem Kapitel haben wir Schablonen-Einbettungen verwendet, um unsere Aufgaben zu lösen. Leider können wir Schablonen-Einbettungen nicht für diese spezielle Aufgabe benutzen, da eine Schablonen-Einbettung eine Abstraktion einer hydraulischen Achse darstellt und somit höchstens für die Verwendung auf dem komprimierten Graphen geeignet ist. Daher müssen wir andere Lösungswege suchen, um identische Achsen zu finden.

Wie beim im Kapitel 3 beschriebenen Verfahren, haben wir auch hier nur Informationen bezüglich Knoten und Verbindungswegen auf dem komprimierten Graphen, so daß wir uns in (fast) exakt derselben Situation befinden. Daher verzichten wir an dieser Stelle auf die Angabe eines Lösungsweges für die Erkennung identischer Achsen und verweisen auf unsere Ergebnisse aus Abschnitt 3.2.

4.3 Vergleich von Pfadsuche und Einbettung

Wir haben nun zwei unterschiedliche Verfahren angegeben, um hydraulische Achsen erkennen zu können, wobei wir eines davon – die Methode der Schablonen-Einbettung – lediglich oberflächlich betrachtet haben. Es stellt sich daher die Frage, welche Methode sich für den praktischen Einsatz am besten eignet.

Wie wir im Kapitel 3 gesehen haben, verursacht die Erkennung der hydraulischen Achsen einen Laufzeitaufwand von $O(|V| \cdot (|V| \cdot |E|))$ bzw. $O(n^3)$ im schlimmsten Fall. Bei der Erkennung der hydraulischen Achsen mittels Schablonen-Einbettung haben wir allerdings einen asymptotischen Laufzeitaufwand von $O(|V_1|^2 \cdot (|E_2| \cdot log|V_2|))$ bzw. $O(n^3 \cdot log(n))$, wenn $G_1 = (V_1, E_1)$ die Schablone und $G_2 = (V_2, E_2)$ den Schaltungsgraph darstellt. Also ist die Methode der Schablonen-Einbettung im schlimmsten Fall um einen logarithmischen Faktor schlechter als die der Pfadsuche.

Nichtsdestotrotz kann die Methode der Schablonen-Einbettung sich als die bessere erweisen, denn wir dürfen nicht außer Acht lassen, daß die bisher gemachten Abschätzungen sehr grob sind, und daß wir ausschließlich vom schlimmsten Fall ausgegangen sind. Da unsere Graphen ganz besondere Eigenschaften besitzen, wie beispielsweise einen beschränkten Grad, können beide Verfahren sich anders verhalten als in unserer Abschätzung – der Simplex-Algorithmus zur Lösung von

Aufgaben der Linearen Programmierung hat eine Laufzeit von $O(n^3)$ im schlimmsten Fall; in der Praxis aber hat er meistens einen Aufwand von $O(n)$, da die dort verwendeten Matrizen eine günstige Form besitzen. Eine genauere Untersuchung unserer Verfahren sprengt den Rahmen dieser Arbeit, so daß wir den Vergleich hier abschließen und uns mit unseren Ergebnissen zufrieden geben müssen.

Um unsere Aufgabe zu lösen, haben wir uns für die Methode der Pfadsuche entschieden. Zum einen ist diese Methode leicht mit schon bestehenden Graphalgorithmen realisierbar, so daß wir keine zusätzlichen Algorithmen entwickeln müssen, was bei der Schablonen-Einbettung der Fall gewesen wäre. Zum anderen hat sich diese Methode, wie oben bereits erwähnt, als die im schlimmsten Fall schnellere erwiesen.

Kapitel 5

Kopplung zwischen hydraulischen Achsen

Zur Lösung der letzten Aufgabe, der Erkennung der Kopplung zwischen hydraulischen Achsen, müssen wir ähnlich wie bei der Erkennung identischer Achsen vorgehen. Der Unterschied hierbei ist, daß Achsen auch in Ketten miteinander gekoppelt sein können. Daher werden wir in diesem Kapitel zunächst die Kopplung von unmittelbar benachbarten Achsen betrachten, um die grundlegende Problematik zu erläutern, und danach untersuchen wir die Möglichkeit der verketteten Kopplung etwas näher. Anschließend werden wir einige mögliche Verbesserungen kurz betrachten.

5.1 Kopplung benachbarter Achsen

Die im Abschnitt 1.2.2 beschriebenen fünf Kopplungsarten besitzen unterschiedliche „Eigenschaften“. Sie werden auf unterschiedliche Weisen erkannt und verursachen daher auch unterschiedliche Aufwände. Im folgenden werden wir drei Vorgehensweisen herleiten, um die fünf verschiedenen Kopplungsarten – die in drei Gruppen aufgeteilt werden können – möglichst effizient erkennen zu können.

5.1.1 Kopplung der Ebenen 0 und 1

Die ersten zwei Kopplungsarten – „keine“ und „informationelle“ Kopplung bzw. Kopplung der Ebenen 0 und 1 – können leicht erkannt werden, da wir nur die zu den Achsen zugehörigen Zusammenhangskomponenten überprüfen müssen. Gerade weil beide Kopplungsarten von den Zusammenhangskomponenten des

Graphen abhängig sind, d. h. die Erkennungsweise ist nahezu identisch, fassen wir sie zusammen in eine Gruppe.

Bei der Kopplung der Ebene 0 ist die Erkennung eine einfache Aufgabe – die Achsen liegen ja in verschiedenen Zusammenhangskomponenten; bei der Kopplung der Ebene 1 ist dies zwar keine einfache Aufgabe, aber trotzdem ohne großen Aufwand zu lösen. Hierzu muß man lediglich überprüfen, ob sich nach der Entfernung der Kontrollkanten etwas an den Zusammenhangskomponenten geändert hat, und dies kann man mit einer weiteren Berechnung der Zusammenhangskomponenten mittels Tiefensuche erreichen. Dieser Schritt schlägt mit einem Aufwand von $O(|E|)$ bzw. $O(n)$ zu Buche. Anschließend vergleichen wir die Zusammenhangskomponenten aller Achsenpaare; da wir $\binom{n}{2} \in O(n^2)$ solche Paare haben, verursacht diese Untersuchung einen Aufwand von $O(n^2)$.

Es gibt allerdings eine etwas günstigere Alternative für die Erkennung der Kopplung der Ebene 1. In der Realisierung unseres Gesamtverfahrens führen wir einmalig die Bestimmung der Zusammenhangskomponenten durch, und zwar nach der Vorverarbeitung. Dies bedeutet, daß zu diesem Zeitpunkt alle erkannten Kontrollkanten aus dem Graphen entfernt wurden, so daß die Endknoten solcher Kanten nun möglicherweise auf verschiedenen Zusammenhangskomponenten liegen.

Es bietet sich daher an, daß wir bei der Graph-Expandierung eine Erkennung von informationellen Kopplungen durchführen. Dazu müssen wir lediglich bei der Wiederherstellung einer Kontrollkante ihre Endknoten näher betrachten: Gehören beide Knoten zu unterschiedlichen Zusammenhangskomponenten, so sind die zugehörigen Achsen informationell gekoppelt. Die Kopplungen aller anderen betroffenen Achsenpaare können später zusammen mit der Erkennung der anderen Kopplungsarten bestimmt werden; die Propagation dieser Information kann dann während der Untersuchung aller $O(n^2)$ Achsenpaare geschehen.

Zugegeben, die Einführung von Schritten zur Erkennung der Kopplungsart in den Expandierungsprozeß verletzt die Modularität der einzelnen Verfahrensschritte und erschwert die Lesbarkeit des Programm-Quellcodes. Auf der anderen Seite hilft hier dieser unsaubere Schritt, einiges an Rechenleistung zu sparen. Da unser Verfahren möglicherweise in verschiedenen Systemen eingesetzt wird, verzichten wir der Code-Lesbarkeit zuliebe auf die durch diesen Schritt gewonnene Rechenleistung.

5.1.2 Kopplung der Ebenen 2 und 3

Die nächsten zwei Kopplungsarten – „parallele“ und „serielle“ Kopplung bzw. Kopplungen der Ebenen 2 und 3 – sind nicht so leicht zu erkennen. Wegen der Ähnlichkeit der Erkennungsweise dieser Kopplungsarten können wir ein Achsen-

paar in einem Vorgang auf beide Kopplungsarten untersuchen, denn es sind in beiden Fällen Pfade miteinander zu vergleichen.

Im ersten Fall teilen sich die Achsen einige Komponenten wie Pumpe und Tank, sie werden aber von unterschiedlichen Kontrollelementen gesteuert und sind somit voneinander unabhängig. Im zweiten Fall sind die Achsen verkettet, d. h. der hydraulische Druck läuft zuerst durch die eine und dann durch die andere Achse. Das bedeutet, daß beide Achsen aufeinander Einfluß nehmen können, wenn die Schaltstellung eines der Kontrollelemente geändert wird (siehe Abbildung 5.1). In beiden Fällen müssen wir die Wege, die die Achsen miteinander verbinden, untersuchen, um festzustellen, ob es sich hierbei um eine parallele oder serielle Kopplung handelt.

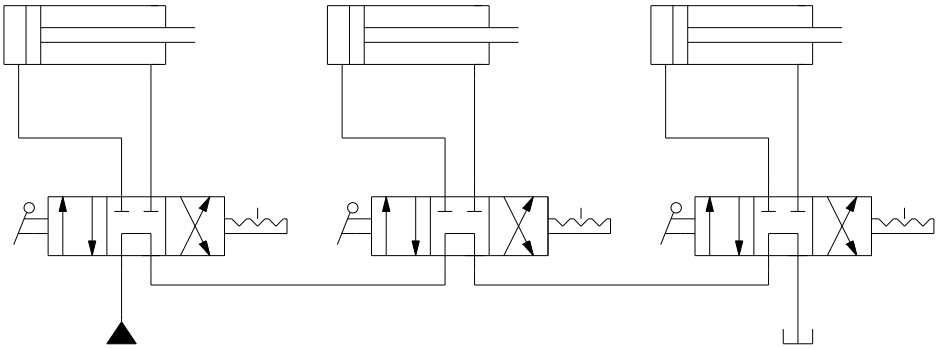

Abbildung 5.1: Drei seriell gekoppelte Achsen

In beiden Fällen betrachten wir die Wege, die die Arbeitselemente mit einer und derselben Pumpe bzw. einem und demselben Tank verbinden. Wie bei der Erkennung identischer Achsen vergleichen wir vier Wege, und zwar von jedem Arbeitselement jeweils zwei Wege, die zu der Pumpe bzw. dem Tank führen. Bei parallel gekoppelten Achsen verschmelzen beide Wege irgendwann miteinander, und gerade die disjunkten Abschnitte sind für die Erkennung entscheidend – enthalten beide Pfad-Abschnitte jeweils mindestens ein Kontrollelement, so handelt es sich hier um eine parallele Kopplung. Bei seriell gekoppelten Achsen enthalten dagegen die gemeinsamen Abschnitte mindestens jeweils ein Kontrollelement jeder Achse. Allerdings muß nicht zwischen zwei seriell gekoppelten Achsen eine bidirektionale Beziehung existieren (siehe Abbildung 5.1), sondern es kann auch eine unidirektionale sein, d. h. nur eine der hydraulischen Achsen kann Einfluß auf die andere nehmen. Abbildung 5.2 zeigt zwei seriell gekoppelten Achsen, von denen nur eine Einfluß auf die andere hat.

Die Untersuchung der Kopplung der Ebene 2 bzw. Ebene 3 verläuft im großen und ganzen wie bei der Erkennung identischer Achsen. Daraus folgt, daß wir auch hierfür einen asymptotischen Aufwand von mindestens $O(n^3)$ in Kauf nehmen

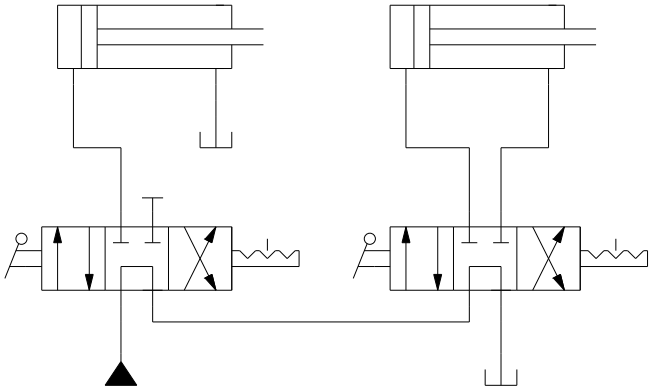

Abbildung 5.2: Eine unidirektionale serielle Kopplung

müssen, da wir $\binom{n}{2} \in O(n^2)$ mögliche Knotenpaare haben, für die jeweils ein Aufwand von $O(|E|)$ bzw. $O(n)$ nötig ist. Wir müssen hier allerdings darauf achten, daß wir beispielsweise für die Untersuchung zweier seriell gekoppelten Achsen gegebenenfalls die Wege zur Pumpe komplett durchgehen müssen – sind aber mehrere Pumpen und Tanks vorhanden, so erhöht sich der Aufwand auf $O(n \cdot n^3) = O(n^4)$, da wir möglicherweise $O(n)$ Pumpen und Tanks haben, und somit auch $O(n)$ verschiedene Pfade, die natürlich auch untersucht werden sollten. Es ist glücklicherweise unwahrscheinlich, daß soviele zusätzliche Pfade untersucht werden müssen, da eine hydraulische Schaltung normalerweise eine oder zwei Pumpen und eine kleine Menge von Tanks besitzt – wir können daher mit einem Aufwand von $O(n^3)$ rechnen.

5.1.3 Kopplung der Ebene 4

Die letzte noch zu untersuchende Kopplungsart – die „sequentielle“ Kopplung bzw. Kopplung der Ebene 4 – besitzt die größte Ähnlichkeit zu dem Fall der identischen Achsen. Wir haben uns entschieden, diesen Fall gesondert und nicht zusammen mit der Erkennung der Kopplung der Ebenen 2 und 3 zu behandeln, da die Kopplung der Ebene 4 nicht nur eine ähnliche Erkennungsweise wie die Erkennung der identischen Achsen erfordert, sondern auch weil die sequentielle Kopplung strukturell eigentlich dem Fall der identischen Achsen näher steht als den anderen Kopplungsarten.

Zwei sequentiell gekoppelte Achsen werden, wie bei identischen Achsen, von genau einer Kontrollkomponente gesteuert. Der Unterschied – abgesehen davon, daß die Achsen nicht identisch sind – liegt darin, daß, obwohl die Achsen zur gleichen Zeit und auf die gleiche Art und Weise angesteuert werden, sie nicht unbedingt gleichzeitig oder auf die gleiche Weise arbeiten. Das besondere an derartig

gekoppelte Achsen ist, daß eine der Achsen verhaltensändernde Komponenten besitzt, wie zum Beispiel hydraulische Widerstände. Solche Komponenten können die Wirkung eines Arbeitselements zeitlich verzögern oder auch nur schwächen bzw. verstärken – man stelle sich dies wie in der Elektrotechnik vor, wo man mit Widerständen, Kondensatoren usw. genau solche Effekte erzielen kann.

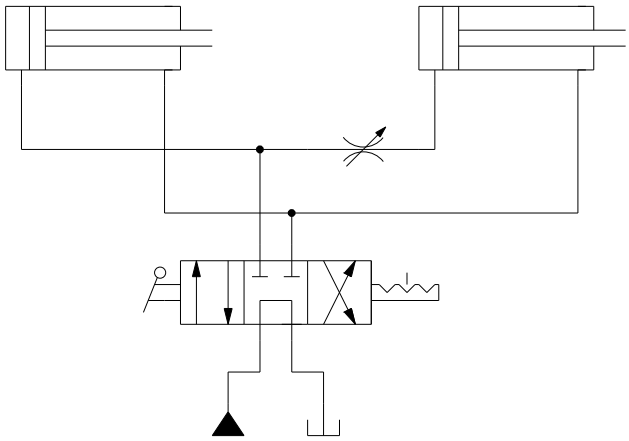

Abbildung 5.3: Zwei sequentiell gekoppelte Achsen

Soweit es die Vorgehensweise betrifft, verfahren wir hier analog zur Erkennung identischer Achsen. Wir vergleichen daher alle Pfade, angefangen mit den Arbeitselementen, bis wir eine gemeinsame Komponente finden. Wie bei der Erkennung identischer Achsen verwerfen wir ein Achsenpaar, wenn sie von unterschiedlichen Kontrollelementen gesteuert werden, d. h. auf dem Pfad zum ersten gemeinsamen Element darf kein Kontrollelement liegen; allerdings untersuchen wir hier die Pfade auf jeden Fall bis zur ersten gemeinsamen Komponente – bei der Erkennung identischer Achsen haben wir die Prozedur beim ersten Auftreten unterschiedlicher Komponenten abgebrochen bzw. beendet.

Da die Untersuchung auf sequentielle Kopplung mit der Erkennung identischer Achsen so verwandt ist, können wir davon ausgehen, daß die oben beschriebene Vorgehensweise ebenfalls einen Aufwand von $O(n^3)$ verursacht. Im Durchschnitt ist dieser Schritt aber doch etwas aufwendiger, da wir jedes Mal die Pfade bis zum ersten gemeinsamen Element ganz durchgehen müssen, um alle Komponenten auf eine potentielle verhaltensändernde Eigenschaft untersuchen zu können.

Leider haben wir bisher nur die Vorgehensweise und den Aufwand für den Vergleich der Pfade angegeben. Auch wenn wir unterschiedliche Elemente auf den Wegen vorfinden, können wir nicht mit Sicherheit sagen, daß diese Komponenten irgendeine Wirkung auf das Verhalten der Achse haben. Bei diesen Komponenten kann es sich beispielsweise um ein Manometer handeln, das lediglich zur Messung verwendet wird. Andererseits kann es sich um eine Komponente mit Widerstand

handeln, die eben das Verhalten der Achse entscheidend bestimmt. Diese zwei Fälle sind ja simpel und daher auch leicht erkennbar. Es kann aber leider der Fall auftreten, daß eine Gruppe von Elementen in einer Teilschaltung auch eine Verhaltensänderung der Achse bewirkt, obwohl die einzelnen Elemente an sich keine Auswirkung auf die Achse haben mögen – dies bleibt leider ein offenes Problem, da wir keinerlei Information bezüglich der Eigenschaften von kombinierten Komponenten besitzen.

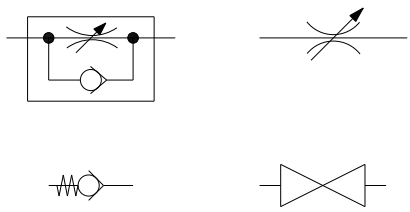

Abbildung 5.4: Einige verhaltensändernde Komponenten

Wir müssen daher, um eine möglichst genaue Erkennung durchzuführen, mindestens ein verhaltensänderndes Element finden. Dazu verfahren wir wie bei der Erkennung identischer Achsen und durchsuchen die Pfade bis zum ersten gemeinsamen Element samt aller Schleifen – tote Zweige sind für diese Untersuchung ohne Belang, wie bei der Erkennung identischer Achsen. Da die verhaltensändernden Komponenten an ihrem Typ bzw. ihrer Bezeichnung erkennbar sind, können wir sie ohne große Mühe bei der Pfadsuche erkennen – daher verursacht diese Untersuchung insgesamt einen Aufwand von $O(n^3)$.

5.1.4 Zusammenfassung der Schritte

Wir haben bisher gesehen, daß die Erkennung der Kopplung benachbarter Achsen zum Teil vergleichbar ist mit der Erkennung identischer Achsen. Der Unterschied zur Erkennung identischer Achsen liegt darin, daß alle Wege gegebenenfalls vollständig durchsucht werden müssen. Daher vergrößert sich der Aufwand für eine durchschnittliche Bearbeitung, aber in Bezug auf den schlimmsten Fall ändert sich nichts.

Da eine Achse darüberhinaus Verbindungen zu mehreren Pumpen und Tanks besitzen kann, müssen wir möglicherweise für ein Achsenpaar mehrmals die oben beschriebene Prozedur zur Behandlung von parallelen und seriellen Kopplungen durchführen, so daß wir für diesen Schritt einen echt größeren Aufwand als für die Erkennung identischer Achsen erwarten dürfen.

In den vorigen Abschnitten haben wir immer den Aufwand für die Erkennung des speziell zu behandelnden Falles angegeben. Man kann sich aber leicht überlegen,

wie groß der Gesamtaufwand für die oben erwähnten Fälle ausfallen würde. Wir fassen daher alle Schritte zusammen und erhalten einen asymptotischen Gesamtaufwand von $O(n^2 \cdot (1 + n + n)) = O(n^2 \cdot n) = O(n^3)$.

5.2 Verkettete Kopplung

Im letzten Abschnitt haben wir die Untersuchung der Kopplung von unmittelbar miteinander benachbarten Achsen näher betrachtet. Das oben beschriebene Verfahren behandelt eigentlich alle Achsenpaare, die irgendwie miteinander verbunden sind, d. h. die gemeinsame Komponenten besitzen. In einer zusammenhängenden Schaltung sind alle Achsen miteinander verbunden, da sie alle Zugang zu mindestens einer bestimmten Pumpe oder einem bestimmten Tank haben. Es können allerdings Achsen existieren, die in Kette miteinander gekoppelt sind – dieser Fall tritt höchstens nur dann auf, wenn eine Schaltung mehrere Zusammenhangskomponenten besitzt, die erst nach der Kontraktion entstanden sind, also wenn Kontrollkanten gelöscht wurden.

Nach der Expandierung des Graphen werden alle gelöschten Komponenten wiederhergestellt, also auch alle vorher entfernten Kontrollkanten. Dadurch sind nun Achsen miteinander verbunden, die vorher in verschiedenen Zusammenhangskomponenten lagen. Wegen der neuen Erreichbarkeit bestimmter Knoten, wie zum Beispiel der Pumpen, müssen wir die Kopplung aller neuen Achsenpaare untersuchen. Da allerdings die einzige Verbindung solcher Achsen über eine Kontrollkante geht, können wir vereinfachend sagen, daß die Kopplung aller solcher Achsenpaare höchstens informationell ist, d. h. wir müssen diese Achsenpaare nicht untersuchen, sondern können davon ausgehen, daß sie informationell gekoppelt sind, da in einer Kette von Kopplungen die schwächste entscheidend ist.

Um all diese Achsenpaare zu bearbeiten, müssen wir einfach alle der Reihe nach betrachten und die Kopplungsart auf „informationell“ setzen. Da wir $\binom{n}{2} \in O(n^2)$ Achsenpaare haben können, haben wir ebenfalls einen Aufwand von $O(n^2)$ für diesen einfachen Schritt.

5.3 Mögliche Verbesserungen

In diesem Abschnitt wollen wir einige Möglichkeiten zur Verbesserung bzw. Beschleunigung unseres Verfahrens aufzeigen. Einige der Ideen sind leicht realisierbar, wogegen andere nicht ohne weitere Informationen realisierbar sind.

5.3.1 Kombination von Erkennung der identischen Achsen und der Kopplungsart

Wie wir in den vorigen Abschnitten gesehen haben, besitzen alle einzelnen Schritte mehr oder minder große Ähnlichkeit zum Verfahren zur Erkennung identischer Achsen. Es stellt sich daher die Frage, ob man nicht beide Verfahren miteinander kombinieren kann, um auf diese Weise in der Praxis eine zügigere Laufzeit zu erreichen, obwohl im O-Kalkül keine Verbesserung zu erwarten ist.

Vor allem die Erkennung von Kopplungen der Ebene 4 verläuft nahezu identisch mit der Erkennung identischer Achsen. Hier ist eine Kombination beider Schritte sehr gewinnbringend, da wir uns eine zweite Untersuchung aller Achsenpaare ersparen können.

Die Kombination der Erkennung der anderen Kopplungsarten mit der Erkennung der identischen Achsen ist zwar möglich, aber leider sehr kompliziert, da man dann viele Fälle unterscheiden und gegebenenfalls unterschiedlich vorgehen muß. Eine Implementierung dieses Kombi-Schrittes wäre äußerst unverständlich; daher beschränken wir uns höchstens auf die Kombination der Erkennung der Kopplung der Ebene 4 mit der Erkennung der identischen Achsen.

Wie wir bereits im Abschnitt 5.1.1 erwähnt haben, hat die Kombination solcher Schritte zwei Seiten: Auf der einen Seite gewinnt man durch die Effizienz- bzw. Geschwindigkeitssteigerung, auf der anderen Seite aber verliert man durch die schlechte Lesbarkeit des Quellcodes, ganz zu schweigen von der Erweiterbarkeit. Also entscheiden wir uns auch hier für eine getrennte Realisierung beider Schritte.

5.3.2 Transitivität der Kopplungsarten

Unsere aktuelle Behandlung der Achsenpaare ist leider nicht sehr effizient, da wir quadratisch viele Paare in der Anzahl der Achsen untersuchen müssen. Da eine Schaltung relativ viele Achsenpaare besitzt und so wenig mögliche Kopplungsarten existieren, stellt sich die Frage, ob nicht viele Untersuchungen identisch verlaufen. Ist dies der Fall, so könnte man bei früher untersuchten Achsenpaaren Information bezüglich der vorhandenen Kopplungen für die noch zu untersuchenden Achsenpaare gewinnen.

Wenn wir die Achsen einer einzigen Zusammenhangskomponente betrachten, so stellen wir fest, daß alle Achsen miteinander irgendwie gekoppelt sind. Um alle diese Kombinationen möglichst schnell zu überprüfen, wäre eine besondere Eigenschaft der Kopplungsarten notwendig, wie beispielsweise die Transitivität von Kopplungen.

Diese Eigenschaft untersuchen wir am besten an einem Beispiel. Man betrachte die hydraulische Schaltung in Abbildung 5.5. Dort kann man leicht drei verschiedene Achsen erkennen, die miteinander gekoppelt sind, daher gibt es auch höchstens drei verschiedene Kopplungen. Man stellt sich nun die Frage, ob bei Kenntnis von zwei dieser Kopplungen auf die dritte geschlossen werden kann. Bei diesem Beispiel trifft das zu, d. h. hier gilt die Transitivität.

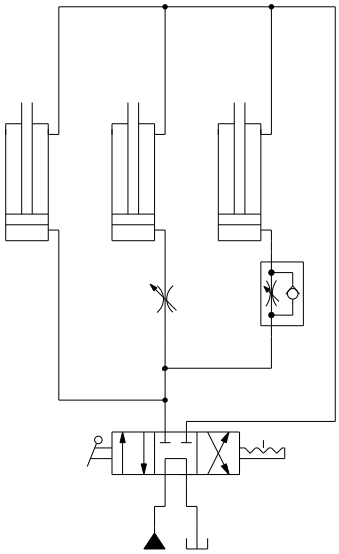

Abbildung 5.5: Gleich gekoppelte Achsen

Leider können wir dieses Ergebnis nicht verallgemeinern; es ist sehr leicht, ein Gegenbeispiel hierfür anzugeben – siehe dazu Abbildung 5.6. Diese Beispielschaltung enthält wiederum drei verschiedene Achsen, die aber diesmal unterschiedlich gekoppelt sind. Leider kann man hier nicht von der oben erwähnten Transitivität ausgehen, da die drei Kopplungsarten nicht gleich sind.

Wenn wir aber die Abbildung 5.6 näher betrachten, so können wir folgendes feststellen: Aus der Kenntnis von zwei der vorliegenden Kopplungen können wir leider nicht auf die dritte schließen, aber wir können die Kopplungsart etwas eingrenzen. Wenn wir zwei der Kopplungsarten kennen, dann können wir davon ausgehen, daß die dritte Kopplung mindestens so stark ist wie die schwächste der zwei bekannten Kopplungen. Eine schwächere Kopplung wäre nicht möglich, denn das Achsenpaar wäre indirekt über die dritte Achse stärker gekoppelt.

Aus diesen Gründen liegt die Eigenschaft der Transitivität leider nicht vor, sondern lediglich eine abgeschwächte Form davon, die man als minimale Transitivität bezeichnen könnte. Diese Eigenschaft ist jedoch von Nutzen, denn sie kann dazu beitragen, daß wir einige Kopplungsuntersuchungen erst gar nicht durchführen müssen.

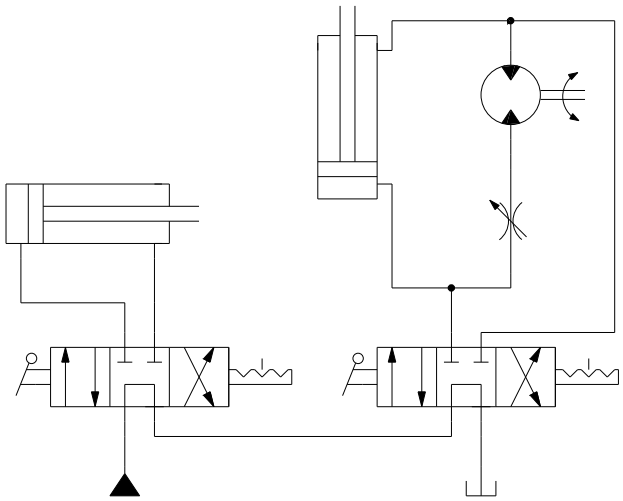

Abbildung 5.6: Unterschiedlich gekoppelte Achsen

Um diese Eigenschaft in der Realisierung unseres Verfahrens auszunutzen, können wir vor der Untersuchung der Kopplungsart eines Achsenpaares die Liste der bereits bestimmten Kopplungen, die diese zwei Achsen miteinschließen, durchgehen und nach einer Kopplung beider Achsen mit einer dritten Achse suchen. Sind bereits zwei Kopplungen zu einer dritten Achse bestimmt worden, so können wir wenigstens auf die schwächste mögliche Kopplung des zu untersuchenden Achsenpaares schließen. Dadurch können wir einige unnötige Untersuchungen auf schwächere Kopplungen vermeiden und dadurch etwas Rechenzeit sparen.

Der einzige zusätzliche Aufwand, den die Ausnutzung der minimalen Transitivität verursacht, stammt aus der Suche in der Liste der bekannten Kopplungen. Dabei werden die zwei Listen bekannter Kopplungen der zwei betroffenen Achsen nach einem gemeinsamen Eintrag durchsucht. Diese Suche verursacht einen Aufwand von $O(n)$, da jeder Eintrag höchstens einmal abgefragt wird – allerdings fällt dieser Aufwand in Anbetracht der gesparten Rechenzeit überhaupt nicht ins Gewicht.

5.3.3 Bildung einer Achsenhierarchie

Im vorigen Abschnitt haben wir uns die Beziehungen zwischen Achsen und Kopplungen etwas näher angeschaut. Wir haben dabei die Kopplungen auf einem sehr abstrakten Niveau untersucht und alle Achsen einzeln betrachtet. Reale hydraulische Anlagen können nun aber weitere interessante Eigenschaften besitzen – eine davon ist, daß Achsen in der Regel nicht alle auf derselben Ebene liegen, sondern sie bilden eine Hierarchie (siehe Abbildung 5.7).

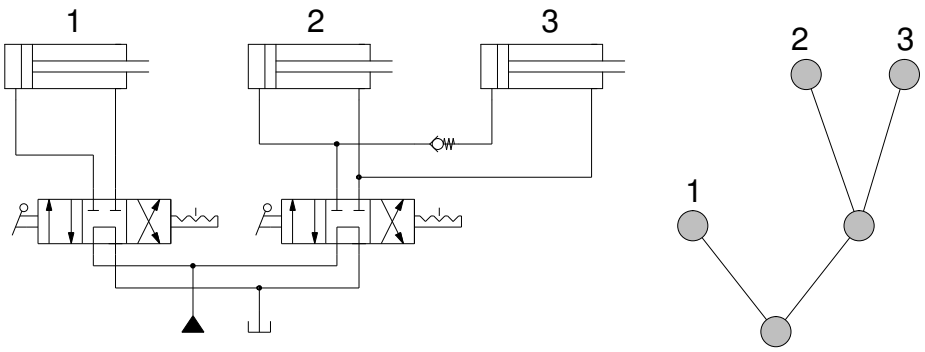

Abbildung 5.7: Eine Schaltung und die entsprechende abstrakte Achsenhierarchie

Durch die Bildung einer Achsenhierarchie können wir ebenfalls etliche Untersuchungsschritte sparen. Man stelle sich die Achsenhierarchie wie ein Baum vor, dessen Blätter gerade die konkreten Achsen und dessen innere Knoten jeweils alle Achsen des Unterbaums darstellen, d. h. die innere Knoten sind „Meta-Achsen“ (siehe Abbildung 5.8). Ist ein solcher Baum vorhanden, so erhalten wir bei Bestimmung der Kopplung zwischen zwei Achsen auch die Kopplung zwischen den Meta-Achsen. Für die nachfolgende Berechnung bedeutet dies, daß die Berechnung der Kopplung zweier Achsen redundant ist, wenn die Achsen auf zwei disjunkten Unterbäumen liegen und die Kopplung der entprechenden Meta-Achsen bekannt ist.

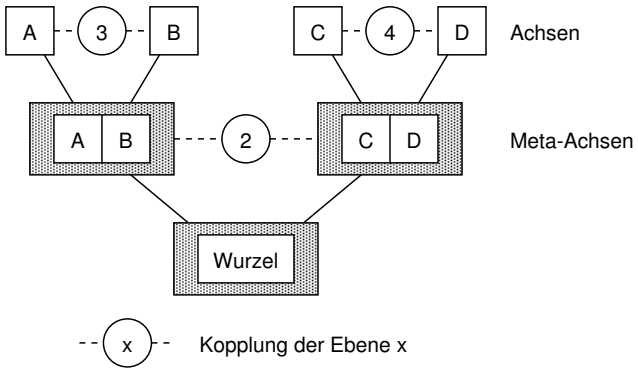

Abbildung 5.8: Abstrakte Achsenhierarchie mit Kopplung der Meta-Achsen

Darüberhinaus können wir mit Hilfe einer Achsenhierarchie genauere Schlüsse ziehen als allein mit der minimalen Transitivität aus dem letzten Abschnitt. Wenn wir wieder drei Achsen betrachten, von denen zwei in einem Unterbaum liegen und die dritte in einem zweiten, vom ersten Unterbaum verschiedenen Unterbaum, so können wir unter Umständen sogar die genaue Kopplung eines Achsenpaares angeben mit Hilfe der Kopplungen der anderen möglichen Achsenpaare.

Dies ist dann der Fall, wenn wir eine Kopplung zwischen Achsen unterschiedlicher Unterbäume und eine Kopplung zwischen Achsen desselben Unterbaums kennen – aus zwei Kopplungen zwischen Achsen unterschiedlicher Unterbäume können wir leider keine Schlüsse ziehen.

Leider stellt die Bildung einer Hierarchie keine triviale Aufgabe dar. Um diese Aufgabe zu lösen, müssen wir leider alle Achsen untersuchen und sie entsprechend in die Hierarchie bzw. in den Baum einfügen. Auch ein inkrementeller Aufbau einer solchen Hierarchie wäre nicht sehr hilfreich, da die Struktur des Baums sich jedesmal ändern könnte.

Die Bildung der oben genannten Hierarchie mag recht schwierig und kostspielig sein, aber man kann trotzdem an diese Kopplungsinformationen herankommen. Wir verzichten auf die explizite Verwaltung eines Baumes und benutzen stattdessen lediglich die Meta-Achsen, die wir inkrementell bestimmen. Hat man die engste Kopplung zweier Achsen bestimmt, so entfernt man diese Achsen aus der Liste und fügt die entsprechende Meta-Achse hinzu. Verfährt man analog mit allen Achsenpaaren, so besteht die Liste der Achsen irgendwann ausschließlich aus Meta-Achsen. Dieser Vorgang kann man auch mit den Meta-Achsen wiederholt durchführen bis am Ende nur noch eine einzige Meta-Achse vorhanden ist.

Kapitel 6

Ergebnisse

In diesem Kapitel möchten wir unser Verfahren als Ganzes zusammenfassen, um einen Überblick über alle Facetten gewinnen zu können, und die Arbeitsweise unseres Verfahrens anhand eines praktischen Beispiels verfolgen. Außerdem möchten wir das Verhalten und die Laufzeit unseres Verfahrens im praktischen Einsatz empirisch untersuchen, um die Einsetzbarkeit beurteilen zu können. Bevor wir allerdings damit anfangen, wollen wir uns unsere Schaltungen und die daraus entstandenen Graphen genauer betrachten.

6.1 Grapheigenschaften

Da unsere Verfahren sowohl auf dem komprimierten als auch auf dem unkomprimierten Graphen arbeiten, müssen wir die Eigenschaften beider Graphen vor und nach der Kontraktion untersuchen, um überhaupt eine Prognose zum Laufzeitverhalten stellen zu können.

Für die weiteren Abschnitte sei G der ursprüngliche Graph mit $G = (V, E)$, und G_K sei der durch die Kontraktion von G entstandene Graph mit $G_K = (V_K, E_K)$. Es gelte dabei $|V_K| < |V|$ und $|E_K| < |E|$.

6.1.1 Eigenschaften vor der Kontraktion

Anders als in den meisten Fällen in der Praxis, besitzen die hydraulischen Schaltungen ganz besondere Eigenschaften. Auf der einen Seite stellen sie in abstrakter Form Multigraphen dar, was meistens zusätzliche Arbeit bedeutet. Auf der anderen Seite haben sie ausschließlich Komponenten von beschränktem Grad, was die Behandlung der einzelnen Komponenten vereinfacht.

Hydraulische Anlagen besitzen einen Schaltkreis, durch den das Öl fließt. Daher bildet eine solche Anlage eine große Schleife. In der für die elektronische Verarbeitung gewählten Darstellung bilden die Schaltungen keine großen Schleifen mehr – zumindest nicht in der Regel –, sie sind aber trotzdem Multigraphen, d. h. zwei Knoten können durch zwei verschiedene Kanten miteinander verbunden sein. Das bedeutet insbesondere, daß eine solche Schaltung Schleifen besitzen darf. Die Abstraktion einer solchen Schaltung in einen Graphen behält all diese Schleifen bei; allerdings stellt dies kein großes Hindernis dar. In all unseren Verarbeitungsschritten haben wir uns ausschließlich für die entfernbaren Schleifen interessiert, also für Schleifen, die keine tragende Rolle in der Schaltung spielen. Wichtige Schleifen besitzen immer bestimmte Komponenten, wie ein Arbeitselement, so daß man solche Schleifen entweder brechen oder einfach ignorieren kann, wie wir es getan haben. Aus diesen Gründen stellen die in den Schaltungen auftretenden Schleifen kein großes Problem dar.

Die hydraulischen Komponenten, die die Knoten des Graphen darstellen, haben den maximalen Grad ≤ 4 in der ursprünglichen Schaltung. Angenommen, alle Knoten des Graphen besitzen den Grad vier, d. h. der Graph ist regulär vom Grad vier. Weisen wir alle Kanten Richtungen zu, wobei jeder Knoten zwei ein- und zwei ausgehende Kanten besitzt, so sieht man leicht, daß $|V| = \frac{1}{2}|E_{AUS}| = \frac{1}{2}|E_{EIN}|$, d. h. es gibt doppelt so viele Kanten wie Knoten. Haben alle Knoten einen Grad ≤ 4, so heißt das, daß der Graph höchstens doppelt so viele Kanten wie Knoten besitzt. In der Praxis hat sich gezeigt, daß die Schaltkreisgraphen in der Regel geringfügig mehr Kanten als Knoten besitzen – eine der größten uns zur Verfügung gestellten Testschaltungen besitzt 115 Knoten und 151 Kanten. Bei den etwas kleineren Schaltungen gilt sogar $|V| \approx |E|$. Eine einfache Folgerung aus der Gradbeschränktheit ist, daß die Kantensortierung gewissermaßen in konstanter Zeit durchgeführt werden kann.

Beide oben erwähnten Eigenschaften stellen keine größere Probleme dar, sondern sie sind auf der einen Seite ein leicht überwindbar, und auf der anderen Seite zeugen sie von der Einfachheit unserer Strukturen. Weiterhin stellt die Eigenschaft des beschränkten Grades eine gewisse Hilfe dar, um das Verhalten und den Aufwand unseres Verfahrens abschätzen zu können.

6.1.2 Eigenschaften nach der Kontraktion

Den Vereinfachungsschritt der Graph-Kontraktion haben wir mit dem Ziel eingeführt, die Verarbeitung auf dem Graphen erst möglich zu machen und zu beschleunigen, indem wir das Problem quantitativ verkleinern. Neben diesen positiven Eigenschaften hat die Graph-Kontraktion leider auch einen Nachteil: Sie macht eine positive Eigenschaft des hydraulischen Graphen zunichte.

Durch die von der Graph-Kontraktion vorgenommenen Transformationen werden auch Knoten miteinander verschmolzen (siehe Abschnitt 2.4.7), so daß der Graph schließlich auch Knoten vom Grad > 4 besitzen darf. Allerdings werden bei einer Verschmelzung zweier Komponenten jeweils zwei Kanten und ein Knoten entfernt, so daß der komprimierte Graph insgesamt echt kleiner als der ursprüngliche Graph ist, sowohl was die Anzahl der Knoten als auch die Anzahl der Kanten angeht. Bei den anderen Kontraktionsschritten werden gleich viele Knoten und Kanten entfernt, und in der Regel wird bei jeder Anwendung einer dieser Regeln der Grad eines Knoten um eins erniedrigt; eine Ausnahme hiervon sind die Regeln zur Ersetzung kritischer Komponenten, die eine Komponente durch mehrere ersetzen.

Also ist der höhere Grad einiger Knoten des komprimierten Graphen die einzige negative Auswirkung der Vorverarbeitung. Allerdings spielt der Verlust des beschränkten maximalen Grades des ursprünglichen Graphen keine Rolle für das Laufzeitverhalten unseres Verfahrens, da der Gewinn durch die Kontraktion des Graphen alles wieder ausgleicht.

6.2 Das Verfahren in seiner Gesamtheit

Nachdem wir in den letzten Kapiteln unterschiedliche Schritte, die zu unserem Verfahren gehören, zum Teil völlig aus dem Zusammenhang gerissen, betrachtet haben, wollen wir sie nun alle zu einem einzigen Algorithmus zusammenfügen, um das in dieser Arbeit entwickelte Verfahren besser überblicken zu können.

Im Abschnitt 2.2 hatten wir folgenden abstrakten Algorithmus zu unserer Verfahrensweise angegeben:

1. Graph-Kontraktion
2. Bestimmung der hydraulischen Achsen
3. Graph-Expandierung
4. Bestimmung der Kopplung zwischen den hydraulischen Achsen

Wir wollen nun den obigen abstrakten Algorithmus etwas verfeinern und jeden Schritt soweit wie möglich aufschlüsseln. Mit den Informationen und Angaben aus den letzten Kapiteln erhalten wir folgenden Algorithmus:

1. **Graph-Kontraktion**: Jeder der Teilschritte der Graph-Kontraktion hat einen asymptotischen Aufwand von $O(n)$. Die letzten Schritte werden gegebenenfalls öfters wiederholt, und zwar solange noch Schleifen vorhanden

sind. Da die Verschachtelung von Schleifen in realen Schaltungen äußerst selten ist, können wir hier mit einem durchschnittlichen Gesamtaufwand von $O(n)$ rechnen.

(a) Ersetzung aller kritischen Komponenten

(b) Löschung von Kontrollkanten

(c) Zusammenfassung von Tanks

(d) Erkennung und Löschung von Ketten gleichartiger Komponenten

(e) Erkennung und Löschung von toten Zweigen

(f) Erkennung und Löschung von Ketten irrelevanter Komponenten

(g) Entfernung von Knoten vom Grad ≤ 2

(h) Verschmelzung benachbarter Verbindungskomponenten

(i) Erkennung und Löschung von Schleifen (gegebenenfalls Wiederholung ab Schritt (f))

2. **Bestimmung der hydraulischen Achsen**: Dieser Schritt besteht lediglich aus der Wiederholung von Pfadsuche und Pfadverfolgung. Da diese zwei Schritte aber sehr oft durchgeführt werden und die dazu verwendeten Algorithmen relativ aufwendig sind, ergibt sich für diesen Schritt ein Gesamtaufwand von $O(n^3)$.

   (a) Für jedes Versorgungselement v tue:

      i. Berechne alle Pfade beginnend von v zu den Arbeitselementen mit dem Algorithmus von Ford

      ii. Ordne alle Knoten auf den Pfaden einer Achse zu

      iii. Schalte eine Kante aller Arbeitselemente aus

      iv. Berechne erneut alle Pfade beginnend von v zu den Arbeitselementen mit dem Algorithmus von Ford

      v. Ordne alle Knoten auf den Pfaden einer Achse zu

   (b) Für jeden Tank t tue:

      i. Berechne alle Pfade beginnend von t zu den Arbeitselementen mit dem Algorithmus von Ford

      ii. Ordne alle Knoten auf den Pfaden einer Achse zu

      iii. Schalte eine Kante aller Arbeitselemente aus

      iv. Berechne erneut alle Pfade beginnend von t zu den Arbeitselementen mit dem Algorithmus von Ford

      v. Ordne alle Knoten auf den Pfaden einer Achse zu

3. **Graph-Expandierung**: Die Graph-Expandierung besteht ausschließlich aus der Wiederherstellung von während der Kontraktion entfernten Knoten und Kanten. Dazu werden alle betroffenen Knoten und Kanten, die bei der Kontraktion auf Stapeln abgelegt wurden, der Reihe nach wieder in den Graphen eingefügt. Dieser Vorgang verursacht einen Aufwand von $O(n)$.

4. **Nachverarbeitung**: Der Schritt der Nachverarbeitung besteht lediglich aus der Erkennung der identischen Achsen. Wie im Abschnitt 3.2 bereits angedeutet, verursacht dieser Schritt einen Aufwand von $O(n^3)$.

   (a) Voruntersuchung der hydraulischen Achsen im komprimierten Graphen

   (b) Bestimmung der identischen Achsen mittels rekursiver Durchsuchung oder schrittweiser Expandierung (dieser Schritt gehört zur Graphexpandierung)

5. **Bestimmung der Kopplung zwischen den hydraulischen Achsen**: Diese letzte Aufgabe besteht aus der Untersuchung eines jeden Achsenpaares auf alle möglichen Kopplungsarten. Laut unseren Überlegungen aus Kapitel 5 verursacht dieser Schritt einen Gesamtaufwand von $O(n^3)$.

   (a) Untersuchung auf Kopplung der Ebenen 0 und 1

   (b) Untersuchung auf Kopplung der Ebenen 2 und 3

   (c) Untersuchung auf Kopplung der Ebene 4

Für unser Verfahren insgesamt ergibt sich dann ein asymptotischer Aufwand von $O(n^3)$ im schlimmsten Fall.

6.3 Ein Anwendungsbeispiel

In diesem Abschnitt möchten wir eine relativ kleine, aber trotzdem informationsreiche hydraulische Schaltung untersuchen, mit der wir jeden Schritt unseres Verfahrens – die Graphkontraktion, die Erkennung der hydraulischen Achsen mittels Pfadsuche, die Erkennung der identischen Achsen sowie die Bestimmung der Kopplungsarten – gut nachvollziehen können. Die Schaltung aus Abbildung 6.1 genügt unseren didaktischen Anforderungen.

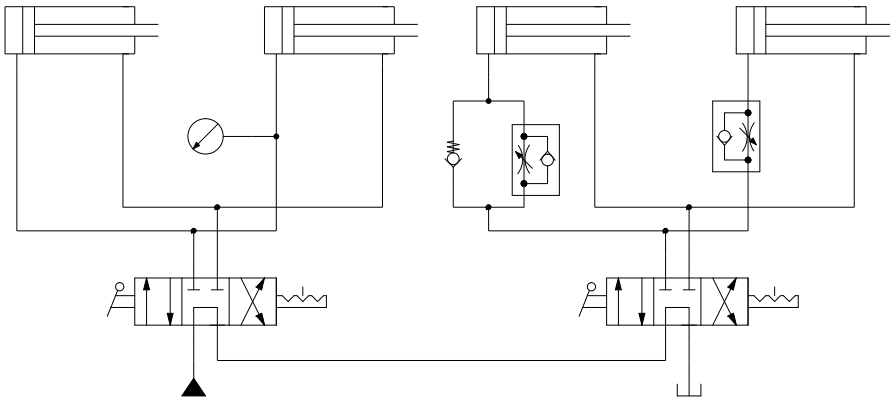

Abbildung 6.1: Eine Beispielschaltung

6.3.1 Graphkontraktion

Die Graphkontraktion ist wohl der wichtigste Schritt unseres Verfahrens – ohne ihn wären die meisten Aufgaben ohne weiteres überhaupt nicht mit bekannten Algorithmen der Graphentheorie lösbar. Dieser Schritt vereinfacht bzw. entfernt einige irrelevante oder störende Konstrukte aus dem zu untersuchenden Graphen. Abbildung 6.2 zeigt die Teile der Beispielschaltung, die von der Graphkontraktion entfernt werden. Es sei angenommen, daß zu diesem Zeitpunkt alle nötige Vorbereitungen, wie die Berechnung der Zusammenhangskomponenten oder Blöcke, bereits abgeschlossen sind.

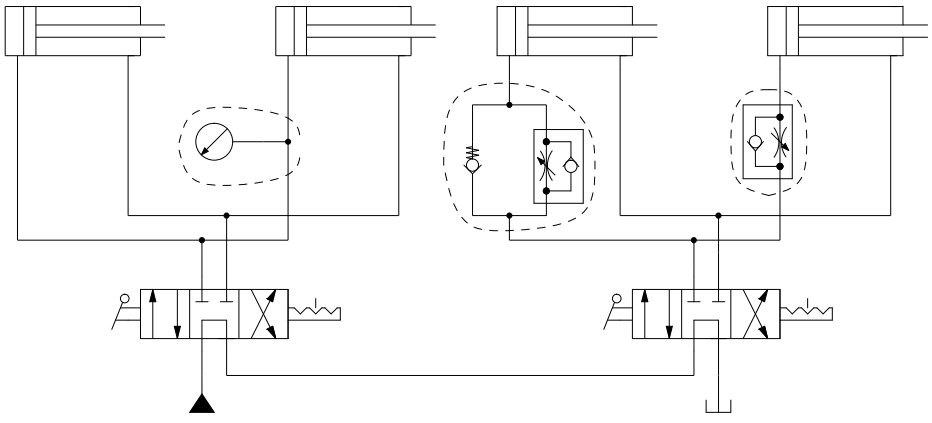

Abbildung 6.2: Die von der Graphkontraktion betroffenen Schaltungsteile

In der Abbildung 6.2 erkennen wir, daß drei Regeln unserer Graph-Grammatik angewendet werden können. Der erste von unserem Algorithmus erkannte Fall ist das Vorhandensein des toten Zweiges. Dieser wird entfernt, und lediglich die Ver-

bindungskomponente bleibt zurück. Man beachte, daß die Entfernung des toten Zweiges in der Entstehung einer weiteren Kette resultiert (siehe Abbildung 6.3).

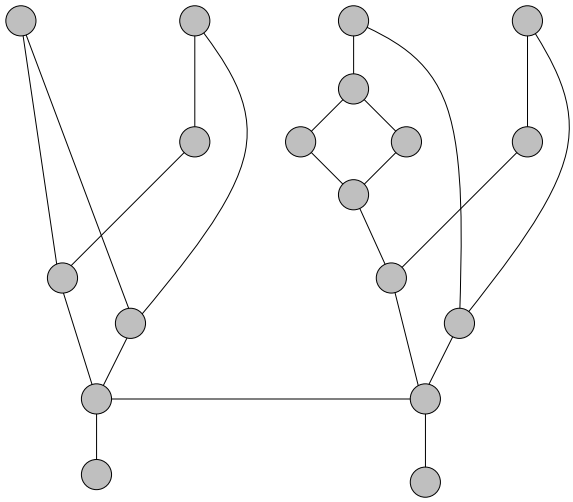

Abbildung 6.3: Abstrakter Graph ohne den toten Zweig

Der zweite erkannte Fall ist das Vorhandensein von Ketten irrelevanter Elemente. Unser abstrakter Graph enthält nun vier solche Ketten, die jeweils aus einer einzigen Komponente bestehen. In den einfachen Fällen sind die Nachbarn der Komponenten ein Arbeitselement und ein Verbindungselement, so daß die zu entfernenden Komponenten gemäß unserer Attraktionsregel zu den Arbeitselementen gruppiert werden (siehe Abbildung 6.4). Im Falle der Schleife können wir leider keine solche Entscheidung treffen, da die Nachbarn der zu entfernenden Komponenten vom gleichen Typ sind.

Der letzte noch erkannte Fall ist das Vorhandensein der (kleinen) Schleife. Die Graph-Grammatik entfernt in diesem Fall alle Knoten außer einer Verbindungskomponente, die anschließend durch die Kettenentfernungsregel zu einem Arbeitselement gruppiert wird. Der abstrakte Graph besitzt nun eine einfache Form, und die Graphkontraktion ist hiermit abgeschlossen.

6.3.2 Erkennung der hydraulischen Achsen

Die Erkennung der hydraulischen Achsen besteht aus der Pfadsuche mit anschließender Wegverfolgung zur Zuordnung der Achsennummern. Abbildung 6.6 zeigt den abstrakten Graph mit den gewonnenen Achseninformationen.

Da die Arbeitsweise des Pfadsuchalgorithmus und der Wegverfolgung im Kapitel 3 bereits detailliert behandelt wurde, verzichten wir an dieser Stelle auf eine erneute Betrachtung.

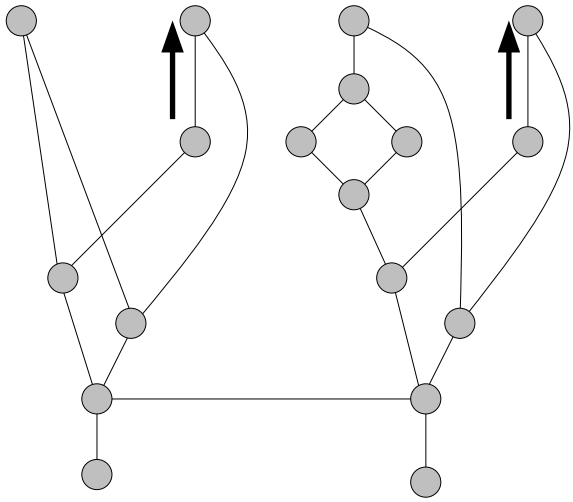

Abbildung 6.4: Kettenelemente und ihre Attraktoren

6.3.3 Erkennung identischer Achsen

Ab diesem Zeitpunkt nehmen wir an, daß unser Graph wieder expandiert wurde.

Unsere Beispielschaltung enthält zwei „identische“ Achsen, die sich allerdings in der Anzahl der Komponenten unterscheiden. Eine der Achsen besitzt einen toten Zweig, der aus einem Druckmeßgerät besteht. Wir haben im Abschnitt 3.2 aber gesehen, daß tote Zweige für die Erkennung identischer Achsen irrelevant sind. Daher erkennt unser Verfahren hier diese zwei identischen Achsen und verschmilzt sie miteinander, so daß beide nun dieselbe Achsennummer besitzen.

Die Erkennung der identischen Achsen geschieht, wie im Abschnitt 3.2 erklärt, durch Verfolgung der Wege, die durch die Nachfolgerinformationen definiert sind. Für weitere Einzelheiten verweisen wir auf den oben genannten Abschnitt.

6.3.4 Bestimmung der Kopplungsarten

Für die letzte noch zu lösende Aufgabe, die Bestimmung der Kopplungsarten, müssen wir ähnlich wie bei der Erkennung identischer Achsen vorgehen, daher verzichten wir auch hier auf eine Erklärung des Verfahrens und beschränken uns auf die einzelnen Schritte, die das Verfahren auf unserer Beispielschaltung durchführt. Wir gehen davon aus, daß alle Achsen in einer geeigneten Datenstruktur, wie zum Beispiel einer Schlange, gespeichert sind, und daß sie in einer günstigen Reihenfolge vorliegen – ist dies nicht der Fall, so macht unser Algorithmus einige Schritte mehr, was keinerlei didaktischen Wert hat.

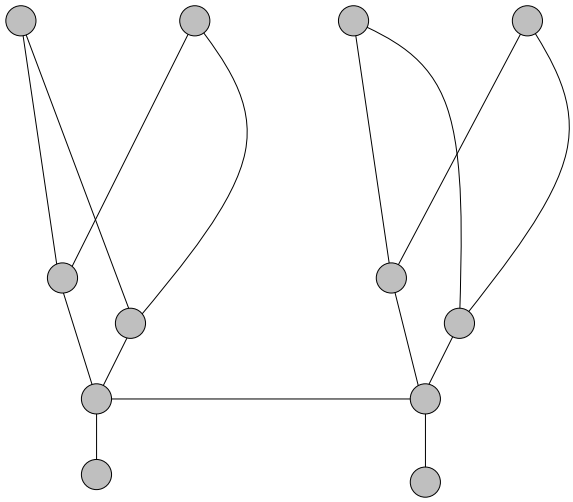

Abbildung 6.5: Der vereinfachte abstrakte Graph

Unser Algorithmus holt die erste Achse – oder besser gesagt den Einstiegspunkt in die Achse, d. h. das Arbeitselement – aus der Schlange und verfolgt den durch die Nachfolgerinformationen definierten Weg, bis eine Komponente gefunden wird, die auch zu einer anderen Achse gehört (siehe Abbildung 6.8).

Daraufhin wird das Achsenpaar untersucht, indem die Wege verfolgt und miteinander verglichen werden. Nach Bestimmung der Kopplungsart wird auch die zweite Achse aus der Schlange entfernt. Beide Achsen werden dann zusammengefaßt zu einer Meta-Achse und in die Schlange eingefügt.

Anschließend holt unser Algorithmus die letzte einfache Achse aus der Schlange und findet, wie im vorigen Fall, eine Komponente, die zu einer anderen Achse gehört, nämlich unserer Meta-Achse. Durch Vergleich der Wege wird die Kopplungsart bestimmt; die Meta-Achse wird aus der Schlange entfernt und mit der einfachen Achse in eine neuen Meta-Achse transformiert. Da diese neue Meta-Achse die einzige übrige ist, hört unser Verfahren hier auf und liefert die Kopplungsarten zwischen allen Achsen, wie in Abbildung 6.10 zu sehen ist.

6.4 Empirische Laufzeitmessungen

Jetzt, wo wir unser Verfahren ausführlich beschrieben und seine Arbeitsweise an einem praktischen Beispiel nachvollzogen haben, möchten wir wissen, inwiefern es in der Praxis einsetzbar ist. Wir haben daher unser Verfahren auf verschiedene Schaltungen angewendet und die Laufzeit dabei gemessen, um zu sehen, wo die Grenzen unseres Verfahrens liegen.

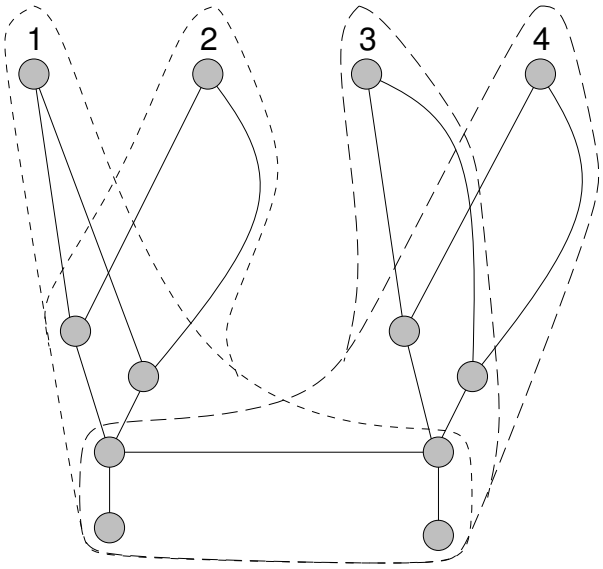

Abbildung 6.6: Die durch die Pfadsuche gefundenen Achsen

Alle Berechnungen wurden auf Workstations des Typs Sun Ultra-1 unter Solaris 2.5 gemacht. Die von uns verwendeten Maschinen haben einen mit 167 MHz getakteten 64 Bit RISC-Prozessor und besitzen 64 MB Arbeitsspeicher. Unsere Ergebnisse stehen auf Tabelle 6.1; alle Zeitangaben sind in Sekunden.

Die auf Tabelle 6.1 angegebenen Zahlen bedeuten: 1) Kontraktion, 2) Achsenerkennung, 3) Expandierung, 4) Nachverarbeitung bzw. Erkennung identischer Achsen, und 5) Erkennung der Kopplungsart. Wir haben für jede Beispieldatei das Verfahren mit und ohne Graph-Kontraktion getestet, wobei die Ergebnisse bei den Tests ohne Graph-Kontraktion nicht immer korrekt waren.

6.5 Bemerkungen zur Implementierung

Ein wichtiges Ziel bei der Implementierung unseres Verfahrens war die Beibehaltung der Modularität. Die Routinen sollen voneinander unabhängig und der Quelltext gut lesbar sein, damit Anwender möglichst keine Schwierigkeiten haben, sich in das Programm einzuarbeiten. Hinzu kommt der Wunsch der einfachen Erweiterbarkeit: Es ist denkbar, daß in der Zukunft weitere hydraulische Komponenten in den hydraulischen Schaltungen verwendet werden; daher müssen unter Umständen einige der Routinen angepaßt werden, um diese Komponenten mitzuberücksichtigen.

Diese Ziele haben wir zu einem bestimmten Grad erreichen können. Auf der einen Seite ist das Programm in Module, die für die einzelnen Aufgaben zuständig

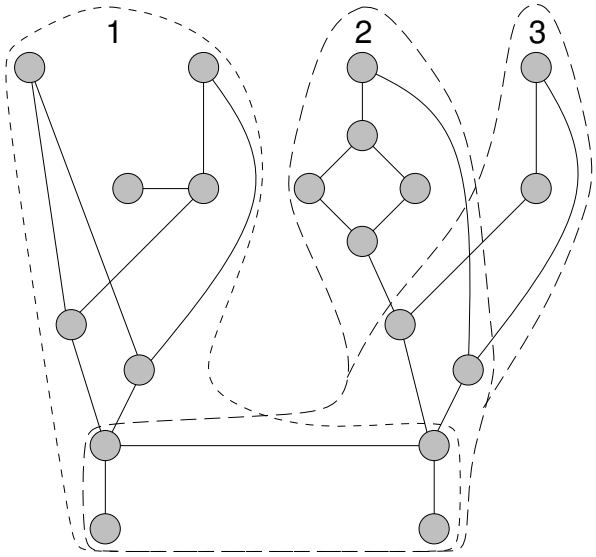

Abbildung 6.7: Die Achseninformation nach Erkennung der identischen Achsen

sind, aufgeteilt; auf der anderen Seite aber benötigen sie oft Informationen und Ergebnisse aus anderen Routinen, und damit wird die Modularität leider verletzt.

Die Erweiterbarkeit des Programms ist, zumindest im aktuellen Zustand, leicht realisierbar. Sind neue, dem Programm unbekannte Komponenten vorhanden, so muß das Programm zunächst nur an einer Stelle erweitert werden, und zwar bei der Ermittlung der Komponentenklasse. Hinzu kommen natürlich Änderungen bzw. Erweiterungen der einzelnen Routinen, die einer gesonderten Behandlung einer solchen Komponenten bedürfen.

Wir haben, wie im Kapitel 2 zur Vorverarbeitung bereits gesehen, die Verwendung von speziellen Hilfslisten befürwortet. Diese Listen ermöglichen eine gezielte Suche nach Komponenten und helfen die Laufzeit zu verringern. Auf der anderen Seite aber müssen sie ständig aktualisiert werden, um die Konsistenz zu wahren. Daher ist bei jeder Erweiterung oder Änderung eine Betrachtung und gegebenenfalls eine Aktualisierung der Hilfslisten notwendig.

Als eine mögliche Verbesserung könnte man in Zukunft von der konkreten Behandlung von einzelnen Komponenten im Programm absehen. Stattdessen sollte es möglich sein, alle Komponenten samt ihren Eigenschaften, Kontext-Bedingungen usw. in einer externen Datei zu verwalten. Dadurch würde die Gefahr, Fehler bei der Erweiterung des Programms einzuführen, beträchtlich reduziert.

Desweiteren wäre eine Erweiterung der internen Datenstrukturen sehr hilfreich, auch wenn dies die Einfachheit der jetzigen Strukturen ein wenig ad absurdum führt. Wir haben bei der Implementierung festgestellt, daß es öfters nötig ist, in dem vom Pfadsuchalgorithmus berechneten Baum zu navigieren. Leider können

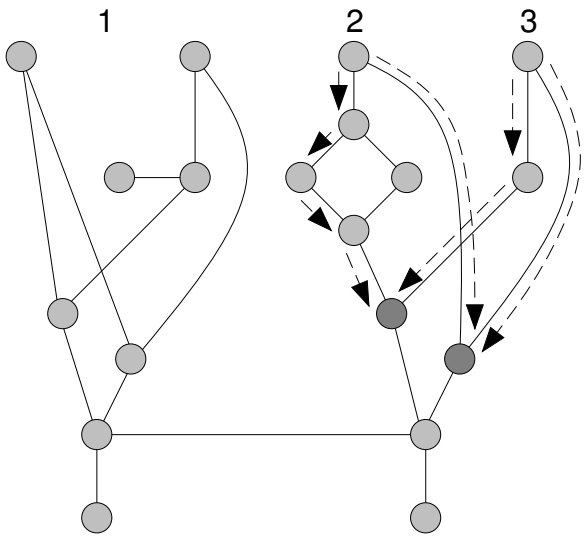

Abbildung 6.8: Wegverfolgung bis zu den gemeinsamen Komponenten

wir momentan dies nur in eine Richtung tun, so daß hin und wieder mehrere Schritte nötig sind, um einen einfachen Rückwärtsschritt zu tun.

6.6 Zusammenfassung und Ausblick

Wir haben für die Probleme der Erkennung der hydraulischen Achsen und der Kopplung Verfahren angegeben, die leider nicht alle Probleme vollständig lösen können, aber auf jeden Fall einen großen Teil davon. Wir haben unser angestrebtes Ziel von 90% Erfolgsquote sogar leicht überschritten.

Um unsere Teilaufgaben zu lösen, haben wir auf verschiedene Konzepte zurückgreifen müssen, wobei die Graph-Grammatiken und die Pfadsuchalgorithmen ungeheuer wichtig für unsere Verfahren sind. Darüberhinaus haben wir auch den Einsatz von Schablonen und deren Einbettung als eine mögliche Alternative zur Pfadsuche in Betracht gezogen und festgestellt, daß dieses Konzept in etwa gleich mächtig bzw. effizient ist.

Es hat sich gezeigt, daß die Verwendung von Graph-Grammatiken zur Kontraktion des Graphen ausschließlich Vorteile mit sich bringt. Wir haben dadurch nicht nur das Problem erheblich vereinfacht und erst lösbar gemacht, sondern auch zum Teil die Verarbeitungsdauer sehr gekürzt, wie man im letzten Abschnitt sehen konnte. Die Effizienz von Graph-Grammatiken ist darüberhinaus mit der ihres Analogons im Bereich der wissensbasierten Systeme – den Produktionsregelsysteme – vergleichbar.

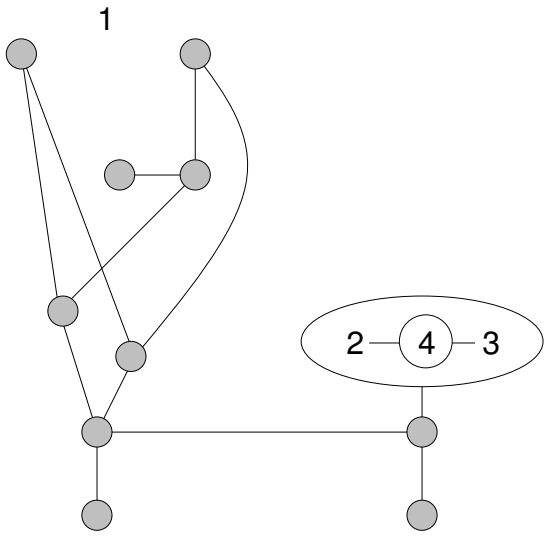

Abbildung 6.9: Der abstrakte Graph mit einer Meta-Achse

Die Pfadsuchalgorithmen stellten leider die „Bottlenecks“ dar und haben daher die Laufzeit unseres Verfahrens wesentlich bestimmt. Daher haben wir bei Graphen, die sich wenig oder gar nicht vereinfachen ließen, relativ schlechte Laufzeiten erhalten.

Für die Zukunft können wir einige Verbesserungsvorschläge machen, um das oben geschilderte Problem zu lösen. Man kann zum einen die Graph-Grammatik erweitern, indem man mächtigere Regeln formuliert, um auch hartnäckige Graphen „klein zu kriegen“. Zum anderen könnte man die Pfadsuchalgorithmen durch ein geeigneteres Konzept ersetzen, so daß bessere Laufzeiten erreicht werden.

Darüberhinaus könnte man allen Regeln der Graph-Grammatik dynamische Prioritäten zuweisen, damit beispielsweise zu einem bestimmten Zeitpunkt immer die Regel gefeuert wird, die dann die höchste Wirkung hat. Dies ist allerdings nur dann sinnvoll, wenn die Zahl der Regeln auch eine bestimmte Grenze erreicht hat – momentan werden lediglich drei Regeln oft benutzt: die Löschung von toten Zweigen, die Löschung von Ketten und die Löschung von Schleifen. Für die Zukunft ist dies aber sicherlich ein gewinnbringendes Konzept.

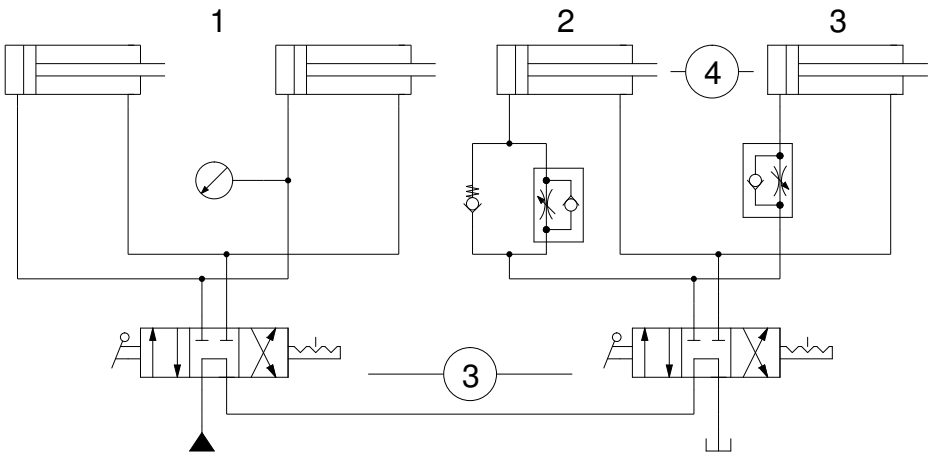

Abbildung 6.10: Die hydraulische Schaltung mit den gefundenen Ergebnissen

Datei	$\|V\|$	$\|E\|$	$\|V_K\|$	$\|E_K\|$	1	2	3	4	5
test	17	20	8	9	0,001	0,005	0,0	0,002	0,001
test	17	20	17	22	0,0	0,007	0,0	0,004	0,001
test14	18	21	12	15	0,001	0,006	0,0	0,006	0,001
test14	18	21	17	22	0,0	0,017	0,0	0,014	0,001
test38	29	36	4	4	0,005	0,005	0,001	0,007	0,001
test38	29	36	30	36	0,0	0,017	0,0	0,007	0.001
test50	39	56	34	49	0,002	0,026	0,001	0,012	0,002
test50	39	56	40	56	0,001	0,028	0,001	0,019	0,002
test37	44	53	20	27	0,006	0,021	0,001	0,009	0,001
test37	44	53	40	58	0,001	0,034	0,0	0,015	0,001
test47	57	78	40	57	0,008	0,041	0,002	0,027	0,004
test47	57	78	58	76	0,001	0,055	0,0	0,025	0,008

Tabelle 6.1: Laufzeitmessungen aus den Testschaltungen

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] S. Bezrukov, W. Unger. Kommunikation und Einbettungen in Netzwerken. Universität - GH Paderborn, 1996.

[2] J. Chen. Algorithmic Graph Embeddings. Texas A & M University, Annual International Conference on Computing and Combinatorics, 1995.

[3] Th. Emden-Weinert et. al. Einführung in Graphen und Algorithmen. Humboldt-Universität zu Berlin, 1996.

[4] D. Jungnickel. Graphen, Netzwerke und Algorithmen. BI Wissenschaftsverlag, 1990.

[5] P. Kelsen, V. Ramachandran. On Finding Minimal Two-Connected Subgraphs. University of Texas, 1992.

[6] D. Kimelman et. al. Reduction of Visual Complexity in Dynamic Graphs, IBM Thomas J. Watson Research Center, 1994.

[7] J. McHugh. Algorithmic Graph Theory. Prentice Hall, 1990.

[8] B. Monien, J. Schulze. Skript zur Vorlesung Parallele Algorithmen. Universität - GH Paderborn, 1995.

[9] M. Nagl. Graph-Grammatiken: Theorie, Anwendungen, Implementierung, Vieweg Verlag, 1979.

[10] Y. Okada, M. Hayashi. Graph Rewriting Systems and their Application to Network Reliability Analysis. NTT Basic Research Lab., NTT Telecommunication Networks Research Lab, International Workshop WG, Band 17, Springer-Verlag, 1991.

[11] A. L. Rosenberg. Issues in the Study of Graph Embeddings. IBM Thomas J. Watson Research Center, International Workshop WG, Springer-Verlag, 1980.

[12] H. J. Schneider. On categorical graph grammars integrating structural transformations and operations on labels. Theoretical Computer Science 109 (1993) 257-274, Elsevier Science Publishers B. V., 1993.

[13] R. Sedgewick. Algorithms in C++. Addison-Wesley Publishing Company, 1992.

[14] W. Shi, D. West. Optimal Algorithms for Finding Connected Components of an Unknown Graph. University of North Texas, University of Illinois at Urbana-Champaign, Annual Conference on Computing and Combinatorics, 1995.

[15] B. Stein. OFT – Objectives and Concepts. Universität - GH Paderborn, 1996.

[16] R. Tamassia, I. G. Tollis. Graph Drawing. DIMACS International Workshop, 1994

[17] E. Vier. Rechnergestützer Entwurf geregelter fluidischer Antriebe. Technical Report MSRT 3/96, Gerhard-Mercator-Universität - GH Duisburg, 1996.

[18] K. Wagner, R. Bodendiek. Graphentheorie, Bd. 3: Zahlen, Gruppen, Einbettungen von Graphen und Geschichte der Graphentheorie. BI Wissenschaftsverlag, 1993.

Diskette zur Diplomarbeit