Auswertung komplexer Design-Anforderungen mittels Scheduling am Beispiel fluidischer Systeme

Diplomarbeit zur Erlangung des Grades eines Diplom-Informatikers des Fachbereichs Mathematik-Informatik an der Universität-Gesamthochschule Paderborn

vorgelegt von:

Carsten H. Thurau

vorgelegt bei:

Prof. Dr. Hans Kleine Büning

Betreuer:

Dr. Benno Stein und Diplom-Mathematiker Marcus Hoffmann

Paderborn, im März 2000

Selbständigkeitserklärung

Hiermit versichere ich, daßich diese Diplomarbeit selbständig und nur unter Zuhilfenahme der angegebenen Quellen und Hilfsmittel angefertigt habe.

Carsten H. Thurau

Paderborn, 9. März 2000

Inhaltsverzeichnis

Abbildungsverzeichnis

Technische Größen

Abkürzung	Einheit	Bedeutung
A_R	mm^2	Kolbenringfläche eines Zylinders
A_K	mm^2	Kolbenfläche eines Zylinders
d	mm	Durchmesser eines Zylinders
F	N	Kraft
M	Nm	Drehmoment
p	$1bar = 100000Pa$; $1Pa = N/m^2$	Druck
P	W (Watt)	Leistung
Q	l/s	Volumenstrom
s	mm	Weg
t	s	Zeit
v	m/s	Geschwindigkeit
V	l	Volumen
W	Nm	Arbeit

Kapitel 1

Motivation und Überblick

1.1. Motivation

Größe und Komplexität technischer Systeme wachsen beständig durch Weiterentwicklung und den parallel dazu steigenden Anforderungen. Dadurch werden die Aufgaben für den Ingenieur schwerer lösbar und zeitlich immer intensiver. Werkzeuge vor allem in Form von Computerprogrammen können die Arbeit des Ingenieurs erleichtern und beschleunigen. Das gilt insbesondere für das Design technischer Anlagen.

Ein Ingenieur besitzt gegenüber dem Computer zwei große, miteinander zusammenhängende Vorzüge: einerseits seine Kreativität — also die Möglichkeit, vollkommen neue und auch ungewöhnliche Lösungswege zu finden —, andererseits seine Fähigkeit, intuitiv Heuristiken zu bilden und zu nutzen. Mit Hilfe dieser Faustregeln versucht der Ingenieur den meist unendlichen Lösungsraum auf einen für ihn halbwegs überschaubaren Bereich zu reduzieren. Diese beiden Fähigkeiten fließen in die Konstruktion einer technischen Anlage ein. Auf der negativen Seite kann ein Mensch den Lösungsraum gerade bei komplexen Anforderungen trotz oder gerade wegen der Heuristiken nicht überblicken, er kann einen vielversprechenden Bereich dieses Lösungsraumes übersehen. Er kann zu einer halbwegs passablen, aber keinesfalls optimalen Lösung kommen. Als weiteres Argument gegen den Ingenieur spricht die menschliche Fehleranfälligkeit. In betriebswirtschaftlicher Hinsicht stört die langwierige Lösungsfindung mit hohen Personalkosten für

den menschlichen Mitarbeiter.

Diese Überlegungen gelten auch und insbesondere im Bereich der fluidischen Systeme. Unter diesem Begriff werden hydraulische und pneumatische Systeme zusammengefasst[1]. Die vorliegende Arbeit und das zugehörige Computerprogramm beschränken sich dabei auf die hydraulischen Systeme. Die Ergebnisse sind mit geringem Aufwand auf den pneumatischen Bereich übertragbar. Dazu müssen nur wenige eingearbeitete physikalische Zusammenhänge und Komponenten-Datenbanken verändert werden.

Der Lehrstuhl „Wissensbasierte Systeme“ von Dr. Kleine Büning an der Universität-GH Paderborn befaßt sich seit Jahren mit der Konfiguration und der Simulation fluidischer Systeme. Das bekannteste Programm ist der hydraulische Simulator FluidSim, auf den in Kap. 5 kurz eingegangen wird In [7] beschreibt Hoffmann, wie der Entwurf eines hydraulischen Systems von den Anforderungen bis zum fertigen Schaltplan mit Hilfe des Computers unterstützt werden kann. Er teilt den Entwurfsprozeß in drei Phasen ein: Anforderungseingabe durch den Benutzer, Abstraktion auf eine funktionale Ebene, Ableitung eines parametrierten Schaltplans.

Die vorliegende Arbeit beschäftigt sich mit den ersten beiden Phasen: die Aufnahme und Analyse der Anforderungen führen zur Darstellung der Funktion des Systems. Die Abbildung 1.1 zeigt die drei Phasen und die dazwischenliegenden Schritte.

1.2. Aufgabe

Hauptpunkt dieser Arbeit ist die Analyse von hydraulischen Anforderungen und damit der Schritt von der Anforderungs- zur Funktionsebene. Beiden Ebenen werden in [7] beschrieben.

Das behandelte Problem P wird in 1.2.1 formal definiert. Aus den hydraulischen Anforderungen soll die totale Ordnung der hydraulischen Phasen — aus denen das Verhalten der hydraulischen Achsen besteht — und der Kopplungsbaum der hydraulischen Achsen erstellt werden.

Daraus entstehen die Aufgaben der Constraintaufstellung und der Constraintverarbeitung.

Die Generierung eines vollparametrierten Schaltplans wird in [7] behandelt.

[1] Hydraulische Systeme arbeiten mit Flüssigkeitsdruck, pneumatische mit Luftdruck.

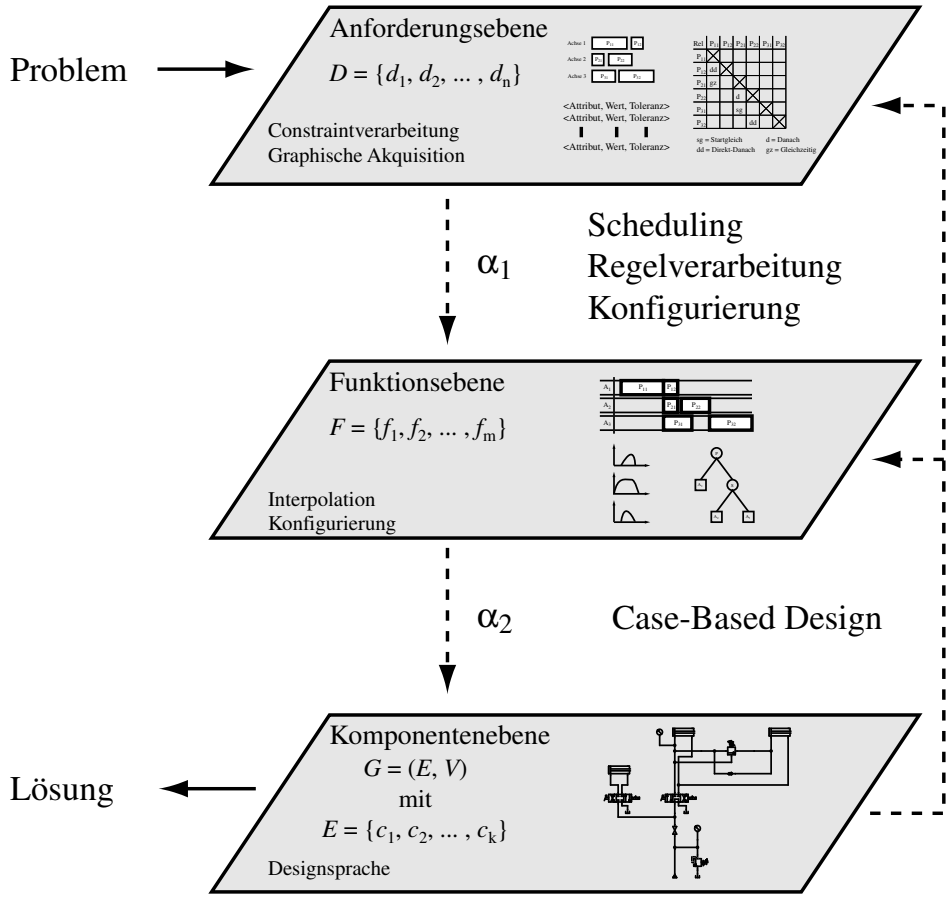

Abbildung 1.1.: Drei Phasen des Entwurfsprozesses aus [7, S.42]

Aufgabe der Arbeit ist einerseits die wissenschaftliche Untersuchung der einzelnen Schritte, andererseits die Entwicklung eines prototypischen Werkzeugs. Letzteres erhält den Namen FluidScheduler.

1.2.1. Problemdefinition

Das zugrunde liegende Problem P kann wie folgt definiert werden:

> Gegeben ist eine Menge A von Achsenanforderungen.
> Jedem $a_i \in A$ ist eine Menge P_i von Phasenanforderungen zugeordnet.
> Die Menge B aller Phasenanforderungen ist die Vereinigung aller P_i.
> Gegeben sei eine Menge C von Constraints über $Pot(A) \times Pot(B)$
>
> Gesucht ist eine Lösung $l =< Z, K >$, die A, B und C erfüllt. Z legt den zeitlichen Verlauf der gegebenen Phasen fest, K den hydraulischen Kopplungsbaum der gegebenen Achsen.
> Die Menge aller Lösungen sei mit L bezeichnet.
>
> Ist ferner eine Gütefunktion $g : L \rightarrow \Re$ bzgl. n Optimierungskriterien gegeben, dann heißt eine Lösung l optimal, wenn $g(l) < g(s) \forall s \in L$.

Die zu lösende Aufgabe besteht also darin, Lösungen für das Problem P zu finden, und aus diesen dann die bestmögliche auszuwählen.

1.2.2. Anforderungen

Die Überlegung, welche Anforderungen vorhanden sind, und in welcher Form sie repräsentiert werden sollen, gehört zur Voraussetzung für die nächsten Schritte. Hierbei wird der Ansatz der hydraulischen Achsen genutzt. Die Definition der hydraulischen Achse folgt auf Seite 25. Grob gesagt wird jede Funktion innerhalb eines hydraulischen Schaltkreises durch genau eine hydraulische Achse ausgeführt. In einer Achse existiert mindestens ein Hydromotor oder Zylinder, der die hydraulische Abtriebsleistung abgibt.

Neben einigen globalen Anforderungen wie dem maximal gültigen Systemdruck beziehen sich die meisten Anforderungen auf das Verhalten einer Achse und die Beziehungen zwischen Achsen. Das Bewegungsverhalten einer Achse kann in verschiedene Phasen zerteilt werden. Mit Hilfe dieser Einheiten braucht der Anwender nur wenige numerische Daten und einige explizite Beziehungen angeben, um die Anforderungen vollständig zu beschreiben. Hier wird viel Zeit gegenüber

herkömmlichen Methoden gespart. Bei diesen braucht der Ingeneur viel Zeit allein für das Zeichnen und Verbessern der Sollprofile . Ein Computerprogramm kann die Profile in Sekunden berechnen.

1.2.3. Ziele

Die beiden Hauptziele nach 1.2.1 sind die zeitliche Einordnung der Phasen und der Aufbau der Abhängigkeiten zwischen den Achsen (Kopplungsbaum). Daraus ergeben sich weitere Ziele: die Ausfahr- und Kraftprofile einzelner Achsen und die Parametrierung einiger Komponenten.

Zeitliche Einordnung der Phasen

Die Reihenfolge der Phasen einer einzelnen Achse ist vorgegeben. Verschiedene Phasen sind jedoch von Phasen anderer Zylinder abhängig. Die zeitliche Einordnung der Phasen und auch ihre zeitliche Länge soll bestimmt werden. Es handelt sich um ein Scheduling-Problem.

Achsenkopplung

Wie erwähnt sind die einzelnen Achsen untereinander nicht unabhängig. Sie sind nach bestimmten Kriterien „gekoppelt“. Die einzelnen Kopplungsarten verursachen unterschiedliche Verhaltensweisen und auch unterschiedliche Kosten. Bei der seriellen Kopplung beispielsweise hängt das Verhalten der zweiten Achse vom Verhalten der erste ab. Dabei sind die Kosten von Bau und Betrieb kleiner als bei der parallelen Kopplung. Die Kopplungen der Achsen können insgesamt als binärer Baum dargestellt werden[2], mit den Achsen als Blättern und der Art der Kopplung an den inneren Knoten. Dabei sind bestimmte Einschränkungen wegen unverträglicher Kopplungsarten und natürlich aufgrund von Anforderungen zu bedenken. Die Berechnung des Kopplungsbaum ist ein weiteres Ziel der Analysephase.

[2]Theoretisch kann es zu parallelen Kopplungen von mehr als zwei Achsen kommen. Durch die Definition der Meta-Achse in Kapitel 2.1.4 werden nur Kopplungen zweier Meta-Achsen erlaubt. Die parallelen Kopplungen mehrerer Achsen können durch hintereinander gestellte parallele Kopplungen simuliert werden.

Sollprofile

Bei den Sollprofilen handelt es sich um Funktionen im klassischen Sinn. Es gibt Bewegungsprofile (Weg-Zeit-Diagramme) und Kraftprofile (Kraft-Zeit-Diagramme) der Zylinder bzw. Hydromotoren. Diese hängen nicht nur direkt von den in den Anforderungen angegebenen Bewegungs- und Kraftgrössen ab, sondern auch von der berechneten zeitlichen Einordnung.

Parametrierung der Arbeits- und Versorgungselemente

Die Auswahl eines geeigneten Zylinders und einer geeigneten Pumpe gehören schon zum Schritt des Designs. Die Auswahl beeinflußt jedoch in starkem Maße die Kosten des Projektes. Dazu kommt, daß die Auswahl an Arbeitselementen und Pumpen [3] den Lösungsraum stark einschränkt. Wenn die leistungsstärkste Pumpe durch einen zulässigen Höchstdruck von 100 bar eingeschränkt wird, sollte in der Funktionsebene kein Systemdruck über diese Größe hinaus verlangt werden, weil sonst in der Synthesephase keine Lösung möglich ist. Die integrierte Parametrierung der Arbeits- und Versorgungselemente ist also wichtig für das Zusammenspiel zwischen dem Analyse- und dem Synthese-Teil.

1.2.4. Methoden

Nun soll ein kurzer Überblick über die benutzten Methoden gegeben werden.

Multi-Kriterien-Analyse

Beim Bau einer hydraulischen Anlage soll der Gewinn maximiert werden. Dabei muß nicht nur der direkt in Geld meßbare Gewinn bedacht werden, sondern auch andere Ziele wie eine lange Lebensdauer, die wiederum von der Größe des Systemdrucks abhängt. Eine Verengung des Zielsektors auf eine minimale geldliche Anschaffungsausgabe zu Anfang kann zum Bumerang werden, wenn es aufgrund dessen zu übermässig langen Produktionszeiten, großen Ausfallzeiten wegen Wartung, oder gar kurzer Lebenszeit kommt. Man muß also verschiedene Ziele in die Lösungsfindung einfließen lassen.

[3] Zylinder in Massenproduktion sind weitaus billiger als für den Sonderfall konzepierte Produkte. Deswegen gibt es standardisierte Baugrößen. Zylinder beliebiger Länge sind zwar möglich (innerhalb physikalischer Grenzen), jedoch teurer.

Die erstrebenswerten Ziele werden in Kapitel 3.6 festgelegt. Sie sollen durch den Benutzer gewichtet werden können.

Constraint-Verarbeitung und Scheduling

Für die Verarbeitung der verschiedenen Beziehungen zwischen Phasen (Beispiel: die Beziehung „Phase 1 muß vor Phase 2 stattfinden“) und für die sukzessive Einschränkung von Größen (Druck, Kraft) bieten sich Constraints an. Nicht nur innerhalb einer Achse hängen Größen voneinander ab (beispielsweise Kraft von Fläche und Druck), das gleiche gilt für miteinander gekoppelte Achsen. Für die Lösung werden die Werte vor allen Dingen durch Propagierung der Contraints eingeschränkt. Durch einfache lokale Propagierung sind Probleme zumeist nicht lösbar, so daß der Lösungsraum durch Auswahlverfahren eingeschränkt werden muß. Diese verursachen häufig hohe Laufzeitkosten. Deswegen sind gute Auswahlverfahren für den nächsten ausgewerteten Constraint und kluge Auswahlkriterien für das Vorantreiben des Backtracking-Algorithmus notwendig. Letzterer ist für die Fälle zuständig, in denen die Constraints nicht genügend einschränkend wirken. Dann muß eine Variable gewählt werden, deren Änderung möglichst viele andere Werte möglichst stark einschränkt. Dazu darf die Variable nicht zu viele Werte haben, damit die Anzahl der Backtrack-Anläufe gering bleibt. Außerdem sollten Backtrackingdurchläufe abgebrochen werden, wenn die bisherige Bestlösung auf jeden Fall besser ist als alle Lösungen, die zum Zeitpunkt des Backtracking-Aufrufs noch möglich wären.

Mit Hilfe der Constraints wird auch das Scheduling-Problem gelöst, denn die verschiedenen Beziehungen zwischen den Achsen können durch Constraints gut dargestellt werden. Während der Constraintverarbeitung werden weitere Beziehungen zwischen Achsen aufgebaut, die nur indirekt (über andere Achsen) in Beziehung stehen. Das Constraintnetz wird dichter, gleichzeitig werden die Variablen des Netzes immer mehr eingeschränkt. Zum Schluß steht die zeitliche Einordnung.

1.3. Aufbau der Arbeit

Verschiedene Bereiche aus Physik, Operation Research und Informatik haben für die Lösungsfindung eine Rolle gespielt. In **Kapitel 2** werden die nötigen Grundlagen des Maschinenbaus, der Graphentheorie, der Constraint-Erfüllung und des Scheduling aufgeführt.

Kapitel 3 handelt über die Auswahl der Anforderungen und Ziele des Verfahrens. Insbesondere werden die hydraulischen Phasen eingeführt und die Zielkriterien für die Lösungssuche angegeben. Die verschiedenen Anforderungen werden in Bereiche aufgeteilt, die sich später auch bei der Modellbildung wiederfinden lassen.

Kapitel 4.1 beschreibt die Abbildung des hydraulischen Modells und der für die Lösung wichtigen Datenstrukturen auf ein Computer-faßbares Modell. Dabei werden die objektorientierten Datenstrukturen, also Objekte und Methoden, angesprochen. Besonderes Augenmerk wird auf das Constraintnetz geworfen.

In **Kapitel 4** geht es um die tatsächlichen Lösungsansätze und -routinen. Dabei wird auch auf die Nutzbarkeit der in den vorherigen Kapiteln genannten Ansätze eingegangen.

Das **Kapitel 5** schließt die Arbeit mit einer kritischen Betrachtung der Ergebnisse ab. Außerdem werden Perspektiven für die Erweiterung aufgezeigt. Die Eingliederung des Programms in den Kontext um den hydraulischen Simulator FluidSim und das Zusammenwirken mit anderen Programmen dieses Kontextes wird beschrieben.

Kapitel 2

Vorbereitung

2.1. Grundlagen des Maschinenbau

2.1.1. Aufgabe einer Hydraulikanlage

Eine hydraulische Anlage soll mechanische Leistung bereitstellen. Diese Leistung besteht aus $P = F \cdot v$, also dem Produkt aus Kraft und Weg bei translatorischer Bewegung (Zylinder), oder aus $P = M \cdot \omega$, also dem Produkt aus Drehmoment und Winkelgeschwindigkeit bei rotarischer Bewegung (Hydromotor). Diese Ausgangsgröße ergibt sich aus mehreren Energieumformungen, bei denen mindestens einmal hydraulische Leistung $P_h = Q \cdot \Delta p$ auftritt. Bei vielen hydrostatischen Anlagen erzeugt ein Elektro- oder Verbrennungsmotor mechanische Antriebsleistung, die eine Pumpe dann in hydraulische Leistung umwandelt. Die Hydropumpe versetzt Flüssigkeit in Bewegung. Die Flüssigkeit durchläuft hydraulische Leitungen und Ventile bis hin zu den Antriebselementen. Diese Hydromotoren und -zylinder geben dann mechanische Abtriebsleistung ab. Das Bild 2.1 zeigt diesen Leistungsfluß.

Die Abbildung verdeutlicht den Verlust von Leistung sowohl beim Durchfluß als auch bei der Umwandlung von Energie. Leistung geht vor allem durch Reibungswärme und Leckströme verloren.

Zur besseren Bearbeitung werden hydraulische Anlagen durch schematische Schalt-

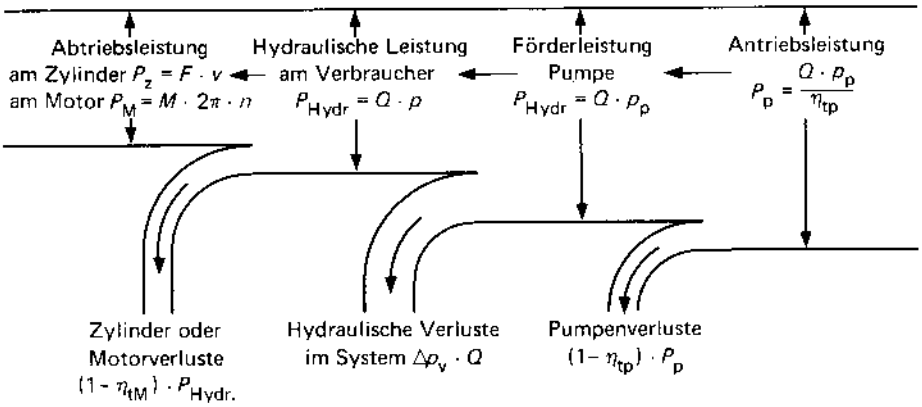

Abbildung 2.1.: Leistungsfluß in einer Hydraulikanlage[11, S. M19]

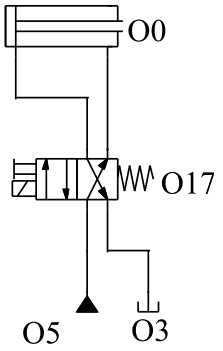

Abbildung 2.2.: Hydraulischer Schaltplan, bestehend aus Zylinder O0, Tank O3, Pumpe O5 und Ventil O17.

pläne dargestellt. Die Abbildung 2.2 zeigt einen solchen Schaltplan. Er besteht aus einem Tank O3, einer Pumpe O5, dem Zylinder O0 und dem Ventil O17. Den Fluß kann man durch Blick auf die Leitungen erkennen. In der momentanen Stellung würde die Flüssigkeit von der Pumpe durch das Ventil in die kleine Kammer von O0 fließen. Aus der größeren Kammer würde gleichzeitig Flüssigkeit durch das Ventil zurück in den Tank geleitet. Jedoch ist die Zylinderstange vollkommen eingefahren und kann sich deswegen nicht in die gewünschte Richtung bewegen. Deswegen findet im eingestellten Zustand kein Fluß statt.

2.1.2. Grundbegriffe

Im folgenden werden nur Grundbegriffe und -zusammenhänge erklärt, die zum Verständnis der Arbeit wichtig sind.

Die Leistung einer Pumpe kann durch Multiplikation von Volumenstrom (in l/s) und Druck (in $bar = 10N/cm^2$) errechnet werden. Der Volumenstrom ist die Flüssigkeitsmenge, die innerhalb eines bestimmten Zeitrahmens durch eine Stelle läuft. Es kann keine Flüssigkeit entweichen, so muß also an allen Stellen des Rohres der Volumenstrom gleich sein.[1] Bei einer Gabelung teilt sich auch der Volumenstrom auf, abhängig von den Widerständen auf dem weiteren Weg.

Für Druck hingegen gilt: wenn auf eine in einen Behälter eingeschlossene Flüssigkeit von außen über eine Fläche eine Kraft einwirkt, dann entsteht in der Flüssigkeit Druck. Dieser ergibt sich direkt aus der Beziehung $p = F/A$. Der Druck wirkt an jeder Stelle des Behälters gleich, er pflanzt sich also gleichmäßig nach allen Seiten fort. Somit ist auch bei einer T-Verzweigung der Druck in allen 3 anliegenden Rohren gleich. Die Richtung des Flüssigkeitsstroms eines Rohres (zur T-Verzweigung hin oder von ihr weg) spielt keine Rolle für den Druck.

Beim Durchlauf des Flüssigkeitstroms durch Kontrollelemente (Ventile) kommt es allerdings zu Druckverlusten, die unter anderem von der Geschwindigkeit abhängen[2] . Genauso kommt es bei hohen Drücken zu Flüssigkeitsverlusten: je höher der Druck wird, desto größer werden auch die Leckagen, der Reibungsverschleiß und der Lärm beim Schalten. Außerdem sinkt die Lebensdauer.

Die Druck- und Flüssigkeitsverluste sind nicht in jedem Fall ungewollt. Es gibt beispielsweise Ventilarten (Druckbegrenzungsventile), die das System gegen zu hohen Druck schützen sollen. Wenn der Druck in Höhe des Ventils zu hoch wird, leitet das Ventil Flüssigkeit direkt in den Tank zurück und hält den Druck auf einem konstanten Grad. Dadurch geht natürlich Volumenstrom verloren. Die Druckverluste, ob gewollt oder Nebenwirkung, müssen in die Überlegungen zur Konstruktion des Schaltplans einbezogen werden.

Der Volumenstrom aus der Pumpe ergibt sich aus der Multiplikation von Umdrehungsgeschwindigkeit (in n/s) und Schluckvolumen (in l/n). Dabei ist n die Drehzahl. Die Antriebsleistung der Pumpe (und damit des Kreislaufs) ergibt sich aus dem Druck und dem Volumenstrom, geteilt durch die Zeit ($P_{an} = p \cdot Q$).

[1] Bei Rissen, bei verschraubten Rohren und in Verteilerstücken entsteht jedoch immer ein kleiner Flüssigkeitsverlust, also ein Volumenstromverlust.

[2] Bei steigendem Volumenstrom steigt der Druckverlust quadratisch. Es gibt dazu weitere Faktoren.

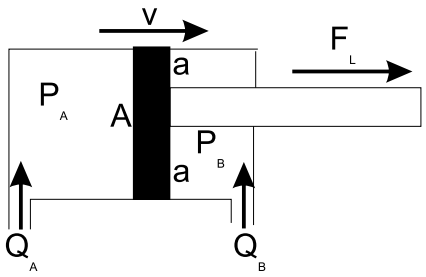

Abbildung 2.3.: Kenngrößen am Zylinder

Die Abtriebsleistung[3] des hydraulischen Kreislaufs besteht aus den Abtriebsleistungen der Arbeitselemente, also der Hydromotoren und Zylinder.

Die Abbildung 2.3 zeigt eine vereinfachte Darstellung eines Zylinders. Die Bezeichung a ist zweimal angegeben, um noch einmal zu verdeutlichen, daß es sich um die gleiche Fläche handelt. In der zweidimensionalen Sicht ist dies leider nicht so leicht ersichtlich.

Bei einem Zylinder wirkt eine auswärtige Kraft auf einen Kolben. Der belastete Kolben stellt für die Flüssigkeit einen Widerstand dar. Der Druck muß also ansteigen, bis die von innen an dem Kolben wirkende genauso groß ist wie die auswärtige Kraft. Die von innen wirkende Kraft wird mit F_L bezeichnet. Wegen $p = F/A$ muß also $p_{A_K} = F_L/A_K$ gelten, wobei A_K die Kolbenfläche ist. Die Geschwindigkeit des Ausfahrens ergibt sich aus dem einströmenden Volumenstrom: $v = Q_A/A_K$. Damit gibt sich auch der Volumenstrom, der in die kleine Kammer einfließt[4]: $Q_B = -v \cdot A_R$. Die Abtriebsleistung des Zylinders ergibt sich aus $P_{ab} = v \cdot F_L$, also aus der Kraft und der Geschwindigkeit.

Bei Hydromotoren muß statt einer Kraft ein Drehmoment eingerechnet werden, das sich gleichsam als Widerstand auswirkt. Im Gegensatz zum Zylinder fließt genauso viel Volumenstrom in den Motor wie wieder heraus, da es keine ungleich großen Kammern gibt wie bei Zylindern.[5] Die Abtriebsleistung ist $P_{ab} = M \cdot 2 \cdot \pi \cdot n$, wobei n die Drehzahl ist.

Der Leistungswirkungsgrad ergibt sich zu einem Zeitpunkt aus der Division von Gesamtabtriebsleistung durch die Antriebsleistung. Die Gesamtabtriebsleistung ist die Summe aller einzelnen Abtriebsleistungen der Hydromotoren und Zylinder.

[3]Die Abtriebsleistung ist die Leistung, die die Anlage durch die Arbeitselemente nach außen gibt.

[4]Man sieht, daß der Volumenstrom nicht in beiden Kammern gleich ist. Es gilt immer das Verhältnis $Q_A/A_K = Q_B/A_R$.

[5]Natürlich muß mit geringen Flüssigkeitsverlusten gerechnet werden.

2.1.3. Hydraulische Achsen

Atomare Komponenten

In [22] werden die atomaren Komponenten in 4 Klassen unterschieden.

- *Versorgungselemente* sorgen für den Flüssigkeitsstrom und den hydraulischen Druck. Sie setzen die Antriebsleistung in hydraulische Leistung um und sind damit Quellen hydraulischer Leistung. In dieser Arbeit werden nur Hydropumpen – kurz: Pumpen – benutzt.

- *Arbeitselemente* sind Quellen mechanischer Leistungen und Senken hydraulischer Leistung. Es handelt sich also um diejenigen Elemente, die Abtriebsleistung abgeben. In dieser Arbeit werden zwei Arbeitselemente benutzt: Zylinder und Hydromotoren.

- *Kontrollelemente* steuern und regeln Druck und Volumenstrom. Alle Arten von Ventilen sind Kontrollelemente. In dieser Arbeit werden keine speziellen Ventile einbezogen. Die Ventile werden allerdings als vorhanden angenommen, ihre Auswirkungen werden mit Hilfe von Heuristiken in die Berechnungen eingeführt.

- *Hilfselemente* sind die Elemente, die nicht in eine der anderen Klassen passen: Speicher, Tanks, Schläuche, Filter etc.

Hilfselementen und Kontrollelementen kommt eine untergeordnete Rolle in dieser Arbeit zu. Die Auswirkungen dieser atomaren Komponenten werden größtenteils durch Heuristiken eingearbeitet.

Hydraulische Achsen

Aus [21]:

> 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.

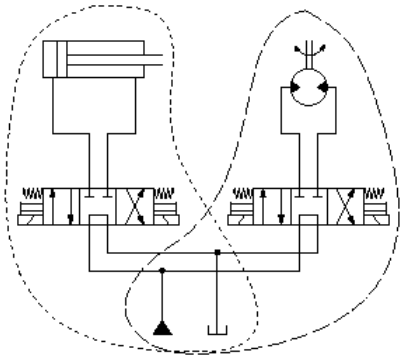

Abbildung 2.4.: Zwei hydraulische Achsen in paralleler Kopplung (aus [15])

Eine hydraulische Achse besteht somit aus mehreren atomaren Komponenten. Sie übernimmt eine Funktion innerhalb einer Anlage. Beispielsweise kann eine Achse für den Antrieb eines Rads verantwortlich sein. Notwendig für jede hydraulische Achse ist mindestens ein Versorgungselement und mindestens ein Arbeitselement. Beim Antrieb eines Rads wäre letzteres ein Hydromotor. Wie man im Anschluß bei den Kopplungsarten sieht, können Achsen sich Komponenten teilen. Eine Ausnahme bilden die Arbeitselemente. Davon wird in dieser Arbeit stark Gebrauch gemacht: bei den erzeugten hydraulischen Anlagen wird jeweils nur eine Pumpe benutzt, die sich sämtliche Achsen teilen müssen. Diese Vorgehensweise wirkt sich positiv auf die geldlichen Kosten aus.

Im Bild 2.2 konnte man nur eine Achse sehen. Die Abbildung 2.4 hingegen zeigt zwei Achsen. Beide teilen sich Pumpe und Tank, besitzen jedoch eigene Kontrollelemente und Arbeitselemente (links ein Zylinder, rechts ein Hydromotor).

2.1.4. Kopplungsarten zwischen hydraulischen Achsen

Innerhalb einer großen, komplizierten Anlage ist das Vorhandensein von mehreren hydraulischen Achsen unabdinglich. Im Zuge der Kostenoptimierung, aber auch des Ineinandergreifens verschiedener Funktionen ist es sinnvoll, bestimmte Komponenten für mehrere Achsen gleichzeitig zu benutzen. Das gilt vor allem für die Versorgungselemente, aber auch für Kontrollelemente.

In [17] und [15] werden fünf Kopplungsebenen für Achsen definiert. Nur drei sind im Rahmen dieser Arbeit interessant. Die formalen Definitionen für diese Kopplungsarten können in den beiden angegebenen Schriften nachgeschlagen werden.

Zwei miteinander gekoppelte Achsen werden zu einer größeren Einheit zusammengefaßt, der sogenannten hydraulischen Meta-Achse. Meta-Achsen können wiederum gekoppelt werden. Zur Vereinfachung definieren wir die Meta-Achse wie folgt:

- Jede hydraulische Achse bildet eine hydraulische Meta-Achse.
- Zwei gekoppelte hydraulische Meta-Achsen bilden eine hydraulische Meta-Achse [6].
- Zu jeder Meta-Achse gehört ein Kopplungs-Attribut für die Art der Kopplung. Dieses trägt den Wert „id“, wenn die Meta-Achse aus einer Achse besteht, ansonsten die Nummer der Kopplungsebene.

Somit ist es einfach möglich, die Kopplungsarten für Achsen und Meta-Achsen zugleich zu definieren.

Ebene 0: keine Kopplung

Wenn A und B keinerlei Verbindung zueinander besitzen, liegt keine Kopplung vor. Es handelt sich eher um zwei verschiedene Schaltungen als um eine Schaltung mit mehreren Funktionen. Diese Nicht-Kopplung ist für die Aufgabenstellung nicht interessant. Zwei getrennte Achsen besitzen u.a. zwei Pumpen. Damit sind sie im Vergleich zu gekoppelten Achsen zu teuer.

Ebene 1: informationelle Kopplung

Zwischen A und B gibt es nur sogenannte Kontrollverbindungen. Das bedeutet: Die beiden Meta-Achsen sind zwar miteinander verbunden (meist über Regelungstechnik), beeinflußen sich aber nicht. Es gibt für jede Meta-Achse eine eigene Pumpe. Die Verbindung dient einzig und allein zu Kontrollzwecken. Wie die Kopplungsebene 0 ist dieser Fall nicht interessant für die Lösung des Problems P.

[6] In der Praxis können mehr als zwei Achsen gleichzeitig gekoppelt sein, sie können u.a. parallel zueinander liegen. Aus Vereinfachungsgründen werden größere Zusammenhänge durch mehrere Meta-Achsen dargestellt. Die parallelen Achsen A, B, C und D können durch die Meta-Achsen A+B und C+D sowie eine weitere, diese beiden Meta-Achsen zusammenfassende Meta-Achse dargestellt werden.

Ebene 2: parallele Kopplung

A und B teilen sich Zu- und Abfluß und damit auch eine Pumpe. Jede Meta-Achse besitzt aber eigene Kontrollelemente, es gibt keine gemeinsamen Kontrollelemente. Die Achsen arbeiten somit unabhängig[7]. Im Gegensatz zu den Kopplungsarten der Ebene 3 und 4 kann die parallele Kopplung durch die Definition nicht in einem Teilbaum auftreten, der mit einem anderen Teilbaum in seriellem oder sequentiellen Verhältnis steht.

Die Abbildung 2.4 zeigt eine parallele Kopplung.

Ebene 3: serielle Kopplung

A und B sind seriell gekoppelt, wenn mindestens eine der Meta-Achsen einen Zu- oder Abfluß enthält, der auf dem Weg zwischen Pumpe und dem Arbeitselement durch mindestens ein Kontrollelement der anderen Meta-Achse läuft, ohne durch das Arbeitselement der anderen Meta-Achse zu laufen.

Ebene 4: sequentielle Kopplung

Bei der sequentiellen Kopplung spricht man auch von einer Folgeschaltung: beide Meta-Achsen besitzen die gleichen Kontrollelemente. Eine der beiden Meta-Achsen besitzt zusätzlich eine Komponente, die ihr Verhalten beeinflußt (beispielsweise Druckminderung oder zeitliche Verzögerung).

2.1.5. Physikalische Auswirkungen der Kopplungen

Die Bedeutungen von Kopplungen kann man an den physikalischen Auswirkungen zeigen. Die bedeutenden Werte im Bereich des Flußes von den Versorgungs- zu den Arbeitselementen sind Druck (in bar angegeben) und Volumenstrom (in l.). Weitere Werte sind beispielsweise Rohrdurchmesser (mit direkter Auswirkung auf Reibung und damit auf Wärmeverluste) und die Winkel bei T-Verzweigungen. Hier sollen nur die beiden wichtigsten Werte Druck und Volumenstrom betrachtet werden. Die Druckverluste werden mit Abzügen vom Zustrom/Abstrom eingearbeitet.

[7] Diese Aussage stimmt nur in regelungstechnischer Hinsicht: unter anderem gilt die Abhängigkeit, daß sich der Flüssigkeitsstrom der zusammenfassenden Meta-Achse auf die beiden gekoppelten Meta-Achsen aufteilt.

Eine Meta-Achse Z ist entweder eine Achse, oder sie besteht aus zwei Meta-Achsen X und Y sowie einer Kopplungsinformation. Im Prinzip (wenn man die Kontrollelemente vernachlässigt) gibt es einen Zustrom zu und einen Abstrom von der Meta-Achse Z. Innerhalb der Meta-Achse gibt es einen Zustrom zu und einen Abstrom von jeder der beiden Meta-Achsen X und Y. Das gilt natürlich nur für Meta-Achsen, die keine Achsen sind. Zustrom und Abstrom laufen nicht die ganze Zeit über die gleichen Leitungen. Das ist einfach zu erklären für den Zylinder in Abbildung 2.3: der Zustrom geht im Anfangszustand in die größere Kammer. Spätestens, wenn diese Kammer voll ist, muß der Zustrom in die andere Kammer und der Abstrom aus der großen Kammer geschehen, sonst kann keine weitere Bewegung stattfinden. Die Leitungen zu den Kammern sind natürlich unterschiedlich.

Bei allen Kopplungen gilt: wenn für die eine Achse kein Zustrom stattfindet, entsprechen Zustrom und Abstrom der anderen Achse dem Zustrom und Abstrom der Meta-Achse.

Bei **paralleler Kopplung** teilt sich der Zustrom zu Z in den Zustrom zu X und Y auf, der Abstrom von Z ist die Summe der Abströme von X und Y. Der Druck im Zustrom von X muß mit dem Druck im Zustrom zu Y und zu Z übereinstimmen, daß gleiche gilt für den Abstrom. [8]

Wenn bei der **seriellen Kopplung** beide Achsen aktiv sind, also Zufluß zu beiden stattfindet, dann passiert folgendes: Flüssigkeit fließt in die Arbeitselemente der ersten Achse. Von dort fließt Flüssigkeit (bei Hydromotoren die gleiche abzüglich Leckverlusten, bei Zylindern die in der anderen Zylinderkammer befindliche) zu den Arbeitselementen der zweiten Achse. Aus diesen kommt dann der Rückfluß zur Pumpe (wiederum gilt: bei Hydromotoren strömt die gleiche Flüssigkeit weiter, bei Zylindern die Flüssigkeit aus der anderen Kammer). Der Druck in der Abstrom-Kammer der Arbeitselemente der ersten Achse muß mit dem Druck in der Zustrom-Kammer der Arbeitselemente der zweiten Achse übereinstimmen. Dieser Druck wirkt natürlich gegen die Ausfahrt des ersten Zylinders (oder für die Einfahrt). Der Zylinder muß also nicht nur gegen die anliegende Last F_L, sondern auch gegen diesen Druck ankämpfen.

Wenn nur eine der seriellen Achsen aktiv ist, fließt sämtliche Flüssigkeit nur in sie hinein bzw. aus ihr heraus.

Bei der **sequentiellen Kopplung** liegen beinahe die gleichen Gegebenheiten vor wie bei der parallelen Kopplung. Durch die verhaltensverändernde Komponente muß aber zusätzlich gelten (wenn Y die veränderte Komponente beinhaltet): entweder gibt es eine zeitliche Verzögerung im Verhalten von Y (gegenüber X), oder

[8] siehe 2.1.2 über die gleichmäßige Verteilung des Drucks

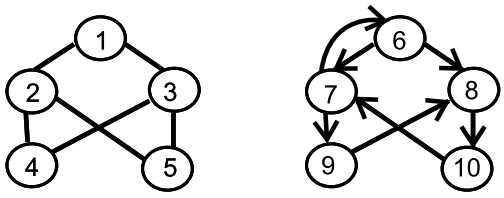

Abbildung 2.5.: Ein ungerichteter und ein gerichteter Graph

der Druck ist geringer nach Durchlauf des Flüssigkeitsstroms durch die Komponente[9].

Zu den skizzierten Auswirkungen müssen in der Praxis noch Leckverluste (an Leitungen und Ventilen) und Druckverluste (vornehmlich an Ventilen) eingerechnet werden. Die Verluste werden heuristisch eingerechnet.

Ein wesentlicher Bestandteil der Lösungsfindung von P ist es, aus bekannten Fakten wie Ausfahrkurven von Zylindern und wirkenden Kräften an Zylindern (sowie weiteren Fakten wie Größe) Schlüsse auf Drücke und Volumenströme zu ziehen. Daraus kann wiederum auf den Kopplungsbaum geschlossen werden.

2.2. Grundlagen der Graphentheorie

Ein ungerichteter Graph $G = (V, E)$ besteht aus einer endlichen Knotenmenge V und einer Kantenmenge $E \in \{\{i, j\} | i, j \in V, i \neq j\}$.

Ein gerichteter Graph $G = (V, E)$ besteht aus einer endlichen Knotenmenge V und einer Kantenmenge $E \in \{(i, j) | i, j \in V, i \neq j\}$.

In Abbildung 2.5 sieht man links den ungerichteten Graph $G_1 = (V_1, E_1)$ mit $V_1 = 1, 2, 3, 4, 5$ und $E_1 = \{\{1, 2\}, \{1, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}\}$. Rechts steht der gerichtete Graph $G_2 = (V_2, E_2)$ mit $V_2 = 6, 7, 8, 9, 10$ und $E_2 = \{(6, 7), (6, 8), (7, 6), (7, 9), (8, 10), (9, 8), (10, 7)\}$. Zu beachten ist dabei die Kante $\{1, 2\}$ bzw. die Kanten $(6, 7)$ und $(7, 6)$: durch die notwendige Richtungsangabe bei gerichteten Graphen können zwei Kanten existieren, die die gleichen Knoten betreffen. Bei ungerichteten Graphen entfällt diese Möglichkeit implizit durch die Definition.

Einige weitere Begrifflichkeiten:

[9] zusätzlich zu den Verlusten, die ohnehin an Ventilen vorkommen

(a) Zwei Knoten $u, v \in V$ sind **adjazent**, wenn sie durch eine Kante verbunden sind.

(b) Eine Kante ist **inzident** mit einem Knoten, wenn der Knoten einer der beiden Endpunkte der Kante ist.

(c) Eine Folge $(v_0, v_1, \ldots, v_m)$ heißt **Weg**, wenn $\forall 0 \leq i \leq m : v_i \in V$ und $\forall 0 \leq i \leq m-1$:

- $(v_i, v_{i+1}) \in E$ bei einem gerichteten Graphen oder
- $\{v_i, v_{i+1}\} \in E$ bei einem ungerichteten Graphen.

m ist die **Weglänge**. (1, 3, 5) und (6, 8, 10) sind Wege von 1 nach 5 bzw. von 6 nach 10. Die Länge ist jeweils 2.

(d) Ein Weg ist **einfach**, wenn kein Knoten zweimal vorkommt. Die Wege (1,3,5) und (6,8,10) sind einfach, der Weg (1, 3, 1, 2) nicht.

(e) Ein **Kreis** ist eine Folge $(v_0, v_1, \ldots, v_m)$, bei der $(v_0, v_1, \ldots, v_{m-1})$ ein einfacher Weg ist und $v_0 = v_m$. Im Beispiel sieht man die Kreise (2, 5, 3, 4, 2) und (7, 9, 8, 10, 7).

(f) Ein gerichteter Graph ohne Kreis heißt **azyklisch**.

(g) Ein ungerichteter Graph ist **zusammenhängend**, wenn es für je zwei Knoten $v, w \in V$ einen Weg von v nach w gibt. Der Beispielgraph ist zusammenhängend.

(h) Ein gerichteter Graph ist **stark zusammenhängend**, wenn es für je zwei Knoten $v, w \in V$ einen Weg von v nach w und einen von w nach v gibt. Der Beispielgraph ist stark zusammenhängend.[10]

(i) Für eine Knotenmenge $V_k \in V$ eines Graphen ist $G(V_k)$ der von V_k **induzierte Graph**; er enthält V_k als Knotenmenge und alle Kanten, bei den beide Endpunkte in V_k liegen. Der induzierte Graph G_3 für die Knotenmenge $\{1, 3\}$ enthält nur die Kante $\{1, 3\}$.

(j) V_k heißt **zusammenhängend** und **Zusammenhangskomponente von G**, wenn $G(V_k)$ **zusammenhängend** ist. G_3 ist eine Zusammenhangskomponente von G_1.

[10] Einfach per Sicht: die Knoten 7,8,9,10 bilden einen Kreis, damit gibt es von jedem Knoten einen Weg zum anderen. Durch die Kanten (7,6) und (6,7) kommt man von 6 in den Kreis oder vom Kreis zur 6. Aber: wenn die Kante (7,6) fehlen würde, wäre der Graph nicht stark zusammenhängend, denn es gäbe keinen Weg von einem Knoten des unteren Kreises zur 6.

	6	7	8	9	10
6	0	1	1	0	0
7	1	0	0	1	0
8	0	0	0	0	1
9	0	0	1	0	0
10	0	1	0	0	0

Tabelle 2.1.: Adjazenzmatrix

(k) Bei ungerichtete Graphen heißen durch eine Kante verbundene Knoten **Nachbarn**. Die Zahl der Nachbarn von v wird als $grad(v)$ bezeichnet. $grad(3) = 3, grad(5) = 2$.

(l) Für eine Kante $(u, v) \in E$ in einem gerichteten Graphen wird u **Vorgänger** von v, v **Nachfolger** von u genannt. Die Zahl der Vorgänger von v wird mit $ingrad(v)$, die Zahl der Nachfolger mit $outgrad(v)$ bezeichnet. $ingrad(7) = 1, outgrad(7) = 1, outgrad(6) = 2$.

Bei der Implementierung von Graphen werden hauptsächlich zwei Möglichkeiten verwendet. Die **Adjazenzmatrix** für einen Graphen mit n Knoten ist eine $n \times n$-Matrix, bei der für jeden Knoten eine Spalte und eine Reihe steht. In der Matrix ist

- $A(i, j) = 1$, wenn $(i, j) \in E$ (bzw. $\{i, j\} \in E$ bei ungerichteten Graphen)
- $A(i, j) = 0$, sonst

Bei gerichteten Graphen muß die Matrix symmetrisch zur Mitteldiagonale (über $(i, i) \forall i$) sein.

Die andere Möglichkeit ist eine Adjazenzliste: für jeden Knoten ist eine Liste angelegt, in der jeder adjazente Knoten steht. Diese Lösung wird besonders bei Graphen mit wenigen Kanten bevorzugt.

Für den gerichteten Graphen zeigt Tabelle 2.1 die Adjazenzmatrix und Tabelle 2.2 die Adjazenzliste.

Ein **Baum** ist ein besonderer Graph. Ein Baum wird am einfachsten durch die rekursive Vorschrift definiert:

1. Ein einzelner Knoten ist ein Baum.

6	$\rightarrow$	7	$\rightarrow$	8
7	$\rightarrow$	6	$\rightarrow$	9
8	$\rightarrow$	10		
9	$\rightarrow$	8		
10	$\rightarrow$	7		

Tabelle 2.2.: Adjazenzliste

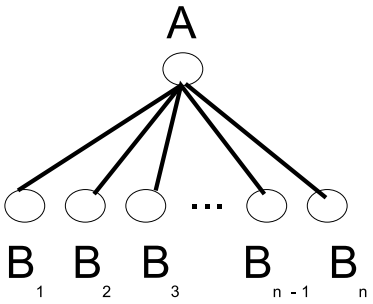

Abbildung 2.6.: Einfache Baumstruktur

2. Wenn A ein einzelner Knoten ist, $B_1, ..., B_m$ Bäume sind, und wenn zwischen A und $B_i \forall i$ eine Verbindung besteht, zwischen zwei Bäumen B_i und $B_j, i \neq j$ aber nicht, dann ist dieses Konstrukt ein Baum. Abbildung 2.6 verdeutlicht dieses.

Man nennt die Unterstrukturen B_i auch Teilbäume. A ist die Wurzel des Baums aus A und den B_i. Teilbäume, die aus nur einem Knoten bestehen, werden Blätter genannt. Alle Nicht-Blätter werden **innere Knoten** genannt.

Einfach gesehen ist ein ungerichteter Graph ein Baum, wenn er zusammenhängend und kreislos ist.

Bei einem gerichteten Baum muß bei 2. die Verbindung immer von A zu B_i hingehen. Ein Graph ist ein gerichteter Baum, wenn er zyklenfrei und zusammenhängend ist, sowie jeder Knoten ingrad 1 und outgrad 1 besitzt. Die Wurzel allerdings ist von ingrad 0, die Blätter von outgrad 0.

2.3. Grundlagen der Constraintverarbeitung

2.3.1. Grundbegriffe

Constraints[11] sind Beschränkungen von Kombinationsmöglichkeiten eines kombinatorischen Problems. Sie sind eine Art der Wissensrepräsentation. Formal ist ein Constraint eine Relation, die die Beziehung zwischen Variablen beschreibt. Formal besteht ein Constraint aus einer Variablen-Menge und einem Attribut über diesen Variablen.

Ein **Constraint-Netz** wiederum besteht aus einer Menge von Constraints. Dabei ist im Normalfall die Schnittmenge der Variablen der Constraints nicht leer. Ein Spezialfall ist der **binäre**[12] Constraint, bei dem höchstens zwei Variablen zugelassen sind. Wenn das ganze Netz aus binären Constraints besteht, kann es leicht auf Graphen abgebildet werden: Variablen werden zu Knoten, Relationen zu Kanten. Bei nur einer Variable spricht man im übrigen von einem **unären**[13] Constraint. Allgemein spricht man von einem n**-Constraint**, wenn n die Anzahl der an dem Constraint anliegenden Knoten ist.

Im allgemeinen wird versucht, eine Lösung für die Variablen zu finden, die durch das Constraint-Netz eingeschränkt werden. [14] Die Suche nach diesen widerspruchsfreien Belegungen nennt man **Constrainterfüllung**.

Constraint-Erfüllungs-Probleme (Constraint Satisfaction Problems / CSP) beschäftigen sich mit der Frage, ob ein Constraint-Netz eine Lösung hat. Ein bekanntes Problem ist das Karten-Färbungs-Problem.[15]

Bei **Constraint-Lösungs-Problemen** (Constraint Solutions Problems) hingegen wird nach einer Lösung gesucht.

Ein **Constraint-Optimierungs-Problem** (Constraint Optimization Problem / COP) ist ein Constraint-Lösungs-Problem mit einer Optimierungsfunktion, die jedes Lösungs-Tupel auf einen numerischen (meist reellen) Wert abbildet. Gesucht wird eine Lösung mit optimalem Wert (je nach Problem Maximum oder Minimum).

[11] direkte Übersetzung: Zwang, Einschränkung
[12] engl.: binary
[13] engl.: unary
[14] Der Wertebereich einer Variable kann durch unäre Constraints ausgedrückt werden.
[15] Beim Karten-Färbungs-Problem (Map Coloring Problem) sollen die Länder einer Karte eingefärbt werden. Nachbarländer müssen mit unterschiedlichen Farben eingefärbt werden. Eine Anzahl von Farben ist vorgeben.

Sofort fällt auf, das das maßgebliche Problem P ein COP ist.

2.3.2. Konsistenz

Seien $V_1, \ldots, V_n$ Wertemengen von Variablen. Ein Tupel $(V_1, \ldots, V_n)$ heißt eine **lokal-konsistente** Lösung eines Constraint-Netzes mit den Constraints $(C_1, \ldots, C_m)$ genau dann, wenn

$\forall i \in (1, \ldots, n) \forall d_i \in V_i \forall C \in (C_1, \ldots, C_m)$
$\exists d_1 \in V_1 \ldots \exists d_{i-1} \in V_{i-1} \exists d_{i+1} \in V_{i+1} \ldots \exists d_n \in V_n :$
C wird durch $(d_1, \ldots, d_n)$ erfüllt

Ein Tupel $(V_1, \ldots, V_n)$ heißt eine **global-konsistente** Lösung eines Constraint-Netzes mit den Constraints $(C_1, \ldots, C_m)$ genau dann, wenn

$\forall i \in (1, \ldots, n) \forall d_i \in V_i$
$\exists d_1 \in V_1 \ldots \exists d_{i-1} \in V_{i-1} \exists d_{i+1} \in V_{i+1} \ldots \exists d_n \in V_n \forall C \in (C_1, \ldots, C_n) :$
C wird durch $(d_1, \ldots, d_n)$ erfüllt,
das heißt: $(d_1, \ldots, d_n)$ erfüllt alle Constraints gleichzeitig

Lokale und globale Konsistenz stimmen dann überein, wenn es keine Zyklen im Netz gibt oder alle Werte eindeutig bestimmt sind. Gesucht wird natürlich eine global konsistente Lösung.

K-konsistent ist ein Netz, wenn gilt: sind k-1 Variablen so belegt, daß alle Constraints erfüllt sind, dann kann man auf jeden Fall einen erfüllenden Wert für eine k-te Variable finden. Ein Netz ist streng k-konsistent, wenn es j-konsistent ist $\forall j \leq k$.

Knoteninkonsistenz[16] liegt für einen Knoten vor, wenn ein unärer Constraint die Lösungsmenge weiter einschränken kann.

Kanteninkonsistenz[17] liegt für eine Kante vor, wenn für eine Belegung des einen Knoten keine Belegung im zweiten Knoten vorliegt, die den Constraint (der durch die Kante beschrieben wird) erfüllt.

Pfadinkonsistenz[18] liegt vor, wenn es ein Werte-Paar für zwei Knoten gibt, das den Constraint zwischen den beiden Knoten zwar erfüllt, aber wegen indirekter Verbindung (also Constraints-Ketten, die vom einen Knoten zum anderen führen) unmöglich ist. Ein Beispiel ist das 2-Färbungs-Problem mit 3 Knoten, wie in der Abbildung 2.7 zu sehen. Wenn die beiden Farben rot und grün sind, so liegt weder Knoten- noch Kanteninkonsistenz vor, wenn bei allen Knoten die Werte

[16] engl.: node inconsistency
[17] engl.: arc inconsistency
[18] engl.: path inconsistency

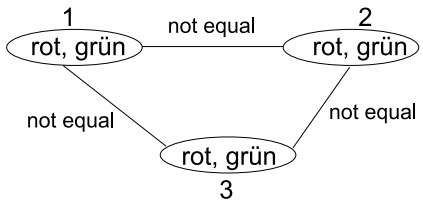

Abbildung 2.7.: 2-Färbe-Problem mit 3 Knoten

rot und grün erlaubt sind. Der Constraint zwischen je zwei Knoten läßt bei rot in dem einen Knoten nur grün im anderen Knoten und umgekehrt zu. Es liegt aber Pfadinkonsistenz vor: wenn der erste Knoten grün ist, muß der zweite rot und wieder der dritte grün sein, wodurch der erste nicht grün sein kann. Damit muß grün im ersten Knoten gestrichen werden. Bei rot im ersten Knoten ergibt sich jedoch ein ähnliches Ergebnis, so daß das Problem unlösbar ist.

Mit der einfachen Kantenkonsistenz kann man schon viele Einschränkungen der Knotenwerte vornehmen. Jedoch fehlt die Möglichkeit, in einem Kreis beide Richtungen zu betrachten, dann hilft der Pfadkonsistenz-Test.

2.3.3. Backtracking und Branch-and-Bound

Backtracking und Branch-and-Bound sind zwei weit verbreitete Techniken, die nicht nur im Bereich der Constraint-Verarbeitung benutzt werden.

Die Backtracking- oder Trial-and-Error-Methode bezeichnet ein Lösungsverfahren, bei dem eine Teillösung systematisch zur Gesamtlösung ausgebaut wird. Falls eine Teillösung L'' nicht mehr weiter ausgebaut werden kann, also auch nicht mehr zu einer Gesamtlösung führen kann, werden ein oder mehrere Schritte rückgängig gemacht zur gröberen Lösung L' (mit $L'' \subset L'$). Dann versucht man die Teillösung auf anderem Weg auszubauen. Dieses Verfahren wird solange wiederholt, bis eine Gesamtlösung gefunden ist, oder feststeht, daß keine Lösung existiert.

Die Rechenzeit wächst bei Backtracking-Verfahren exponentiell zur Suchtiefe. Das Branch-and-Bound-Verfahren[19] ist für Optimierungsprobleme gedacht. Ein Problem wird schrittweise in Teilprobleme (branch) zerlegt. Diese Teilprobleme kann man sich als Baum vorstellen (siehe S.32). Jedem Knoten innerhalb des Baums wird ein Wert zugeordnet, der eine Schranke für den Wert der Zielfunktion des Optimierungsproblems darstellt. Es werden zunächst diejenigen

[19] engl. für „Verzweigen und Beschneiden“

Zweige bearbeitet, die den besten Zielwert erwarten lassen. Wenn schon eine Lösung gefunden und damit ihr Wert bekannt ist, werden Zweige mit schlechteren Schrankenwerten gar nicht mehr in Betracht gezogen.

2.3.4. Lösungsmethoden

Zur Lösung des Constraint-Problems geht man i.a. schrittweise vor. Bei sequentiellen Lösungsverfahren versucht man, lokal konsistente Lösungen zu finden und auf global konsistente zu erweitern. Das Wort **Constraint-Propagation** bezeichnet die Weiterleitung der Einschränkungen, die ein Constraint bewirkt, an andere Constraints. Mit Hilfe der Constraint-Propagation kann man die Werte der Variablen immer weiter einschränken. Diese Vorgehensweise ist eine Grundtechnik der Constraint- Verarbeitung. Ein Knoten wird dabei in der Regel häufiger besucht. Bei der **Rückwärts-Propagierung** [20] werden bei Unverträglichkeiten in späteren Schritten diese zurückgespielt (zurückpropagiert). Dadurch werden die noch möglichen Interpretationen vorhergehender Knoten eingeschränkt.

Lösungen mit Backtracking sehen etwas anders aus. Bei diesen beginnt man mit Constraint-Propagation, bis sich keine Änderungen mehr einstellen. In vielen Fällen wird dadurch noch keine Lösung gefunden. Deswegen wird ein Knoten genommen. Für diesen werden nacheinander sämtliche mögliche Werte ausprobiert und jeweils als Anstoß für die Propagierung der dem Knoten anliegenden Constraints genommen. Die Laufzeit einer solchen Vorgehensweise hängt stark von der Reihenfolge ab. Deswegen gibt es über die zufällige Reihenfolge hinaus weitergehende Techniken der Auswahl.

Vorausschauende Techniken (look-ahead schemes) wirken auf die Variablen-Auswahl. Dabei kann die Variable genommen werden, die den Rest des Suchraums am meisten einschränkt (variable ordering). Man versucht dieses meistens durch Auswahl der Variable zu erreichen, die in der größten Zahl von Constraints zu finden ist. Sinnvoll ist auch die Auswahl des Variablen-Wertes, der die Anzahl an Optionen in zukünftigen Zuweisungen maximiert (value ordering).

Zurückschauende Techniken (look-back schemes) beschäftigen sich mit Backtracking bei Sackgassen. Einerseits kann man versuchen, die Entscheidungen zu entdecken und zu ändern, die zu der Sackgasse führte (Go back to source of failure). Andererseits kann man auch die Gründe für Sackgassen aufzeichnen, damit sie nicht nochmal passieren (constraint recording / learning).

Bei sogenannten lokalen Suchverfahren wird versucht, angefangen mit einer vollständigen Belegung durch eine Folge von kleinen Veränderungen in der Belegung

[20] engl.: Back-Propagation

das Problem zu lösen. Somit bleibt die Lösung im lokalen Bereich der Anfangsbelegung. Die Anfangsbelegung kann heuristisch oder auch zufällig erfolgen. Diese Vorgehensweise ist insofern problematisch, als bei den meisten dieser Ansätze die Lösungen um lokale Minima des Suchraums pendeln. Deswegen werden lokale Suchverfahren i.a. mehrmals mit unterschiedlichen Anfangsbelegungen gestartet. Zusätzliche Meta-Heuristiken wie Tabu-Search können die Suche verbessern. Bei letzterer werden einzelne Lösungen, wenn sie erreicht werden, für die nächsten n Belegungen gesperrt. Somit kann ein Verharren auf einem lokalen Minimum oder in dessen Umgebung vermieden werden.

Wie man sieht, liegt das eigentliche Problem gar nicht bei der Constraint-Propagierung selbst, sondern darin, daß die Propagierung im Normalfall noch nicht zur Lösung führt. Nötig wird die Selektion der richtigen Variable und das Durchspielen der möglichen Lösungswerte. Die kostspieligen Suchvorgängen, vor allem der exponentiell wachsende Suchbaum führen zu langwierigen Lösungsroutinen.[21]

2.4. Scheduling

Scheduling-Probleme sind Probleme der Resourcen-Allokation über die Zeit, um bestimmte Menge von Aufgaben zu erfüllen. Es handelt sich um ein Entscheidungsproblem mit ein oder mehreren Zielen. Damit gehören Scheduling-Probleme zur großen Gruppe der kombinatorischen Optimierungsprobleme. Beispiele sind die Arbeitsaufteilung auf Maschinen in einer Werkstatt und die Aufgabenverteilung auf Prozessoreinheiten in einem Netzwerk. Häufig geht es wie in vielen Fabriken um eine Aufgabe (Auto zusammenbauen), die auf Unteraufgaben aufgeteilt werden kann (Tür einsetzen, Fenster einsetzen, Hupe einbauen). Die Teilaufgaben können auch von parallelen Maschinen erledigt werden (beispielsweise 2 Maschinen für die Türen und 3 für die Fenster). Aufgabe ist es nun, die Arbeitsschritte nach gewissen Kriterien auf die Maschinen aufzuteilen (große Auslastung der Maschinen, gleichmäßige Auslastung gleichartiger Maschinen, maximal viele Autos an einem Tag, ...).

Scheduling Probleme kann man nach bestimmten Kriterien unterteilen. Eine Aufteilung in drei Felder $\alpha|\beta|\gamma$ aus [12] erweist sich als sehr tauglich.

[21] Dabei führt die Constraint-Propagierung zu einer durchaus schnellen Lösung. Ansonsten müßte man mit Hilfe von reinem Backtracking suchen: in jedem Schritt wird eine Variable ausgewählt. Sämtliche Werte dieser Variable werden durchgespielt. Wenn die erste Variable neun Werte hat, nach Constraint-Propagierung schon 6 davon ausgeschlossen werden können, muß nach Constraint-Propagierung nur noch ein Drittel des reinen Backtracking-Baums betrachtet werden!

- α enthält die Maschinenzahl und -zusammenhänge: einfache Fälle sind einzelne Maschinen oder parallel laufende identische Maschinen (u. U. mit verschiedenen Geschwindigkeiten). Die wichtigsten Scheduling-Probleme betreffen jedoch

  - Flow Shops: m Maschinen sind in Serie geschaltet, alle Arbeitsaufträge müssen nacheinander alle m Maschinen in derselben Reihenfolge durchlaufen.
  - flexible Flow Shops: anstelle von m Maschinen gibt es m Stufen, in denen parallel arbeitende Maschinen zur Verfügung stehen. Die Anzahl der parallelen Maschinen variiert in jeder Stufe.
  - Open Shops: jeder Job muß auf jeder der m Maschinen ausgeführt werden, allerdings ist 1. die Reihenfolge nicht vorgegeben und kann 2. für einen Job bei einigen Maschinen die Durchlaufzeit 0 betragen.
  - Job shop: wieder gibt es m Maschinen. Jeder Job hat jedoch eine vorgegebene Route.

- β enthält Einschränkungen und Constraints. Hier sind mehrere Eintragungen möglich. Wichtige Möglichkeiten sind

  - Vorrang-Constraints: ein oder mehrere Jobs müssen vor einem oder mehreren anderen Jobs erledigt werden. Diese Constraints können sehr gut schriftlich oder per Graph angegeben werden.
  - Verdrängung: ein Job kann einen anderen von einer Maschine verdrängen. Die Zeit, die der andere dort verbracht hat, ist allerdings nicht verloren, er muß nur noch die Restzeit später dort ablaufen.
  - Startzeiten: für einige Jobs sind Startzeiten vorgegeben. Die Jobs können erst dann anlaufen.
  - Kreislauf: In Job Shops müssen einige Jobs mehrmals zu einer Maschine.
  - mit Deadline: einzelne Jobs müssen zu einem bestimmten Zeitpunkt fertig sein.

- γ enthält die Zielfunktionen. Häufig ist Minimierung gewünscht: Kriterien sind Gesamtzeit[22], Fertigstellungszeit (die Summe der Laufzeiten der einzelnen Jobs), gewichtete Fertigstellungszeit (die einzelnen Laufzeiten werden vor der Summierung gewichtet) oder Summe der Säumigkeit (die Zeit, die Jobs über die Deadline hinaus brauchen).

[22] engl.: minimum makespan

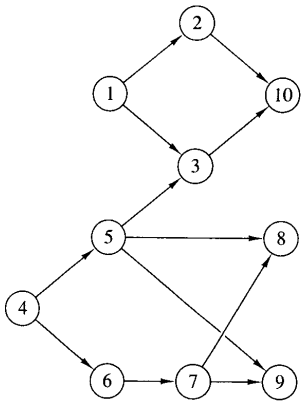

Abbildung 2.8.: Vorrang-Constraints des Problems S

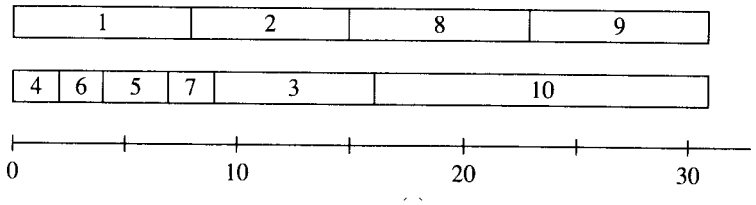

Abbildung 2.9.: Eine optimale Lösung des Problems S

Mit Hilfe dieser Beschreibungsmöglichkeit kann man Scheduling-Probleme einfach einordnen. Wie man in Kapitel 4.2 sieht, gibt es jedoch Probleme, die nur mit Erweiterung in dieses Schema passen.

Ein einfaches Scheduling-Problem S aus [12, 16ff.] soll zur Verdeutlichung dienen. Es handelt sich um ein $P2|prec|C_{max}$ mit 10 Jobs. Die Laufzeiten der Jobs folgen nun.

Job	1	2	3	4	5	6	7	8	9	10
p_j	8	7	7	2	3	2	2	8	8	15

Abbildung 2.8 zeigt die Vorrang-Constraints für S.

Ein einfacher Algorithmus zum Scheduling wird nun definiert.

- Sei J die Menge der Jobs.

- Sei J'_i die Menge der schon verarbeiteten Jobs zum Zeitpunkt i.
- Sei K_i die Menge der Jobs k_{ij}, für die zum Zeitpunkt i alle Jobs schon verarbeitet wurden, die Vorgänger von K_i sind.
- Sei $\#M$ die Anzahl der Maschinen.

Begonnen wird der Algorithmus mit i = 0.

- berechne K_i mit Hilfe von J'_i
- $\forall$ Maschine: Wenn die Maschine nicht durch einen vorherigen Auftrag gespeert ist,
  - weise der Maschine einen Job $x_j \in K_i$ zu, nehme dabei den kürzesten Job zuerst
  - sperre jede Maschine für die Zeitdauer z_j des Jobs x_j
  - füge x_j in J'_k ein $\forall k \geq i + z_j$

Grob gesehen wird also immer der kleinste schon zu verarbeitende Job einer arbeitslosen Maschine zugewiesen.

Eine durch den Algorithmus gefundene optimale Lösung ist in Abbildung 2.9 zu sehen. Die Gesamtlaufzeit ist klar optimal: es gibt keine Zeit, in der eine der beiden Maschinen ohne Job ist, und somit auch keine Zeit, die weiter eingespart werden könnte.

Ein überraschendes Ergebnis erzielt der Algorithmus, wenn die Anzahl der Maschinen auf 3 erhöht wird. Das so erhaltene Problem S' wird in der Gesamtzeit 36 gelöst, im Vergleich zu 31 bei S. Dies liegt an der Tatsache, daß immer der kürzeste Job genommen wird, und daß keine Leerlaufzeiten von Maschinen akzeptiert werden. Mit einem anderen Algorithmus können hier mindestens 5 Zeiteinheiten gespart werden.

Das Beispiel zeigt die Wichtigkeit, einen guten Algorithmus zu finden.

Nun werden einige Lösungsansätze für das Scheduling genannt. Auf die einfachsten Ansätze, die keine Rücksicht auf Constraints nehmen, wird verzichtet.

Der schon gezeigte Ansatz kann SPT (shortest processing time) genannt werden: der kürzeste Job wird zuerst verarbeitet. Hierbei dürfen allerdings nur nicht durch einen Vorgänger blockierte Jobs genommen werden. Sinnvoller wäre LPT (longest processing time): kleinere Jobs können später hinten angehängt werden, längere

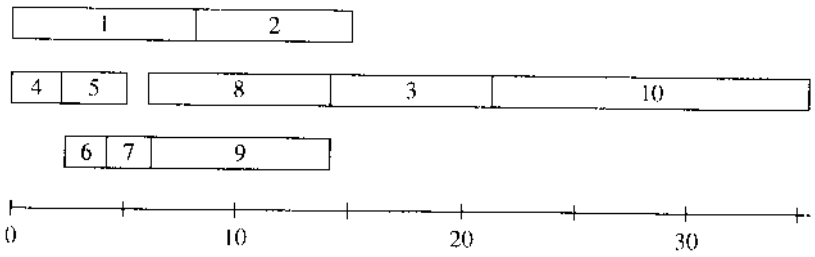

Abbildung 2.10.: Eine nicht optimale Lösung des Problems S'

Jobs hingegen, die zum Schluß übrig sind, können zu sehr langer Gesamtlaufzeit führen.

Die kritische Pfad-Methode CPM (critical path method) besteht darin, den längsten Pfad zu berechnen. Man kann für jeden Job j die früheste Startzeit C'_j errechnen. Beim vorliegenden Beispiel S sind die frühesten Startzeiten

Job	1	2	3	4	5	6	7	8	9	10
C'_j	0	8	8	0	2	2	4	6	5	15

Damit ist auch die minimale Gesamtlänge von 30 bekannt

Nun kann man auch berechnen, welche späteste Startzeit ein Job hat.

Job	1	2	3	4	5	6	7	8	9	10
C'_j	0	8	8	16	19	18	20	22	22	15

Die kritischen Jobs sind diejenigen, bei denen die früheste und die späteste Startzeit übereinstimmen. Es handelt sich um 1, 3, 10. Damit ist auch der kritische Pfad ermittelt: 1, 3, 10.

Ein einfacher Algorithmus zur Lösung bietet sich also an: nehme bei der Auswahl zwischen mehreren Jobs zu einer bestimmten Zeit i immer denjenigen, bei dem die Differenz zwischen spätester Startzeit und i am geringsten ist. Erweitert werden kann der Algorithmus, indem nicht nur i, sondern auch die nachfolgenden Zeitpunkte schon betrachtet werden. Wenn am Zeitpunkt $k > i$ ein Job frei ist, für den die Differenz zwischen spätester Startzeit und i geringer ist, so wird auf diesen Job gewartet, dieser wird vor den schon zu Verfügung stehenden Jobs ausgeführt. Eine Ausnahme ist natürlich sinnvoll: sollte ein Job zwischen i und k ausgeführt werden können, so wird dieser auch vorher ausgeführt.

In [12, Kap. 4] sind weitere Lösungsansätze zu finden, die aber im Rahmen dieser Arbeit nicht berücksichtigt werden können.

2.5. Multi-Kriterien-Analyse

Der Ursprung der Multi-Kriterien-Analyse[23] liegt in der Betriebswirtschaftslehre, genauer gesagt im Operations Research.

Ziel der Multi-Kriterien-Analyse ist, die nach mehreren Gesichtspunkten (Zielen, Kriterien) beste Lösung für ein Problem zu finden. Im allgemeinen ist die Lösung nicht die objektiv beste, weil es keine feste, objektive Vergleichbarkeit der einzelnen Kriterien gibt. Die verschiedenen Kriterien werden in verschiedenen Maßstäben gemessen. Häufig widersprechen sich die Ziele: eine Verbesserung der Lösung hinsichtlich eines Ziels bewirkt die Verschlechterung hinsichtlich eines anderen.

Gesucht wird die Lösung, die der Benutzer unter Einbeziehung aller Ziele bevorzugt. In [25, S.25] werden zwei Arten von Multi-Kriterien-Problemen unterschieden. Multi-Attribut-Entscheidungen (MADM[24]) lösen das Problem durch Auswahl einer Handlungsalternative. Diese Möglichkeit ist nur bei endlich vielen[25] und explizit bekannten Alternativen brauchbar. Die zweite für diese Diplomarbeit relevante Gruppe bilden die Multi-Objektive-Entscheidungen (MODM[26]). Hierbei ist die Menge der Lösungen unendlich und nicht vorher bekannt: alle Lösungen, die gewisse Nebenbedingungen erfüllen, werden in Betracht gezogen. Die Ziele sind durch Zielfunktionen[27] beschrieben. Bei MADM gibt es also einen diskreten, bei MODM einen stetigen Lösungsraum. Für diese Arbeit sind nur MODM interessant.

Notwendig für einen Entscheidungsprozeß ist die Vergleichbarkeit der Lösungsalternativen. Es wird eine Präferenzordnung benötigt, eine zweistellige Ordnungsrelation, mit der die Alternativen in eine Reihenfolge gebracht werden können. Diese Ordnung sollte mit den Vorstellungen des Benutzers übereinstimmen.

In der Nutzentheorie wird diese Präferenzordnung durch eine reelle Funktion beschreiben, der **Nutzenfunktion**. Bei zwei Alternativen sollte diejenige mit dem höheren Wert auch die bessere sein, die Alternative mit dem insgesamt höchsten Wert sollte die beste sein. Die Zielfunktionen bei MODM können durch solche Nutzenfunktionen beschrieben werden. Gerne werden auch Kostenfunktionen benutzt. Diese sind nichts anderes als invertierte Nutzenfunktionen: je höher der

[23] engl.: Multi Criteria Analysis; andere relevante Begriffe sind Multi Criteria Decision-Making (Entscheidungtreffen) und Multi Criteria Optimization (Optimierung).
[24] MADM = Multi Attribut Decision Making
[25] Aufgrund der Rechengeschwindigkeit sollten die Auswahlmengen nicht nur endlich, sondern auch möglichst klein sein.
[26] MODM = Multi Objective Decision Making
[27] englisch: objectives

Wert, desto schlechter ist die Alternative. Kostenfunktionen bieten sich gerade dann an, wenn tatsächlich Kosten eingerechnet werden, z. B. Betriebs- oder Anschaffungskosten.[28]

Philipp Vincke teilt in [23] die Lösungsansätze von MODMs in 3 Familien auf. Dabei erklärt er, daß die Grenzen zwischen diesen unscharf sind:

1. interaktive Methoden (interactive methods)
2. Outranking-Methoden[29] (outranking methods)
3. Multi-Attribut-Nutzen-Theorie (multiple attribute utility theory)

Interaktive Methoden setzen auf die Mitwirkung des Benutzers, der fortlaufend kleine Entscheidungen in Richtung der Lösung treffen muß.

Bei den **Outranking-Methoden** werden die Alternativen direkt miteinander verglichen. Eine Alternative A ist besser als die Alternative B, wenn mehr Ziele für A sprechen. A ist die Lösung oder eine der Lösungen, wenn es keine bessere Alternative gibt.

Bei der **Multi-Attribut-Nutzen-Theorie** kann man wiederum drei Ansätze unterscheiden: die Präferenzmethode sowie den vektorellen und den skalaren Ansatz..

Die **Präferenzmethode** sieht eine genaue Wertung der Kriterien untereinander vor. Bei der Lösungsfindung werden zunächst sämtliche Alternativen gesucht, die für das wichtigste Ziel die besten sind. Nur dann, wenn mehrere gleich gute Lösungen vorhanden sind, wird die Lösungsmenge sukzessiv eingeschränkt, indem das jeweils nächstwichtige Kriterium betrachtet wird.

Bei den meisten MODM-Problemen allerdings sollen alle k reellwertigen Zielfunktionen gleichzeitig maximiert werden:

$$max\{z(x) = \begin{pmatrix} z_1(x) \\ z_2(x) \\ ... \\ z_k(x) \end{pmatrix}\}$$

[28] Der Vorteil liegt allerdings nur im besseren, intuitiven Verständnis für den Menschen: jede Kostenfunktion wird zu einer Nutzenfunktion, wenn man jeden Wert mit -1 multipliziert.

[29] Im deutschsprachigen Raum wird auch der Begriff Prävalenzsatz verwendet, jedoch überwiegt auch in deutschen Büchern der Begriff Outranking.

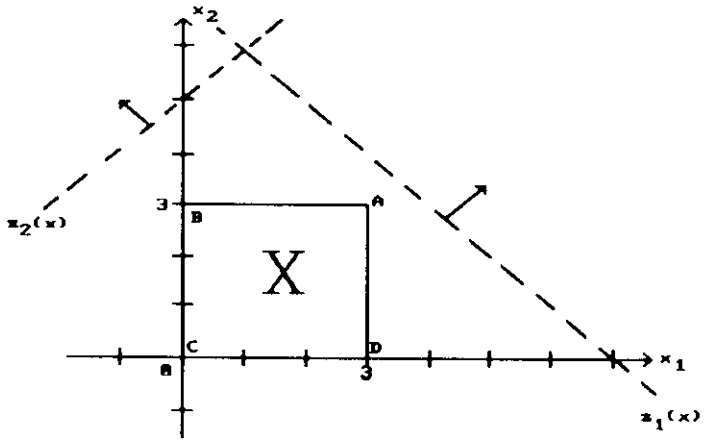

Abbildung 2.11.: Darstellung des Vektormaximumproblems aus [25, S.100]

Wie man sieht, bilden die k Zielfunktionen einen k-dimensionalen Vektor. Man spricht hierbei von einem Vektormaximumproblem. Es greift der **vektorelle Ansatz**. Die ideale Lösung ist gegenüber allen Zielfunktionen gleichzeitig maximal. Sie wird auch utopische Lösung genannt, weil sie aufgrund von Nebenbedingungen und gegenseitiger Beeinflußung der Ziele nur selten zulässig ist. Stattdessen muß man also eine Lösung finden, für die gilt: es gibt keine Lösung, die bezüglich mindestens eines Zieles besser ist, und bei allen anderen Zielen mindestens gleich gut.

Ein Beispiel gibt [25, S.100 f.] mit sehr einfachen Zielfunktionen. Zwei Werte x_1, x_2 liegen jeweils im Bereich von 0 und 3. Die beiden Zielfunktionen sind

$$z_1(x) = x_1 + x_2$$
$$z_2(x) = -x_1 + x_2$$

In Bild 2.11 ist der zulässige Bereich durch das Quadrat gegeben. Die beiden unterbrochenen Striche zeigen die Lage der jeweiligen Zielfunktion, und in welcher Richtung die Zielfunktion wächst. Leicht sehen oder berechnen kann man, daß der Punkt (3,3) eine individuelle Optimallösung bzgl. Funktion z_1 ist. Das bedeutet: die Funktion erreicht in diesem Punkt ihr Maximum im Zulässigkeitsbereich. Genauso ist (0,3) eine individuelle Optimallösung für z_2. Alle Punkte auf der Linie zwischen diesen Punkten (beispielsweise (1,3)) sind **funktional-effiziente**

Lösungen. Diese Lösungen zeichnen sich dadurch aus, daß eine Verbesserung bzgl. eines Ziels immer eine Verschlechterung bzgl. eines anderen bewirken. Es gibt also keine besseren Lösungen. Im Beispiel gilt: eine Verbesserung von z_1 bewirkt immer eine Verschlechterung bzgl. z_2 und umgekehrt. Die ideale Lösung liegt bei $(1\frac{1}{2}, 3\frac{1}{2})$, und damit außerhalb des Bereichs.

Der Nachteil des vektorellen Ansatzes liegt auf der Hand: es gibt meistens viele effiziente Lösungen, unter denen wiederum eine Auswahl getroffen werden muß.

Der **skalare Ansatz** ist eigentlich eine Vereinfachung des vektorellen: Das Vektormaximumproblem wird durch ein Kompromißmodell ersetzt. Die sogenannte Kompromißzielfunktion bildet die k-dimensionalen Zielfunktionsvektoren durch eine eindimensionale Funktion in den reellen Raum ab. Die Kompromißzielfunktion ist meist additiver oder multiplikativer Natur. Bei ersterem werden also die Funktionen einfach nur addiert. Zwei Bedingungen gelten: das Modell muß mit Hilfe der Funktion numerisch lösbar sein, und unter den Optimallösungen muß sich in jedem Fall eine Lösung befinden, die bezüglich des Vektormaximumproblems funktional-effizient ist.

Kapitel 3

Anforderungen und Zielwerte

Anmerkung

In den nächsten Kapiteln werden meistens nur Zylinder betrachtet, wenn es um hydraulische Zusammenhänge geht. Zum Ziel gehört auch für das Design von hydraulischen Systemen mit Hydromotoren. Zur Vereinfachung wird jedoch auf ein genaues Eingehen auf Hydromotoren verzichtet. Diese sind einfacher zu handhaben als Zylinder: es gibt keine Weg-Zeit-Diagramme, sondern nur Geschwindigkeits-Zeit-Diagramme und Beschleunigungs-Zeit-Diagramme.

3.1. Einführung

Die Vorgehensweise beim Design hydraulischer Anlagen kann vergröbert so aussehen: der Experte nimmt die Anforderungen des Kunden entgegen. Dazu gehört vor allen Dingen der Bewegungsablauf der Arbeitselemente zusammen mit dem Kraft- oder Momentbedarf. Eine mit Worten beschriebene Bewegungsfolge ist meist schwer verständlich und unübersichtlich. Das gilt um so mehr für die Bewegungsabläufe mehrerer Verbraucher, bei denen sich die Bewegungen überschneiden. Deswegen präzisiert der Experte — immer in Absprache mit dem Auftraggeber — die Aufgabenstellung und entwickelt daraufhin die Sollprofile. Die Abbildung 3.1 zeigt ein einzelnes Weg-Zeit-Diagramm, auf dem die einzelnen

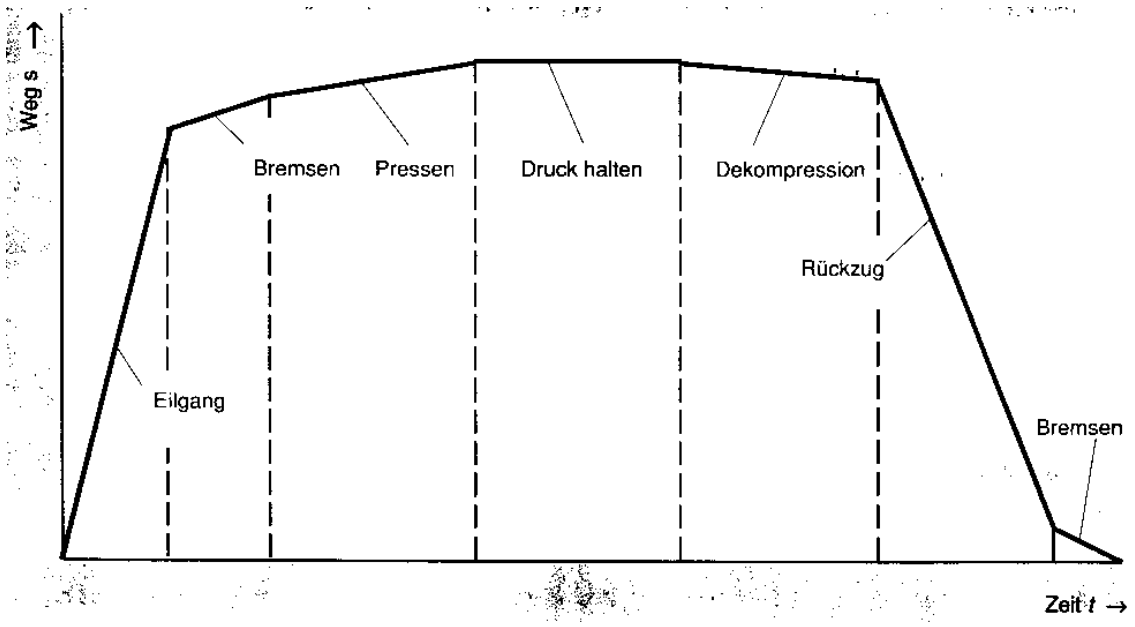

Abbildung 3.1.: Weg-Zeit-Diagramm eines Zylinders aus [5, S.18]

erfüllten Aufgaben verzeichnet sind. Die Abbildung 3.2 zeigt ein Zustandsdiagramm einer Anlage mit Bohrspindel und Hydrozylinder. Auch die Schaltstellungen der Wegeventile, die der Experte schon für den Schaltplan vorsieht, sind eingezeichnet. Aus den Zustandsdiagrammen entwickelt der Fachmann auf Grund seiner Erfahrungen den Schaltplan zur Anlage. Meistens durchläuft die Lösung dabei mehrere Phasen des Grobentwurfs und immer wieder verbesserter Schaltpläne. Manchmal fallen Fehler erst beim Bau der Anlage auf, so daß das Design wiederholt werden muß. Die Kosten- und Zeitintensivität ist also groß.

Es soll nun eine den Experten entlastende Vorgehensweise gefunden werden. Diese wird zunächst in zwei Teile aufgeteilt: der erste Teil führt zu den Sollprofilen, der zweite erstellt daraus den Schaltplan. Letzterer Teil wird in der Arbeit [7] von Hoffmann beschrieben.

Ein grundlegender Ansatz ist die Aufteilung des Bewegungsablaufs des Arbeitselementes einer Achse in sogenannte Phasen. Wie schon in Bild 3.1 angedeutet, kann man den Bewegungsablauf leicht aufsplitten. Zu Anfang fährt der Zylinder im Eilgang, also möglichst schnell, auf eine gewisse Position aus. Danach folgt eine Bremsen-Phase, bevor das Pressen beginnt. Man kann davon ausgehen, daß ab dort eine Gegenkraft am Zylinder wirkt. Deswegen steigt der Druck, die Geschwindigkeit sinkt. Um einen Gegenstand richtig zu pressen, muß eine kurze Zeit der Druck aufrecht erhalten werden. Danach folgen Dekompression, Rückzug (wieder als Eilgang) und nochmals Bremsen. Diese einzelnen Teilstücke

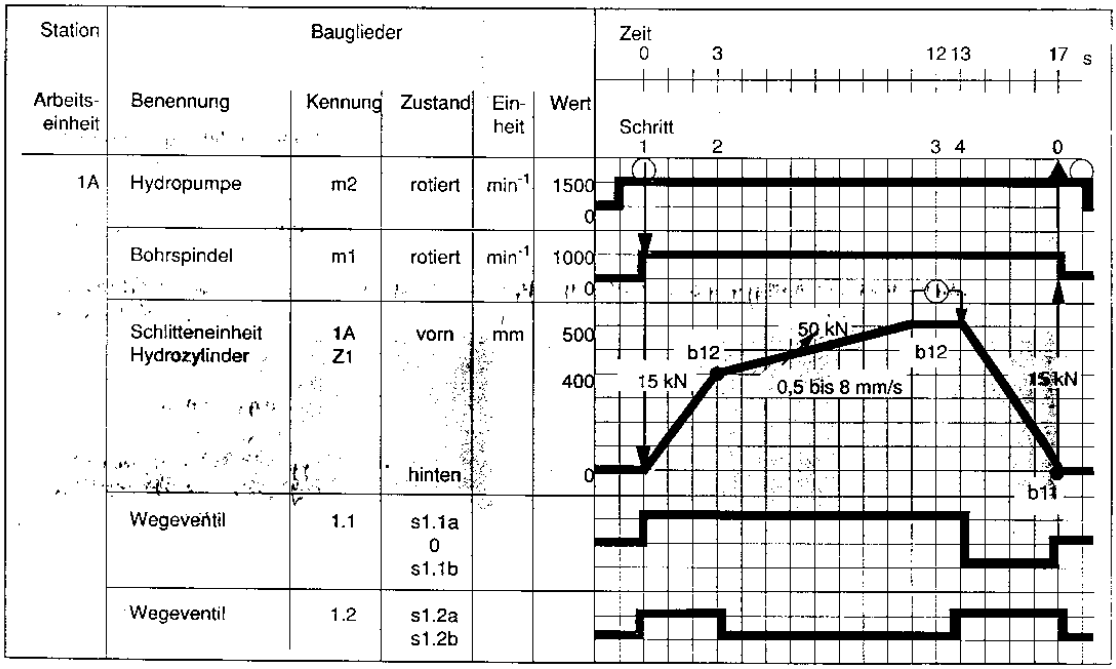

Abbildung 3.2.: Zustands-Diagramm eines Zylinders aus [5, S.18]

werden Phasen genannt.

Eine Phase ist ein vom Benutzer definiertes Teilstück des zu einer hydraulischen Phase gehörenden Verhaltens. Mit Hilfe von Phasen kann der Benutzer Veränderung der Parameter einer hydraulischen Achse angeben. Durch die Gangart kann die Art und Weise genannt werden, in der diese Veränderungen stattfinden.

Generell empfiehlt es sich, die größtmögliche Phase zu nehmen. Es wäre durchaus möglich, in Abbildung 3.1 die Phase „Halten“ in zwei kleine Phasen aufzuspalten. Diese Vorgehen erhöht die Zahl der Phasen und die Rechendauer, ohne einen Vorteil zu bringen. Deswegen unterscheiden sich für gewöhnlich zwei aneinanderliegende Phasen durch ihre Gangart.

Die generelle objektorientierte Aufteilung sieht also wie folgt aus: Eine Anlageanforderung besteht aus ein oder mehreren Achsen, deren Verhalten in Phasen beschrieben wird. Die Reihenfolge der Phasen einer Achse ist vorgegeben.

Der Experte muß aus den Anforderungen einige meist numerische Werte ableiten, die für Achsen und Phasen gelten. Diese werden als Eingabedaten für das für das unter 1.2.1 definierte Problem P genommen. Die Aufnahme der Daten ist einfacher und schneller als die Verarbeitung in manueller Weise: die Beschreibung der Phasen (Eingabe der angestrebten Position oder Geschwindigkeit, dazu Art

der Bewegung) ist nicht so komplex wie das Zeichnen eines Weg-Zeit-Diagramms. Die Beziehungen zwischen Phasen können leicht ausgedrückt werden durch Constraints der Art „Phase 1 darf erst nach Phase 2 stattfinden“. Die Berechnung der Weg-Zeit-Diagramme unter Berücksichtigung dieser Einschränkungen ist im Gegensatz zur manuellen Methode nicht mehr Sache des Experten. Dieser wird damit in zeitlicher Hinsicht entlastet. Dazu wird auch die Achsen-Kopplungs-Hierarchie errechnet, die bei der automatischen Generierung eines vollparametrisierten Schaltplans in [7] benötigt wird, aber auch dem Experten viel über den generellen Zusammenhang der Achsen sagt. Dieser ist ohne diese Information kaum, bei größeren Achsenmengen aber auch dann schwierig für den Menschen zu erkennen.

In den nachfolgenden Abschnitten werden die Anforderungs- und Zielwerte für die drei Objektklassen „hydraulische Anlage“ (im weiteren Verlauf taucht diese in Form von globalen Anforderungen und Zielen auf), Achse und Phase beschrieben. Wie die Beziehungen der Phasen untereinander ausgedrückt werden können, folgt in 3.5. Zum Abschluß des Kapitels werden die Kostenfunktionen beschrieben, nach denen die beste Lösung gefunden werden soll. Zunächst jedoch soll ein Beispiel eingeführt werden, das in den nachfolgenden Abschnitten erweitert wird.

Anforderungsbeispiel

Die im Anhang aufgeführte Anforderungsdatei (siehe Anhang B) soll in den nächsten Kapiteln für Beispiele genutzt werden. Gefordert werden 3 Achsen (A_1, A_2, A_3). Die Parameter dieser Achsen führen bei Betrachtung durch einen menschlichen Experten zur Einsicht, daß A_2 und A_3 sequentiell hintereinander liegen. Die daraus entstehende Meta-Achse M_1 liegt danach seriell hinter A_1.

In der Abbildung 3.3 sieht man stark vereinfachte Sollprofile. Diese Sollprofile zeigen weder die Gangart, noch können die Profile gegeneinander verglichen werden. Gezeigt werden soll damit nur das Ausfahrverhalten. Längen bezeichnen die Ausfahrposition des Endpunkts einer Phase. Kraftangaben (F) bezeichnen einen Kraftsprung direkt am Anfang einer Phase. Kraftangaben mit Pfeil bedeuten: die Kraft steigt während der Phase vom Anfangs- auf diesen Endwert. Ereignis-Zeitpunkte[1] sind durch Kästchen gekennzeichnet, mit dem Wert, der den Zeitpunkt kennzeichnet. Anfangs- und Endpunkte von Phasen sind durch Kreise markiert.

Zwischen P12 und P13 existiert ein Teilstück ohne Phasenkennung. Das bedeutet, daß hier keine Aktion stattfindet, die Phasen P12 und P13 nicht direkt verbunden

[1] engl.: events

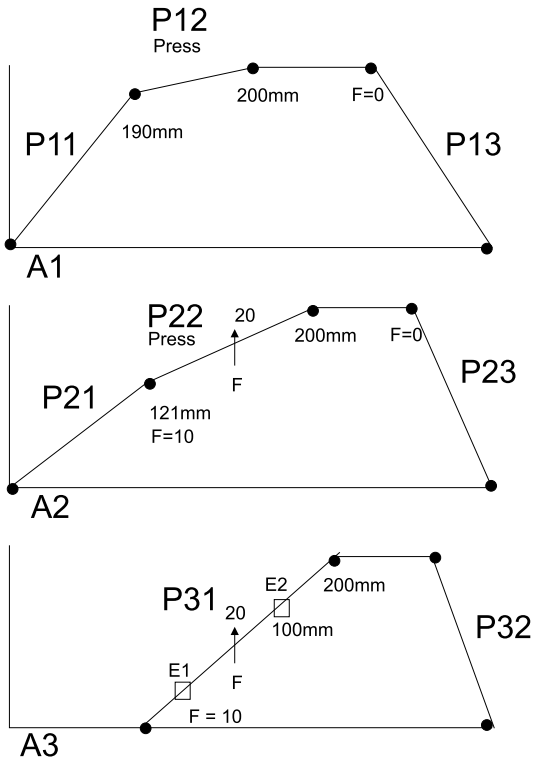

Abbildung 3.3.: Anforderungsbeispiel: vereinfachte Sollprofile

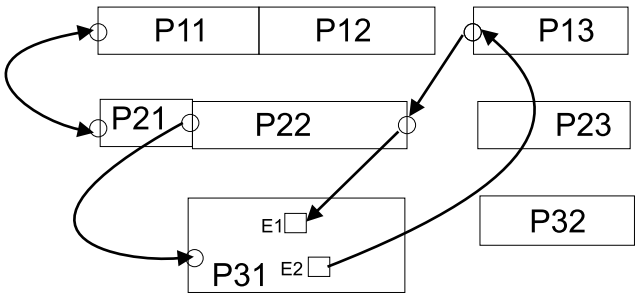

Abbildung 3.4.: Anforderungsbeispiel: Abhängigkeiten

sind. Dieses Teilstück kann während der Berechnung auch auf die zeitliche Länge 0 gesetzt werden!

Die Abbildung 3.4 zeigt die Beziehungen zwischen Phasen. Direkt aneinanderliegende Phasen wie P11 und P12 bedeuten, daß P12 ohne zeitlichen Abstand auf P11 folgt. Alle anderen Constraints sind mit Pfeilen angegeben. Dabei bedeutet ein Pfeil, daß der erste Punkt vor dem zweiten kommt, wie beispielsweise der Anfang von P13 vor dem Ende von P22 liegt. P11 und P21 sollen gleichzeitig sein, so daß auf beiden Seiten Pfeile stehen.

3.2. Globale Anforderungen und Ziele

Global werden nur wenige Anforderungen eingegeben. Dazu gehören die Art des Einsatzgebietes (dadurch wird der Systemdruckbereich eingeschränkt) und der maximale Systemdruck. Letzterer ist eine optionale Angabe.

Die Ziele sind vielfältig: das Ziel Soll-Diagramm ist jedoch eher den Achsen zuzurechnen. Globale Ziele sind der Kopplungsbaum und das globale Scheduling der Phasen. Letzteres wirkt sich natürlich direkt auf die Soll-Diagramme aus. Dazu kommt die Dimensionierung der Pumpe, also die Auswahl einer Pumpe aus der Datenbank.

Am Rande werden auch der maximale Systemdruck und die Gesamtkosten errechnet. Beide fließen direkt in die Kostenfunktionen ein.

3.3. Achse

Die wichtigsten Vorgaben für eine Achse sind natürlich Art und Anzahl[2] der Arbeitselemente. Zusätzlich kann man für einen Zylinder die maximale Ausfahrlänge angeben.

Das direkt den Achsen zuzurechnende Ziel ist natürlich die Errechnung der Soll-Diagramme. Das Kraft-Diagramm und das Weg-Zeit-Diagramm sollen nach der Berechnung als Aneinanderreihung von Polynomstücken ersten bzw. zweiten Grades vorliegen. In Abb. 3.1 ist das Weg-Zeit-Diagramm in der Natur gar nicht zu erreichen: die Funktion ist nicht stetig ableitbar, deswegen ist die Geschwindigkeit nicht stetig. Das Programm muß genauer arbeiten. Es wird zweifache Ableitbarkeit, also auch einfache stetige Ableitbarkeit gefordert. Dazu werden für

[2] In gewissen Situationen werden mehrere parallel arbeitende Arbeitselemente benötigt.

die Position (Weg-Zeit-Diagramm) Polynome zweiten Grades benutzt, die für die Geschwindigkeit zu Polynomen ersten Grades abgeleitet werden (also Geraden) und für die Beschleunigung zu Polynomen nullten Grades (also Geradenstücken parallel zur Zeitachse). Für die Beschleunigung ist diese Darstellung ausreichend.

In der Abbildung 3.2 sind auch Statusprofile für die Ventile zu sehen. Diese zeigen die jeweiligen Einstellungen eines Ventils. Sie werden nicht errechnet: es wäre Wissen über die Art der Ventile erforderlich, vor allem über die Anzahl der Einstellungsmöglichkeiten. Das Programm aus der Arbeit [7] benötigt jedoch nur die Soll-Diagramme und der Kopplungsbaumes, die Ventile werden in der Konfigurationsphase ausgewählt.

Auch die Dimensionierung der Arbeitselemente muß errechnet werden, also ein Hydrozylinder oder Hydromotor aus der Datenbank der verwendbaren Arbeitselemente übernommen werden. Damit hat man gleichzeitig die Kosten für das Element und (bei einem Zylinder) die Ausfahrlänge.

3.4. Phase

Hier wird nur das Verhalten des Arbeitselementes behandelt. Die Beziehungen zwischen Phasen sind so vielfältig, daß sie im folgenden Abschnitt 3.5 erklärt werden.

Wie schon in der Einleitung dieses Kapitels berichtet, wird vom Auftraggeber ein Bewegungsablauf vorgegeben. Statt der Erarbeitung eines Diagramms mit dem Experten reicht jetzt die Eingabe der Endpunkte einer Phase aus. Man kann für die Phase die zu erreichende Position und/oder Geschwindigkeit[3] angeben. Die vorherige Geschwindigkeit und Position ist durch den Endpunkt der Vorgängerphase bestimmt (oder 0, falls es die erste Phase ist). Bei Hydromotoren kann nur die Umdrehungsgeschwindigkeit gefordert werden.

Zusätzlich wird die Belastung am Arbeitselement angegeben: für Hydromotoren das Bewegungsmoment, für Zylinder die Kraft. Anfangs- und Endwert für die Phase müssen explizit angegeben werden. Es kann nämlich zu sprunghafter Änderung kommen, wenn beispielsweise ein Zylinder ohne Widerstand ausfährt (0 N) und dann gegen etwas prallt, daß er pressen soll. Zu Beginn der Pressphase steigt dann die Lastkraft sprunghaft an (auf beispielsweise 15 N), innerhalb der Pressphase wird sie kontinuierlich erhöht (auf ihren Endwert, zum Beispiel 30 N).

[3]Bei gleichzeitiger Angabe kommt es jedoch häufig dazu, daß die beiden Angaben sich gegenseitig ausschließen. Dieses muß ausgeschlossen werden.

Entscheidend ist neben Belastung und zu erreichendem Endwert auch die Art des Erreichens, die Gangart. In 3.1 gibt es unter anderem die Phasen Eilgang (Accelerate), Pressen (Press) und „Druck halten“ (Hold Pressure). Eilgang ist gleichbedeutend mit schneller Beschleunigung bis hin zu möglichst maximaler Geschwindigkeit, um eine Position zu erreichen (ob zum Schluß mit maximaler negativer Beschleunigung gebremst wird, hängt davon ab, ob der Zylinder danach weiter ausfahren soll). Pressen steht für langsames, kontinuierliches Ausfahren. Bei „Druck halten“ sollte sich der Zylinder nicht bewegen und der Druck sich nicht ändern.

Es gibt fünf weitere Gangarten:

- Bei der Punktfahrt (Point Drive) wird energieoptimale Bewegung verlangt, also möglichst kleine positive (zu Anfang) und negative (zum Ende) Beschleunigung. Meistens tritt dabei gar keine konstante Geschwindigkeit auf.
- Bei der Konstantfahrt (Constant Drive)wird eine möglichst lange konstante Geschwindigkeit erwartet.
- Dekompression (Decompress) ist das Gegenteil von Pressen.
- Bei der Positionsfahrt (Position Drive) wird schnelles Erreichen der Position erwartet, also schnelle Beschleunigung und schnelles Bremsen.
- „Position halten“ (Hold Position) bedeutet: keine Geschwindigkeit.

Neben den drei Haupteingaben Endpunkte, Bewegungsart und Belastung sollen zusätzlich noch Einschränkung zur Zeit (Minimum oder Maximum) gemacht werden können. Außerdem kann der Benutzer eine Sicherheitspause hinter einer Phase einschieben. Innerhalb dieser zeitlich von ihm festgelegten Pause ändert sich die Position (Zylinder) oder die Bewegung (Hydromotor) nicht. Die Pause dient Sicherheitsgründen oder auch als Erholung für das Arbeitselement.

Die genannten Eingaben werden in dieser Arbeit auch **Aktionsteil der Phase** genannt, im Gegensatz zum **Constraint-Teil**, der jetzt folgend beschrieben werden soll.

Im Beispiel aus Abbildung 3.3 und 3.4 sind 8 Phasen zu sehen. Für 2 Phasen ist die Gangart angegeben: für P12 und P22 wird die Gangart „press“ gefordert. Die Endposition für P11 ist 121mm, diese ist gleichzeitig die Anfangsposition von P22. Am Beginn von P22 wird die Kraft sprunghaft auf F = 10 N gesetzt. Vermutlich wird an dieser Position der Zylinderkolben gegen einen Widerstand gepresst. Während der Phase P22 steigt die Kraft langsam auf 20N.

3.5. Constraints zwischen Phasen

Im **Constraint-Teil einer Phase** werden die Beziehungen der Phase zu anderen Phasen beschrieben. Implizit gibt es noch eine weitere Beziehung zur Vorgänger- und Nachfolgerphase innerhalb der Achse, und damit auch zu allen anderen vorherigen oder nachfolgenden Phasen der Achse. Dabei handelt es sich aber nur um eine vorher/nachher-Beziehung.

Zwischen allen Phasen eines Schaltkreises können weitere Beziehungen eingebaut werden. Weil Phasen Intervalle über die Zeit sind, bieten sich sieben einfache Beziehungen an. Diese sind aus [1] entnommen.[4]

Eigentlich sind es 4 Beziehungen mit ihrer invertierten Variante (A und B sind folgend zwei Phasen):

- **A vor (before) B** ist gleichbedeutend mit **B nach (after) A**. Die Phase A muß also enden, bevor B beginnt.
- **A direkt vor (direct before) B** bedeutet auch **B direkt nach (direct after) A**. Hier wird gefordert, daß das Ende der Phase A mit dem Beginn von B übereinstimmt.
- **A enthält (holds) B** indiziert **B liegt in (during) A**. Das Intervall B darf also frühestens mit dem Start von A beginnen, muß aber spätestens zusammen mit dem Ende von A enden.
- Bei **A gleicht (equals) B** handelt es sich um eine symmetrische Relation, also gilt auch **B gleicht A**. A und B weisen die gleiche Start- und die gleiche Endzeit auf.

Diese Beziehungen sollen im weiteren Verlauf **Constraint-Makros** genannt werden.

Zusätzlich sollen komplexere Beziehungen zugelassen werden. Diese können den Beginn oder das Ende einer Phase betreffen, aber auch einen sogenannten Ereignis-Zeitpunkt[5]. Ein Ereignis tritt innerhalb einer Phase auf und wird durch eine Relation zwischen einer Variable der Phase (Position, Kraft, Geschwindigkeit) und einem Wert beschrieben, zum Beispiel „F $>$ 5“. Durch diese Relation wird

[4] Sechs der 13 Beziehungsarten aus der Arbeit von Allen sollen allerdings nicht genutzt werden. A überlappt B beispielsweise kann für Phasen einer Achse nicht gelten, für Phasen unterschiedlicher Achsen kann man dieses Verhältnis einfacher und präziser ausdrücken durch Constraints über Ereignis-Zeitpunkte.

[5] engl.: event

ein Ereignis-Zeitpunkt beschrieben, der zum Vergleich mit anderen Zeitpunkten geeignet ist.

Zwischen zwei Zeitpunkten (Ereignis, Beginn oder Ende) können die Relationen $<, \leq, ==, \geq, >$ gelten. Es kann also gefordert werden, daß der erste Zeitpunkt vor oder nach dem zweiten stattfindet. Damit ist zum Beispiel möglich, daß beim Erreichen einer bestimmten Ausfahrposition des ersten Zylinders der zweite Zylinder auch auszufahren beginnt.

Zusätzlich können Verzögerungen ausgewählt werden, so daß beispielsweise eine Aktion erst genau 5 Sekunden nach einem Zeitpunkt passiert.

Nach der Definition der möglichen Constraints ist auch der Begriff Constraint-Makros einfach zu erklären. Constraint-Makros können leicht in ein oder zwei einfache Constraints verwandelt werden. „A vor B“ ist nichts weiteres als „Ende von A liegt vor Beginn von B“. „A gleicht B“ kann ersetzt werden durch „Beginn von A gleich Beginn von B“ und „Ende von A gleich Ende von B“.

Im Beispiel gibt es sehr viele Beziehungen. P12 muß direkt nach P11 beginnen, also gilt „P11 direct before P12“ oder umgekehrt „P12 direct after P11“. Dieses Constraint-Makro kann auch ersetzt werden durch „Zeitpunkt P11 Ende“ == „ Zeitpunkt P12 Anfang“. Auch zwei Ereignis-Zeitpunkte sind eingebaut. Der Zeitpunkt E1 liegt genau dort, wo die Kraft 10 N groß wird. Dieses Ereignis darf erst nach dem Ende von P22 passieren, also „ Zeitpunkt P22 Ende“ < „Zeitpunkt P31 F == 10“.

3.6. Auswahl der Zielkriterien

Es müssen Zielkriterien gefunden werden, die eine große Relevanz besitzen, und in die Berechnung einfließen können (beispielsweise fehlen bei der Umweltverträglichkeit von Anlagen genaue Daten). Viele Kriterien beeinflußen sich gegenseitig.

1. Minimierung der Achsenanzahl: Wenn die Arbeitselemente zweier Achsen identischer Klasse sind (also entweder beide Hydromotoren oder beide Zylinder) und dazu die Weg- und Kraftdiagramme[6] und die Baugrösse (bei Zylinder: beide maßgebenden Flächen) übereinstimmen, dann können zwei vorgegebene Achsen zu einer verschmolzen werden. Dadurch können viele Kosten (u.a. mehrere Baukomponenten) gespart werden. Ausserdem sinkt die Komplexität der Anlage.

[6] bei Hydromotoren: Drehmomentdiagramme

2. Nutzung sparsamer Kopplungen: Die Kopplungen werden häufig von Anforderungen implizit eingeschränkt. Wenn jedoch die Wahl zwischen verschiedenen Kopplungsarten getroffen werden kann, gilt u.a. aus Kostengründen: sequentiell ist besser als seriell, seriell ist besser als parallel.

3. Minimierung der Kosten: Generell führen kleinere Komponenten zu kleineren Kosten. Als Nebenwirkung führen beispielsweise kleinere Fläche zu Veränderung von Druck und Volumenstrom. Dadurch werden andere Punkte negativ modifiziert. Es können nicht alle Kosten eingerechnet werden, sondern nur die Kosten für Hydromotoren, Zylinder und Pumpen. Dazu kommen in der Praxis weitere Anschaffungskosten für beispielsweise Rohre genauso wie Wartungs- und Instandsetzungskosten. Letztere hängen u.a. vom Druck ab.

4. Minimierung der Laufzeit: In der Produktion bedeutet schnellere Laufzeit mehr produzierte Ware und aufgrund dessen mehr Einnahmen. Dafür müssen meistens hohe Drücke und Volumenströme in Kauf genommen werden.

5. Maximierung des Leistungswirkungsgrades: Der Leistungswirkungsgrad gibt an, wieviel Prozent der Pumpenleistung (also Antriebsleistung) tatsächlich von Zylindern und Hydromotoren wieder abgegeben wird (Abtriebsleistung). Der Rest ist Leistungsverlust vor allen Dingen durch Wärme und Umsetzung. Verlorene Leistung ist gleichzusetzen mit verlorenem Geld.

6. Minimierung der Varianz der Pumpen-Leistung: eine gleichmässige Belastung schont die Pumpe und führt zu weniger Ausfällen. Dadurch sinken Ausfall- und Instandsetzungskosten.

7. Minimierung des Systemdrucks: In erster Linie soll durch die Minimierung des Systemdrucks die Dissipation (Verlust durch Reibung) verringert werden. Reibungsverluste entstehen durch hohe Drücke und durch enge Querschnitte. Die Rohr-Querschnitte werden im hydraulischen Simulator FluidSim bisher außer Acht gelassen. Deswegen soll die Minimierung des Systemdrucks reichen. Es gibt aber noch mehr Auswirkungen des Systemdrucks: je höher der Druck, desto größer der Lärm beim Schalten und die Leckage. Außerdem steigt mit sinkendem Druck die Lebenserwartung der Anlage.

8. Minimierung der ϵ-Schläuche an den Diagrammen (Gesamtgüte): es ist in vielen Fällen nicht nötig, die Anforderungen zu 100% einzuhalten. Deswegen kann die Toleranzgrenze vorgeben werden. Jedoch sollte eine 90%ige Lösung nicht so gut wie eine 100%-Lösung bewertet werden, wenn sie nach den anderen Kriterien gleichartig zu sein scheint. Die Bezeichnung ϵ-Schlauch ist folgend zu deuten: wenn $f(x)$ die einzuhaltende Funktion

ist, dann ist $[f(x) \cdot (1 - \epsilon), f(x) \cdot (1 + \epsilon)]$ die mögliche Lösung. Dabei soll ϵ möglichst klein sein.

Es gibt im Bereich der Hydraulik noch weitere Ziele. Es wurden an dieser Stelle nur die wichtigsten ausgewählt. Die Kosten während der Nutzung werden durch andere Kriterien berührt, aber nicht direkt eingerechnet. Beispielsweise fallen bei höherer Druckzahl größere Wartungs- und Instandsetzungskosten an.

Der Benutzer soll eine Gewichtung der Kriterien vornehmen können. Es gibt durchaus Anforderungen, bei denen nur die Kostenminimierung verlangt wird, die Laufzeit ist uninteressant (weil der Arbeitsvorgang nur selten an einem Tag ausgeführt wird).

Kapitel 4

Berechnungen, Verfahren und Lösungen

4.1. Modellbildung

Durch die Anforderungen und die Objektorientierung ist die Modellbildung in weiten Teilen schon vorgeben. Es ist klar, daß Objekte der Typen Schaltkreis, Achse und Phase existieren müssen. Auch muß eindeutig der Schaltkreis eine Liste seiner Achsen, jede Achse eine Liste ihrer Phasen enthalten. Dazu muß ein Baum aus Meta-Achsen während der Lösungssuche aufgebaut werden, der dem Schaltkreis zugeordnet wird. Zur Implementierung kann gesagt werden, daß die Verkettungen immer über Zeiger erfolgen.

Sämtliche hydraulische Zusammenhänge sollen mit Hilfe von Constraints eingearbeitet werden. Es ist durchaus sinnvoll, beispielsweise die Kopplungsbedingungen mit Hilfe von Regeln und Regelverarbeitung einzuführen. Das schafft jedoch die Notwendigkeit einer Erweiterung des Programms um diese Regelverarbeitung. Stattdessen wird auch an dieser Stelle auf Constraints gesetzt: es werden — wie auch bei Regeln — gewisse Bedingungen für jede Kopplungsart eingesetzt.[1] Diese Bedingungen sind ebenso Constraints des Constraint-Netzwerks wie auch

[1] In gewisser Hinsicht sind Constraints Regeln: ausgehend von anderen Variablen wird eine Aussage über eine Variable getroffen. Das ähnelt einer Regel „FALLS folgende Variablenwerte vorliegen DANN setze bestimmte Variable auf bestimmten Wert“.

die Kopplungsmöglichkeiten. Somit kann durch die Einschränkung der Werte der Bedingungsinformationen auch die Kopplungsart eingeschränkt werden. Das hydraulische Wissen in Form von Formeln kann sehr schnell eingearbeitet werden.

Die Constraints können in verschiedene Kategorien eingeteilt werden: die inneren Constraints betreffen nur Variablen eines Objektes (einer Phase, einer Achse). Beispiel: $F = p \cdot A$. Äußere Constraints betreffen mehrere Objekte, beispielsweise $p_1 = p_2$ für zwei untereinanderliegende Meta-Achsen. Daneben gibt es Kopplungs-Constraints, die unten beschrieben werden.

Die Constraints werden als Funktionen eingearbeitet.

Die Einordnung der Zeitpunkte in das Modell ist schwierig. Jeder Zeitpunkt gehört zu genau einer Phase. Jede Phase besitzt die zwei Zeitpunkte Anfang und Ende. Dazu kommen Ereignis-Zeitpunkte, die von Variablenwerten abhängen, und erst dann stattfinden können, wenn eine der die Phase beschreibenden Funktionen einen bestimmten Wert erreicht oder überschreitet. Die Lösung besteht in der Modellierung der Zeitpunkte als eigene Objekte. Diese Objekte zeigen auf ihre Phasen zurück, jede Phase besitzt auch eine Liste ihrer Zeitpunkt-Objekte. In späteren Schritten sollen Zeitpunkte zusammenfaßbar sein, wenn sie gleichzeitig sein müssen (siehe 4.8.1).

Neben den inneren und äußeren Constraints existieren Kopplungsconstraints, die die Beziehung zweier Phasen beschreiben. Sie betreffen immer die Zeitpunkte der jeweiligen Phasen, nicht die Phasen direkt. Auch im Modell sollen sie nicht auf die Phasen angewandt werden. Es wird ein Graph gebildet, bei dem die Zeitpunkte als Knoten und die Verbindungen als Kanten fungieren. Constraint-Makros werden zu diesem Zweck in ihre Constraints aufgespalten. Die Kanten sind vom gerichteten Typ. Zusätzlich enthalten sie als Wert ein Interval, das die Größe des Abstands zwischen den beiden Zeitpunkten angibt. Eine Kante mit Wert $[-1; 3]$ zwischen zwei Zeitpunkten Z_1, Z_2 bedeutet also, daß der zweite Zeitpunkt Z_2 frühestens 1 Sekunde vor, aber spätestens 3 Sekunden nach Z_1 liegen darf. Wenn nur eine obere oder nur eine untere Schranke gegeben ist, so wird dieses durch Einsetzen von ∞ angegeben: $[-1; \infty]$. Beim letzteren Beispiel liegt Z_2 also frühestens 1 Sekunde vor Z_1, oder beliebig dahinter. Feste Werte wie Gleichheit können durch ein auf einen Punkt beschränktes Interval $[a; a]$ angegeben werden: $[0; 0]$ steht für die Gleichheit.[2]

Die Aussage, daß diese Kanten gerichtet sind, ist nur bedingt wahr. In Wirklichkeit kann jede Kante mit Wert $[a; b]$ durch eine umgekehrte Kante mit Wert

[2] Um nicht mit ∞ direkt arbeiten zu müssen, wofür sich eine symbolische Darstellungsform anbieten würde, wird ∞ im Programm durch eine hohe, aus dem normalen Wertebereich herausragende Zahl ersetzt. Dabei muß beachtet werden, daß bei der Addition zweier Intervalle kein Wert über ∞ (bzw. der Darstellung von ∞) erreicht wird.

$[-b;-a]$ ersetzt werden. Im Beispiel kann die Kante $[-1;3]$ zwischen Z_1, Z_2 in die Kante zwischen Z_2, Z_1 mit dem Wert $[-3;1]$ verwandelt werden. Deswegen sind für viele Teilprobleme während der Verarbeitung effiziente Algorithmen für ungerichtete Graphen nutzbar.[3] Angenommen, es bestände eine Kante von E2 zu E1 im Beispiel, dann würde es einen Kreis mit „Begin P13", „Ende P22", E1 und E2 geben. Die Kanten gehen alle gegen den Uhrzeigersinn (in der Abbildung 3.4) bis auf die Kante zwischen E2 und E1. Für Längenberechnungen kann man dann das invertierte Interval der Kante von E2 und E1 berechnen, um das Interval zwischen E1 und E2 zu erhalten.

Die Einarbeitung der Funktionen für Weg, Kraft, Leistung etc. ist ebenfalls etwas schwieriger als die Verwendung bloßer numerischer Daten. In Kapitel 3 wurde schon festgelegt, daß die Funktionen mit Hilfe von Polynomstücken beschrieben werden sollen. Der Grad dieser Funktionen ist unterschiedlich. Für die Ausfahrposition eines Zylinders beispielsweise wird ein Grad von 2 gefordert. Außerdem soll die Funktion stetig und differenzierbar sein. Ebenso soll die Ableitung der Posititon, die Geschwindigkeit, stetig und differenzierbar sein, sie hat natürlich nur den Grad 1. Für die Beschleunigung, die Ableitung der Geschwindigkeit, reicht damit die Angabe von Geradenstücken parallel zur x-Achse, also von konstanten Werten. Bei anderen Funktionen wie dem Druck reicht auch der Grad 1 für die Polynomstücke. Die Funktionen können gut als Liste angegeben werden, in der für die einzelnen Stücke nur die Koeffizienten angegeben werden. Für $\frac{1}{2} \cdot x^2 + x + 3$ reicht also das Tripel $(\frac{1}{2}, 1, 3)$.

Die Bandbreite der möglichen, noch einzuschränkenden Funktionen ist nicht trivial. Abbildung 4.1 zeigt vier verschiedene Möglichkeiten für Positionsfunktionen. Es handelt sich um Funktionen höchstens 2. Grades. Nicht nur ändert sich die Länge der Phase (2, 3 oder 4), auch ist die Art der Funktion anders: bei den ersten beiden Beispielen wird zunächst große Geschwindigkeit verlangt, die am Ende immer stärker abgebremst wird. Die dritte Funktion ist linear. Ein besonderer Fall ist die vierte Funktion, die aus zwei Polynomstücken besteht (2 Kurven, also Funktionen 2. Grades hintereinander). Ursache kann eine parallele Achse sein, deren größte Ausfahrposition zwischen Zeitwert 2 und 3 erreicht wird. Danach kann die Achse dann schnell ausfahren, weil sie sämtlichen Volumenstrom aufnimmt, den sie sich vorher mit ihrer Schwesterachse teilen mußte.

Die möglichen Funktionen für eine Phase sind weder aufzählbar noch in sinnvollem Maße mit Hilfe von anderen Funktionen einschränkbar. Deswegen müssen andere Daten gemerkt werden, beispielsweise Mindest- und Höchstlänge (zeitlich), Anfang- und Endwert, vorgegebene Gangart (die sich auf die Art der Funktion stark auswirkt).

[3] Bei den meisten Problemen sind die Lösungen für die ungerichteten Graphen schneller als die gleichen oder ähnliche für einen gerichteten.

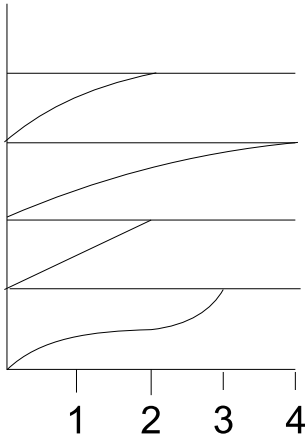

Abbildung 4.1.: Funktionsbandbreite für eine Phase

Wie schon auf Seite 59 gezeigt, ist der Kopplungsbaum Teil des Constraintnetzwerkes. Durch die Meta-Achsen des Kopplungsbaums fließt Volumenstrom mit Druck. Der genaue Fluß ist in Kapitel 2.1.5 dargestellt. Der Fluß muß mit Hilfe von Constraints eingearbeitet werden. Weil die Kopplung zu Beginn nicht feststeht, ergeben sich (zu) viele Möglichkeiten, die schnell abgebaut werden müssen. Deswegen ist die Lösung der Kopplungsconstraints vorrangig zu sehen.

Zusammengefaßt besteht das Modell also aus mehrere Teilen, die jedoch miteinander (auch durch Constraints) verbunden sind: grundsätzliche Objekte (Achsen und Phasen), Kopplungsbaum und Zeitpunkte mit Beziehungen.

Während der Constraintlösungsphase im folgenden Kapitel werden Schritte vollzogen, in denen sich die Werte der Objekte verändern können. Diese Schritte werden wegen des Branch-and-bound-Ansatzes auch wieder zurückgespielt. Um eine schnelle Verarbeitung zu gewährleisten, wird für jedes Objekt ein eigener Stack bereitgestellt. Bei einer Wertveränderung wird ein neues Stackobjekt aufgelegt, das die veränderten Werte enthält (zusätzlich natürlich alle nicht veränderten Werte). Bei zusammengelegten Zeitpunkt-Objekten liegt nur noch ein Stackobjekt auf allen verschmolzenen Zeitpunkt-Stacks zusammen.

4.2. Einordnung der Scheduling-Methoden

Man kann das Problem P wie folgt als Scheduling-Problem einordnen: es gibt m Maschinen, nämlich die Achsen. Es liegt dabei keine Parallelität vor: es gibt also keine zwei Maschinen, die das gleiche leisten können. Für jede Maschine gibt es

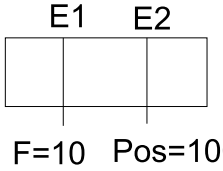

Abbildung 4.2.: Zwei Ereignis-Zeitpunkte in einer Phase

n_i Jobs oder Phasen, n_i ist die Anzahl der Phasen der Achse i. Jeder Job braucht nur eine Maschine. Zwischen den einzelnen Jobs gibt es Constraints: implizit zwischen den Phasen einer Achse und explizit durch den Benutzer angegeben. Somit kann man das Problem P bis zu diesem Punkt als Flow Shop, Open Shop oder Job Shop einordnen. Die Anzahl der Maschinen ist normalerweise größer als 1. Es gibt Vorrang-Constraints zwischen den Jobs. Die Zielfunktion besteht aus 6 Teilen, die in 4.3 erklärt werden. Eine Teilfunktion ist dabei die Gesamtzeit-Minimierung.

Zwei Eigenschaften erweitern die Vorrang-Constraints im Vergleich zu herkömmlichen Scheduling-Problemen. Erstens ist bei den normalen Vorrangs-Constraints nicht vorgesehen, daß diese mit einer Zahl oder einem Interval belegt werden können. Es soll aber möglich sein, daß eine Phase erst eine gewisse Anzahl von Sekunden nach dem Ende einer anderen beginnt (z. B. aufgrund von eingegebener Sicherheitszeit). Bei positiver Verschiebung (A soll 5 Sekunden nach Beendigung von B beginnen) kann man dieses Problem durch Einfügen eines Zwischenjobs lösen. Bei negativer Verschiebung (A soll 5 Sekunden vor Ende von B beginnen) ist dieses nicht möglich.

Zweitens sprengen die Ereignis-Zeitpunkte den Rahmen des Scheduling. Eine erste Idee ist einleuchtend: die Ereignis-Zeitpunkte geschehen immer innerhalb einer Phase. Deswegen kann man die Phase aufstückeln. In Beispiel 4.2 sieht man zwei Ereignisse. [4] So könnte man die angegebene Phase in drei Unterphasen (vom Beginn zum 1. Ereignis, vom 1. zum 2. vom 2. zum Ende) unterteilen. Leider kann diese Idee nicht umgesetzt werden: es steht in vielen Fällen die Reihenfolge der Ereignisse nicht fest, sie ergibt (und verschiebt) sich erst während der Berechnung. So kann das Ereignis $Pos = 100$ vor das Ereignis $F = 10$ rücken. Es ist sogar möglich, daß beide Zeitpunkte gleich liegen.

Man sieht also, daß die direkte Umsetzung auf ein bekanntes Scheduling-Problem und -Verfahren nicht möglich ist.

Ein kleines Problem bedeutet auch die Tatsache, daß die eigentliche Länge der Phasen nicht bekannt ist. Dieses könnte durch Scheduling-Verfahren, wie sie

[4] $F = 10$ und $Pos = 100$: Kraft ist gleich 10 N bzw. Position ist 100mm.

auch bei Prozessoren gebraucht werden, gelöst werden.[5] Jedoch ist der Einbau des hydraulischen Wissens und der Kopplungen schwierig bei diesen Algorithmen.

Ein gebräuchlicher Scheduling-Ansatz geht von der Einordnung der kritischsten Jobs als erstes aus. Damit sind bei Scheduling-Problemen mit Deadlines die Jobs mit früher Deadline gemeint. Auf Seite 42 wird gezeigt, wie auch bei Problemen ohne vorgegebene Deadlines eine ähnliche Vorgabe berechnet wird, nämlich die spätestmögliche Startzeit. Die Kritische Pfad-Methode soll beim vorliegenden Problem P genutzt werden. Sie ist nicht direkt wie auf S. 42 beschrieben umsetzbar, kann aber als Leitmotiv eingesetzt werden. Constraints mit wenigen Möglichkeiten sollten zuerst eingeschränkt werden. Vor allen Dingen gilt dies für Phasen, für deren Funktion nur wenige Möglichkeiten bestehen. Diese werden als erstes bearbeitet.

Während des Scheduling muß auch erkannt werden, ob Events überhaupt stattfinden können. Beispielsweise kann die Forderung „F >5“ für eine Phase nie erfüllt werden, wenn die Kraft in der Phase von 2 auf 4 wächst.

4.3. Kostenfunktionen

4.3.1. Auswahl des Lösungsansatzes

Folgende Bedingungen gelten für die Lösung des Problems P:

- Die Lösung muß die beste hinsichtlich mehrere Ziele sein. Die einzelnen Ziele werden durch Zielfunktionen angegeben und durch den Benutzer gewichtet.
- Die Lösung muß sämtliche Nebenbedingungen erfüllen. Nebenbedingungen bestehen aus Einschränkungen des Wertebereiches (explizit oder implizit angegebene Minima und Maxima für die zeitliche Länge einer Phase) und den Constraints zwischen den Phasen (beispielsweise: Phase 1 muß vor Phase 2 stattfinden).
- Der Lösungsraum ist stetig.
- Bei den Daten handelt es sich nicht nur um numerische Werte, sondern auch um Polynome bis zum Grad 2.[6]

[5] Auch dort sind nicht immer die benötigten Zeiten bekannt.

[6] Es gibt auch Nominalobjekte wie das Arbeitsfeld, die sich jedoch nur vor der eigentlichen Berechnung verändernd auf einige numerische Werte verhalten.

Dabei schränkt vor allen Dingen der Punkt der Gewichtung den Lösungsansatz ein.

Interaktive Methoden sind von vorneherein ausgeschlossen, da bei diesen die Trennung zwischen Anforderungseingabe und Lösungssuche entfällt. Gewollt ist jedoch eindeutig eine automatisierte Lösungsfindung, mit der auch Nicht-Experten schnell und sicher eine gute Lösung finden können.

Bei **Präferenzmethoden** ist die Gewichtung schon vorausgegeben. Jedoch sind Ziele geringer Wichtigkeit nur dann interessant, wenn mehrere Alternativen für das erste Ziel gleich gut sind. Ziele gleicher Wichtigkeit sind nicht möglich.

Gewichtung ist bei **Outranking-Methoden** möglich: statt die Anzahl der für A sprechenden mit der Anzahl der für B sprechenden Kriterien zu vergleichen, werden einfach die Gewichtungen der für die jeweilige Alternative sprechenden Kriterien zusammengezählt. Das Problem dabei ist leicht an einem Beispiel zu erkennen: die drei Ziele 1, 2 und 3 sollen gleich gewichtet sein. Wenn die Kriterien 1 und 2 leicht für die Alternative A sprechen, 3 schwer für B, dann wird auf jeden Fall A genommen. Es wird also keine Rücksicht darauf genommen, wie sehr sich die Alternativen betreffend eines Ziels unterscheiden. Damit kommt eine reine Outranking-Methode nicht in Frage.

Es bleiben also der skalare und der vektorelle Ansatz. Dabei ist der skalare Ansatz zu bevorzugen. Er ist einfacher und schneller in der Berechnung, handelt es sich doch um eine Vereinfachung des vektorellen Ansatzes. Ebenso ist der skalare Ansatz bei Gewichtung besser geeignet. Dazu kommt, daß bei Constraint-Optimierungs-Problemen per Definition eine reellwertige Funktion genutzt wird. Um das Problem P direkt als ein Constraint-Optimierungs-Problem zu nehmen, ist der skalare Ansatz also sehr geeignet. Anzumerken ist dennoch, daß die Berechenbarkeit in gewisser Weise immer in Frage gestellt werden muß. Das liegt vor allen Dingen daran, daß ein Mensch nur schwerlich in jedem Fall eine sichere Unterscheidung der Lösungsgüte erreichen kann. Eine gute Nutzenfunktion hängt jedoch von der Güte des Experten ab, der sie entwickelt [7]

4.3.2. Die benutzte Kostenfunktion

Hier soll nun kurz erklärt werden, wie die festgelegten Optimierungskriterien in die Kostenfunktion eingesetzt werden.

[7] In vielen Fällen sind natürlich mehrere Experten am Werk, so daß die Objektivität größer werden sollte.

Wie schon in 3.6 erwähnt, hat der Achsenkopplungsbaum die höchste Priorität. Die Struktur des Achsenbaums hat nur indirekt Auswirkung auf die Kostenfunktion. Unterschiedliche Kopplungen bewirken bei fast allen Kriterien auch unterschiedliche Werte, unter anderem ist der Systemdruck bei paralleler Kopplung häufig höher als bei den anderen Kopplungsarten. Die Beziehungen zwischen den Achsen wirken sich auf die Kopplungen aus (beispielsweise sollten vollkommen unabhängige Phasen [8] nur parallel verbunden sein.).

Ein weiteres Hauptkriterium ist die Erfüllung aller Constraints. Dieses Kriterium ist höher zu bewerten als alle anderen, nämlich als Voraussetzung für die Lösung.

Die anvisierte Kostenfunktion soll additiv aus Teilfunktionen zusammengerechnet werden. Für jedes Ziel gibt es eine dieser Teilfunktionen. Dazu kommt aber eine Gewichtung zwischen 0 und 1, die einfach mit dem Teilfunktionswert multipliziert wird. Somit wird das Minimum von $z(x) = \sum_{i=1}^{6} w_i \cdot z_i(x)$ gesucht.

Es gibt wiederum zwei mögliche Ansätze:

1. Es gibt eine direkt angegebene Beziehung zwischen den einzelnen Funktionen. Beispielsweise: 1 bar Systemdruck entspricht 10 DM Kosten.

2. Jede Funktion bildet den für das einzelne Ziel interessanten Wert der Lösung in eine reele Skala ab. Die Abbildung ist jedoch von den Anforderungen abhängig. Beispielsweise soll in den Bereich zwischen 0 und 1 abgebildet werden. Bei gleichmäßiger Verteilung und Belegung der Randwerte wird z. B. bei Lösungswerten zwischen 0 und $100bar$ der Druckwert $50bar$ auf $0,5$ abgebildet, bei Lösungswerten zwischen 0 und 1000 jedoch auf $0,05$. Wenn die Funktion für die DM-Kosten gleichbleibt, ist bei der zweiten Variante $1bar$ nur noch 1/10 soviel Wert wie bei der ersten. Man versucht also die Optimierung des Machbaren und berücksichtigt das schon zu Anfang Ausschließbare nicht.

Der erste Ansatz setzt voraus, daß die Beziehungen zwischen den Werten gut abgeschätzt werden können. Dies ist aber nicht immer der Fall. Außerdem werden Werte als erreichbar vorausgesetzt, die im Endeffekt wegen der gegenseitigen Beeinflußung der Ziele nicht erreicht werden können. Beim zweiten Ansatz hingegen ist die Abschätzung des Erreichbaren wichtig. Die optimale Methode liegt meistens wahrscheinlich in der Mitte: die Abschätzung der Beziehungen ist nicht einfach, dafür sehr fehlerträchtig. Der zweite Ansatz läßt wirklich klare Beziehungen nicht zu. Der Kompromiß verlangt zwei Schritt. Vergleichbare

[8] „Vollkommen unabhängige Achsen“ sind auch indirekt über andere Phase nicht voneinander abhängig

Teil-Zielfunktionen sollten im ersten Schritt miteinander verrechnet werden. Im zweiten Schritt werden die unabhängigen Ergebnisse des ersten Schrittes aufaddiert.

Zum im Kapitel 3 vorgestellten Beispiel sollen folgende Kriterien gefordert werden: die Geldkosten sollen höchstwichtig sein und die Gewichtung 10 erhalten. Die Dissipation soll mit 5 gewichtet werden. Die restlichen Kriterien sind eher unwichtig und werden alle mit 1 belegt; damit sind sie für die Berechnung noch relevant. Bei einem Wert von 0 würden sie ignoriert werden.

Aufgrund des Umfangs wird für die Zielfunktion auf die Auflistung der gesamten Teil-Zielfunktionen verzichtet. Verrechnet werden die Funktionen wie gesehen additiv mit Gewichtung. Als Beispiel werden zwei Teil-Funktionen erläutert:

Minimierung der Kosten

Die möglichen Mindestkosten setzen sich aus den Kosten des billigsten Zylinders multipliziert mit der Anzahl der benötigten Zylinder, den Kosten des billigsten Hydromotors multipliziert mit der Anzahl der benötigten Hydromotoren und den Kosten der billigsten Pumpe zusammen. Die möglichen Maximalkosten setzen sich wiederum aus Zylinder-, Hydromotor- und Pumpenkosten zusammen, dabei muß natürlich die jeweils teuerste Komponente genommen werden. Der Wertebereich zwischen Mindestkosten und Maximalkosten wird gleichmäßig auf den Wertebereich zwischen 0 und 1 abgebildet.

Maximierung des Leistungswirkungsgrades

Dieser hängt von den Leistungswirkungsgraden der Pumpen und der Arbeitselemente ab. Die Berechnung ist einfach. Die durchschnittlichen Leistungen der Arbeitselemente werden zusammengezählt und durch die durchschnittliche Leistung der Pumpe geteilt. Das Ergebnis liegt zwischen 0 und 1, dabei ist 1 jedoch das Optimum. Bei diesem Wert würde sämtliche Leistung der Pumpe auch an den Arbeitselementen ankommen. Dieses ist natürlich wegen Leckagen und Druckverlusten utopisch. Da bei der Verrechnung der Funktionen ein niedriger Wert als besser gilt, wird das Ergebnis von 1 abgezogen. Dadurch ist der Wert errechnet.

4.3.3. Anmerkung zur Vergleichbarkeit von Lösungen

Da die Kostenfunktionen von den Anforderungen abhängen, ist die Vergleichbarkeit von Lösungen nur dann gegeben, wenn sie zu den gleichen Anforderungen gehören. Zwei Lösungen zu verschiedenen Anforderungen, die beide zum gleichen Kostenwert führen, sind selten wirklich gleich gut nach der Einschätzung des Betrachters. Auch mag die Lösung für eine Anforderung den Wert 0, diejenige für eine andere den Wert 2 erreichen. Trotzdem kann die zweite Lösung eine (nach Expertenmeinung) sehr gute, die erste eine mittelmäßige sein.[9] Die Vergleichbarkeit von Lösungen, wie sie z. B. bei fallbasierten Systemen oder in der Fuzzy-Logic vorkommt, ist also nicht gegeben. Für den Anwender hat der Kostenwert überhaupt keine Relevanz, er ist nur intern für die Lösungssuche wichtig.

4.4. Lösungsansatz

Der Grundalgorithmus Stepping läuft rekursiv ab. Er wird zu Anfang mit Solve(0, Anfangsbelegungen, NULL) aufgerufen. NULL bedeutet dabei: es gibt noch keine gefundene Lösung, also auch keine beste Lösung. VB enthält die Belegungen sämtlicher Variablen des Constraintsnetzes (meistens Intervalle).

```
Lösung Solve(int Schritt, ConstraintNetz VB, Lösung besteLösung)
```

- kopiere VB nach tVB
- während es noch propagierbare Constraints in tVB gibt
  - propagiere die Constraints in tVB
- falls das Netz tVB nicht erfüllbar ist
  - beende diese Prozedur und gebe besteLösung als Ergebnis zurück
- falls sämtliche Variablen in tVB nur noch genau einen Wert besitzen
  - falls die Lösung von tVB besser ist als besteLösung
    - ▷ beende diese Prozedur und gebe Lösung von tVB als Ergebnis zurück
  - sonst

[9] Die mittelmäßige sollte natürlich die bestmögliche sein; manchmal gibt es aber keine wirklich guten Lösungen.

```
        ▷ beende diese Prozedur und gebe besteLösung als Ergebnis
          zurück
```

- `wähle eine Variable aus tVB aus`
- `setze die Variable tempLösung auf besteLösung`
- `für alle noch möglichen Werte der Variable`
  - `rufe Solve(Schritt+1, tVB, besteLösung auf) und setze tempLösung auf den Rückgabewert`
- `beende Prozedur und gebe tempLösung als Ergebnis zurück`

Im Idealfall ist bei allen Constraintproblemen die Propagierung ausreichend, im Normalfall ist die Suche nicht nur nötig, sondern auch laufzeitbestimmend. Deswegen werden in den folgenden Abschnitten Propagierung und Suche getrennt betrachtet.

Gemein ist beiden Teilen jedoch, daß die einzelnen Constraints und die einzelnen Variablen nicht gleich wichtig sind für das Constraintnetz. Alle Constraints müssen eingehalten werden, aber einige Constraints wirken auf viele Variablen, und einige Variablen wirken über die anliegenden Constraints besonders einschränkend auf einzelne Variablen. Die verschiedenen Variablen besitzen auch ein ungleich großen Wertebereich.

Wie in Kapitel 4.1 beschrieben, kann man das Constraintnetz in verschiedene Teilbereiche mit verschiedenen Hauptvariablen einteilen:

- Globale Daten: Kopplungsinformationen
- Achsenbezogene Daten: Typ des Zylinders/Hydromotors
- Phasenbezogene Daten: Funktionen für Position, Kraft, etc.

Diese Teilbereiche sind nicht unabhängig: es bestehen neben den inneren Constraints eines Objektes auch äußere Constraints zu anderen Objekten. Dazu kommen die Beziehungen zwischen den Phasen.

Die Anzahl der Möglichkeiten bei jedem Teilbereich wird nun abgeschätzt.

Die Anzahl der möglichen Kopplungsbäume ohne Einbeziehung der Art der Kopplungen liegt in $\Omega(n!)$, sie ist also größer oder gleich $n!$. Dabei ist n die Anzahl der Achsen. Zu beachten ist: bei paralleler Kopplung ist die Reihenfolge der beiden

Unter-Meta-Achsen uninteressant. Tatsächlich kann parallele Kopplung nicht in 2. oder tieferer Ebene vorkommen. Bei sequentieller und serieller Kopplung ist die Reihenfolge entscheidend, die linke Teil-Achse soll dann vor der rechten stehen. Somit liegt die Anzahl der Möglichkeiten der Kopplungen zwischen $2^(n-1)$ und 3^{n-1}. Damit gibt es für 5 Achsen mehr als $60 \cdot 16 = 480$ Möglichkeiten. Tatsächlich ist die Zahl weitaus höher. Im einfachen Beispiel aus Kapitel 3 gibt es nur 3 Achsen. Dann ist die Auswahl geringer, es gibt 61 Möglichkeiten (per Hand errechenbar).

Die Auswahl eines Baumes oder Teilbaumes hat Auswirkungen auf die Achsen, also auch die Typenauswahl, und damit indirekt auf die Phasen.

Für die Typenauswahl Zylinder / Hydromotor / Pumpe kann man die Anzahl der Möglichkeiten durch die Multiplikation von

- Anzahl Pumpen in der Datenbank
- Anzahl Zylinder in der Datenbank hoch Anzahl der benötigten Zylinder
- Anzahl Hydromotoren in der Datenbank hoch Anzahl der benötigten Hydromotoren

berechnen. Schon bei 5 Pumpen und 10 Zylindern in der Datenbank und nur 5 Achsen gibt es also $5 \cdot 10^5 = 500000$ Möglichkeiten. Durch die Auswahl eines Zylinders werden sämtliche Phasen der Achse ebenfalls eingeschränkt. Im Beispiel gab es drei Achsen, in den Beispielsdatenbanken gibt es 3 Pumpen und 4 Zylinder. Damit gibt es in diesem Bereich $3 \cdot 4^3 = 192$ Möglichkeiten.

Die Anzahl der möglichen Funktionen für die Variablen Kraft, Position, Leistung etc. der Phasen ist unendlich. Außerdem sind diese Funktionen nur schwer einschränkbar.

Bei den Zeitpunkten muß zunächst einmal darauf geachtet werden, daß die Datenstruktur mit den Zeitpunkten und den Verbindungen zwischen den Zeitpunkten nicht mit den für die Constraint-Verarbeitung definierten Knoten und Kanten übereinstimmt. Die Verbindungen tragen selbst Informationen, nämlich den zeitlichen Abstand zwischen den anliegenden Zeitpunkten. Somit sind die Verbindungen genauso wie die Zeitpunkte in Wirklichkeit als Knoten eines Constraintnetzwerkes zu sehen, in dem 3-Constraints verwendet werden, die jeweils 2 Zeitpunkte und eine Verbindung betreffen. Für die weitere Verarbeitung werden weiterhin die Begriffe Knoten und Kanten gebraucht. Die Verwendung der Knoten und Kanten erweist sich als günstig. Die Länge der Kanten und die Länge der Phasen beeinflussen sich gegenseitig. Die Anzahl der Möglichkeiten für eine

Kantenlänge ist theoretisch unendlich, praktisch von der Genauigkeit und der als ∞ gebrauchten Länge abhängig. Diese ist jedoch so groß, daß man faktisch von Unendlichkeit sprechen kann.

Vor der Beschreibung der Constraintpropagierung und der Suche soll kurz die Frage erörtert werden, warum keine lokalen Suchverfahren benutzt werden. Bei diesen wird zunächst eine Lösung geraten und dann nach Verbesserungen gesucht. Das Raten erweist sich jedoch aufgrund der Komplexität als sehr schwierig. Vor allem die durch Polynom-Stücke beschriebenen Funktionen können nur schwer „erraten“ werden. Somit ist die Gefahr zu groß, eine nicht ausreichende Lösung angeboten zu bekommen.

Die Berechnung der genauen Polynom-Stücke hat sich allerdings im Verlaufe der Arbeit als nicht notwendig erwiesen. Stattdessen wird zunächst mit annähernden Daten gearbeitet, die das Verhalten einer Achse während der Phase beschreiben. Statt einer Kette von Polynomstücke bzw. unendlich vielen Polynomstück-Möglichkeiten werden nur bestimmte Charakteristiken benutzt, beispielsweise den Volumenstrom, der innerhalb der Phase fließt, oder die maximale Geschwindigkeit während der Phase. Die exakte Berechnung der Phasen findet nach der eigentlichen Berechnung statt.

4.5. Vorverarbeitung

Die erste Aufgabe der Vorverarbeitung ist der Ausschluß nicht-konsistenter Daten, darunter der Ausschluß von Anforderungen, die keine Lösung ermöglichen. Der Ausschluß nicht-konsistenter Daten wird größtenteils schon während der Eingabe vorgenommen. Beispielsweise darf die als Endpunkt einer Phase geforderte Position nicht größer sein als die maximale Ausfahrposition, die in der Achse angegeben werden kann. Andere auszuschließende Daten sind beispielsweise geforderte Ausfahrlängen, die kein Zylinder in der Datenbank erreichen kann. Widersprüche können sich auch an anderer Stelle ergeben: die Länge der Phase, die Beschleunigung und die Geschwindigkeit können sich widersprechen. Der Widerspruch sollte günstigerweise vor der eigentlichen Berechnung erkannt werden. Dann wird die Berechnung gar nicht erst gestartet. Während der Verarbeitung würden diese Widersprüche erst spät erkannt werden, also nach langer Laufzeit. Die Vorverarbeitungsschritte hingegen können in linearer Zeit zur Phasenzahl ausgeführt werden.

In der Vorverarbeitung müssen Grenzen für verschiedene Parameter berechnet werden. So kann für jede Phase eine Mindestlänge errechnet werden, die vom zurückzulegenden Weg und der maximalen Ausfahrgeschwindigkeit für Zylinder

abhängt. Dazu können auch die Abstände zwischen dem Anfang und einem Ereignis (bzw. einem Ereignis und dem Ende) eingeschätzt werden.

Nicht möglich sind Kreise innerhalb des Zeitpunkt-Geflechts, wenn für diese die Länge 0 ausgeschlossen ist. Dieses passiert, wenn alle Kreise nur negative oder nur positive Längen (mit Ausschluß von 0) enthalten — dann müßten einzelne Zeitpunkte vor sich selbst liegen. Benutzt werden kann für diesen Zweck der Algorithmus von Floyd (siehe [16, S.541]) zur Berechnung aller kürzesten Wege in einem Graphen. Dieser Algorithmus hat ungünstigerweise eine Laufzeit von V^3, wenn V die Anzahl der Zeitpunkte ist. Außerdem erweist er sich auch für die Propagierungsphase unter Umständen als günstig. Deswegen wird er nicht während der Vorverarbeitung genutzt, sondern unter bestimmten Umständen bei Schritten der Constraintverarbeitung, auf jeden Fall aber beim 1. Schritt.

Interessant ist auch ein Test auf den Zusammenhang des Zeitpunkt-Graphen. Es gilt: jeder Zeitpunkt einer Phase ist mit jedem Zeitpunkt jeder anderen Phase der Achse zusammenhängend. Denn es wird für jede Phase in den Graphen eingefügt:

- eine Kante vom Beginn zum Ende. Der Beginn muß vor dem Ende liegen oder mit diesem übereinstimmen. Der Wert der Kante entspricht der Länge der Phase (meistens als Interval angegeben). Wenn diese nicht bekannt ist, werden 0 und ∞ als Grenzen gesetzt.
- eine Kante mit Wert $[0; \infty]$ vom Ende der Phase zum Anfang der nächsten Phase der Achse (die nächste Phase darf frühestens beginnen, wenn die Phase endet). Manchmal kann der Wert eingeschränkt werden. Wenn das Constraint-Makro „P1 direkt vor P2“ auftritt, kann in der Kante von P1 zu P2 $[0; 0]$ eingetragen werden.
- für jeden Ereignis-Zeitpunkt: eine Kante vom Anfang der Phase zum Ereigniszeitpunkt und vom Ereigniszeitpunkt zum Ende. Bei beiden darf der Wert nicht nicht größer sein als die Länge der Phase.

Daraus folgt der Zusammenhang zwischen allen Zeitpunkten einer Achse.

Zusätzlich werden in das Netzwerk aus Zeitpunkten und Kanten natürlich die Kanten eingetragen, die explizit durch den Benutzer angegeben wurden.

Wenn keine Verbindung zwischen den Achsen zweier Phasen besteht, dann sind die Achsen vollkommen unabhängig und können auch so behandelt werden. Der fehlende Zusammenhang deutet auf zwei verschiedene Schaltkreise hin. Dennoch empfiehlt es sich, einen Schaltkreis zu nehmen, um eine Pumpe und damit Kosten zu sparen. Die zusammenhängenden Phasen und damit die zusammenhängenden

Achsen können in der Zeit $V \cdot logV$ errechnet werden. Für den Zusammenhang werden die Kanten als nicht gerichtet gewertet. Im Normalfall wird der gesamte Graph zusammenhängend sein. Im Einzelfällen erhält man mehrere Zusammenhangskomponenten, auf die man die einzelnen Achsen aufteilen kann. Kopplungsfindung muß nur noch für die Achsen jeder Zusammenhangskomponente geschehen. Die Meta-Achsen, die jeweils alle Achsen einer Zusammenhangskomponente enthalten, werden parallel zueinander gestellt.

4.6. Propagierung

Bei der Constraint-Propagierung wird auf lokale Propagierung gesetzt. Das heißt: die jeweiligen Variablen geben ihre Änderungen an ihre Nachbarn weiter. Das geschieht solange, bis sich keine Änderung mehr ergibt. Dabei gibt es Variablen und Werte, die größere Auswirkungen haben. Beispielsweise wirkt sich der Ausschluß einer Baugröße für einen Zylinder auf sehr viele Werte und teilweise sehr stark auf diese aus. Wenn der besagte Zylindertyp einige Parameter besitzt, die eine untere oder obere Grenze für einen Wert bilden, beispielsweise für die Kolbenfläche, ändern sich z. B. die Möglichkeiten für den Druck (der abhängig von der Fläche ist), somit bestehen Auswirkungen auf die Kopplungsmöglichkeiten, den maximalen Systemdruck, etc.

Man muß zwischen Auswirkungen während der Propagation und Auswirkungen für die nächste Auswahl unterscheiden: bei einer höheren unteren Grenze für den maximalen Systemdruck ändert sich die betreffende Kosten-Teilfunktion zu Ungunsten, so daß bei der nächsten Variablenauswahl im Schritt 3 die Wertezahl vermutlich stark beschnitten werden kann. Dabei gilt: zunächst sollten Werte genommen werden, die sich direkt auf die Propagierung auswirken. Werte, die nur oder fast nur die Auswahl beschränken, sollten erst genommen werden, wenn sich die Notwendigkeit einer Auswahl abzeichnet (also Schritt 3 des vereinfachten Algorithmus). Somit ist die Reihenfolge der Propagierung:

1. propagiere Werte, die viele andere Werte möglichst stark einschränken
2. propagiere Werte, die zu weiterer Propagierung führen
3. propagiere Werte, die sich nur oder fast nur auf die Auswahl auswirken

Die Constraints werden als Funktionen eingearbeitet.

Die erste, einfachste Lösung der Einarbeitung eine Prozedur für jeden Knoten des Constraintnetzes. Diese wird immer dann aufgerufen, wenn sich der Wert

des Knotens ändert. Sie setzt die Variable dann auf den neuen Wert und weist — wieder per Funktionsaufruf — den anliegenden Knoten neue Werte zu nach den durch die Constraints definierten Vorschriften. Diese Methode funktioniert rekursiv. Bei Constraints A, B, C und Aufruf der A verändernden Funktion mag diese die Veränderung von B aufrufen, diese wiederum die Funktion für C. Letztere kann wieder die Funktion für A aufrufen. Somit kann eine rekursive Aufrufschleife entstehen, bei der langsam die Werte A, B und C eingeschränkt werden. Meistens hat die Änderung eines Wertes jedoch mehrere andere Wertveränderungen zur Folge. Wenn A nach der Veränderung von B die Änderung von D aufruft, so muß diese so lange warten, bis die Änderungsschleife A,B,C endlich durchgelaufen ist. Diese lokale Propagierung des Unternetzes aus A, B, C ist nicht unbedingt unerwünscht, denn die gemachte Einschränkung von A, B und C muß ja zwangsläufig später während der Propagierung erfolgen. Jedoch kann es sinnvoller sein, D vorher aufzurufen, weil die Änderung von D A, B und C vielleicht mehr beschränkt als die ganze Propagierungsschleife. Desweiteren kommt es bei den Aufrufen irgendwann zu Speicherproblemen, wenn die Veränderungsfunktionen sich immer wieder aufrufen.[10]. Mindestens kommt es zu langen Laufzeiten.

Aufgrund dessen wird die Funktion aufgeteilt. Die Funktion change_A(new_value) verändert den Wert des Knotens A. Dabei wird nicht einfach der new_value zugewiesen, sondern im Normalfall der alte mit dem neuen Wert geschnitten. Beispielsweise kann als alter Wert ein Wertinterval von $[-1;3]$ stehen, als new_value wird $[1;5]$ zugewiesen. Dann darf der alte Wertebereich natürlich nicht verlassen werden, weil dadurch Constraints verletzt würden. Stattdessen wird der Schnitt $[1;3]$ als neuer Wert des Knotens genommen. Es muß keine Änderung vorgenommen werden, wenn das übermittelte Werteinterval das alte Interval beinhaltet. Wenn es eine wirkliche Änderung ist, dann wird der Knoten als geändert markiert und ein Verweis auf den Knoten in die Liste der zu propagierenden Knoten aufgenommen.

Die Funktion propagate_A propagiert den Wert des Knotens A. Er gibt die Änderung somit weiter an die Nachbarknoten, indem nach der Vorschrift des Constraints change_B_i für alle anliegenden Knoten B_i aufgerufen wird.

Eine globale Funktion kümmert sich um die Liste der zu propagierenden Knoten und arbeitet diese solange ab, bis keine veränderten Knoten mehr vorliegen und die Propagierung somit zur Ruhe kommt.

Der Vorteil dieser Aufteilung: bis ein Wert propagiert wird, kann er schon durch mehrere andere Knotenpropagierungen verändert worden sein. Beim obrigen Beispiel hätte A zunächst B, dann aber schon D verändert. Danach wären die

[10] Je nach Größe können 100e von Aufrufen für A, B und C kommen

Funktionen propagate_B, propagate_C und propagate_D aufgerufen worden. propagate_C und propagate_D hätten A verändert. Im alten Verlauf wäre A immer wieder von C verändert worden.

Eine große Verbesserung bedeutet zusätzlich die Einführung von Prioritäten in die Liste der zu verarbeitetenden Propagierungen. So kann man Knoten, die sehr stark einschränkend auf das Constraintnetz wirken, zuerst propagieren. Dabei sollte es egal sein, ob diese starke Einschränkung von der Art des Knotens abhängt — beispielsweise wirken sich Kopplungsinformationen immer sehr stark auf das Netz aus —, oder ob er selbst eine starke Änderung erfahren hat, die weitergegeben werden kann.

Bei äußeren Constraints muß man immer im Auge haben, welche Objekte und welche Variablen der Objekte gemeint sind.

Ein nebensächliches Problem ergibt sich dadurch, daß viele Variablen im Fließkommabereich liegen. Dieses läßt sich dadurch umgehen, daß die Variablen in einen ganzzahligen Bereich gebracht werden. Z. B. können 0,5 m in 50 cm umgewandelt werden. Teilweise wird jedoch hohe Genauigkeit und damit hohe Anzahl von Stellen gefordert. Rundungsfehler müssen während jeglicher Berechnung bedacht werden, also wird eine Toleranz gegeben, die für verschiedene Variablen auch verschieden hoch ist.

4.7. Suche

Die Suche muß drei Ansprüchen genügen. Sie

- muß schnell sein
- muß nachvollziehbar sein — wird das Programm zweimal auf das gleiche Problem angesetzt mit exakt der gleichen Optimierungsgewichtung, so muß die gleiche Lösung gefunden werden
- darf keine vielversprechenden Lösungsansätze auslassen

Punkt 2 und 3 bedeuten zusammengenommen, daß zufallsbedingtes Verhalten möglichst ausgeschlossen werden soll.

Die erste Aufgabe der Suche sollte bei Einbeziehung des Branch-and-Bound-Gedankens die Einschränkung des Lösungsraums sein, also das Beschneiden des

Baums. Dieses kann nicht durch die Propagierung geschehen, sondern nur durch die Suche.

Für Branch-and-Bound muß es möglich sein, eine untere Grenze zu berechnen für den Kostenwert aller Lösungen in dem entsprechenden Zweig des Suchbaums. Wenn dieser Minimalwert schlechter, also höher ist als der Wert für die schon gefundene Lösung, dann ist die Suche in dem Teilbereich sinnlos.

Für vier Teilkostenfunktionen ist es möglich, diese Grenze zu berechnen. Beispielsweise kann man die Minimierung der Geldkosten durchaus vornehmen. Für jede Komponente wird einfach die günstigste noch anwendbare Version genommen. Schwierig sind die Berechnung der Varianz der Pumpenleistung und die Berechnung des Leistungswirkungsgrades. Diese können nur schwierig abgeschätzt werden, wenn die Kopplungsinformationen noch nicht komplett sind. Beide Teilkosten können 0 nicht unterschreiten. Jedoch werden zu viele aussichtslose Lösungsteilbäume durchschritten. Als Kompromiß wird die untere Grenze mit den schon vorhandenen Mitteln abgeschätzt und jeweils die schlechteste (in Bezug auf Leistung) Kopplung angenommen.

Die Berechnung der Teilkostenfunktionen ist etwas zeitintensiv. Dagegen gibt es mehrere Mittel:

1. bei jeder Änderung von Variablen werden die Teilkosten verändert, falls überhaupt eine Auswirkung durch die Variablenänderung vorliegt
2. die Berechnungsfunktion wird nur jeden i. Schritt gestartet
3. die Berechnungsfunktion wird jeden Schritt gestartet, jedoch nur, wenn eine Teilfunktion vorher markiert wurde.

Möglichkeit 1 ist in der Praxis zu aufwendig, weil bei fast jeder Änderung eine Neuberechnung starten müßte oder die Änderungen explizit angegeben werden müssen. Letzteres ist in vielen Fällen gar nicht möglich. Der 2. Lösungsansatz ist unpraktisch, weil sich bei nur 3 Schritten Abstand zwischen den Berechnungen und nur 5 Verzweigungen pro Schritt $5^3 = 125$ Verzweigungen ergeben; wenn im 1. Schritt schon 4 von 5 entfallen, werden $4 \cdot 5^2 = 100$ Verzweigungen umsonst bearbeitet. Die eingesparte Laufzeit wird dadurch leicht wieder ausgegeben, eher sogar überschritten.

Die dritten Möglichkeit ist eine Variante der ersten: die Änderungen werden nicht direkt vorgenommen, sondern erst am Ende der Constraint-Propagierung. Das geschieht nur, wenn sich die untere Grenze der Kostenfunktion wahrscheinlich verändert hat. Änderungen der oberen Grenze sind beispielsweise uninteressant:

wenn für einen Zylinder der teuerste Typ ausgeschlossen wird, dann hat das für die untere Grenze und damit die Berechnung keine Relevanz. Durch ihr Laufzeitsparendes Verhalten bietet sich diese Möglichkeit an.

Genauso wichtig ist die Auswahl der zu betrachtenden Variable und der Werte für die Suchfunktion. Auch hier werden Prioritäten gesetzt. Dabei werden drei Stufen unterschieden. Die Kopplungsinformationen sind am wichtigsten und schränken die restlichen Möglichkeiten am meisten ein. Außerdem gibt es für eine einzelne Kopplung sehr wenige Möglichkeiten. Ausgewählt werden zunächst Kopplungen, bei denen sequentielle Lösungen möglich sind. Diese sind die günstigsten. Danach folgen restliche Kopplungen.[11] Die zweite Stufe ist die Parametrierung der Komponenten. Danach folgen restliche Variablen. Die genaue Reihenfolge soll aus Zeitgründen nicht erläutert werden.

Die Auswahl der Reihenfolge der Werte hängt von der jeweiligen Variable ab. Bei den Kopplungen werden in der Reihenfolge sequentielle, serielle und parallele Kopplung abgearbeitet. Somit ist die Reihenfolge von der günstigsten zur ungünstigsten Art gehalten, und unter Umständen können dann die ungünstigen ausgelassen werden (wenn die günstigeren schon bessere Lösungen ermöglichen).

Auch bei der Parametrierung und den weiteren Variablen wird immer mit der günstigsten Komponente angefangen. Dafür wird die untere Kostenschranke berechnet, die entsteht, wenn der jeweilige Wert benutzt wird. Die insofern günstigste Belegung wird als erstes untersucht, dann folgen in aufsteigender Reihenfolge die weniger günstigen, mit Ausnahme derer, deren untere Kostenschranke schlechter ist als die bis dato gefundene Lösung.

4.8. Einfache Techniken zur Verbesserung von Propagierung und Suche

4.8.1. Grouping

Sinnvoll ist die Verschmelzung von Zeitpunkten. Dafür gibt es eine Abstraktionsmethode für Graphen, wie sie u.a. [15] verwendet: das Grouping. Hierbei werden zusammengehörende Knoten eines Graphen zusammengefaßt als Meta-Knoten. Grouping bringt sehr viel Flexibiltät, u.a. die Möglichkeit, auf mehrere

[11] Wenn sequentiell nicht möglich ist, dann müssen seriell und parallel möglich sein. Wäre das nicht der Fall, gäbe es nur eine mögliche Kopplung, und damit gar nicht die Notwendigkeit einer Auswahl für diese Kopplung

Knoten verteilte Informationen zusammenzufassen. Informationsverlust gibt es an dieser Stelle nicht.

Dafür muß natürlich die Zusammengehörigkeit zunächst definiert werden. Die Verschmelzung soll genau dann und nur dann passieren, wenn die Zeitpunkte gleichzeitig sein müssen. Das gilt bei folgenden Bedingungen:

- Die Zeitpunkte bilden einen Kreis im Graphen (wenn dieser ungerichtet betrachtet wird).
- Wenn man diesen Kreis in einer Richtung betrachtet — also gegebenenfalls das invertierte Interval für alle Kanten mit falscher Richtung benutzt —, dann dürfen nur positive oder nur negative Werte in den Intervallen vorkommen, zusätzlich in jeder Kante die 0. Also sind alle Intervale (gegebenfalls nach Richtungsänderung) vom Typ $[-a; 0]$ oder alle vom Typ $[0; a]$, mit $a > 0$. Wenn die 0 in nur einem Interval fehlt, dann kann es überhaupt keine Lösung geben für diesen Bereich des Suchbaums, die Berechnung kann sofort abgebrochen werden. In diesem Fall müßte ein Zeitpunkt vor oder nach sich selbst liegen.

Zusätzlich kann es auch zu Kantenverschmelzungen kommen. Es muß nur eine Kante zwischen zwei Zeitpunkten existieren, die Richtung ist dabei egal. Sollte es zwei geben, werden sie miteinander verschmolzen. Wenn zwei Kanten von A zu B existieren mit den Werten $[-3; 3]$ und $[1; 5]$, dann werden sie zu einer Kante verschmolzen mit dem Wert $[1; 3]$. Falls die Kanten unterschiedliche Richtungen haben, muß für den Schnitt natürlich ein Interval invertiert werden.

Auch bei der Verschmelzung der Zeitpunkte kommt es zu einer Kantenverschmelzung: jede Kanten, die einen der Zeitpunkte berührt, wird in den neuen Meta-Zeitpunkt übernommen. Wenn die Punkte A und B verschmolzen werden und beide eine Verbindung zu C haben, müssen die Verbindungen AC und BC ebenso verschmolzen werden. Zusätzlich können Kanten zwischen zu verschmelzenden Zeitpunkten gestrichen werden, also eine mögliche Verbindung AB.

Die Verschmelzung wirkt sich natürlich positiv auf die weitere Laufzeit aus, immerhin wird es mindestens einen Zeitpunkt oder eine Kante weniger geben. Es können sogar mehrere Zeitpunkte oder Achsen gleichzeitig verschmolzen werden. Im Beispiel aus 3 kann es dazu kommen, daß zwischen E1 und E2 eine Kante gelegt wird mit dem Wert $[0; a]$, mit a positiv. Dann entsteht ein Kreis zwischen „P13 Anfang“, „P22 Ende“, E1 und E2. In dieser Richtung sind alle Kanten positiv (einschließlich 0). Dann können diese vier Zeitpunkte verschmolzen werden. Die Kanten zwischen den ursprünglichen Zeitpunkten entfallen dadurch. Verschmolzen werden muß keine Kante. Der neue Meta-Zeitpunkt erbt jedoch alle nicht entfallenen Kanten.

4.9. Nachberechnung

Wie auf S. 4.4 beschrieben, werden in dem eigentlichen Algorithmus keine genauen Profile in Form der verlangten Polynomstück-Ketten errechnet. Stattdessen werden bestimmte Charakteristiken genommen, die diese Verläufe genügend genau dokumentieren (Minima, Maxima, Volumenstrom, ...). In der Nachberechnung werden diese dann u.a. mit Hilfe der Gangarten in richtige Profile umgesetzt. Dabei muß das Ergebnis des Constraintnetzwerkes noch vorliegen, denn die Profile sind aufgrund der Koppelungen nicht unabhängig. Mit den vorliegenden Ergebnissen und den Kopplungen, die ja durch den Kopplungsbaum gegeben sind, können die Profile hinreichend genau errechnet werden.

Eine kleine Anmerkung scheint zum Thema Nachoptimierung angebracht: eine Nachoptimierung findet tatsächlich nicht statt. Während der Propagierung und der Suche wird möglichst die beste Lösung gesucht. Wenn die Optimierung gelingen würde, so wäre bewiesen, daß die gefundene Lösung nicht die beste war. Tatsächlich ist dieser Fall möglich, da die Lösungsmenge nicht vollständig, sondern nur mit Hilfe von Heuristiken durchsucht wird. Dennoch brachten Experimente mit verschiedenen Nachoptimierungen keinen Erfolg.

Kapitel 5

Resultate

5.1. Bemerkungen zur Implementierung

5.1.1. Unterschiede zwischen beschriebenen und genutzten Verfahren

Die benutzten Algorithmen wurden mit nur kleinen Änderungen im Programm umgesetzt. Diese mußten zum größten Teil wegen der benutzten Datenstrukturen geschehen, wirken sich aber nicht in großem Maße auf das Laufzeit- und Berechnungsverhalten aus.

5.1.2. Anbindung an andere Programme

Das erarbeitete Programm läuft momentan noch unabhängig von anderen Programmen. Die Ergebnisse jedoch können direkt von dem in [7] beschriebenen Programm eingelesen und zu einem Schaltkreis verarbeitet werden. Denkbar ist in Zukunft die direkte Einbindung beider Programme in den hydraulischen Simulator FluidSim. Allen drei Programmen ist u.a. das Speicherformat ASE (siehe [19]) gemein, so daß der Datenaustausch sehr einfach ist (von FluidScheduler zu dem Programm zu [7], von dort zu FluidSim). Mit dem Einbau der in [13]

vorgestellten Algorithmen zur Reparatur von Schaltkreisen, die ein ungewolltes Verhalten zeigen, kann ein mächtiges Werkzeug zur Konstruktion, Wartung und Verbesserung von hydraulischen Anlagen geschaffen werden.

Denkbar sind auch Varianten für andere Bereiche wie Pneumatik oder Elektrotechnik. Das gilt vor allen Dingen für die Pneumatik, denn für diesen Bereich ist schon eine Version des Simulators FluidSim vorhanden. Außerdem ist die Veränderung der Hydraulik-Constraints zu Pneumatik-Constraints wegen ähnlicher physikalischer Eigenschaften sehr einfach.

5.2. Zusammenfassung und kritischer Rückblick auf den Verlauf der Arbeit

Die beiden Hauptziele dieser Arbeit war die zeitliche Einordnung von Phasen und die Eingliederung von Achsen in einen Strukturbaum. Aus diesen beiden Zielen und der Form der Verarbeitung folgten als Nebenziele die Ermittlung von Sollprofilen und die Parametrierung einiger Komponenten des zu bildenden Schaltplans.

Die Anforderungen an eine hydraulische Anlage sollten in möglichst einfacher und umfassender Weise als Eingabe für das Problem P dienen. Als Besonderheit gegenüber ähnlichen Problemen mußten Ereignis-Zeitpunkte in die Beziehungen zwischen zwei Phasen einbezogen werden.

Nach der Anforderungsaufnahme sollte sich eine Verarbeitung anschließen, die neben physikalischen Zusammenhängen auch in Constraint-Form gefaßte Heuristiken einbeziehen würde. Die Suche nach der besten Lösung führte aufgrund verschiedener Zielvorstellungen zur Multi-Kriterien-Analyse, genauer zu MODMs.

Generell konnte in dieser Arbeit und mit Hilfe der Ergebnisse aus [7] gezeigt werden, daß ein computergesteuertes Design hydraulischer Anlagen innerhalb einer vernünftigen Laufzeit möglich ist. Es besteht die Hoffnung, daß sich aus den Erkenntnissen eine industrielle Nutzung ergeben wird.

Das Programm FluidScheduler kann nur mit Hilfe des zu [7] entwickelten Programms (oder ähnlichen, noch zu implementierenden Werkzeugen) zu sofortiger sinnvoller Nutzung führen. Allein mit den Ergebnissen des FluidScheduler müßte ein Experte die weiteren Schritte zum Bau des Schaltplans in manueller Weise finden. Jedoch ergibt sich ein Bruch zwischen der Berechnung in FluidScheduler und der darauf folgenden Berechnung des Schaltplans, wie es häufig ist bei der Partitionierung von Programmen. Es könnte sich u. U. als günstig erweisen, diese beiden Teile mehr miteinander zu verschmelzen. Jedoch sind die Fallbasierten

Methoden in [7] und die Constraint-basierten Methoden in dieser Arbeit nicht auf einfache Weise zu verschmelzen.

5.3. Perspektiven

Zum Schluß sollen noch einige Möglichkeiten zur Erweiterung aufgezeigt werden, deren Implementierung den Rahmen der Arbeit gesprengt hätten. Sie können als Anhaltspunkte für weitere Arbeiten oder als Grundidee für tiefergehende Bearbeitung einiger Teilbereiche dieser Arbeit dienen.

5.3.1. Weiche Constraints

Momentan werden sämtliche Constraints als harte Constraints behandelt, d.h., sie müssen erfüllt werden. Im Zweifelsfall kann dies dazu führen, daß keine Lösung möglich ist. In der Realität gibt es nicht nur harte Constraints (Mußbedingungen, Mindestforderungen), sondern auch weiche Constraints, die man als Möglichst-Bedingungen oder Wünsche bezeichnen könnte. Diese Erweiterung bringt also eine weitaus grössere Anforderungsbandbreite und eine größere Realitätsnähe.

Der Einbau kann mit Hilfe der Erweiterung der Kostenfunktion geschehen. Dann wird ein weiteres zu gewichtendes Kriterium eingeführt. Die Anzahl der erfüllten weichen Constraints wird als Teilkostenfunktion genommen (wieder mit einer Abbildung in den Bereich zwischen 0 und 1). Ansonsten kann die Einhaltung der Constraints oder möglichst vieler Constraints vor der eigentlichen Kostenfunktion gefordert werden, genauso, wie die harten Constraints vor der Kostenfunktion ausgewertet werden. Sie müssen eingehalten werden, bevor im verbleibenden Lösungsraum die Kostenfunktion betrachtet wird.

Eine ähnlich gelagerte Erweiterungsmöglichkeit wäre für einen Fall denkbar, der aufgrund von widersprüchlichen Constraints noch keine Lösung zulässt. Dann könnte man auf eine Ersatzlösung hinarbeiten, in der möglichst wenige harte Constraints verletzt werden.

5.3.2. ϵ-Abweichungen statt ϵ-Schläuche

Die möglichen Abweichungen werden momentan mit Hilfe von ϵ-Schläuchen dargestellt. In einem ϵ-Schlauch $\epsilon(f)$ um eine Funktion f liegen an einem Punkt x alle Werte y, für die gilt: $(1-\epsilon)\cdot f(x) \leq y \leq (1+\epsilon)\cdot f(x)$. Dabei ist $0 \leq \epsilon \leq 1$ (0

ist gleichbedeutend mit keiner Abweichung, 1 gleichbedeutend mit einer 100prozentigen Abweichung).

In der Praxis kann es günstig sein, die obere und untere Abweichung zu unterscheiden. Dann wählt man statt ϵ die obere Abweichung ϵ_1 und die untere ϵ_2. Dann gilt: In einem ϵ-Schlauch $\epsilon(f)$ um eine Funktion f liegen an einem Punkt x alle Werte y, für die gilt: $(1 - \epsilon_2) \cdot f(x) \leq y \leq (1 + \epsilon_1) \cdot f(x)$. Mit dem momentanen Programm muß man diese Tatsache damit umgehen, daß man für den ϵ-Schlauch immer die engere Schranke nimmt, also die näher am Sollwert liegende. Für spätere Programmversionen könnte man dieses verändern.

5.3.3. Erweiterung des Hydraulik-spezifischen Wissens

Wie auch im Simulator FluidSim ist in FluidScheduler nur begrenztes physikalisches Wissen eingeflossen. Zum Beispiel sind zeitliche Einschwingphasen der Komponenten nicht berücksichtigt. Es gilt: je mehr fachliches Wissen einfließt, desto genauer wird das Abbild der Realität. Auf der negativen Seite steigt jedoch die Komplexität der Programmschritte.

Weiterhin werden nur die Anschaffungskosten, nicht aber die laufenden Kosten der hydraulischen Anlage berechnet. Wartungskosten und -intervalle, Ausfallzeiten wegen Reparaturnotwendigkeit, und auch einfache Betriebskosten wie Strom oder Ölersatz werden nicht eingerechnet. Vielfach könnten diese Kosten mit statistischen Mitteln einbezogen werden.

Möglich wäre auch — zusätzlich zu den Angaben der Datenbank – die Einbeziehung individueller Zylinder-, Pumpen- und Hydromotorengrößen. Dabei müßte eine einfache Kostenfunktion vorliegen, mit der man ausgehend von der Baugröße die Anschaffungskosten der Komponenten berechnen kann. Im allgemeinen sind Zylinder in Massenfertigung billiger. Im Einzelfall kann ein spezifisch gebauter diese Schwäche dadurch ausmerzen, daß er besonders gute Werte hinsichtlich der anderen Kriterien besitzt.

5.3.4. Weitere Optionen zur Einschränkung der Lösung

Denkbar sind auch weitere Einschränkungen wie z. B. ein bei Bedarf angebbares Maximum für die Gesamtkosten. Diese Erweiterung wird sich nur auf die Anforderungseingabe und die Aufbereitung der Anforderungsdaten vor den Lösungsalgorithmen auswirken. Die Lösungsverfahren müssen nicht geändert werden. Es werden die Anfangsintervalle einiger Variablen kleiner sein, und damit wird die

Laufzeit eher sinken. Allerdings kann durch die Einschränkungen die normalerweise beste Lösung unbrauchbar werden, dadurch kann im Einzelfall die Laufzeit steigen.

5.3.5. Verwendbarkeit in anderen Bereichen

Neben der Verwendung im pneumatischen Bereich, der dem hydraulischen sehr ähnlich ist, kann über eine Verwendung in allen Bereichen nachgedacht werden, bei denen mehrere, verschachtelte Teilfunktionen existieren, deren Verhalten jeweils in einzelne Phasen aufgeteilt werden kann.

Literatur

[1] Allen, James F.: Maintaining Knowledge about Temporal Intervals; in: Communications of the ACM, November 1983, Volume 26, Number 11, S.832-843

[2] Allen, James F.: Toward a General Theory of Action and Time, in: Artificial Intelligence, 1984, 23(2), S. 123-154

[3] Beck, J. Christopher; Mark S. Fox: A Generic Framework for Constraint-Directed Search and Scheduling; in: AI Magazine, Winter 1998, S.101-130

[4] Dechter, Rina; Judea Pearl: Network-Based Heuristics for Constraint-Satisfaction Problems; in: Artificial Intelligence, 1988, Volume 34, S. 1-38

[5] Drexler, P. et al: Der Hydraulik Trainer Band 3: Projektierung und Konstruktion von Hydroanlagen, Mannesmann Rexroth GmbH, Lohr am Main/BRD, 1988

[6] Freuder, Eugene C.: Synthesizing Constraint Expressions; in: Communications of the ACM, New York, Band 21, 1978, S. 958-966

[7] Hoffmann, Marcus: Konzepte zur Automatisierung des Designprozesses fluidischer Systeme; Doktorarbeit, Universiät-GH Paderborn, Fachbereich Informatik, 1999

[8] Kumar, Vipin: Algorithms for Constrains-Satisfaction Problems: A Survey; in: AI Magazine, 1992, S. 32-44

[9] Müller-Merbach, Heiner: Operations Research; 1973, 3. Auflage, Verlag Franz Vahlen, München

[10] Oevel, Walter: Einführung in die Numerische Mathematik; Spektrum Akademischer Verlag, Heidelberg, 1996

[11] Paetzold, Wolf; Hemming, Werner: Hydraulik und Pneumatik, 1992, Verlag Christiani, Konstanz

[12] Pinedo, Michael: Scheduling - Theory, Algorithms, and Systems; 1995, Prentice Hall Inc., Eaglewood Cliffs, New Jersey 07632

[13] Schlotmann, Thomas: Formulierung und Verarbeitung von Ingeneurwissen zur Verbesserung hydraulischer Systeme; Diplomarbeit, 1998, Universität-GH Paderborn

[14] Schmitt, G.; H. Noltemeier, M.Widera: Inkrementelle temporale Constrainprogrammierung; in: KI, 1/1997, S.7-13

[15] Schulz, Andre: Graphenanalyse hydraulischer Schaltkreise zur Erkennung von hydraulischen Achsen und deren Kopplung; Diplomarbeit, 1997, Universität-GH Paderborn

[16] Sedgewick, Robert: Algorithmen in C++; 1992, 1. Auflage, Addison-Wesley, Bonn

[17] Stein, Benno: OFT - Objectives and concepts, Universität Informatik, Fachbereich Mathematik - Informatik, Bericht tri-ri-97-189, 1997

[18] Stein, Benno: Wissensbasierte Systeme; Vorlesungsfolien, Wintersemester 1998/99, Universität-GH Paderborn

[19] Stein, Benno; Hoffmann, Marcus; Thurau, Carsten: Das ASE-Format zur Beschreibung fluidischer Schaltkreise, Universität Paderborn, Fachbereich Informatik, 1998

[20] Stützle, Thomas: Lokale Suchverfahren für Constraint Satisfaction Probleme: die min conflicts Heuristik und Tabu Search; in: KI, 1/1997, S.14-20

[21] Vier, Elmar: Rechnergestützter Entwurf geregelter fluidischer Antriebe; Technical Report MSRT 3/96, Gerhart-Mercator-Universität-GH Duisburg

[22] Vier, Elmar; Stein, Benno; Hoffmann, Marcus: Strukturelle Formulierung von Anforderungen an hydrostatische Antriebe, Forschungsbericht 8/97,

[23] Vincke, Philipp: Multicriteria Decision-aid; John Wiley & Sons Ltd, Chichester / West Sussex, 1992

[24] Winston, Wayne: Operations Research, Duxbery Press, Belmont, California, 1994

[25] Zimmermann, Hans-Jürgen; Gutsche, Lothar: Multi-Criteria-Analyse, Springer Verlag, Berlin, 1991

Kapitel A

ASE-Format

Das ASE-Format (Art-Deco-System-Exchange-Format) ist in [19] beschrieben. Es dient bisher als Speicherungsformat für das Programm ArtDeco-FluidSim, ist jedoch auch für größere Aufgabenbereiche geeignet. In [19] wird auch die Nutzung des Programms für den hydraulischen Kontext erklärt. [7] erweitert diese Nutzung. Die nachfolgenden Beschreibungen der einzelnen Objektarten sind im gleichen Stil wie in [19] gehalten.

A.1. Globale Informationen

Die nötigen globalen Informationen für FluidScheduler werden als Objekte des Typs Global gespeichert. Diese sollen den Namen **GlobalDemands** für die Anforderungen und **GlobalSolution** für die Lösungsdaten tragen.

Folgende Einträge gehören zu **GlobalDemands**:

- Status: STATE (TYPE INT) - in welchem Zustand das Objekt abgespeichert wurde: 0 = leer, 1 = Anforderungsphase, 2 = Ergebnisphase
- Name: NAME (TYPE STRING)
- Einsatzbereich: FIELD (TYPE STRING)

- Maximaler Systemdruck: MAX_PRESSURE (TYPE FLOAT) - muß nicht gesetzt sein (dann entfällt der Wert)
- Beschreibung, Kommentare und Anmerkungen: DESCRIPTION (TYPE STRING)
- momentan gültige Optimierungsmethode: CRITERIA (TYPE LIST); in dieser Liste sollte für jedes Kriterium in einer Unterliste der Name und der Prozentwert (als Int-Wert zwischen 0 und 10) angegeben werden. Die Kriterien sind momentan:
  - "MIN_TIME": Gesamtzeit minimieren
  - "MAX_EFFECTIVITY": maximale Effektivität
  - "MIN_COST": niedrige Kosten
  - "MIN_VARIETY": Minimierung der Varianz der Pumpenleistung
  - "MIN_DISSIPATION": minimale Dissipation
  - "MIN_EPSILON_HOSE": enge ϵ-Schläuche

Folgende Einträge gehören zu **GlobalSolutions**:

- SYSTEM_PRESSURE (TYPE FLOAT): maximaler Systemdruck
- MAX_VOLUME (TYPE FLOAT): maximaler Volumenstrom
- PUMP_NAME (TYPE STRING): Name der ausgewählten Pumpe
- COST (TYPE FLOAT): Kosten der Anlage

A.2. ProtoAchsen

(OBJECTTYPE (TYPE STRING)(VALUE "PROTOAXIS"))

Bei den Anforderungen werden Achsen mit ihren zugehörigen Phasen beschrieben. Die Achsenbeschreibung ist jedoch gegenüber den in [19] eingeführten Beschreibungen stark eingeschränkt. Es sind nur folgende Informationen bekannt:

- der Name der Achse
- die Art der Arbeitselemente in der Achse (Zylinder / Hydromotor – nur eine Art kann in einer Achse vorkommen)

- die Anzahl dieser Arbeitselemente
- *spezifische Daten der Arbeitselemente, z. B. Durchmesser des Zylinders.*
- die Phasen mit ihren Beschreibungen

Im Gegensatz dazu ist bei den bisher bekannten Achsen die Liste sämtlicher Komponenten angegeben. Diese sind ebenfalls in der ASE-Datei genauer beschrieben. Aufgrund dessen kann kein Objekt vom Typ "AXIS" verwendet werden. Der Begriff "PROTOAXIS" beschreibt den Zustand der Achse zum Zeitpunkt, an dem sie für FluidScheduler interessant ist: eine Protoachse, also die Urform einer Achse, ein Gerippe, ein Achsenskelett.

NAME (TYPE STRING)

Dieser Slot enthält den Namen, den der Benutzer der Achse gegeben hat. Er ist i.a. nicht identisch mit dem Identifier der Achse.

WORKING_ELEMENT (TYPE STRING)

Hier wird der Typ des Arbeitselementes angegeben. Es gibt nur zwei Möglichkeiten: "compCylinder" für einen Zylinder, "compHydraulicMotor" für einen Hydromotor.

NUMBER (TYPE INT)

Die Anzahl der Arbeitselemente wird angegeben. Alle Arbeitselemente einer Phase müssen identisch sein und arbeiten.

PHASES (TYPE LIST)

Hier werden in einer Liste die Identifikatoren der einzelnen Phasen angegeben. Durch diese Liste ist auch die richtige Reihenfolge der Phasen festgelegt.

Weitere

Name	Typ	Art	Beschreibung
SOLUTION_NAME	STRING	L	Name der Lösungskomponente
COST	FLOAT	L	Kosten der Lösungkomponente
MAX_POS	FLOAT	ZA	maximale Ausfahrposition
FORCE	FLOAT	ZL	maximale Kraft
POS	FLOAT	ZL	maximale Ausfahrposition
FRICTION	FLOAT	ZL	Reibung
SQUARE_A	FLOAT	ZL	Kolbenfläche
SQUARE_B	FLOAT	ZL	Kolbenringfläche
SPEED	FLOAT	ZL	maximale Geschwindigkeit
ACCEL	FLOAT	ZL	maximale Beschleunigung
VOLUME	FLOAT	HL	Schluckvolumen
MAX_PRESSURE	FLOAT	HL	maximaler Druck
SQUARE	FLOAT	HL	maximale Winkelgeschwindigkeit
EFFECTIVITY	FLOAT	HL	Wirkungsgrad

Dabei besteht die Art aus Z für Zylinder oder H für Hydromotor, dazu A wie Anforderung oder L wie Lösung.

A.3. Eingabe: Phasen

(OBJECTTYPE (TYPE STRING)(VALUE “PHASE“))

AXIS (TYPE IDENTIFIER)

Dieser Slot enthält den Idenfifier der zugehörigen Protoachse.

NAME (TYPE STRING)

Dieser Slot enthält den Namen, den der Benutzer der Phase gegeben hat. Er ist i.a. nicht identisch mit dem Identifier der Achse.

TIME_MIN, TIME_MAX (TYPE FLOAT)

Diese beiden Slots enthalten die Mindest- bzw. Maximalzeit, die die Phase brauchen darf. Diese Zeiten müssen nicht angegeben werden, dann fällt der jeweilige Slot weg. Bei einer exakt vorgeschriebenen zeitlichen Länge müssen beide Werte gleich sein.

WALK (TYPE STRING)

Hier wird die Gangart angegeben.

SAFETY (TYPE FLOAT)

Gibt einen etwaigen Sicherheitszeitraum an, der nach der Phase eingehalten werden muß. In diesem Sicherheitszeitraum darf keine Bewegung des Arbeitselementes stattfinden, wenn es sich um einen Zylinder handelt. Bei einem Hydromotor darf stattdessen keine Geschwindigkeitsänderung passieren.

FORCE_BEGIN, FORCE_END (TYPE FLOAT); TORQUE_BEGIN, TORQUE_END (TYPE FLOAT)

Force_Begin und Force_End geben die Kräfte an, die zu Anfang und zu Ende der Phase auf den Zylinder wirken. Wenn diese unterschiedlich sind, wird im Verlauf der Phase einer lineare zum Ausfahren verlaufendes Kurve angenommen. Bei Hydromotoren wird statt der Kraft das Drehmoment angegeben.

ACTIONS (TYPE List)

Beschreibt die Aktionen, die in dieser Phase ablaufen. Eine Aktion besteht in der Änderung eines Slots: Geschwindigkeit, Position oder Kraft. Es können mehrere Aktionen angegeben werden.[1]

- Für jeden Slot, der verändert werden soll, wird angegeben:
  - der Slot, für den die Änderung gilt: “P“ (Position), “S“ (Geschwindigkeit) — für Hydromotoren nur “S“

[1] Achtung: diese können sich aber widersprechen.

- die Beschreibung, wie der Zahlenwert interpretiert werden muß
  - ▷ "diff": die Angabe ist als Differenz zwischen Alt- und Neuwert zu verstehen
  - ▷ "to": die Angabe ist der exakte Wert, der erreicht werden soll
- "relativ", falls eine prozentualer Wert gemeint ist; "absolut", wenn ein fester Wert gewählt wird
- die Zahl (FLOAT), die die Änderung beschreibt. Die Interpretation hängt von den beiden vorherigen Punkten ab. Wenn beispielsweise die Zahl 150 angegeben wurde, bedeutet das:
  - ▷ "diff" und "relativ": der Slot soll am Ende der Phase um 150% des Anfangswertes erhöht worden sein; er soll also 2 1/2 mal so groß werden
  - ▷ "diff" und "absolut": der Slot soll am Ende der Phase um 150 höher sein als zu Anfang
  - ▷ "to" und "relativ": der Slot soll am Ende der Phase um 150% des Anfangswertes erreichen; er soll also 1 1/2 mal so groß werden
  - ▷ "to" und "absolut": der Slot soll am Ende der Phase den Wert 150 erreichen
- die maximale Abweichung in Prozent (FLOAT), die der Endwert haben darf. Wenn also der Endwert 150 sein soll, aber eine maximale Abweichung 10 (%) angegeben ist, sind Endwerte zwischen 135 und 165 akzeptabel.

CONSTRAINTS (TYPE List)

Diese Beziehungen werden mit Hilfe von Zeitpunkten definiert. Als Zeitpunkte kommen in Frage:

- BEGIN
- END
- Ereigniszeitpunkte (siehe unten)

Zwischen 2 Zeitpunkten können folgende Relationen gelten: ">", ">=", "<", "<=", "==".

Ein jeder Constraint wird als Liste angegeben. In dieser steht:

- der Zeitpunkt in der Phase, also "BEGIN", "END" oder eine Liste mit der Beschreibung des Ereignis-Zeitpunktes
- der Delay-Wert dieses Zeitpunktes: wie im obrigen Beispiel angegeben, kann es sinnvoll sein, erst eine gewisse Anzahl von Sekunden nach dem Zeitpunkt einzusetzen, z. B. 2 Sekunden nach dem Beginn der Phase. Der Delay-Wert entspricht der Anzahl der Sekunden, kann also auch negativ sein. Wenn der Zeitpunkt direkt gemeint ist, wird als Delay-Wert einfach 0 angegeben.
- die Relation als String: ">", ">=", "<", "<=" oder "=="
- der Name der anderen beteiligten Phase
- der Zeitpunkt in der anderen beteiligten Phase, also wiederum "BEGIN", "END" oder eine Liste mit der Beschreibung des Ereignis-Zeitpunktes
- der Delay-Wert des Zeitpunktes der anderen beteiligten Achse

Statt der festen Zeitpunkte Begin und End kann man auch einen Ereignis-Zeitpunkt wählen, beispielsweise: "wenn die Kraft den Wert 5 unterschreitet". Ein Ereignis-Zeitpunkt besteht aus folgenden Angaben:

- dem Slot, beispielsweise "F" für Kraft
- einem Relationssymbol: ">", ">=", "<", "<=", "=="
- einem Float-Wert
- "relativ", falls ein prozentualer Wert gemeint ist; "absolut", wenn ein fester Wert gewählt wird

Beispiele:
(F ">" 5 "absolut"): wenn die Kraft den Wert 5 überschreitet
(F ">" 110 "relativ"): wenn die Kraft 110% ihres Startwertes überschreitet

Ein Ereignis-Zeitpunkt kann mit dem Beginn (oder Ende) der Phase übereinstimmen, wenn z. B. die Kraft schon vorher auf 6 stand. Ein Ereignis-Zeitpunkt kann theoretisch auch nie sein, wenn z. B. die Kraft immer unter 5 bleibt. **Während des Scheduling muß ein nicht stattfindendes Ereignis-Zeitpunkt erkannt werden!**

Anmerkung: zwischen 2 Phasen können mehrere Constraints gelten.

CONSTRAINT_MACROS (TYPE List)

Für einige einfache Zusammenhänge gibt es sinnvollere (weil weitaus kürzere) Beschreibungen der Zusammenhänge zwischen zwei Achsen. Dafür muß nur der Name der anderen Achse und die Relation zwischen den beiden angegeben werden:

- "before" / "after"
- "direct before" / "direct after"
- "during" / "holds"
- "equals" / "equals"

Die Angabe ist somit kürzer und auch verständlicher für den Benutzer. Die Constraint-Makros stehen für folgende Constraints (dabei soll Phase X die Phase sein, in der das Constraint-Makro steht, Y die Phase, deren Name angegeben wurde):

Makroname	Bedeutung
before	Ende (X) < Anfang(Y)
direct before	Ende (X) = Anfang(Y)
after	Anfang(X) > Ende (Y)
direct after	Anfang(X) = Ende(Y)
during	Anfang(X) > Anfang(Y) und Ende(X)<Ende(Y)
holds	Anfang(X) < Anfang(Y) und Ende(X)>Ende(Y)
equals	Anfang(X) = Anfang(Y) und Ende(X)=Ende(Y)

START (TYPE FLOAT)

Gibt den Startpunkt für die Phase an. Es handelt sich um den absoluten Startpunkt, also die Zeit seit dem Beginn der ersten Phase.

SOLUTION_F, SOLUTION_P (TYPE List)

In diesem Slot werden die Daten der Lösung angegeben, und zwar als Diagramm. Abgespeichert werden das Kraft-Zeit-Diagramm (SOLUTION_F) und das Weg-Zeit-Diagramm (SOLUTION_P). Die anzugebende Liste besteht jeweils aus einer Liste pro Polynomstück. In dieser wird angegeben: (BeginRelativ Length

a b c). Diese Zahlen sind alle vom Typ FLOAT. Dabei bedeutet BeginRelativ den relativen Begin zum Anfang der Phase (!), Length die zeitliche Länge des Polynomstücks, und a, b, c bilden das Polynom ($a \cdot x^2 + b \cdot x + c$). Beispiel: (SOLUTION_P (0 1 1 0 0) (1 1 2 0 -1)) bedeutet: das erste Polynomstück geht vom Anfang (Sekunde 0) bis zur Sekunde 1 und ist in der Form x^2. Das zweite Polynomstück beginnt in Sekunde 1, dauert ebenfalls 1 Sekunde bis Sekunde 2 und ist von der Form $2 \cdot x^2 - 1$.[2]

A.4. Eingabe: Meta-Achsen

(OBJECTTYPE (TYPE STRING)(VALUE "META_AXIS"))

Dieser Objekttyp wird schon in [19] beschrieben. Für die Kopplung ist es unerheblich, ob es sich um Achsen oder Proto-Achsen handelt. Deswegen werden für die Kopplung von Proto-Achsen einfach Meta-Achsen genommen. Es wird jedoch ein Wert PROTO (TYPE BOOLEAN) eingeführt, der genau dann wahr sein soll, wenn sich die Meta-Achse auf Proto-Achsen bezieht.

Die gebildeten Meta-Achsen müssen also nur die Namen der Proto-Achsen enthalten.

Erweitert wird der Slot COUPLING (TYPE STRING). Er wird um die Möglichkeit **"id"** ergänzt. Diese steht für identische Achsen.

[2] Wie leicht nachzurechnen ist, ist die angegebene Funktion stetig, allerdings nicht stetig differenzierbar.

Kapitel B

Beispieldatei

Folgend ist die in den Kapiteln 3 bis 5 benutzte Beispieldatei im ASE-Format angegeben.

```
 (GlobalDemands
  (OBJECTTYPE (TYPE STRING)(VALUE "GLOBAL"))
  (NAME (TYPE STRING)(VALUE "Testdatei aus Diplomarbeit Anhang C"))
  (DESCRIPTION(TYPE STRING)(VALUE "Es sind drei Achsen gefordert. A2
und A3 liegen wahrscheinlich sequentiell, die  daraus entstehende
Meta-Achse wahrscheinlich seriell hinter A1."))
  (STATE (TYPE INT)(VALUE 0))
  (CRITERIA(TYPE LIST)(VALUE
    (
       ("MIN_TIME" 0)
       ("MAX_EFFECTIVITY" 0)
       ("MIN_COST" 0)
       ("MIN_VARIETY" 0)
       ("MIN_DISSIPATION" 0)
       ("MIN_EPSILON_HOSE" 0)
    ))
  )
 )
```

```
 (idA1
  (OBJECTTYPE (TYPE STRING)(VALUE "PROTOAXIS"))
  (WORKING_ELEMENT (TYPE STRING)(VALUE "compCylinder"))
  (NAME (TYPE STRING)(VALUE "A1"))
  (NUMBER (TYPE INT)(VALUE 1))
  (PHASES (TYPE LIST)(VALUE (idP11 idP12 idP13 )))
 )
 (idP21
  (OBJECTTYPE (TYPE STRING)(VALUE "PHASE"))
  (NAME (TYPE STRING)(VALUE "P11"))
  (AXIS (TYPE IDENTIFIER)(VALUE idA1))
   (FORCE_BEGIN (TYPE FLOAT)(VALUE 0))
   (FORCE_END (TYPE FLOAT)(VALUE 0))
  (ACTIONS (TYPE LIST)(VALUE (
      ("P" "to" "absolut" 190 0)
)))
  (CONSTRAINTS (TYPE LIST)(VALUE
   (
    ("BEGIN"  0  "==" idP21 "BEGIN"  0 )
   ))
  )
  (CONSTRAINT_MACROS (TYPE LIST)(VALUE
   (

   (idP12 "DIRECT BEFORE")
   ))
  )
  )

 (idP12
  (OBJECTTYPE (TYPE STRING)(VALUE "PHASE"))
  (NAME (TYPE STRING)(VALUE "P12"))
  (AXIS (TYPE IDENTIFIER)(VALUE idA1))
   (FORCE_BEGIN (TYPE FLOAT)(VALUE 100))
   (FORCE_END (TYPE FLOAT)(VALUE 100))
   (TIME_MIN (TYPE FLOAT)(VALUE 2))
   (WALK (TYPE STRING)(VALUE "PRESS"))
  (ACTIONS (TYPE LIST)(VALUE (
      ("P" "to" "absolut" 200 0)
)))
  (CONSTRAINTS (TYPE LIST)(VALUE
   (
   ))
```

```
  )
  (CONSTRAINT_MACROS (TYPE LIST)(VALUE
   (
   ))
  )
  )

 (idP13
  (OBJECTTYPE (TYPE STRING)(VALUE "PHASE"))
  (NAME (TYPE STRING)(VALUE "P13"))
  (AXIS (TYPE IDENTIFIER)(VALUE idA1))
   (FORCE_BEGIN (TYPE FLOAT)(VALUE 0))
   (FORCE_END (TYPE FLOAT)(VALUE 0))
   (WALK (TYPE STRING)(VALUE "POSITIONDRIVE"))
  (ACTIONS (TYPE LIST)(VALUE (
      ("P" "to" "absolut" 0 0)
)))
  (CONSTRAINTS (TYPE LIST)(VALUE
   (
    ("BEGIN"  0  "<" idP22 "END"  0 )
   ))
  )
  (CONSTRAINT_MACROS (TYPE LIST)(VALUE
   (
   ))
  )
  )



 (idA2
  (OBJECTTYPE (TYPE STRING)(VALUE "PROTOAXIS"))
  (WORKING_ELEMENT (TYPE STRING)(VALUE "compCylinder"))
  (NAME (TYPE STRING)(VALUE "A2"))
  (NUMBER (TYPE INT)(VALUE 1))
  (PHASES (TYPE LIST)(VALUE (idP21 idP22 idP23 )))
 )
 (idP21
  (OBJECTTYPE (TYPE STRING)(VALUE "PHASE"))
  (NAME (TYPE STRING)(VALUE "P21"))
  (AXIS (TYPE IDENTIFIER)(VALUE idA2))
   (FORCE_BEGIN (TYPE FLOAT)(VALUE 0))
   (FORCE_END (TYPE FLOAT)(VALUE 0))
```

```
   (WALK (TYPE STRING)(VALUE "ACCELERATE"))
  (ACTIONS (TYPE LIST)(VALUE (
      ("P" "to" "absolut" 121 0)
)))
  (CONSTRAINTS (TYPE LIST)(VALUE
   (
   ))
  )
  (CONSTRAINT_MACROS (TYPE LIST)(VALUE
   (

   (idP22 "DIRECT BEFORE")
   ))
  )
  )

 (idP22
  (OBJECTTYPE (TYPE STRING)(VALUE "PHASE"))
  (NAME (TYPE STRING)(VALUE "P22"))
  (AXIS (TYPE IDENTIFIER)(VALUE idA2))
   (FORCE_BEGIN (TYPE FLOAT)(VALUE 10))
   (FORCE_END (TYPE FLOAT)(VALUE 20))
   (WALK (TYPE STRING)(VALUE "PRESS"))
  (ACTIONS (TYPE LIST)(VALUE (
      ("P" "to" "absolut" 200 0)
)))
  (CONSTRAINTS (TYPE LIST)(VALUE
   (
    ("END"  0  "<" idP31 ("F" "==" 10 "absolut")  0 )
    ("BEGIN"  0  "<" idP31 "BEGIN"  0 )
   ))
  )
  (CONSTRAINT_MACROS (TYPE LIST)(VALUE
   (
   ))
  )
  )

 (idP23
  (OBJECTTYPE (TYPE STRING)(VALUE "PHASE"))
  (NAME (TYPE STRING)(VALUE "P23"))
  (AXIS (TYPE IDENTIFIER)(VALUE idA2))
   (FORCE_BEGIN (TYPE FLOAT)(VALUE 0))
```

```
   (FORCE_END (TYPE FLOAT)(VALUE 0))
   (WALK (TYPE STRING)(VALUE "POINTDRIVE"))
  (ACTIONS (TYPE LIST)(VALUE (
      ("P" "to" "absolut" 0 0)
)))
  )



 (idA3
  (OBJECTTYPE (TYPE STRING)(VALUE "PROTOAXIS"))
  (WORKING_ELEMENT (TYPE STRING)(VALUE "compCylinder"))
  (NAME (TYPE STRING)(VALUE "A3"))
  (NUMBER (TYPE INT)(VALUE 1))
  (PHASES (TYPE LIST)(VALUE (idP31 idP32 )))
 )
 (idP31
  (OBJECTTYPE (TYPE STRING)(VALUE "PHASE"))
  (NAME (TYPE STRING)(VALUE "P31"))
  (AXIS (TYPE IDENTIFIER)(VALUE idA3))
   (FORCE_BEGIN (TYPE FLOAT)(VALUE 10))
   (FORCE_END (TYPE FLOAT)(VALUE 10))
  (ACTIONS (TYPE LIST)(VALUE (
      ("P" "to" "absolut" 200 0)
)))
  (CONSTRAINTS (TYPE LIST)(VALUE
   (
    (("P" "==" 100 "absolut")  0  "<" idP13 "BEGIN"  0 )
   ))
  )
  (CONSTRAINT_MACROS (TYPE LIST)(VALUE
   (
   ))
  )
  )

 (idP32
  (OBJECTTYPE (TYPE STRING)(VALUE "PHASE"))
  (NAME (TYPE STRING)(VALUE "P32"))
  (AXIS (TYPE IDENTIFIER)(VALUE idA3))
   (FORCE_BEGIN (TYPE FLOAT)(VALUE 0))
   (FORCE_END (TYPE FLOAT)(VALUE 0))
  (ACTIONS (TYPE LIST)(VALUE (
```

```
      ("P" "to" "absolut" 0 0)
)))
  )

)
```

Kapitel C

CD mit Programm