KfK 5306
Juni 1994

Ein wissensbasierter Planungsansatz für Reihenfolgeprobleme in einer Roboterzelle

K.-A. Ricken
Institut für Angewandte Informatik

Kernforschungszentrum Karlsruhe

Kernforschungszentrum Karlsruhe

Institut für Angewandte Informatik

KfK 5306

Ein wissensbasierter Planungsansatz für Reihenfolgeprobleme in einer Roboterzelle*

Karl-Albrecht Ricken

*von der Fakultät für Informatik der Universität Karlsruhe (TH) genehmigte Dissertation

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640, 76021 Karlsruhe

ISSN 0303-4003

Kurzzusammenfassung

Ein allgemeiner Ansatz integriert die Darstellung unterschiedlicher Reihenfolgeprobleme wie die des Travelling-Salesman-Problems, der Maschinenbelegungsplanung oder der Projektplanung. Er basiert auf Aktivitäts-Zustandsnetzen, die weiterentwickelt und graphentheoretisch formalisiert wurden. Dabei vereinigt der Planer herkömmliche Ansätze wie die Zustandsraumrepräsentation und den UND/ODER-Graphen. Die Darstellung unterstützt und integriert die Erarbeitung von Lösungsstrategien und stellt für ihre Entwicklung eine allgemeine Vorgehensweise bereit.

Speziell für Zeitprobleme entstand ein nicht-linearer Zeitplaner mit einer Komplexität von $O(n^4)$, der die qualitative und quantitative Zeitplanung integriert. Ein qualitatives temporales Netzwerk enthält Zeitbeziehungen wie vorher, während, überlappt usw. und ihre Disjunktionen. Es basiert auf J. Allens Schlußfolgerungsliste und vereinfacht das Netzwerk zu transitiven Ketten. An Kreuzungen transitiver Ketten werden weitere Vereinfachungen vorgenommen. Auch die quantitative Information (Dauern und Zeitpunkte) kann unvollständig sein oder in unscharfer Form ("nicht vor 9 h") vorliegen. Zwischen quantitativer und qualitativer Schlußfolgerung besteht eine gegenseitige Rückkopplung: Die quantitative Schlußfolgerung beschränkt die qualitative und umgekehrt.

Der Planer löst verschiedene Reihenfolgeprobleme in einer Roboterzelle zum Gußputzen am Kernforschungszentrum Karlsruhe.

A knowledge based planning approach for sequencing problems and temporal problems

A general approach integrates the representation of various sequencing problems such as the Travelling Salesman Problem, Scheduling, or Project Planning. It is based on Activity-State-Networks which were elaborated and formalized in a graph-theoretical approach. The planner integrates conventional approaches such as the state-space-represenation or AND/OR-Graphs. The approach supports and integrates the elaboration of problem-solving strategies and provides a general framework for the development of these strategies..

A non-linear temporal planner with a complexity of $O(n^4)$ is provided for solving a broad variety of time-problems. The planner integrates qualitative and quantitative planning. The qualitative temporal network includes relations such as before, during, overlaps etc. and their disjunctions. It is based on J. Allen's temporal propagation algorithm and simplifies the network with transitive chains. At the intersection of tranisitve chains further reductions are made. Also the quantitative information such as durations and time points can be incomplete or partially constraint (such as "not before 9 o'clock"). There is a feed back loop between the two propagations: The quantitative propagation constrains the qualitative one and vice versa.

The planner solves various sequencing problems in a robot cell for fettling at the Nuclear Research Center Karlsruhe.

1. Thema

Thema der vorliegenden Arbeit ist die *Entwicklung und Implementierung eines Reihenfolgeplaners für die COMETOS-Roboterzelle. Der Planer soll unterschiedliche Darstellungsformen und Lösungstrategien integrieren, um die Entwicklung spezifischer Lösungen zeitsparender zu machen.*

In der Literatur über Operations-Research (vgl. etwa [Müller-Merbach 70]) werden Reihenfolgeprobleme unter verschiedenen Gesichtspunkten klassifiziert. Man impliziert damit, daß nicht nur die Problemlösung selbst, sondern auch die Darstellung der verschiedenen Probleme grundsätzlich verschieden sein muß.

Lösungsverfahren für Reihenfolgeprobleme sind oft Such- oder Sortierverfahren. Deswegen liegt es nahe, nach einem gemeinsamen Ansatz zur Darstellung und Lösung zu forschen: *Es soll ein allgemeiner Repräsentations- und Lösungsansatz gefunden werden, mit dem die Bandbreite der Reihenfolgeprobleme in einer Roboterzelle zusammengefaßt werden kann.* Der Ansatz soll für Reihenfolgeprobleme typische Strukturen wie Zeit, Kausalität und Hierarchie integrieren.

Hier liegt die Chance eines allgemeinen Werkzeugs zur Reihenfolgeplanung gegenüber herkömmlichen Systemen zur Modellierung, die entweder inflexibel sind, da sie nur spezielle Probleme bearbeiten können oder nur mit erheblichem Programmieraufwand auf eine vorliegende Aufgabenstellung applizierbar sind. Wissensbasierte Planer sind besser geeignet, schlecht strukturierte Gebiete darstellen: "Knowledge-based system technology is particularly useful for solving problems that one doesn't understand very well to begin with. That is the typical situation when one is building an application system"[1].

[1] Richard Fikes in IEEE expert October 1991, S. 13.

2. Ziel

Die COMETOS-Roboterzelle ist ein hochflexibles Handhabungssystem zur Bearbeitung mittelgroßer Gußteile. Sie soll um ein System ergänzt werden, das die dort anfallenden Planungsprobleme löst. Dabei treten verschiedene Planungsprobleme auf, die alle zur Klasse der Reihenfolgeprobleme gehören. Bis jetzt ist weder vollständig abzusehen, welche Bedingungen genau an die Planung gestellt werden, noch an welcher Stelle in der Roboterzelle weitere Reihenfolgeprobleme auftreten. Deshalb soll das Planungssystem für weitere Anforderungen offen sein und möglichst allgemein gehalten werden.

Die COMETOS-Roboterzelle[2] ist ein hochflexibles Handhabungsgerät zum Gußputzen großer Werkstücke. Reihenfolgeplanungsprobleme treten in folgenden Bereichen auf:

1. Auf Zellebene soll die Kapazität so gut wie möglich ausgenutzt und Fristen für die Bearbeitung von Werkstücken eingehalten werden[3]. Durch die Technologie in der Zelle und andere Faktoren werden einer Planung jedoch Grenzen gesetzt, die unbedingt beachtet werden müssen[4]. Für eine große Bandbreite von Werkstücken sind Arbeitspläne aufzustellen, wobei Alternativen berücksichtigt werden müssen. Diese Tätigkeit ähnelt der Projektplanung.
2. Die Routen des autonomen Fahrzeugs müssen geplant werden.
3. Auf Maschinenebene müssen die Bahnelemente, die beim Einlernen erzeugt werden, auf intelligente Weise so zusammengestellt werden, daß die Bearbeitungszeiten möglichst kurz sind.

Das System zur Reihenfolgeplanung muß damit folgende Anforderungen erfüllen:

1. Es muß verschiedene Klassen von Reihenfolgeproblemen einheitlich darstellen und lösen können. Bei der Darstellung sind sehr unterschiedliche Strukturen zu berücksichtigen, wie etwa Vorrang- und Zeitbeziehungen, Kausalitäten, technologische und wirtschaftliche Randbedingungen, Prioritäten, verschiedene Arten von Maschinen und Hilfsmitteln, Vorgängen und Anwendern.
2. Die Komplexität soll gegenüber dem Anwender durch dynamische, leicht erstellbare Strukturen reduziert werden.
3. Problemdarstellung und Lösungsstrategien des Systems sollen bei möglichst vielen Reihenfolgeproblemen Unterstützung bieten. Gleichzeitig sollen sie einheitlich, leicht erweiterbar und mit vertretbarem Aufwand gestaltet werden.

Ein Schwerpunkt der Arbeit liegt in der Zeitpropagation, die für einige Reihenfolgeprobleme wie Projektplanung oder die Belegungsplanung eine große Rolle spielt.

Das System zur Reihenfolgeplanung soll auch auf andere Reihenfolgeprobleme übertragbar sein.

---

[2]Die Zelle wird im Anhang beschrieben.

[3]Diese Ziele stehen im allgemeinen miteinander im Konflikt.

[4]Ein weiterer Konflikt entsteht, wenn eine Teillösung für einen Roboters der Zelle mit Teillösungen für andere Roboter im Konflikt steht.

3. Vorgehensweise

Bis heute wurde noch nicht versucht, ein allgemeines Werkzeug zur Planung von Reihenfolgen zu bauen, weil

- die Anwendungsumgebungen von Reihenfolgeproblemen sich sehr stark unterscheiden,
- verschiedene, unabhängige Verfahren zur Lösung herangezogen werden müssen,
- bereits kleine Reihenfolgeprobleme sehr komplex sind.

Daraus sind einige wichtige Fragen ableitbar:

1. Soll ein derartiger Planer im Rahmen eines wissensbasierten Systems oder in anderer Form realisiert werden?
2. Was haben Anwendungsumgebungen von Reihenfolgeproblemen gemeinsam?
3. Was haben Lösungsverfahren für Reihenfolgeprobleme gemeinsam?
4. Wie läßt sich das, was wir an Gemeinsamkeiten gefunden haben, zusammenfügen?

Frage 1 diskutiert die Vorgehensweise und ist Gegenstand dieses Kapitels. Die Fragen 2 und 3 werden in den Kapiteln 4 und 5 diskutiert. Das Zusammenfügen der Konsequenzen wird in Kapitel 6 zu einem Konzept zusammengefaßt. Die notwendigen Grundtechniken dazu beschreibt Kapitel 5.

3.1. Randbedingungen für den Einsatz des wissensbasierten Systems

F. Puppe nennt Kriterien, um zu entscheiden, ob ein Problem mit einem wissensbasierten System realisiert werden sollte [Puppe 89, S. 132].

1. Problemcharakterisierung[5]: Reihenfolgeplanungsprobleme sind gegenüber dem Allgemeinwissen klar abgrenzbar. Die Datenerfassung erfolgt in der COMETOS-Zelle meist automatisch, z.B. durch Einlesen von Gratinformationen mit einem Laserscanner. Aus Sicht des Reihenfolgeplaners ist die Planungsumgebung verhältnismäßig statisch. Zumindest während der Berechnung ist nicht mit durchgreifenden Änderungen der Planungsumgebung zu rechnen. All das spricht für die Entwicklung eines wissensbasierten Systems. Dagegen spricht, daß auch menschliche Experten nicht immer Reihenfolgeprobleme gut lösen können. Der Planer ist auch nicht auf einen bestimmten Typ von Reihenfolgeproblemen festgelegt. Vielmehr soll er durch Anpassungsfähigkeit die Planung unterstützen, etwa dann, wenn ein völlig neues Reihenfolgeproblem auftaucht, das bisher nicht einmal modelliert wurde.
2. Praktischer Nutzen: Laien werden Reihenfolgeprobleme wahrscheinlich schlechter lösen, was zu wesentlich höheren Kosten der Bearbeitung führt. Dem praktischen Nutzen abträglich ist, daß ein geeignetes Klima für die Einführung von wissensbasierten Systemen fehlt. In Gießereien stößt bereits die Einführung konventioneller Computersysteme in der Produktion auf Widerstand. Das System wird deswegen im Hintergrund bleiben und für den Endanwender nicht sichtbar sein.
3. Eignung für den Wissenserwerb: In der Maschinenbelegungsplanung hat der Autor selbst mehrere Jahre Erfahrung sammeln können. Um Bahnelemente zu planen, ist das Wissen aus Bedienungshandbüchern und Gesprächen mit der COMETOS-Mannschaft zu erwerben. Für die Bahnelementeplanung gibt es in der Literatur keine

[5] Die Strukturierung dieses und der folgenden Punkte lehnt sich eng an Puppes Kriterienkatalog an.

Quellen, weil das Problem erst mit der COMETOS-Zelle aufkam. Verwandte Probleme finden sich dagegen reichlich.

Sicher ist damit die Reihenfolgeplanung nicht die ideale Einsatzumgebung für ein wissensbasiertes System. Die Alternative wäre aber nicht, konventionell zu implementieren, sondern auf jede Implementierung zu verzichten und die Probleme weiterhin manuell zu lösen. Da das auf Dauer für den Anwender der Roboterzelle unbefriedigend ist, soll mit dieser Arbeit der Versuch gemacht werden, ihn bei der Planung zu unterstützen und dabei die Planungsgüte zu verbessern.

Zu den Wissensdomänen zählen Planungsverfahren des OR und der KI und die Umgebung von COMETOS. Quellen für den Wissenserwerb sind die Standard- und Fachliteratur, die internen Berichte, Interviews und Ergebnisse eigener Versuche und Forschungsanstrengungen. Für Wissensrepräsentation und Inferenz fehlt bisher eine angemessene Lösung. Die Forschung wurde auch durch Diskussionen mit Kollegen am Institut für Logik, Komplexität und Deduktionssysteme der Universität Karlsruhe (Prof. Deussen) stimuliert.

Das Wissen für den Reihenfolgeplaner steht größtenteils zur Verfügung. Unbekannt und damit Gebiete für die eigene Forschung sind die Optimierung von Bahnelementen und die Repräsentations- und Inferenzformen für den allgemeinen Reihenfolgeplaner.

3.2. Implementierung

Für die Entwicklung stehen Programmiersprachen und spezialisierte Werkzeuge, sog. Tools zur Verfügung. In der Abteilung Industrielle Handhabungssysteme am KFK, in der diese Arbeit entstand, wird mit Knowledge Craft™ der Firma Carnegie Group, Inc. gearbeitet. Praktische Probleme der Implementierung ergeben sich aus den Grenzen der zur Verfügung stehenden Entwicklungsumgebung.

1. COMMON Lisp und das Tool Knowledge Craft sind zwar hinsichtlich der Verarbeitungsmöglichkeit mächtige Werkzeuge, aber im Vergleich zu klassischen Programmiersprachen wie C oder Pascal recht langsam. Da aber besonders die Propagation von Zeiten rechenzeitintensiv ist, wünscht man sich eine schnelle Plattform für die Implementierung. Die Zeitpropagierung ist deswegen in C mit einer Schnittstelle von KnowledgeCraft realisiert. Das C-Modul kann separat benutzt und vertrieben werden.

2. Für die objektorientierte Programmierung existieren bis heute keine durchgängigen Werkzeuge, die den Entwickler von der Analyse bis zur Implementierung begleiten. Das ausgewählte Werkzeug Innovator™ der Firma MID ist weder für Lisp noch für objektorientierte Programmierung oder Datenmodellierung geeignet. Es wird nur verwendet, weil der Autor glaubt, daß es besser ist, mit einem schlechten Werkzeug zu arbeiten als ohne.

4. Grundlagen

Welche Gemeinsamkeiten besitzen Reihenfolgeprobleme? Die COMETOS-Umgebung als Anwendungsfall stellt typische Probleme. Danach werden die OR-Grundlagen der Reihenfolgeproblematik dargestellt und analysiert.

4.1. Darstellung der COMETOS-Umgebung

4.1.1. Überblick über das Gußputzen

Beim Gußputzen werden die Grate abgetrennt, die beim Gießen entstanden sind. Isele hat Gußgrate systematisiert (vgl. [Isele 87, S. 27f]). Er untersuchte Großgußteile, insbesondere Achsbrücken für Lastkraftwagen bei der Firma Schubert & Salzer in Ingolstadt. Die Werkstücke wiegen bis zu 250 kg und sind aus globularem Grauguß.

Isele unterscheidet zwischen Trennen und Putzen. Anschnitte und Speiser trennt man. Putzen gliedert er weiter in Innen- und Außenputzen sowie in geplanten und ungeplanten Putzaufwand. Aufgrund der Kinematik kommt für die Bearbeitung mit Robotern nur Außenputzen infrage, das Innenputzen erfolgt weiterhin von Hand. Geplanter Putzaufwand ist durch die Gußtechnologie bedingt, z.B. bei Speisern und Anschnitten. Ursache ungeplanten Putzaufwandes sind Sandausbrüche, angebrannter Sand und Abdrücke von ungenau positionierten Kernstützen.

4.1.2. Die COMETOS-Roboterzelle[6]

Am Institut für Angewandte Informatik wurde eine Roboterzelle zum Gußputzen entwickelt (vgl. auch Anhang). Sie besitzt zwei Portalroboter, einen für die Bearbeitung und einen zum Messen. Die Werkstücke sind auf Paletten gespannt, die ein fahrerloses Transportsystem bewegen soll. Ein Anwender programmiert die Roboter mit einer Art aufgabenorientierten Programmierung[7]: Er führt den Roboterarm an einer Sensorkugel über den Grat des Werkstückes, wobei er nur auf 20 mm genau führen muß. Beim konventionellen Einlernen wären höchstens Abweichungen von 0,5 mm zulässig. Die Genauigkeitsanforderungen an den Anwender bei COMETOS im Vergleich zu herkömmlichen Systemen stellt die Tabelle 4.1 dar.

Führungsgenauigkeit der Sensorkugel bei der Programmierung	±20mm
Meßgenauigkeit des Laserscanners	±0,05mm
erforderliche Einlerngenauigkeit bei konventionellem Einlernen: bei COMETOS:	 ±0,5mm ±20mm

Abb. 4.1.: Vergleich der Genauigkeitsanforderungen: Aufgrund der Kombination von Laserscanner und Sensorkugel reicht bei COMETOS grobes Einlernen aus.

Die Flexibilität ermöglicht ein am Arm des Roboters angeflanschter Laserscanner, der anschließend auf 0,05 mm genau Lage und Höhe des zu entfernenden Grates mißt. Aus diesen Daten wird automatisch ein Meß- und später ein Putzprogramm erstellt. Für das Putzprogramm gibt der Anwender zusätzliche, aus der Messung nicht ableitbare Daten, wie Werkzeugwahl und -orientierung ein. Das System senkt die Einlernzeit von mehreren hundert Stunden bei konventioneller off-line Programmierung auf 4 bis 5 Stunden.

[6] Eine detaillierte Beschreibung der Zelle findet sich bei [Lawo 89], [Lawo 90a-c], [Lawo 91]. Grundbegriffe der Robotik erklärt Naval [Naval 89].

[7] Der Anwender programmiert nicht konventionell, sondern spezifiziert nur die Aufgaben.

Der Anwender führt den Roboter nicht in einen Zug um das Werkstück. Vielmehr unterteilt er den Weg des Roboters in mehrere Abschnitte, die Bahnelemente heißen. Dabei handelt es sich um sinnvoll zusammengefaßte Sequenzen von Bahnpunkten. Jeden der Bahnpunkte kann der Roboter mit Hilfe der Robotersteuerung ansteuern. Bei den Bahnelementen unterscheidet man weiterhin zwischen Luft- und Gratbahnen. Nur auf Gratbahnen wird der Laserscanner messen. Gratbahnen heißen beim Messen Meßbahnen und beim Putzen Bearbeitungsbahnen. Zwischen Bearbeitungsbahnen können Luftbahnen liegen. Auf ihnen ist das Werkzeug nicht im Eingriff. Man generiert sie beispielsweise für Wege zur Werkzeugwechseleinrichtung oder zum Werkstück hin.

COMETOS entspricht einer allgemeinen Klasse von Problemen. Hochflexible Handhabungssysteme verfolgen unter hohem Programmieraufwand das Ziel, Werkstücke mit Industrierobotern zu bearbeiten. Die Aufgabe läßt sich vereinfachen, wenn man sie hierarchisch in Teilaufgaben strukturiert. Die Folge der Teilaufgaben beschreibt Abbildung 4.2.

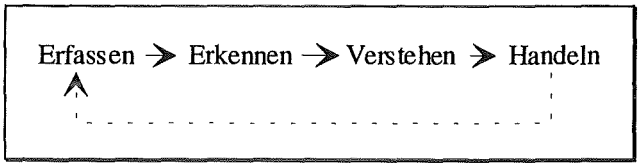


Abb. 4.2: Das allgemeine Problem, hochflexible Handhabungssysteme zu programmieren, läßt sich durch Strukturierung vereinfachen.

Die hierarchische Strukturierung wird am Ende dieses Kapitels weiterverfolgt.

4.1.3. Reihenfolgeplanungsprobleme in der COMETOS-Zelle

Um einen ausreichenden Durchsatz zu gewähren, Leerzeiten zu vermeiden und Bearbeitungsfristen einzuhalten, muß die Maschinenbelegung geplant werden. Der Anwender lernt Bahnelemente nicht immer in der Reihenfolge ein, die für die Bearbeitung sinnvoll ist, sondern in der Reihenfolge, die für das Einlernen bequem ist. Deswegen müssen die Bahnelemente umsortiert werden, um die Bearbeitung schneller und wirtschaftlicher zu machen. Das sind Aufgabenstellungen, die in den nächsten Abschnitten erläutert werden.

4.1.3.1. Maschinenbelegungsplanung für COMETOS

Wichtigstes Ziel der Maschinenbelegung ist, eine möglichst hohe Auslastung der Zelle zu gewährleisten. Fristen einzelner Werkstücke spielen keine so große Rolle, weil üblicherweise eine Tagesproduktion aus der Gießerei auch wirklich abgearbeitet werden kann.

Ein naiver Ansatz würde in der COMETOS-Fertigungszelle einen Flow Shop[8] von zwei Maschinen sehen. Zuerst mißt und lernt ggf. der Einlern-Meßroboter ein, dann bearbeitet der Putzroboter das Werkstück. Die reale Problemstellung ist aber um einiges komplexer:

- Jedes Werkstück muß auf einer Palette transportiert und bearbeitet werden. Die Anzahl der Paletten ist begrenzt.
- Die Roboter sind aufgrund identischer Kinematiken untereinander austauschbar. Der Putzroboter kann auch messen, wenn man an seine Hand den Laserscanner anflanscht. Umgekehrt könnte der Meßroboter auch Bearbeitungsaufgaben überneh-

[8]Begriffe wie Flow Shop und Job Shop sind der Theorie der Maschinenbelegungsplanung entnommen und werden im Abschnitt 4.3 erklärt.

men. Dafür müßte er aber wie der Putzroboter in einer entsprechend geschützten Arbeitszelle stehen.

- Die Zelle kann aus mehr als zwei Robotern bestehen, die sich die anstehenden Aufgaben aufteilen.
- Einige Werkstücke müssen vor dem Messen getrennt werden.
- Innenputzen und ungeplanter Putzaufwand werden manuell erledigt.

Bei näherer Betrachtung handelt es sich hier um einen komplexen Job Shop, der durch folgende Einflüsse noch schwieriger wird:

- Die Einlernzeiten sind unbekannt und lassen sich nur schätzen.
- Ist für ein Werkstück die Einlernzeit unbekannt, so sind dies auch die Meß- und Bearbeitungszeiten. Weil die Bandbreite an Werkstücken regelmäßig erweitert wird, kommt das oft vor.
- Die Arbeitspläne neuer Werkstücke sind unbekannt. Arbeitspläne bekannter Werkstücke sind flexibel[9] und sind situationsabhängig zu bestimmen.
- Zum Einlernen des Putzprogramms werden zwei Roboter benötigt. Es gilt also nicht "eine Operation - eine Maschine".

Das Problem, Maschinen zu belegen, wird speziell durch die Forderung nach flexiblen Arbeitsplänen sehr komplex. Bevor Operationen in eine Reihenfolge gebracht werden können, muß zuerst untersucht werden, welche Operationen überhaupt in Frage kommen.

4.1.3.2. Optimale Reihenfolgen der Bahnelemente

Die Bearbeitung eines Werkstücks soll möglichst schnell gehen. Die Roboterzelle steht im Wettbewerb mit dem Gußputzer, der bei seiner Arbeit opportunistisch vorgehet und nur dort ansetzt, wo eine Bearbeitung notwendig ist. Der Bearbeitungsroboter kann andererseits sehr rasch Grate abtrennen. Um diesen Vorteil zu halten, müssen die Bearbeitungsnebenzeiten wie Werkzeugwechsel und Orientierungsänderungen des Roboters minimiert werden.

Der Anwender unterteilt Bahnelemente nach eigenem Gutdünken. Wahrscheinlich lernt er immer am Werkstück entlang ein. Durch geschicktes Zusammenfassen von Bearbeitungsbahnen kann Zeit gespart werden. Im Rahmen einer Diplomarbeit (vgl. [Lotter 92]) wurde dieser Frage nachgegangen und eine Lösung ausgearbeitet:

1. Unter der Voraussetzung, daß ein bestimmtes langsameres Werkzeug weniger gut zugängliche als auch gut zugängliche Bahnen bearbeitet, die auch mit einem schnelleren Werkzeug bearbeitet werden könnten, und daß für einen Werkzeugwechsel 20 s (s: Sekunden) benötigt werden, gilt: Für die Bearbeitung eines Grats mit der Länge l soll ein Werkzeug mit der Vorschubgeschwindigkeit $v_{langsam}$ durch ein anderes mit dem Vorschub $v_{schnell}$ ersetzt werden, wenn gilt:

$$l > \frac{20s * v_{langsam} * v_{schnell}}{v_{langsam} - v_{schnell}}.$$

Dabei wird vorausgesetzt, daß beide Werkzeuge den Grat bearbeiten können.

[9] Ob z.B. ein Werkstück getrennt wird, kann man nur für das konkrete Werkstück entscheiden. Manchmal fallen die abzutrennenden Steiger und Speiser von selbst ab. Voraussagen läßt sich das jedoch nicht.

2. Werkzeugwechsel sind mit höchster Priorität zu minimieren.

3. Drehungen der Roboterhand (Achsen a, b, c) sind zeitaufwendig. Ein Orientierungswechsel der Hand um k Grad soll, wenn durch Einfügen einer Luftbahn ein weiteres Bahnelement mit gleicher Orientierung abgefahren werden könnte, durch eine solche der Länge l ersetzt werden, wenn gilt:

$$k * 30\ mm > l.$$

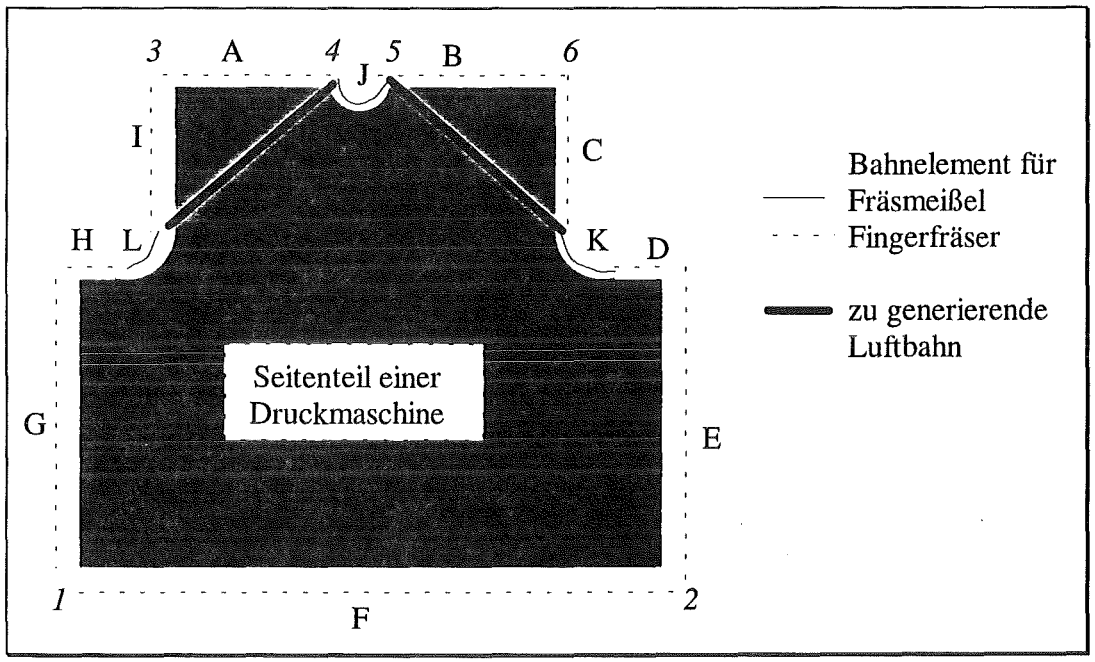


Abb. 4.3: Beispiel für Bearbeitungsbahnen an einem Druckmaschinenseitenteil

Das Beispiel[10] aus Abbildung 4.3 soll die Aufgabe illustrieren. Die Bahnen A bis I werden mit einem hohe Bahngeschwindigkeit zulassenden Fräsmeißel, die Bahnen J bis L mit dem wesentlich langsameren Fingerfräser bearbeitet. Würde man bei Punkt 1 starten und die Bahnen um das Werkstück herum abfahren, wären sechs Werkzeugwechsel nötig, um das Werkstück abzuarbeiten. Faßt man dagegen die Bearbeitung der Bahnen A bis I und die der Bahnen J bis L zusammen, dann ist nur ein Werkzeugwechsel erforderlich.

Zeiten für die Orientierungsänderung kann man sparen, wenn man parallele Bahnen zusammenfaßt, z.B. die Reihenfolge A - B - F anstelle von E - F - G, wobei F von Punkt 1 nach Punkt 2 (anstatt in Gegenrichtung) abgefahren wird. Allerdings müssen Fräsmeißel, Trenn- und Schruppscheiben die richtige Orientierung zum Werkstück haben. Bei der Entscheidung für die Orientierung eines Werkstücks zum Werkzeug sind zu berücksichtigen:

- Kollisionsprobleme
- Grenzen des Arbeitsraums
- Fräser und Schleifscheibe können nicht beliebig in Gleich- oder Gegenlauf arbeiten
- die Aufspannung des Werkstückes.

Die Mehrzahl der Wege, die sich durch Permutation der Bahnelemente ermitteln, ist aufgrund technischer Beschränkungen unzulässig.

[10] Das Beispiel wurde keineswegs willkürlich gewählt. Es war Thema genauer Studien zum Gußputzen des Teams Häfele, Isele, Kohlhepp und Lawo (vgl. [Häfele 91a, b]).

Außerdem sind zusätzliche Luftbahnen zu generieren. So ist ein möglicher Weg von J nach K zwar schon bekannt - er führt entlang der Bahnen B und C - aber er ist zeitraubend im Vergleich zum direkten Weg (fette Linie). Der direkte Weg ist jedoch unbekannt, denn er wurde nie eingelernt. Er kann auch nicht errechnet werden, weil vom Werkstück außer dem Gratverlauf keine Daten bekannt sind. Insbesondere kann für das Werkstück kein CAD-Modell erstellt werden. Auch für diese Aufgabe ist eine Lösung gesucht.

Das Planungssystems soll also folgendes unterstützen (Tabelle 4.1):

Minimierungsziel	bisheriges Vorgehen	angestrebtes Vorgehen
Einlernzeit (größter Zeitblock)	Der Bediener muß beim Einlernen das spätere Meß- und das Putzprogramm berücksichtigen. Der erfahrene Bediener lernt das Meßprogramm vom Putzprogramm aus ein und erzeugt wegen der mehrfachen Werkzeugwechsel komplizierte Bahnen.	Rund-um-fahren reicht. Der Bediener kann sich auf die Optimierung der Einlernzeit zum Messen konzentrieren.
Meßzeit	Das Meßprogramm entsteht aus den eingelernten Bahnen, die jedoch mit Blick auf die spätere Bearbeitung mit zusätzlichen Orientierungsänderungen generiert werden.	Der Bediener konzentriert sich auf die Optimierung der Einlernzeit zum Messen.
Bearbeitungszeit	Aus den eingelernten Bahnen und den Ergebnissen der Messung entsteht ein Putzprogramm. Der Bediener konnte aber nicht jeden Zusammenhang und alle Beeinflussungen a priori berücksichtigen. Das Bearbeitungsprogramm dauert länger als nötig.	Die Struktur (Reihenfolge der Bahnelemente) des Meßprogramms wird vom System minimiert aufgrund der Reduktion von Orientierungsänderungen und Werkzeugwechseln.

Tab. 4.1.: Optimierungsaufgaben, bei denen der Planer Unterstützung anbieten soll

4.2. Grundlagen der Reihenfolgeplanung

Die Planung verwaltet die Knappheit von Ressourcen wie Zeit und Raum. Im Gegensatz zur Layoutplanung sind der räumliche Gesichtspunkt oder Fragen, wie Funktionseinheiten technisch zusammenpassen, bei der Reihenfolgeplanung von untergeordnetem Interesse. *Reihenfolgeplanung ist die Verwaltung knapper Ressourcen über die Zeit zur Realisierung von Zielen einer Anwendung.*

4.2.1. Klassifizierung von Reihenfolgeplanungsproblemen

Eine im deutschsprachigen Raum bekannte Gesamtdarstellung der Reihenfolgeprobleme schuf Müller-Merbach [Müller-Merbach 70]. Er ordnet Reihenfolgeprobleme nach der Art der Problemrepräsentation und der Komplexität der Lösung [Müller-Merbach 70 S. 2f]:

A alle vorkommenden Elemente sind bekannt, alle Elemente müssen in die Reihenfolge aufgenommen werden. AK: die Kosten zwischen den Elementen sind a priori bekannt und konstant. AV: die Kosten sind nicht bekannt und variabel.

B alle vorkommenden Elemente sind bekannt, aber nicht alle Elemente müssen in die Reihenfolge aufgenommen werden.

U vorkommende Elemente sind möglicherweise bezüglich ihrer Lage und ihrer Anzahl nach unbekannt.

Das Problem des optimalen Weges in einem Netz ist vom Typ B. Ein Netz besteht aus Knoten und mit Kosten belegten Kanten. Ein optimaler Weg ist je nach Aufgabenstellung der kürzeste oder längste Weg zwischen zwei Knoten. Ein Spezialfall dazu ist das Travelling Salesman Problem, das vom Typ AK ist. Start- und Endknoten sind identisch und ein Lösungsweg muß alle Knoten wenigstens einmal passieren (Hamilton'scher Zyklus). Probleme der Maschinenbelegungsplanung zählen zu den komplexesten Problemen der Reihenfolgeplanung überhaupt; sie sind ebenfalls vom Typ AK.

Probleme kürzester und längster Wege in einem Netz sind heute gut verstanden und beherrschbar, d.h. es gibt schnelle Verfahren zu ihrer Lösung. Gut untersucht, aber weniger gut beherrscht sind die übrigen Probleme insbesondere das TSP und die Maschinenbelegungsplanung. Für sie existieren nur Optimierungsverfahren von exponentieller Komplexität. Bereits bei kleineren Problemen muß man sich deswegen mit suboptimalen Lösungsverfahren zufriedengeben, die in Einzelfällen erheblich vom Optimum abweichen. Verfahren, die diese Probleme stets optimal lösen, sind von exponentieller Komplexität und bis auf wenige Ausnahmen aufgrund der ausufernden Laufzeit nicht zu verwenden.

4.2.2. Darstellung von Problemen

In der Literatur werden Reihenfolgeprobleme fast immer mit Zielfunktionen und Nebenbedingungen dargestellt. Diese Form ist für Probleme der mathematischen Optimierung geeignet, für die klassische Verfahren existieren (vgl. z.B. [Neumann 75a]). Eine Darstellung des TSP und des Problems der Maschinenbelegungsplanung in der üblichen Form enthält der Anhang. Probleme der Reihenfolgeplanung sind binäre Optimierungsprobleme, für deren Lösung Judea Pearl zwei wesentliche graphentheoretische Darstellungsformen nennt:

- den Zustandsraumrepräsentationsgraphen
- den UND/ODER-Graphen (vgl. [Pearl 84, S. 21ff]): Ein Problem kann entweder durch Alternativen (ODER) oder Unterprobleme (UND) reduziert werden.

Die Zustandsraumdarstellung teilt ein gegebenes Problem rekursiv in Teillösungen und Restprobleme (Zustände) auf. Ein Zustandsraumgraphen stellt es dar, dessen Knoten ein Teilproblem und dessen Kanten Verfeinerungsoperatoren sind, die das Problem weiter auflösen, bis keine Probleme mehr auftreten. Die Entscheidungssequenzen einer Lösung sind entlang eines Weges durch den Zustandsraumgraphen ablesbar (vgl. [Pearl 84, S. 20f]). Die Problemreduktion ist eine Konjunktion paarweise unabhängiger Unterprobleme. Der Suchraum des Graphen besteht aus zwei Arten von Verbindungen: Verbindungen, die alternative Ansätze für den Umgang mit dem Problemknoten, von dem sie ausgehen, darstellen (ODER) und Verbindungen, die einen Vaterproblemknoten mit den individuellen Unterproblemen verbinden, mit dem sie verbunden sind (UND). Ein Lösungsgraph ist ein UND/ODER-Teilgraph mit den folgenden Eigenschaften:

1. Er enthält den Startknoten
2. Jedes seiner Blätter (Knoten ohne Nachfolger) sind primitive Probleme, deren Lösung bekannt ist.
3. Falls der Teilgraph zu den Blättern hin eine UND-Verbindung L enthält, muß er die gesamte Gruppe von UND-Verbindungen enthalten, die Brüder von L sind.

Ein Problemknoten ist gelöst, wenn:

1. er ein terminaler Knoten (Blatt) ist oder
2. ein nichtterminaler ODER-Knoten ist, von dem mindestens eine ODER-Verbindung auf einen gelösten Knoten zeigt oder
3. er ein nichtterminaler UND-Knoten ist, dessen UND-Verbindungen alle auf gelöste Knoten zeigen (vgl. [Pearl 84, S. 21ff]).

Die Auswahl der Problemdarstellung ist problemabhängig. Ein UND/ODER-Graph bildet ein Problem in einer Abstraktionshierarchie ab, die zur mehrstufigen Planung verwendet werden kann. Hier besteht eine Verbindung zu mehrstufigen Planungsansätzen.

4.2.3. Planungsansätze aus dem OR

Das Operations Research stellt Methoden für die Reihenfolgeplanung zur Verfügung, die man gliedern kann in enumerative, spezielle und heuristische Verfahren. Zu den enumerativen Verfahren zählen neben der vollständigen Enumeration solche, die die Enumeration abkürzen, aber von gleicher Komplexität sind. Dazu zählen Branch-and-Bound-Verfahren oder Methoden der impliziten Enumeration (vgl. [Neumann 75a, S. 311ff], [Garfinkel 79] [Lageweg 77]).

Unter Spezialverfahren seien Methoden verstanden, die nur für spezialisierte Probleme anwendbar sind. Ein Beispiel dafür ist der Johnsonalgorithmus, der auf spezielle Schedulingprobleme mit zwei Maschinen begrenzt ist (vgl. [Brucker 81, S. 121ff], wo auch die Grenzen spezieller Verfahren beschrieben werden).

Unterschiede bestehen im Hinblick auf Komplexität, Optimalität und die Breite der Anwendbarkeit, wie Tabelle 4.2. zeigt

Kriterium	**Enumeration**	**Spezialverfahren**	**Heuristik**
Komplexität	exponentiell	polynomial	polynomial
Optimalität	optimal	optimal	höchstens gute Lösung
Anwendbarkeit	breit	sehr eng	Für Problemklasse breit

Tab. 4.2.: Klassen von Optimierungsverfahren hinsichtlich ihrer Komplexität, der Optimalität erzielbarer Lösungen und ihrer Anwendungsbreite

Was haben nun die Verfahren gemeinsam? Dazu zwei Beispiele für klassische Verfahren:

a) zur Maschinenbelegungsplanung:

```
1. Suche alle freigegeben Operationen.
2. Sortiere sie nach ihrer Dauer aufsteigend.
3. Ordne jede Operation in dieser Reihenfolge den Ressourcen zu.
```

b) zur Lösung eines kürzeste-Wege-Problems:

```
1. Suche alle Wege, die von der aktuellen Position aus
   begehbar sind.
2. Sortiere sie aufsteigend nach ihrer Weglänge.
3. Plane in dieser Reihenfolge die Wege und ihre Nachfolger
   ein.
4. Bewerte die Planung: Wähle die beste Lösung aus.
```

Gemeinsame Verfahrenselemente sind in dieser Reihenfolge 1. Suchen, 2. Sortieren, 3. Zuweisen, 4. Bewerten.

4.2.4. Analyse der Verfahren

Bei der Darstellung von Reihenfolgeproblemen unterscheidet man zwischen dem Gegenstand der Planung und den Randbedingungen. Planungsgegenstand ist z.B. ein Auftrag oder ein Bahnelement. Randbedingungen treten auf in Aussagen über den Planungsgegenstand ("Keine Maschine kann gleichzeitig zwei Werkstücke bearbeiten").

Wesentliche Bausteine einer Heuristik sind die Suche oder das Sortieren und das Bewerten von Elementen einer Lösungssequenz. Der Verfasser hat sämtliche Verfahren zur Reihenfolgeplanung untersucht, die H. Müller-Merbach beschreibt [Müller-Merbach 70] sowie Reihenfolgeplanungsverfahren aus zwei Übersichten von Neumann [Neumann 75 a, b]. Ihre Bausteine lassen sich mit der Abfolge *Suchen und/oder Sortieren - Bewerten* beschreiben, wobei selten ein Verfahren alle Bausteine verwendet.

Einige Verfahrensbausteine verwalten eine ganze Sequenz, andere wählen aus der Sequenz ein einzelnes Element. Die Verwaltung von Sequenzen steuert die Auswahl eines Elements, indem sie globale Informationen bereitstellt. Außerdem vergleicht man auf dieser Ebene Lösungen hinsichtlich ihres Zielfunktionswertes. Bei der Verwaltung einzelner Elemente wird das nächste Element aufgrund globaler und lokaler Informationen gesucht. Beispiel für globale Information: Wähle einen Knoten, der bisher in der Lösungsmenge fehlt. Beispiel für lokale Information: Wähle den nächstgelegenen Knoten.

Vorgeschaltete Sortierheuristiken vereinfachen die Suche, deren Ergebnisse anhand einer Zielfunktion verglichen werden. Zwischen Suche und Bewertung steht das Einplanen des Elements, also die Frage, wie das gesamte System sich verändert, wenn man das gewählte Element an dieser Stelle der Sequenz wählt. Eine Übersicht über Bausteine von Heuristiken gibt die Tabelle 4.3.:

Verfahren	Verwaltung ganzer Sequenzen	Verwaltung einzelner Elemente
Vorsortieren	Regeln wie Kürzeste Operationszeit, Eröffnungsverfahren, Verbesserungsverfahren	
Suchen	Speichern bisheriger Lösungssequenzen, um mehrfache Untersuchung der gleichen Sequenz zu vermeiden. Dijkstra-Algorithmus zur Erzeugung kürzester Wege (vgl. [Müller-Merbach 70, S. 39f]), Depth-First und Breadth-First Suche	Suchen des nächsten Planungsschritts, d.h. Erweiterung der bisherigen Sequenz oder Rücknahme von Planungsschritten, z.B. beim Backtracking.
Planungsschritt ausführen		Die Konsequenzen eines Planungsschritts berechnen
Bewerten	Bewerten einer erzeugten Sequenz, z.B. der Weglänge bei kürzeste-Wege-Problemen.	

Tab. 4.3.: Typische Verfahrensteile von Heuristiken zur Reihenfolgeplanung verarbeiten ganze Sequenzen oder Elemente einer einzelnen Sequenz zur Problemlösung

Von einem Reihenfolgeplaner ist zu verlangen, daß er die dargestellten Verfahrensschritte bereitstellt oder Raum dafür gibt.

4.2.5. Konsequenzen für ein Werkzeug

Wie kann der Reihenfolgeplaner gegenüber herkömmlichen Werkzeugen Zeit sparen? Er kann eine Modellierungsumgebung vorgeben, die die Darstellungsformen Zustandsraumrepräsentationsgraph und UND/ODER-Graph unterstützt. UND/ODER-Graphen benötigen eine Darstellung, die Vorgänge in entsprechenden Bäumen darstellen kann. Zwischen Vorgängen verschiedener Ebenen bestehen hierarchische Beziehungen. Vorgänge in Zustandsräumen können Vorrangbeziehungen besitzen, weil sie einander bedingen. Dieser Zusammenhang entspricht einer kausalen oder zeitlichen Beziehung. Um eine Darstellung zu bewerten, nennt Sathi [Sathi 85, S. 550f] drei Kriterien:

1. Vollständigkeit: Deckt die Darstellung die Anwendungsdomäne ab?
2. Präzision: Wird Wissen in angemessener Granularität dargestellt, d.h. sind alle wesentlichen Dinge sichtbar und wird auf Unwesentliches verzichtet?
3. Klarheit: Gibt es genau eine Darstellung für eine gegebene Situation? Vieldeutigkeit entsteht durch Inkonsistenz und Unvollständigkeit, wobei Vollständigkeit mit dem ersten Kriterium abgedeckt ist. Inkonsistenz impliziert, daß es zwei oder mehr Beschreibungen gibt, die, wenn sie zusammengefügt werden, Konflikte verursachen.

Ein Werkzeug soll darüberhinaus die Bausteine von Heuristiken bereitstellen, um neue Verfahren schneller zu programmieren. Es muß Randbedingungen verschiedenster Art verarbeiten, die in Darstellungen klassischer Reihenfolgeprobleme nicht zu finden sind. Das Bahnelementeproblem der COMETOS-Zelle beispielsweise ist kein TSP, weil zusätzlich Randbedingungen wie Werkzeuge zu berücksichtigen sind und das Abfahren einer Bearbeitungsbahn (= Knoten des TSP) die gesamte Entfernungsmatrix ändert. Es ähnelt aber dem TSP.

Probleme sollten so dargestellt werden, daß dabei die Lösung unterstützt wird. Diese Forderung erfüllen Simplextableaus für binäre Optimierungsprobleme nicht. Die lineare Optimierung ist daher für Reihenfolgeprobleme ungeeignet.

5. Planungsansätze

Die im vorigen Absatz beschriebenen Darstellungs- und Lösungsverfahren besitzen, wie gezeigt wurde, Gemeinsamkeiten, für die ein umfassender Ansatz gesucht wird. Dieses Kapitel beschreibt Konzepte, die die Integration der Verfahren erleichtern. Am Ende läßt sich abschätzen, ob ein einziges Werkzeug zur Reihenfolgeplanung sinnvoll ist und welche Konzepte dafür infrage kommen.

5.1. Darstellung von Planungsproblemen

Gesucht ist eine Repräsentation, die sehr unterschiedliche Arten von Wissen (zeitliches, hierarchisches und kausales Wissen vgl. Ergebnisse von Kapitel 4) integriert. Typische Darstellungsformen aus der Literatur sind Prädikate, Regeln, Petrinetze, Frames und Semantische Netze. Die diskrete Simulation betont besonders die Kausalität und baut Prädikaten/Transitionsnetze auf. Aktivitäts-/Zustandsnetze beschreiben Planungsprobleme auf einer höheren Ebene. In vielen Ansätzen findet man Hierarchien, die das Wissen strukturieren, z.B. auch Blackboard-Architekturen, die den Austausch von Wissen formalisieren. Einige Systeme vereinigen mehrere Darstellungsformen zu einem hybriden Ansatz.

5.1.1. Planung mit diskreter Simulation

Bei der diskreten Simulation betrachtet man ein Planungsproblem als eine Folge von Aktivitäten und Zuständen. Aktivitäten überführen einen Zustand in einen anderen. Eine Lösung ist eine Folge von Aktivitäten, die einen Startzustand in einen Zielzustand überführen. Die heute wichtigste Form diskreter Simulation sind Petri-Netze.

Sie dienen zur Darstellung und Simulation verteilter diskreter Systeme, bei denen zwischen Zuständen und Transitionen unterschieden wird. Die Systeme können nichtdeterministischen und nebenläufigen Charakter haben. Mit Petri-Netzen stellt man Betriebssysteme, Bauanleitungen, Automatenbedienung, Computerkommunikation, Organisationsstrukturen uvm. dar.

Ein Vorgang besteht aus diskreten Aktivitäten, denen Vor- und Nachbedingungen zugeordnet werden können. Die Nachbedingung einer Aktivität kann Vorbedingung einer anderen Aktivität sein. Petri-Netze lassen sich grafisch als Netzwerke darstellen.

5.1.1.1. Zeitdarstellung mit Prädikaten/Transitionsnetzen

Zeitdauern kann man in Petri-Netzen sowohl den Stellen als auch den Transitionen zuweisen. Eine Marke muß auf einer Stelle oft eine Mindestdauer belegen, bevor sie entfernt werden darf. Es ist möglich, die Marke sofort einer Transition zuzuweisen, die man später mit ihr feuern lassen will. Fehlt bis dahin einer anderen Transition diese Marke zum feuern, darf sie die Marke dennoch nicht entfernen. Analog kann man Transitionen eine Zeitdauer zum Schalten zuweisen.

B. Baumgarten zählt zu den Nachteilen von Netzen mit Zeitbegriffen, daß Erreichbarkeitsanalysen schwierig sind [Baumgarten 90, S. 257]. Der Autor gibt selbst eine einfache Zeitmodellierung für Petri-Netze an, die auf offenen Zeitintervallen oder endlichen Vereinigungen davon beruht [Baumgarten 90, S. 258ff]. Die Zeit ist ausschließlich auf die Stellen modelliert, Transitionen vollziehen sich zeitpunktartig. Wenn eine Transition dennoch Zeit verbraucht, dann kann man sie dahingehend verfeinern, daß man einen zusätzlichen Zustand "Transition dauert an" einführt. Die Zeit vergeht im ganzen Netzwerk auf idealisiert gedachten Uhren gleichschnell. Für Transitionen kann man Vorbedingungen derart formulieren, daß sie nur dann schalten, wenn seit einem Ursprungszeitpunkt mindestens oder höchstens eine vorgegebene Zeitspanne vergangen ist. Mit diesem Netzwerk kann man die Zeit in einem Ereignisnetz simulieren. Die

Planungsmöglichkeiten sind dagegen sehr beschränkt. So kann man sich für ein System von Vor- und Nachbedingungen einen Ablauf darstellen lassen. Man kann aber nicht einem bestimmten Vorgang ein festes Zeitintervall zuweisen und von dort aus andere Vorgänge zeitlich anpassen.

Camurri und Frixione [Camurri 90] benutzen für die Arbeits- und Belegungsplanung in einem Flexiblen Fertigungssystem sog. *Structured Time-Colored Petri-Nets* (STCPN). Jeder Auftrag erhält eine andere Farbe. Die Markierungen einer Farbe stellen den Zustand eines bestimmten Auftrags dar. Eine Transition ist entweder ein Bearbeitungsprozeß oder eine Warteschleife, d.h. der Auftrag wartet eine Weile. Transitionen besitzen Bearbeitungs- bzw. Wartezeiten, einen Zustand sowie eine Menge erlaubter Schaltfarben. Stellen entsprechen Ressourcen oder Warteschlangen. Ressourcen besitzen eine Liste der Zeiten, zu denen sie noch verfügbar sind. Eine Transition kann schalten, wenn alle Inputstellen mit der gleichen Farbe oder der neutralen Farbe belegt sind und die Ressourcen unter den Inputstellen zum gegenwärtigen Zeitpunkt in einem vernünftigen Zeithorizont verfügbar sind. Um die Eingabe zu erleichtern, geben Camurri und Frixione bereits einige Makrostrukturen, variable Teilnetze, vor. Der Anwender kann daraus Instanzen generieren, die er lediglich spezifizieren muß. Für Zwecke der Belegungsplanung ist eine derartige Darstellung sicherlich hilfreich. Kompliziertere Zeitbeziehungen werden aber nur schlecht darstellbar sein. Insbesondere ist die Möglichkeit zur Darstellung von Zeitbeziehungen im Vergleich zu Allens Ansatz (vgl. S. 38f) eher rudimentär

Auf der Basis von KEE entwickelten Sanoff und Poilevey ein Subsystem für CIM zur Maschinenbelegungsplanung und Produktionssteuerung, das ebenfalls auf Time-Colored Petri-Nets aufbaut [Sanoff 91]. Die Stellen können ihre Eigenschaften von Objekten des Fabrikmodells oder von allgemeinen Stellen, etwa von einer allgemeinen Input-"Transition", erben. Die Autoren nennen ihr System deswegen ein hybrides, parametrisiertes Petri-Netz. Die Vererbungsstrategie erleichtert die Arbeit mit dem Netz. Der Anwender generiert ein Netz schneller, weil er nicht jede Stelle vollständig neu eingegeben muß.

Immer wiederkehrende Aufgaben in Roboterzellen sind als Zyklen darstellbar. U.U. sind mehrere Zyklen alternativ, so daß ein Konflikt entsteht. P. Freedman und A. Malowany wählen den entsprechenden Zyklus mit einer separaten Wissensbasis aus [Freedman 90, 91].

5.1.1.2. Hierarchien in Petri-Netzen

Unterschiedliche Detaillierungsgrade lassen sich schon mit der einfachsten Form von Petri-Netzen realisieren. Eine Transition wird verfeinert, indem man sie durch ein transitionsberandetes Teilnetz ersetzt; eine Stelle ersetzt man analog durch ein stellenberandetes Teilnetz. Ein erfolgreiches Beispiel dieser Detaillierung findet man z.B. bei [Leinemann 91] für die Simulation einer Roboterumgebung in einem Fusionsreaktor. Für Reihenfolgeplanungsprobleme wird man sich jedoch eine mächtigere Darstellung von Hierarchien wünschen: Hierarchien stellt man in Petri-Netzen indirekt durch Stellen- oder Transitionserweiterung dar. Das schließt auch Konzepte wie UND- und ODER-Verknüpfungen ein, aber der Anwender wird dabei kaum unterstützt. Eine explizite Darstellung konjunktiver und disjunktiver Hierarchien fehlt.

Eine Menge nicht-hierarchischer, miteinander in Bezug stehender Petri-Netze bezeichnen Huber, Jensen und Shapiro [Huber 91] als Seiten ("pages"). Die Semantik der hierarchischen Konstrukte wird über die Äquivalente in nicht-hierarchischen Petri-Netzen erklärt. Die Autoren nennen vier Arten, Hierarchien zu etablieren: Durch Substitution einer Transition oder einer Stelle, durch rekursive Substitution eines Knotens und durch Fusionsmengen. In allen Fällen wird ein Knoten des Netzes durch ein weiteres

Netz ersetzt, wobei die Gestaltung der Schnittstelle zwischen dem Netz und seinem Unternetz eindeutig sein muß. Zwischen Unternetzen der gleichen Ebene gibt es keine direkten Beziehungen; Querverbindungen sind verboten. Das Modell hierarchischer Petri-Netze erinnert an die hierarchische Gestaltung der Prozeduren eines Computerprogramms bei einem prozeduralen Programmieransatz.

Es ist prinzipiell also möglich, einen UND/ODER-Graphen (vgl. S. 11f) mit hierarchischen Ansätzen für Petri-Netze darzustellen. Im Vergleich zu Aktivitäts-/Zustands-Netzen (s.u.) ist das umständlich, weil Petri-Netze nur kausale, nicht aber hierarchische Beziehungen explizit darstellen. Andererseits sind Petri-Netze zur Modellierung sehr weit verbreitet.

5.1.1.3. Wissensbasierte Systeme und Petri-Netze

Petri-Netze sind von ihrer Struktur her ein Werkzeug zur Simulation, nicht zur Entscheidungsfindung. Einige Ansätze kombinieren ein regelbasiertes System mit einem Petri-Netz zu einem wissensbasierten System. Die Regelbasis wählt Wege aus und nimmt qualitative Simulationen sowie die automatische Bewertung der Simulationsergebnisse vor. Noronha und Sarme halten Petri-Netze für ein geeignetes Werkzeug zur hierarchischen Abstraktion und als Input für die Simulation [Noronha 89].

SAGE von Freedman und Malowany löst Reihenfolgeprobleme sich ständig wiederholender Operationen in einer Roboterzelle [Freedman 88]. Sie unterlegen ein farbiges Petri-Netz mit einem prologbasierten entscheidungsunterstützenden System. Seine Aufgabe ist es, an Stellen, an denen das Petri-Netz sich nicht-deterministisch verhält, eine Entscheidung herbeizuführen. Es untersucht die Konsequenzen verschiedener Sortierungen der Zelloperationen und wählt eine gute Lösung. Bei der Darstellung von Constraints unterscheiden die Autoren zwischen Vorrang- und Belegungsconstraints. Einschränkungen bei den Vorrangbeziehungen werden auf die Vorbedingungen einer Transition über eine mehrwertige Zustandsvariable abgebildet, die für jede Operation einen anderen Wert besitzt. Auch Belegungsconstraints werden über Vorbedingungen ausgedrückt, indem für jede Ressource eine andere Zustandsvariable geschaffen wird. In beiden Fällen besitzen die Zustandsvariablen symbolische statt bool'scher oder numerischer Werte. Insofern wurde auch hier eine Erweiterung vorgenommen.

Entscheidungssituationen bei der Belegungsplanung unterteilten A. Camurri und F. Frixione in vier Kategorien: 1. topologische Konflikte, 2. Farbkonflikte, 3. nichtdeterministische Dauern, 4. gemischte Situationen [Camurri 90, S. 61f]. Ein topologischer Konflikt liegt vor, wenn gleichzeitig zwei Transitionen feuern könnten, die Belegung ihrer gemeinsamen Inputstellen aber nur für eine Transition ausreicht. Bei Farbkonflikten auf einer Stelle mit mehr als einer Farbe braucht der Planer eine Entscheidung über die Farbe, die er als nächste feuern soll. Die Konfliktlösung wird an ein System delegiert, das außerhalb des Petri-Netzes steht, in diesem Fall eine Wissensbasis.

Aufträge, Ressourcen und Constraints in der Maschinenbelegungsplanung versuchen Carlier, Chretienne und Girault mit einem einzigen Formalismus darzustellen [Carlier 84]. Die mit Petri-Netzen möglichen Analysen sind damit auf ihr Modell übertragbar und beziehen auch Aufträge, Ressourcen und Constraints mit ein. Ein Ereignisgraph stellt Reihenfolgeconstraints jeder Maschine in Unternetzen dar. Auftragsgraphen verbinden die Unternetze um zu verhindern, daß die Reihenfolgerestriktionen der Operationsfolge eines Auftrags verletzt werden. Voraussetzung zum Aufbau des Petrinetzes ist jedoch, daß all diese Beziehungen bekannt sind. Es dient zur Simulation, kann aber keine Entscheidungen treffen.

5.1.2. Aktivitäts/Zustands-Netze (A/S-Netze)

A. Sathi, M. S. Fox und M. Goldberg beschreiben ein Modell zur Repräsentation von Wissen über Aktivitäten für das Projektmanagement [Sathi 85]. Es ist nach Ansicht der

Autoren ebenso auf andere Problembereiche anwendbar, z.B. Maschinenbelegungsplanung, Fahrzeug- und Robotersteuerung. Die Autoren wollen verschiedene bisherige Ansätze integrieren. Zu den Schlüsselkonzepten gehören

- ein verallgemeinernder Ansatz, um Wissensstrukturen darzustellen,
- Repräsentationen von Aktivität, Zustand, Ziel
- Zeit und Kausalität.

Sechs strukturelle Beziehungen zwischen Objekten werden bereitgestellt: *is-a* als Default, *subset-of* für die Klassifikation, *elaboration-of* für die Ausarbeitung, *part-of* für die Dekomposition, *revision-of* für die Weiterentwicklung und *instance* für die Individuation.

Aktivität ist die "basic unit of action" zum Zwecke der "transformation of the world from one situation or state to another" [Sathi 85, S. 537]. *Aggregierte Aktivitäten* werden mit *has-sub-activity*, einer Spezialisierung von *has-part*, disjunktiv oder konjunktiv dekomponiert. *Einzelaktivitäten* können über *has-elaboration* weiter ausgearbeitet werden. Die Autoren betonen, daß eine Unteraktivität grundsätzlich mehreren Oberaktivitäten zugeordnet werden kann. Die hierarchischen Geflechte sind also nicht notwendigerweise Bäume im strengen Sinne.

Ein Zustand "defines a fact which holds as of some point in time (...) or for a period of time" [Sathi 85, S. 538]. Analog zu Aktivitäten werden auch Zustände in Einzelzustände und aggregierte Zustände unterteilt. Ein aggregierter Zustand ist eine zusammenfassende Beschreibung von Einzelzuständen, die gelten müssen, damit die Voraussetzungen zur Ausführung einer Aktivität gelten, oder eine Beschreibung von Einzelzuständen, die nach der Ausführung einer Aktivität gelten werden.

Die dekompositionelle *has-part* Beziehung wurde für die Beziehung zwischen Zuständen zu *has-sub-state* verfeinert. Aggregierte Zustände entsprechen *UND*-Knoten eines Netzwerks, weil jede ihrer Unterbedingungen erfüllt sein muß, damit der aggregierte Zustand gilt. Einzelzustände entsprechen *ODER*-Knoten, da wenigstens eine Ausarbeitung zugesichert werden muß, damit der Einzelzustand gilt.

Einzelzustände werden weiter zu Besitz- und Zustandsprädikaten spezialisiert. Zustandsprädikate beschreiben den Zustand der Aktivität, Besitzprädikate verlangen, daß eine Ressource über einen endlichen Zeitraum für die Aktivität zur Verfügung steht, gewissermaßen im Besitz der Aktivität ist. Unter einem Aktivitätscluster versteht man eine Aktivität mit zwei assoziierten Zustandsbäumen: Der eine Baum besteht aus Zuständen, welche die Aktivität ermöglichen (Vorbedingungen), der andere aus Zuständen, die von der Aktivität verursacht werden (Nachbedingungen).

Das Konzept der Instantiation und Manifestation unterscheidet drei Arten von Aktivitäten und Zuständen: Prototypen beschreiben Standardprozeduren, z.B. Richtlinien zur Ausführung einer Tätigkeit. Instanzen sind Aktivitäten und Zustände für den Einzelfall. Instanzen von Aktivitäten können tatsächlich ausgeführt werden. Manifestationen stellen Instanzen zu einem bestimmten Zeitpunkt oder in einer bestimmten Phase des Planungsfortschritts dar. Um den Fortschritt darzustellen, kann eine Instanz mehrere Manifestationen besitzen.

Kausale und temporale Beziehungen verbinden Aktivitäten mit Zuständen. *Enable* und *Cause* verknüpfen Zustände mit einer Aktivität. Zustände im Sinne einer Vorbedingung, die eine Aktivität ermöglichen (*enable*), müssen wahr sein, damit die Aktivität wahr werden kann. Ist eine Aktivität wahr, dann werden die Nachbedingungen, die sie über die Beziehung *cause* verursacht, zugesichert. Schließlich ist eine Relation *cause-enable* notwendig, die zwei Zustände verbindet, die einander direkt bedingen. Zur temporalen Darstellung sind die üblichen Konzepte qualitativer und quantitativer Zeit,

also Zeitpunkt, Zeitintervall und Zeitgranularität notwendig. Besitzprädikate sind Zustände, die nur über einen begrenzten Zeitraum hin gelten.

Bei den A/S-Netzen[11] handelt es sich um eine außerordentlich mächtige, sehr flexible Darstellung, die mehrere Arten von Wissen vereinigt. Insbesondere werden hierarchische, kausale und temporale Bezüge explizit und auf einheitliche Weise dargestellt. Besonders attraktiv erscheint darüber hinaus die Möglichkeit, Alternativen über *ODER*-Knoten darstellen zu können.

5.1.3. Objekte und Hierarchien

Die Identifizierung von Objekten und die Klassifizierung der Objekte hängt vom Problembereich eines Reihenfolgeproblems ab. Für Traveling Salesman Probleme kommen als Objekte die Knoten und Kanten des Problems in Frage. Außerdem kann man Nebenbedingungen, die an die Nutzung einer Kante gestellt werden, als Objekte formulieren. In der Maschinenbelegungsplanung zählen zu den Objekten meist Ressourcen, Aufträge und Operationen, Werkstücke sowie Datenelemente der Umgebung (Arbeitspläne, Teilebeschreibung etc.). Objekte werden oft in Klassen zusammengefaßt, die wiederum Teil einer Klasse sind. Daraus entstehen Hierarchien, zu deren Modellierung man zwischen den Objekten Relationen schafft. Sehr breite Anwendung finden taxonomische (a-kind-of) und kompositionelle Relationen (part-of). Darüberhinaus ist es oft möglich, über Relationen Slots und Werte durch die Hierarchie hindurch weiterzugeben.

5.1.3.1. Objekthierarchien in einem Aktivitäts-Zustands-Netz

Der Maschinenbelegungsplaner ISIS von Fox (vgl. [Fox 84]) stellt neben den üblichen Klassen für Maschinenbelegungsprobleme alle Constraints und Relationen ebenfalls als Objekte dar und verbindet sie mit vielfältigen Relationen. Er verwendet für den Aufbau ein Schichtenmodell. Sathi nennt für das CALLISTO-Projekt, einem ISIS-Vorgänger, fünf Schichten. Auf der Implementierungsschicht werden die Datenstrukturen der untersten Ebene definiert. Die logische Schicht erklärt die logische Interpretation von Informationen wie das Schema-Slot-Wert Tripel, das als Zusicherung eines Wertes für ein Attribut interpretiert wird. Die epistomologische Ebene definiert Konzepte wie Vererbung, Menge, Prototyp und Aggregation. Dazu zählen auch sechs strukturelle Relationen, aus denen alle anderen Relationen abgeleitet werden. Sie werden in ihrer Transitivität, Reflexivität, Symmetrie und in ihrem Definitions- und Bildbereich erklärt. Die meisten framebasierten Ansätze vereinfachen dagegen das Objekt "Relation", um es für die Inferenz leichter benutzbar zu machen. Oft werden auch nur zwei Klassen von Relationen angeboten: *is-a* für taxonomische Hierarchien und *part-of* für kompositionelle. ISIS mit sechs hierarchischen Grundrelationen, aus denen sich alle anderen Relationen entwickeln lassen, ist hier vollständiger [Sathi 85]. Relationen werden mit Frames explizit als Objekte dargestellt. Sie haben alle Eigenschaften, die andere Schemen auch besitzen: Relationen können vererbt werden, Relationen können Constraints oder Wissen zugeordnet werden. Insbesondere ist es möglich, aus seinem Ansatz alle notwendigen Relationen zwischen Objekten der angeführten Objektklassen abzuleiten.

Die vierte Ebene, der Semantic Layer, enwirft Konzepte für Aktionen, Objekte und Agenten. Zu den aktionsbezogenen Konzepten zählen die Definitionen von Aktivität, Zustand, kausalen und temporalen Beziehungen. Objektorientierte Konzepte sind die Verfeinerung oder Ausarbeitung von Objekten und die dazugehörigen Constraints. Der Besitz von Objekten wird in einem agentenorientierten Objekt erklärt.

Die Domänenebene schließlich definiert, welche Aktivitäten, Zustände und Ziele das konkrete Problem enthält. Sie spezifiziert allgemeine Relationen der epistomologischen Ebene und entwirft Propagationsmechanismen [Sathi 85].

[11] "A/S" bedeutet "Activity/State", auf deutsch "Aktivität/Zustand"

Neben der Objektebene besitzt ISIS auch zwei Suchhierarchien, auf die im Absatz über Suchstrategien (s.u.) detailliert eingegangen wird. Sie sind von den Hierarchien der Objekte zu unterscheiden.

5.1.3.2. Modellierung organisatorischer Hierarchien

Das PATRIARCH System wurde für umfangreiche und komplexe industrielle Umgebungen geschaffen. Es ist ein Echtzeitsystem zur integrierten Arbeitsplanung und Maschinenbelegungsplanung. Bei PATRIARCH werden die Planungsebenen entsprechend den Planungsebenen des Betriebs entworfen. Das schließt organisatorische und temporale Gesichtspunkte ein [Lawrence 86]. Vorgegeben sind dagegen bei RESS-I die Planungsebenen, die sich in Grob-, Mittel- und Feinplanung aufteilen lassen [Liu 90].

5.1.3.3. Hierarchien in Blackboard-Architekturen

Unter einem Blackboard versteht man die globale Datenbasis einer Blackboard-Architektur, die Zwischenergebnisse aus sogenannten Wissensquellen abspeichert. Die Wissensquelle stellt einen kognitiven Prozeß dar, der unabhängig von anderen Wissensquellen abläuft. Wissensquellen kommunizieren nur über das Blackboard. Sie besitzen einen Bedingungs- und einen Ausführungsteil. Im Bedingungsteil ist beschrieben, unter welchen Umständen der Aktionsteil ausgeführt wird. Der Aktionsteil modifiziert und generiert anschließend Einträge im Blackboard. Die Einträge repräsentieren die Objekte des Systems oder ihre Beziehungen untereinander. Eine Blackboard-Architektur besteht aus Wissensquellen, die unabhängig voneinander Einträge in das Blackboard vornehmen. Sie werden von einem intelligenten Steuermechanismus angestoßen (vgl. [Kloth 88]).

OPIS von P. S. Ow und S. F. Smith [Ow 87], ein Nachfolger von ISIS, besitzt zur Problemdekomposition eine Hierarchie von Wissensquellen ("Knowledge Sources"). Sie sind Problemlösungskomponenten, die das Problem zerlegen und Teilprobleme an untergeordnete Komponenten weitergeben. Ein *Top-Level Manager* entscheidet opportunistisch, ob ressourcen- oder auftragsorientiert zu planen ist und gibt entsprechende Teilprobleme an den *Resource Scheduler* oder den *Order Scheduler* weiter. Der *Resource Scheduler Manager* koordiniert die Arbeit der Wissensquellen *Resource Assigner* (Zuweisung von Aufträgen), *Resource Reserver* (Reservierungen) und *Sequence Developer* (Reihenfolgeplaner). Analog koordiniert der *Order Scheduler Manager* die Arbeit des *Coarse Scheduler* (Grobplanung), *Reservation Reserver* (Reservierung) und des *Detailed Scheduler* (Detailplanung) [Ow 87, S. 437ff]. Durch die Aufteilung des Planers in verschiedene Problemlösungskomponenten strukturiert OPIS gleichzeitig den Problemlösungsprozeß. In eine ähnliche Richtung geht auch der Ansatz von Dagli und Hoffman [Dagli 89].

Der Maschinenbelegungsplaner CIBELE von P. Calvo, T. Caminati, P. Fraternali und S. Miriello besitzt ebenfalls eine Blackboard-Architektur mit Suchagenten [Calvo 89]. Sie erhalten bestimmte Aufgaben in einem linearen Problemlösungsprozeß; zwischen zwei Agenten gibt es keine Konkurrenz. Jeder Agent arbeitet in einem "Context" des Lösungsprozesses und diese Kontexte sind als Baum angeordnet. Ein Kindkontext ist eine Weiterentwicklung seines Vaterkontextes. Der Kindkontext wird entweder als Bestandteil der endgültigen Lösung akzeptiert oder verworfen. Im zweiten Fall geht ein Backtrackingalgorithmus zurück zum Vaterkontext und versucht, die Lösung in eine andere Richtung weiterzuentwickeln.

Das System SONIA von A. Collinot, C. Le Pape und G. Pinoteau [Collinot 88] hat eine Blackboard-Architektur mit zwei Blackboards. Ein *Domain Blackboard* enthält Informationen, die mit der Entwicklung von Maschinenbelegungen zu tun haben. Dazu zählen die Speicherung von Ergebnissen, die Kapazitätsanalyse und die Lösung von Konflikten. Das Blackboard zur Steuerung sammelt Informationen, die mit der Steuerung der Planung im Zusammenhang stehen. Dazu gehören die Problemanalyse, die

Aufteilung des gegebenen Problems in Unterprobleme, ein Modul mit Heuristiken zur Problemlösung und Regeln über ihre Anwendbarkeit, ein Agendamechanismus und Speicher für Planungspolitiken und die ausgewählten Aktionen. Darüberhinaus besitzt SONIA eine Menge von Wissensquellen. Sie dienen zur prediktiven oder reaktiven Planung[12]. Jedes Gebiet eines Blackboards kann eine Wissensquelle aufrufen, die das Blackboard wiederum verändert [Collinot 88, S. 92f]. In eine ähnliche Richtung gehen CRONOS-II und sein Nachfolger CRONOS-III, zwei Werkzeuge zur Maschinenbelegungsplanung, die für eine breite Palette industrieller Anwendungen gedacht sind [Canzi 90]. Die Problemlösungsmodule sind aber nicht wie bei SONIA hierarchisch strukturiert, sondern stehen auf gleicher Ebene nebeneinander. Dafür werden Arbeitspläne in strengen UND/ODER-Bäumen in immer kleinere Aufgaben geteilt, denen Constraints zugeordnet werden.

Die Blackboard-Architektur des Maschinenbelegungsplaners HORIZON von Reece besitzt vier Wissensquellen zur Problemanalyse und Wartung der Belegungsplanung, die nicht wie bei SONIA alternativ, sondern der Reihe nach aufgerufen werden [Reece 90]. Eine fünfte Wissensquelle enthält Heuristiken über die Problemdomäne, die ein Anwender austauschen oder modifizieren soll, um Gelegenheiten zu nutzen, die eine bestimmte Anwendung bietet.

5.1.3.4. Architekturen hybrider Systeme

Wissensbasierte Systeme greifen nur in wenigen Fällen noch auf eine einzige Form der Wissensdarstellung und Inferenz zurück. Die Vorteile einer einzigen Darstellungs- und Schlußfolgerungsmethode liegen in der Einfachheit und leichteren Analyse des Systemverhaltens. Nachteil ist, daß Wissen oft nicht angemessen und klar dargestellt werden kann. Damit wird die Erstellung des Systems sehr aufwendig, es sei denn, man begnügt sich mit einer eingeschränkten Problemdarstellung.

Deswegen vereinigt man oft vielfältige Wissensstrukturen in Entwicklungswerkzeugen und -umgebungen. Der Anwender erhält die Möglichkeit, die geeignetste Wissensstruktur für die Darstellung der unterschiedllichen Domäneninformationen zu benutzen, was eine klarere Systemorganisation fördert und die schrittweise Programmerstellung und -wartung unterstützt. Das hat aber auch Nachteile: Es gibt keine eindeutige Art mehr, Wissen darzustellen, das System wird komplexer und seine Korrektheit ist nur sehr schwer zu beweisen.

Hindi und Singh nennen drei Möglichkeiten, Wissen, mathematische Modelle und algorithmische Methoden in einem System zu kombinieren [Hindi 90]:

1. Ein wissensbasiertes System kann als globale Steuerinstanz fungieren und Entscheidungen von lokaler Bedeutung an Algorithmen delegieren.
2. Ein globales System, das auf Algorithmen basiert, benutzt wissensbasierte Systemtechniken, um lokale Entscheidungen zu treffen.
3. Ein wissensbasiertes System erzeugt einen globalen Plan, dessen Fehler von Algorithmen behoben werden.

Oft findet man Kombinationen von Regeln und Frames. Frames gruppieren zusammengehörige Informationen und repräsentieren hierarchische oder netzwerkartige Strukturen, z.B. Vorrangrelationen, Netzwerke von Aktivitäten oder Beschreibungen auf verschiedenen Abstraktionsebenen. Die logikbasierte Repräsentation hilft bei der Sammlung anderer problembeschreibender Fakten, wie dem Aufbau einer Regelbasis heuristischer Reihenfolgeregeln, der Deduktion und der Beantwortung von Fragen.

[12]Reaktive Planung: Der Vorgang des Umplanens, wenn ein unerwarteter Wechsel in der Umgebung während der Planausführung den alten Plan ungültig macht [Noronha 91, S. 167].

CORTES von Fox, ein Nachfolgeprojekt von ISIS, ist ein integrierter Rahmen zur Produktionsplanung, Maschinenbelegungsplanung und Produktionssteuerung [Fox 90]. Es verbindet Module für die Planung, die Maschinenbelegung, die Modellierung einer Fabrik und für Unsicherheitsanalysen. Das Ergebnis wird an Dispatcher weitergegeben, welche die zeitliche Zuweisung von Operationen an die Maschinen vornehmen.

Oft wird als Rahmen für ein hybrides System eine Blackboard-Architektur verwendet, bei der Wissensquellen oder Abstraktionsebenen unterschiedliche Wissensstrukturen besitzen. OPIS besitzt eine Hierarchie sich ergänzender und konkurrierender Wissensquellen, die über eine gemeinsame Datenstruktur kommunizieren. Das Expertenwissen wird damit verteilt [Ow 87]. SONIA besitzt zwei Blackboards, eines für die Koordination von Informationen der Problemdomäne, ein anderes für die Steuerung der Problemlösungsstrategie [Collinot 88].

Ein allgemeiner Reihenfolgeplaner hat mit einer Blackboard-Architektur und den zugehörigen, unterschiedlichen Wissensquellen die Möglichkeit, sehr verschiedenes Wissen zu strukturieren. Darüberhinaus kann auch die Wissensstruktur unterschiedlich sein, sofern man sich auf ein Kommunikationsprotokoll einigt, mit dem auf dem Blackboard kommuniziert wird.

5.1.3.5. Zusammenfassung der hierarchischen Konzepte

Die Identifizierung von Objekten und die Klassifizierung der Objekte hängt vom Problembereich eines Reihenfolgeproblems ab. Für Traveling-Salesman-Probleme kommen als Objekte die Knoten und Kanten des Problems in Frage. Außerdem kann man Nebenbedingungen, die an die Nutzung einer Kante gestellt werden, als Objekte formulieren. In der Maschinenbelegungsplanung zählen zu den Objekten meist Ressourcen, Aufträge und Operationen, Werkstücke sowie Datenelemente der Umgebung (Arbeitspläne, Teilebeschreibung etc.). Objekte werden oft hierarchisch klassifiziert. Zur Modellierung schafft man zwischen den Objekten Relationen.

In neueren Ansätzen findet man drei Arten von Hierarchien: Objekthierarchien, Darstellungshierarchien und Wissenshierarchien. Objekthierarchien, mit der Objekte zueinander in Bezug gesetzt werden, sind Teil eines umfassenderen relationalen Konzepts. Über hierarchische Beziehungen hinaus stehen Objekte meist in kausalen oder zeitlichen Beziehungen zueinander. Objekthierarchien erlauben bestimmte Arten der Inferenz wie Verfeinerung, Ausarbeitung oder Vererbung. Besonders häufig findet man kompositionelle (part-of) und taxonomische Hierarchien (is-a). Als Objekte kommen nicht nur physikalische oder betriebsorganisatorische Ganzheiten (wie bei PATRIARCH [Lawrence 86]) in Frage. Sie eignen sich darüberhinaus zur internen Organisation des Planers in Darstellungshierarchien.

Darstellungshierarchien ermöglichen die schrittweise Spezifizierung eines allgemeinen Planers hin zu speziellen Planern. Im System SRL von M. S. Fox heißen die verschiedenen Hierarchieebenen Schichten [Fox 85]. Unterschiedliche Wissensquellen schließlich können zu Wissenshierarchien zusammengefaßt werden. Sie stehen in engem Zusammenhang mit der Möglichkeit, opportunistisch zu schließen, d.h. der Auswahl einer geeigneten Wissensquelle.

Einerseits verlangen Hierarchien von einer Repräsentationssprache komplexere Darstellungsmöglichkeiten und komplizieren den Inferenzprozeß. Andererseits ermöglichen sie, vorgegebene hierarchische Strukturen darzustellen und den Problemlösungsprozeß erheblich zu vereinfachen. Sowohl nichtlineares (s.u.) wie auch hierarchisches Planen läßt sich mit bereits vorgegebenen Hierarchien, insbesondere mit Bäumen effizient realisieren. Sie unterstützen so die Modellierung von Reihenfolgeproblemen.

5.2. Lösungsstrategien

Die Planungsmethoden des Operations Research und der Künstlichen Intelligenz überlappen sich häufig. T. J. Grant sieht den wesentlichen Unterschied darin, daß das Operations Research einen Systemansatz verfolgt, während man in der Künstlichen Intelligenz versucht, die Erfahrungen und Fähigkeiten des Menschen zu modellieren [Grant 86, S. 45].

Der erste Abschnitt gibt einen Überblick über grundlegende Inferenzmechanismen zur Planung. Spezielle Verfahren für Probleme der Reihenfolgeplanung sind Gegenstand des zweiten Abschnitts. Bekannteste Anwendung der Schlußfolgerungsmechanismen sind wissensbasierte Systeme. Der dritte Teil beschreibt Anwendungen in der betrieblichen Praxis und zeigt, daß bisher im wesentlichen einfache prioritätsgesteuerte Verfahren verwendet werden.

5.2.1. Kalküle für die Planung

J. Hertzberg faßte die wichtigsten Planerstellungsmethoden, die auf dem Situationskalkül beruhen, zusammen [Hertzberg 86]. Sehr grundsätzlich analysierte Gilhooly das Planen im Situationskalkül. Er entwickelt es aus der behavouristischen (verhaltensorientierten) Denktheorie [Gilhooly 82]. Detailliert beschreibt N. Nilsson Planungssysteme [Nilsson 82, Kap. 7, 8]. Der Planung im Situationskalkül liegen eine Menge S der Situationen und eine Menge O der Operatoren zugrunde. Jedem atomaren Operator $o \in O$ sind Vor- und Nachbedingungen P und $Q \subseteq S$ derart zugeordnet, daß die Implikation gilt:

$$P(s) \rightarrow Q(o(s)).$$

Die Operatoren sind unmittelbar ausführbar. Ein Problem ist durch eine Menge von Start- und Zielsituationen gegeben. Unter der Lösung eines Problems versteht man eine Sequenz von Operatoren, die die Start- in die Zielsituation überführt.

Um die Lösungssequenz von Operatoren zu erhalten, wurden zwei grundsätzliche Strategien entwickelt. Beim einstufigen Planen kann man versuchen, durch Permutation der Operatoren eine Sequenz zu finden, die das Problem löst. Der Aufwand dafür ist exponentiell und bereits für kleinere Probleme undurchführbar, weil alle Sequenzen durchprobiert werden müssen. Mehrstufiges Planen bedeutet, zunächst eine Grobplanung für das gesamte Problem zu entwickeln, dann das Problem aufzuteilen und für die Teile die Lösung zu verfeinern. Zwischen Teilplänen entstehen leicht Interaktionsprobleme, die zusätzlich zu lösen sind. Der Planungsprozeß selbst kann so umfangreich werden, daß man ihn mit einem Metaplan strukturiert, der nach den gleichen Regeln erzeugt werden kann.

Hertzberg sieht die Probleme der Planung im Situationskalkül hauptsächlich in Unzulänglichkeiten des Situationskalküls selbst [Hertzberg 86, S. 157ff]:

- Voraussetzung für die sog. "Strips assumption" ist, daß wirklich alles, was ein Operator ändert, angegeben wird. Für reale Probleme kann das sehr mühsam sein.
- Zeitkontinuierliche Prozesse werden nicht angemessen dargestellt. Vielmehr geht man davon aus, daß jeder Operator sofort wirksam wird.
- In der realen Welt kommt es oft zu Unstimmigkeiten mit der Planung, weil stets nur ein Ausschnitt der Welt dargestellt werden kann. Es fehlt eine Rückkopplung und die Möglichkeit zum Umplanen. Formal können neue Fakten durch nichtmonotones Schließen eingebaut werden. Umplanmöglichkeiten bieten NOAH von E. D. Sacerdoti [Sacerdoti 77] und Wilkins' SIPE [Wilkins 83] (s.u.).

Georgeff und Ingrand bauten einen Planer, der bei Fehlen von Wissen über die Umgebung auf Defaultpläne zurückgreift [Georgeff 89]. In dem Augenblick, in dem dieses Wissen wieder zur Verfügung steht, werden die Defaultpläne überschrieben.

5.2.1.1. Nichtlineares Planen

Domänenunabhängiges Planen mehrerer Teilziele begann 1973 mit HACKER von G. J. Sussman [Sussman 73]. Er versucht, Abhängigkeiten zwischen Teilplänen zu erkennen und sie dann so umzuordnen, daß sich die Interaktionen auflösen. Sussman konnte das noch nicht garantieren, aber seine Arbeit führte zu einer Serie schneller Entwicklungen wie WARPLAN von D. Warren [Warren 74] und INTERPLAN von A. Tate [Tate 74]. Waldinger baute Teilpläne unter Beibehaltung der Ergebnisse, die in der Zielsituation verlangt wurden, in vorgegebene Pläne ein [Waldinger 75]. Er hängte den Teilplan an das Ende des Gesamtplans. Kommt es zu Unverträglichkeiten, wird der Teilplan durch den Gesamtplan solange nach vorn durchgeschoben, bis sich die Interaktionsprobleme auflösen.

Beim nicht-linearen Planen wird eine Reihenfolge nur dann festgelegt, wenn es dafür einen zwingenden Grund gibt. Interaktionen zwischen Teilplänen werden als Constraints über die Objekte beschrieben. Sie werden erst dann aufgelöst, wenn eine eindeutige Entscheidung möglich ist. Dies wird durch Constraint-Propagierung ermittelt. Eine Reihenfolge zwischen zwei Operationen wird erst dann bestimmt, wenn ihre Interaktionen entdeckt worden sind. "The important idea, due to Sacerdoti, is that a plan (at least while it is being constructed) does not have to specify fully the order of execution of its steps. In other words, a plan is only a partial order on steps; this is what is meant by *nonlinear* planning." [Chapman 87, S. 335].

Den ersten nichtlinearen Planer NOAH baute 1975 E. D. Sacerdoti [Sacerdoti 77]. Ein Plan wird als eine a priori teilweise geordnete Folge von Operatoren angesehen. Man erzeugt partiell geordnete Teilpläne und erweitert dann die Ordnung so, daß ein Gesamtplan entsteht, der frei von Interaktionsproblemen ist. Planvariablen werden erst so spät wie möglich instanziiert. Tate erweiterte NOAH. Sein Programm NONLIN wurde von einem Backtracking-Algorithmus gesteuert [Tate 76].

Stefik fügte dem 1982 Constraints zu [Stefik 80, 81]. Stefiks MOLGEN sammelt im Laufe der Planung immer mehr Constraints, die den Lösungsraum einschränken. Interaktionen können so leicht ermittelt und aufgelöst werden. Eine Verallgemeinerung nichtlinearen Planens gelang D. Chapman [Chapman 87]. Für seinen Planer TWEAK bewies er die Korrektheit und die Komplexität. TWEAK besteht aus drei Schichten: der Planrepräsentation, der Lösungsstrategie und einer globalen Steuerung. Der Planer stellt zwei Arten von Constraints auf. Temporale Constraints beschreiben die Reihenfolge der Operatoren. "Codesignation Constraints" binden eine Variable an eine Konstante oder an eine andere Variable. Bei der Suche werden schrittweise unzulässige Alternativen ausgeschlossen. Ein unvollständiger Plan p heißt *notwendig*, wenn er für alle weiteren Vervollständigungen wahr ist. p heißt *möglich*, wenn er nur für einige Vervollständigungen wahr ist. Zwei Aussagen eines unvollständigen Plans binden notwendigerweise, wenn sie für alle Vervollständigungen gebunden werden können, unabhängig davon, welche Constraints dazukommen. Eine bisher unbekannte Aussage heißt in einer gegebenen Situation wahr, wenn sie auf eine andere Aussage gebunden werden kann, deren Wahrheit in der Situation bereits zugesichert wurde. Chapman entwickelt einen polynomialen Algorithmus, mit dem die Wahrheit für eine Aussage in einer Situation überprüft werden kann.

TWEAK baut nicht-lineare Pläne durch Aufstellen von Constraints auf. Ein Plan wird schrittweise durch Spezifizierung von Teilbeschreibungen spezifiziert. Der Vorteil ist, daß keine Entscheidung für Objekteigenschaften gefällt werden muß, solange dafür

keine logisch zwingende Begründung vorliegt. Der Planer trifft keine Annahmen, nur um den Problemlösungsprozeß weiterzubringen. Dieses Verfahren eines Planaufbaus heißt Constraintpropagierung. Legt ein Planer dagegen vorzeitig Variablen fest, plant er linear.

5.2.1.2. Hierarchisches Planen

Für ein Problem ermittelt man in einem ersten Schritt eine Groblösung oder abstrakte Lösung und verfeinert sie im zweiten Schritt heuristisch. Abstrahieren kann man von Situationsmerkmalen oder von Operatoren.

Für die Abstraktion von Situationen werden die Merkmale einer Situation priorisiert. Eine Planung löst zunächst das Problem für die wichtigsten Merkmale. Danach werden inkrementell weniger wichtige Merkmale dazugenommen und verplant. Von oben gibt man ein Planskelett nach unten und verfeinert es dort. Scheitert die Verfeinerung, wird auf der höheren Ebene nach einer Alternative gesucht. Ein wichtiger Vertreter dieser Planungsform ist ABSTRIPS von Sacerdoti [Sacerdoti 73]. Die Situationsabstraktion kann lange Pläne beträchtlich effektiver erzeugen als die einstufige Planung. Problematisch ist, daß bereits auf der abstrakten Ebene ein möglichst korrekter Plan gefunden werden muß, um zeitaufwendiges Backtracking zu vermeiden. Dies wird noch schwieriger, wenn man optimale Pläne sucht. Ein abstrakter Plan muß eine optimale Konkretisierung ermöglichen.

Abstrakte Operatoren lassen sich zu Sequenzen konkreter Operatoren dekomponieren. Schon auf abstrakter Ebene erkennt man Reihenfolgerestriktionen. Auf einer abstrakten, weniger komplexen Ebene erfolgt die wesentliche Planungsarbeit. Sacerdotis NOAH verbindet nichtlineares Planen mit der Operatorabstraktion [Sacerdoti 77]. Stefiks MOLGEN enthält darüber hinaus die Möglichkeit zur Situationsabstraktion [Stefik 80, 81]. Eine andere Möglichkeit ist, mit Skeletalplänen zu arbeiten, die bereits eine Grobstruktur des endgültigen Plans vorgeben. Friedlands MOLGEN gibt einen Allgemeinen Plan zur Analyse der DNA-Struktur einer molekulargenetischen Struktur vor [Friedland 79, S. 287]. Jeder der vier Schritte ist die Wurzel eines Baums von Spezifizierungen, die ausgewählt werden müssen.

5.2.1.3. Meta-Planen

Die Planungsstrategie kann genauso wie der eigentliche Plan entwickelt werden. Stefiks MOLGEN stellt vier Metaoperatoren zur Verfügung [Stefik 81]: 1. Der Operator FOCUS versucht, einen Planungsoperator auf den bisher entwickelten Plan anzuwenden. RESUME nimmt die Planungsarbeit an einem Operator wieder auf. GUESS instanziiert eine Planvariable. UNDO macht eine Instanziierung durch GUESS rückgängig.

5.2.1.4. Constraintsatisfizierung

Mit Hilfe einer Constraint-gesteuerten heuristischen Suche werden Objekte für die Planung ausgesucht. Heuristiken, die Objekte suchen und auswählen, steuern die Suche. Sie entstanden aus Metawissen über den Suchprozeß selbst. Bei Constraintsatisfizierungsproblemen gibt es eine Menge von Variablen mit vorgegebenen Definitions- und Abbildungsbereichen. Zu dieser Beschränkung kommen Konsistenzrelationen zwischen Variablen. Für die Variablen wird eine Wertebelegung, eine Instanziierung, gesucht, die mit dem Definitions- und Abbildungsbereich und den Konsistenzrelationen übereinstimmt. Um die Lösungssuche effizient zu gestalten, schränkt man mit bestimmten Techniken den Suchraum ein.

ISIS von M. S. Fox [Fox 84] unterscheidet zwischen harten und weichen Constraints. Hart sind Constraints, die die Zulässigkeit der Lösung zusichern (z.B. Vorrangbeziehungen zwischen Operationen). Weiche Constraints drücken Präferenzen für die Lösung

aus (Fristen, Bevorzugung einer Maschine). Sie sind relaxierbar, falls sonst keine Lösung gefunden werden kann. Das ist dann der Fall, wenn zwei Constraints sich gegenseitig widersprechen, beispielsweise wenn die Einhaltung einer Frist der Kostenminimierung entgegensteht. Eine Nützlichkeitsfunktion vergleicht die Constraints und relaxiert den Constraint mit dem geringeren Nutzen. Ein Constraintpropagationsalgorithmus untersucht Constraints hierarchisch auf verschiedenen Abstraktionsebenen (s.u.).

Die Unterscheidung zwischen harten und weichen Constraints, die Klassifizierungsmöglichkeiten und die Relaxierungstechniken machen die Constraintsatisfizierung zu einem sehr flexiblen Darstellungsverfahren. Eine Bedingung muß lediglich formuliert werden und wird dann dem entsprechenden Objekt zugeordnet. Mit Constraintsatisfizierung kann der Suchraum erheblich eingeschränkt und die Lösungssuche effizienter gestaltet werden. Die Methode macht damit Reihenfolgeplaner effizienter, erhöht ihre Flexibilität und erlaubt in vielen Fällen eine geeignetere Wissensdarstellung.

5.2.1.5. Bewertung der Planungskalküle

Vorteil der hier vorgestellten Verfahren ist, daß sie prinzipiell für alle Arten von Reihenfolgeproblemen geeignet sind. Im Gegensatz zu Verfahren des Operations Research kann der Problembereich sehr unstrukturiert sein, weil allen KI-Planern eine Suchstrategie zugrunde liegt. Sie finden stets die Lösung, wenn auch mit beträchtlichem Aufwand, der meist von exponentieller Größenordnung ist. Um Effizienz zu gewinnen, kann man nichtlinear, hierarchisch oder mit Metaplänen arbeiten. Ist das Problem nur groß genug, verbessert jede der Methoden die Lösungsstrategie. Um die Effizienz weiter zu erhöhen, müßten die Suchstrategien verbessert werden.

Die hier vorgestellten Methoden sind also für einen Reihenfolgeplaner sehr interessant, sie sind aber in ein System mit geeignter Wissensdarstellung und verbesserten Suchstrategien einzubetten.

5.2.2. Planungsmethoden

Außer für das Problem kürzester bzw. längster Wege sind die meisten in der Realität auftretenden Reihenfolgeprobleme NP-vollständig. In den letzten zwanzig Jahren hat sich die Forschung darum bemüht, die Grenze zwischen NP-vollständigen Problemen und Problemen polynomialer Größenordnung genauer zu ziehen, Heuristiken auf ihre Tauglichkeit hin zu überprüfen und für einzelne Probleme schnellere Verfahren zu entwickeln.

In der Praxis der Fertigung spielt die Frage der Optimalität meist keine so große Rolle, man kann sich mit Heuristiken zufrieden geben. Das liegt daran, daß unvorhergesehene Ereignisse aus jeder noch so gut optimierten Planung Makulatur machen können. Dazu kommt, daß in der Praxis oft spezifische Randbedingungen und Zusammenhänge eine Rolle spielen, die nicht immer einfach zu modellieren sind und auf die die optimierenden Verfahren erst einmal transformiert werden müssen. Wie das aussehen kann, beschreiben die folgenden Abschnitte.

Einige Planer klassifizieren Planungsprobleme und suchen intelligent ein OR-Verfahren aus. Andere versuchen, die Differenz zwischen Ausgangszustand und Ziel zu minimieren. Oft werden beide Ansätze integriert. Spezielle Fragen wirft die zeitliche Planung auf, die besonders unter Berücksichtigung der Unschärfe behandelt wird.

5.2.2.1. Lösungen für die Maschinenbelegungsplanung

Zu den Standardwerken für die Maschinenbelegung zählen die Arbeiten von Conway und Maxwell [Conway 67], von Baker [Baker 74] und im deutschsprachigen Raum die Arbeit von P. Brucker [Brucker 81]. Brucker untersucht die Komplexität von Spezialverfahren. Einen breiten Überblick über Prioritätsregeln geben Panwalkar und Iskander [Panwalkar 77]. J. E. Day und M. P. Hottenstein untersuchen nicht nur die klassischen

statischen Schedulingprobleme. Bei statischen Problemen ist die Anzahl der Aufträge vorgegeben und endlich. Im dynamischen Fall treffen die Aufträge kontinuierlich ein. U.a. berichten die Autoren, daß die KOZ (kürzeste Operationszeit) Prioritätsregel nicht optimal ist, wie das von Baker behauptet wurde [Day 71 S. 21ff], [Baker 84]. Einige Autoren vergleichen Prioritätsregeln mit Hilfe der Simulation, kommen aber auch nicht zu wesentlich anderen Ergebnissen [Keck 68], [Dincmen 74]. Fisher analysierte das worst-case Verhalten von Heuristiken für verschiedene Aufgabenstellungen. Eine derartige Analyse garantiert dem Anwender einer Methode eine Mindestperformance [Fisher 82]. Kanet zeigt, daß es aus taktischen Gründen günstig sein kann, bei der Verplanung von Aufträgen Leerzeiten vorzusehen [Kanet 86]. Es ist sogar eher unwahrscheinlich, daß eine optimale Maschinenbelegungsplanung ohne taktische Leerzeiten für ein gegebenes Problem existiert. Er nennt aber kein polynomiales Verfahren, um die optimalen Leerzeiten zu bestimmen.

Methoden für Prozesse, die nur aus einer Operation bestehen, vergleichen Lageweg, Lenstra und Rinnooy Kan [Lageweg 79]. Ihr Ziel ist, die maximale Verspätung zu minimieren. McNaughton untersucht Aufträge, die aus einer einzigen Operation bestehen, hauptsächlich für parallele Prozessoren. Die Prozesse haben Fristen. Für Verspätungen wird eine Verlustfunktion zugrunde gelegt. R. K. Arora und S. P. Rana überführen einen halbgeordneten Flow Shop, in dem keine Zwischenlager erlaubt sind, in ein TSP [Arora 80]. Verschiedene Probleme mit parallelen Maschinen werden z.B. von B. Simons [Simons 82], C. Martel [Martel 82] und E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan [Lawler 82] untersucht. Um zu ermitteln, welches Verfahren für ein gegebenes Problem geeignet ist, berechnen Miller und Stockman die Anzahl der Lösungen, die nicht die Vorrangrelationen verletzen. Falls der dann ermittelte Lösungsraum klein genug ist, kann man z.B. mit Branch und Bound Methoden unschwer eine optimale Lösung ermitteln. Ein Überblick über probabilistische Methoden findet sich in einem Conference Proceeding, das von M. A. H. Dempster, J. K. Lenstra und A. H. G. Rinnooy Kan herausgegeben wurde [Dempster 82].

Der wichtigste Algorithmus für den Spezialfall von zwei Maschinen ist der Johnson-Algorithmus. Er liefert für den Flow Shop o/2/F/o/max y(i) eine garantiert optimale Lösung (Beweis bei [Johnson 54]). In der Folgezeit wurde er in zweierlei Hinsicht erweitert. Der Jackson-Algorithmus optimiert den allgemeineren Fall des Job Shop mit zwei Maschinen o/G/m(i)<=2/o/max y(i), sofern jede Tätigkeit aus höchstens zwei Teiltätigkeiten besteht. Außerdem entdeckte man, daß für eine bestimmte Struktur der Operationszeiten die Sortierung des Johnson-Algorithmus auf Probleme mit drei Maschinen o/3/F/o/max y(i) erweitert werden kann.

Einen anderen Denkansatz verfolgen Modelle zur Kapazitätsplanung. Sie gehen davon aus, daß sich ein ständiger Strom an Arbeit durch die Fertigung bewegt. Meist ist das Volumen des Arbeitsstroms stochastisch verteilt. Vor den Maschinen entstehen Warteschlangen die optimiert werden können oder man kann Engpässe optimieren. Solberg beschreibt die Beziehung zwischen einem deterministischen Engpaßmodell und einem stochastischen Warteschlangenmodell [Solberg 81, S. 116ff]. Choong und Maimon zeigen, wie man einen Arbeitsstrom mit stochastischen Ankunftsraten über die Zeit und über den Maschinenpark verteilt, in dem Aufträge dynamisch durch das System geschleust werden. Wenn die einem Auftrag zugedachte Station in der Zelle überlastet ist, sucht das Verfahren nach ähnlichen Stationen in anderen Zellen [Choong 86].

Besonders in flexiblen Fertigungssystemen und großer Fertigungsbreite kann die Rüstzeit Gegenstand der Optimierung sein. Formal kann man die Minimierung der Rüstzeit für eine Maschine wieder auf ein TSP zurückführen, bei der die Aufträge Knoten sind und die Rüstzeiten bewertete Kanten. Ursachen für Rüstzeiten sind beispielsweise Werkzeugwechsel. Viele Maschinen verfügen über ein Magazin, das eine

begrenzte Anzahl von Werkzeugen aufnehmen kann. So entstehen zwei Arten von Rüstzeiten: 1. Die Zeit, um ein Werkzeug, das im Eingriff war mit einem anderen Werkzeug aus dem Magazin zu wechseln. 2. Die Zeit, um das Magazin mit anderen Werkzeugen aus dem Lager zu bestücken. Tang entwickelte für den Fall mit einer Maschine eine Branch und Bound Methode [Tang 86, S. 152ff]. Den Zwei-Maschinen-Fall in einem Flow Shop untersuchten E. J. Lee und P. B. Mirchandani. Als dritte Art von Rüstzeit nennen sie die Systemrüstzeit. Darunter verstehen sie die Zeit, die notwendig ist, um einen neuen Teiletyp zu produzieren. Sie entwickelten dafür Heuristiken [Lee 86].

Je umfangreicher ein Problem ist, je mehr Aufträge und Maschinen zu verplanen sind, desto schwieriger wird seine Handhabung. Proth hat vorgeschlagen, ein Problem zu unterteilen. Sämtliche vorhandenen Teile werden zu Teilefamilien, Maschinen zu "Subsystemen in der Produktion" zusammengestellt. Dann werden Teilefamilien den Subsystemen so zugeordnet, daß die Zeit so kurz wie möglich ist. Die Subsysteme können dann getrennt verplant werden. Das Dekompositionsverfahren wird von J. M. Proth [Proth 86] beschrieben.

Für Probleme in der Praxis eignet sich besonders der Überblick von Siegel, der sich in seiner Darstellung auf einfachere Verfahren beschränkt [Siegel 74]. Eine gute Zuordnung von Problemen und schnellen Lösungen enthält der Bericht von Lamatsch, Morlock, Neumann und Rubach [Lammatsch 86].

Vereinfachend kann man folgendes sagen: Für eine ganze Reihe von Problemen mit ein oder zwei Maschinen gibt es Spezialverfahren. Darüberhinaus kann man meist nur noch mit enumerativen oder heuristischen Verfahren arbeiten. Enumerative Verfahren lösen Problemgrößen etwa bis zu 5 Maschinen und 10 Aufträgen in vertretbarer Zeit. Größere Probleme löst man meist mit Prioritätsregeln wie "Kürzeste Operationszeit", "Längste Operationszeit", "Most Work Remaining Rule".

5.2.2.2. Lösungen für das TSP

Enumerative Verfahren, Verfahren der ganzzahligen Optimierung und Branch und Bound Verfahren sind nur für kleinere Rundreiseprobleme geeignet. Heuristische Verfahren teilt man in Eröffnungs- und Verbesserungsverfahren ein. Mit Eröffnungsverfahren findet man eine zulässige Lösung, die mit Verbesserungsverfahren verbessert werden kann. Bekannte Heuristiken zur Eröffnung sind die des besten Nachfolgers, der sukzessiven Einbeziehung und die Heuristik von Tomescu (alle beschrieben z.B. in [Kramer 89]). Ausgehend von einem Kurzkreis mit zwei Knoten wird der Kreis in jedem Schritt um einen Knoten erweitert, bis ein vollständiger Weg konstruiert wurde. Das ebenfalls dort beschriebene Verfahren von Tomescu beginnt mit einem minimalen 1-Baum und vertauscht Kanten solange, bis von jedem Knoten zwei Kanten ausgehen.

R-optimale Verfahren verbessern eine Anfangslösung durch Vertauschen von r Kanten. Erstmals wurde das Verfahren von Croes vorgestellt [Croes 58]. Ein schnelles 3-optimales Verfahren entwickelte Lin [Lin 65]. Sammlungen von Verfahren findet man in den Arbeiten von Christofides, Eilon und Mingozzi [Christofides 69, 72, 79]. Rossier und Troyon nähern sich dem optimalen Kantenaustausch über Wahrscheinlichkeiten [Rossier 86].

Eine ausführliche Darstellung von exakten Verfahren findet man bei Domschke [Domschke 85]. Die Arbeit von Bellmore und Memhauser verfolgt einen eher theoretisch motivierten Ansatz. Die Autoren geben einen Überblick über die Problemdefinition und die zugrundeliegenden Theoreme. Sie klassifizieren Lösungstechniken, beschreiben bewährte Methoden und vergleichen ihre Rechenzeiten [Bellmore 68]. Approximationsverfahren beschreibt Mehlhorn [Mehlhorn 84, S. 218 - 223].

5.2.2.3. OR-Methoden intelligent auswählen

Einige wissensbasierte Systeme wählen für ein gegebenes Problem anhand einer Regelbasis passende Lösungsverfahren aus. Die Wissensbasis besitzt Informationen über die Eignung der Verfahren für bestimmte Problemcharakteristika wie die Problemgröße und Randbedingungen. Ein Problem wird analysiert und entsprechend den vorhandenen Problemcharakteristiken kategorisiert. Danach sucht man den für den Problemtyp besten Algorithmus, appliziert ihn auf das Problem und löst es.

Das System SCHEDULER zur Maschinenbelegungsplanung von Lamatsch enthält 19 Problemkategorien, die in Richtung der Problemverallgemeinerung aufeinander zurückgeführt werden können. Ein Einmaschinenproblem o/1/o/o/o z.B. ist ein Mehrmaschinenproblem o/m/o/o/o mit m=1. Insgesamt 37 Methoden stehen zur Verfügung, die regelbasiert einem konkreten Problem zugeordnet werden und es dann lösen [Lamatsch 86].

KPS (Knowledge-Based Approach To Planning and Scheduling) sucht zuerst eine zulässige Lösung, die anschließend verbessert wird [Gilmore 89]. Zur Ermittlung der zulässigen Lösung wurde ein Greedy-Algorithmus gewählt, der nur dafür sorgt, daß keine unzulässigen zeitlichen Überschneidungen auftreten. Ein Regelsystem empfiehlt und vollzieht Änderungen an der Ausgangslösung mit dem Ziel, sie zu verbessern.

PAC von C. Copas und J. Browne erzeugt Belegungspläne für Montagezwecke und bezieht bei der Auswahl der besten Heuristik technische Kriterien ein. So wird überprüft, ob das notwendige Material für den Auftrag zur Verfügung steht, die Kapazität von Maschinen ausreicht und ob die Belegung einer Maschine zeitkritisch ist. Aufgrund des Analyseergebnisses wird dann eine Methode ausgewählt und ausgeführt [Copas 90].

Wu und Wysk verwenden ein Multi-Pass Scheduling, d.h. von jedem Entscheidungspunkt aus wird die beste Schedulingstrategie mit Hilfe einer Simulation neu ermittelt [Wu 87]. Die Strategie mit dem besten Ergebnis wird ausgewählt.

Alle diese Verfahren trennen zwischen Planungswissen, das in sequentiellen Algorithmen dargestellt wird, und Wissen über die Anwendung des Planungswissens, das meist in einer Regelbasis steckt. Der Ansatz von Wu und Wysk hat den Vorteil, daß die Planung mit dem entsprechenden Algorithmus sehr effizient ist. Sie ist aber auch starr; die Fähigkeit zur Suche bei der Planung fehlt, weil die Algorithmen nur straight forward planen. Will man sie dennoch dem Problem gut anpassen, benötigt man eine sehr große Zahl von Algorithmen und eine entsprechend große Regelbasis mit Wissen über ihre Anwendbarkeit. In der Praxis ist das Spektrum der Probleme, die von einem solchen System bearbeitet werden können verhältnismäßig gering. Entweder funktioniert das System nur für eine bestimmte Anwendungsdomäne (z.B. PAC und KPS), oder ein Problem muß für die Lösung erst abstrahiert und angepaßt werden (z.B. SCHEDULER). Schon durch die Ähnlichkeit zum opportunistischen Schließen ist die Auswahl von Verfahren ein mächtiges Werkzeug, das aber im Verbund mit anderen Methoden verwendet werden sollte.

5.2.2.4. Suchstrategien zur Constraintanalyse und -propagierung

MOLGEN, ISIS und OPAL sind drei Systme, bei denen die Constraintanalyse von zentraler Bedeutung ist. MOLGEN betrachtet die Planung als einen Vorgang, der durch Operationen auf Constraints charakterisiert ist und plant hierarchisch durch das Aufstellen von Constraints. Mit ihnen werden Interaktionen zwischen lose gekoppelten Unterproblemen ausgedrückt. Dazu besitzt MOLGEN drei Operationen:

- Constraintaufstellung: neue Constraints werden dem Planungsprozeß hinzugefügt.
- Constraintpropagierung: Neue Constraints werden aus alten abgeleitet.

- Constraintsatisfizierung: Variablen werden gebunden, ohne einen Constraint zu verletzen.

Ein *Search Manager* steuert die Suche in ISIS Top-Down durch verschiedene Abstraktionsebenen eines Schedulingproblems. Die Ebenen tauschen untereinander Constraints aus. Auf jeder Ebene besteht die Suche aus den Phasen Voranalyse (Auswahl von Operatoren und Bindung von Constraints), Suche (Beam-Search) und Nachanalyse (Auswahl, Diagnosen). In der Voranalysephase bestimmt ISIS die Grenzen des Suchraums. In der Suchphase wählt es einen Operator, nach dem es in der Postanalyse das Ergebnis der Suche ausgewertet hat. Ist das Ergebnis zufriedenstellend, gibt ISIS es als Constraints an die nächstniedrigere Ebene weiter. Andernfalls muß es den Suchraum der aktuellen oder nächsthöheren Ebene ändern [Fox 84, S. 37].

ISIS sucht auf vier Ebenen. Ebene 1 identifiziert die aktuelle Auftragsmenge und sortiert sie nach Prioriät. Die Priorität eines Auftrags hängt von seiner Wichtigkeit und Dringlichkeit ab. Die Aufträge plant Ebene 2 in Abhängigkeit von der Maschinenkapazität und ihrer Operationszeit ein. Mit einer CP/M-Analyse (siehe S. 37) und geschätzten Operationszeiten gibt Ebene 3 Zeitintervalle für die Operationen als Constraints vor. In der Voranalyse entscheidet sie entsprechend den Auftragsconstraints, ob der Auftrag vorwärts oder rückwärts einzuplanen ist. Während der Suche generiert sie neue Zustände, die die Teilbelegung weiter spezifizieren. Jeder neue Zustand erhält eine Bewertung. Wenn ISIS keine akzeptable Belegung gefunden hat, dann relaxiert es Constraints [Fox 84, S. 38ff].

AIPLAN, ein Planungs- und Schedulingsystem für Aluminiumgießereien unterscheidet zwischen Heuristiken zur Reihenfolgeplanung und zur Zuweisung. Reihenfolgeplanungsheuristiken sind entweder lokal oder global. Lokale Heuristiken stellen Spezialwissen über einen Auftrag dar, während globale Heuristiken die gesamte Planungssituation berücksichtigen, um eine Reihenfolge unter den Aufträgen zu bestimmen (z.B. Aufträge mit der kürzesten Operationszeit vorzuziehen). Zuweisungsheuristiken enthalten Wissen darüber, welche Ressource welchen Auftrag ausführen kann [Arff 89].

Der gleiche Autor unterscheidet in einem späteren Projekt Heuristiken zur Suchraumbegrenzung und zur Abstraktion [Arff 91]. Der Suchraum wird begrenzt, indem man

- feststellt, ob Constraints verletzt wurden,
- partielle Pläne und untere Grenzen von Suchpfaden auswertet oder
- Zustände auswählt.

Dabei schneidet der Planer Zweige des Suchbaums ab. Abstraktionsheuristiken helfen, durch wenige erlaubte Kombinationen, die auf höherer Ebene festgelegt werden, die Komplexität zu reduzieren. Dazu gehören

- Gruppierheuristiken: Auftragsgruppen, die sich gegenseitig nicht beeinflussen (z.B. weil sie keine Ressource gemeinsam benutzen), werden getrennt voneinander verplant. A. Ricken zeigte, daß Gruppierheuristiken die Komplexität reduzieren, ohne optimale Lösungen auszuschließen [Ricken 89]. Der Gruppierung kommt allerdings in der Praxis wegen fehlender Voraussetzungen wenig Bedeutung zu.
- Ordnungsheuristiken: Eine bestimmte Sortierung der Aufträge wird vorgegeben. Damit wird zwar die Lösungssuche erheblich beschleunigt, aber nur ein lokales Optimum gefunden.

In SONIA gibt es einen etikettierten Graphen, der die Auswahl von Operationen darstellt. Er wird in einer Vorauswahlphase mit gewichteten Produktionsregeln konstruiert.

Der Bedingungsteil der Regeln nimmt auf den aktuellen Zustand der Planung und verfügbare Operationen Bezug [Collinot 88, S. 89].

SOJA verwendet Regeln auf drei verschiedene Arten: Vorwärtsverkettende Regeln ändern den Zustand des Systems, rückwärtsverkettende Regeln beweisen Literale aus dem Bedingungsteil anderer Regeln, und Dämonenregeln ändern den Kontext [Le Pape 85]. Ebenfalls mit gewichteten Produktionsregeln arbeitet OPAL, um Vorrangbeziehungen zwischen Operationen festzulegen, die nicht mit der constraintgesteuerten Analyse etabliert werden konnten. Das Gewicht einer Operation hängt ab vom Beitrag zur Realisierung eines bestimmten Ziels.

Bei Suchstrategien dienen ggf. zusätzliche Teile im Bedingungsteil von Regeln als Anhaltspunkte für eine Auswahl. Dabei fügt man Regeln gern eine Art Kontext hinzu, der über ihre sinnvolle Anwendbarkeit eine Aussage macht. Bei SOJA fallen Regeln heraus, die nicht zum aktuellen Kontext passen, OPAL und SONIA gewichten Regeln, bevor eine endgültige Auswahl getroffen wird. Die Suche läuft in zwei Phasen ab: in der Vorauswahl werden Regeln bewertet. Unter den Regeln mit positiver Bewertung wird danach diejenige bestimmt, die feuert. ISIS fügt eine Nachanalysephase hinzu, mit der die Suche bewertet wird. Gleichzeitig zeigt ISIS, wie flexibel diese Strategie ist, indem es sie auf sehr unterschiedlichen Ebenen der Problemlösung einsetzt. Der Grund für den Einsatz von Regeln liegt in der Verbesserung der Laufzeit eines Produktionssystems. Zur Auswertung der nötigen Zusatzinformationen verwendet ISIS den Schlußfolgerungsmechanismus, der auch sonst das Produktionssystem bewegt.

5.2.2.5. Lokale Suchtechniken

Zur Steuerung der Suche bevorzugt man Strategien, die Wissen über das Problem nutzen, um die Lösung zu beschleunigen, anstatt blind alle denkbaren Alternativen auszuprobieren. Meist handelt es sich dabei um heuristisches Wissen. Heuristiken treffen eine Auswahl über den nächsten Suchschritt oder entscheiden, wann eine Suche abgebrochen wird. Man kann zur Verbesserung der Planungsgüte und/oder Verminderung des Planungsaufwands auf problemspezifisches Wissen zurückgreifen.

Eine bekannte Suchtechnik ist die Best-First Methode, bei der stets der Schritt ausgewählt wird, der die größte Wertsteigerung im Sinne einer Zielfunktion erreicht. In Verbindung mit einer Wissensbasis läßt sich das Split-and-Prune Verfahren aus dem OR verbessern, z.B. durch das Abschneiden von Zweigen, für die hergeleitet werden kann, daß sie nicht zur optimalen Lösung führen. Für eine detaillierte Darstellung dieser Verfahren vgl. [Pearl 84].

ISIS verwendet für die Maschinenbelegungsplanung eine heuristische Strahlsuche ("Beam Search") innerhalb eines Zustandsraums von Teilbelegungen. Während jeder Suchiteration dienen die n besten Zustände als Grundlage für den nächsten Schritt. Außerdem verwendet ISIS eine Heuristik, um die Wichtigkeit eines Auftrags zu bestimmen. Das geschieht (s.o.) in Abhängigkeit seiner Wichtigkeit und Dringlichkeit. Eine Strahlsuche benutzt auch OPIS, um bei der Planung der Operationen einen Arbeitsplan für die Aufträge zu entwickeln. Dabei berücksichtigt der Planer alternative Belegungszeiten, Vorrangbeziehungen und andere Präferenzen. Auch die Auswahl von Constraints für die Relaxierung erfolgt heuristisch, indem OPIS die Constraints ihrer Bedeutung nach gewichtet. Ein vergleichbarer Ansatz findet sich auch bei [Parunak 86].

MAESTRO, ein Maschinenbeleger von Britt, Geoffroy, Schaefer und Gohring, wählt die nächste zu verplanende Aktivität heuristisch aus [Britt 87]. Im Unterschied zu ISIS dient hier ihre relative Beschränktheit und ihr Erfolgsniveau als Auswahlkriterium. Die Beschränktheit ergibt sich aus der Anzahl der Möglichkeiten, eine Aktivität einzuplanen. Zur Ermittlung des Erfolgsniveaus für eine mögliche Planung werden Zielvorgaben mit dem Ergebnis des Planungsschritts verglichen und gewichtet. Wurde eine Aktivität

ausgesucht, dann bestimmt eine Heuristik ihre Platzierung anhand der bisherigen Belegung.

Das System PATRIARCH von Lawrence und Morton [Lawrence 86] rechnet Opportunitätskosten gegen Strafkosten für die Verspätung eines Auftrags auf. Opportunitätskosten entstehen, wenn eine Ressource knapp ist. Sie werden einem Auftrag k zugerechnet, wenn andere Aufträge sich deswegen verspäten, weil er zum Zeitpunkt t eingeplant wird. Dagegen stehen die Kosten, die durch die Verspätung von k entstehen. Übersteigen die Verspätungskosten die Opportunitätskosten, so plant PATRIARCH den Auftrag k zum Zeitpunkt t ein, andernfalls schiebt er ihn auf.

SONIA verwendet in der reaktiven Komponente eine heuristische Strategie, um Konflikte zwischen der ursprünglichen Planung und Unterbrechungen in der Werkstatt zu lösen. Je nach Anzahl der Unterbrechungen entscheidet das System, ob es nur einen Teil des ursprünglichen Plans umplant, oder ob es die Planung neu aufwirft [Collinot 88]. Die Produktionsregeln in SONIA (s.o.) sind ebenfalls Heuristiken.

Darüberhinaus ließen sich noch eine Fülle von Suchtechniken einschließlich der einschlägigen Sortierverfahren, wie z.B. die kürzeste Operationszeit, nennen. Die Fülle der Techniken ist ein Hinweis darauf, daß es bisher keine beste Suchtechnik gibt, weil das Reihenfolgeplanungsproblem nicht vollständig beherrscht ist. In keinem Fall gibt es eine Gewähr, auch nur in die Nähe des globalen Optimums vorzustoßen. Ein wenig mehr Erfolg verspricht eine domänenabhängige Suche, wie bei ISIS, MAESTRO oder PATRIARCH. Sie ist deswegen domänenabhängig, weil zumindest die Zielfunktion vorgegeben sein muß, um die Qualität des Suchergebnisses zu messen. Die Parameter einer Zielfunktion sind dazu meist vom System vorgegeben und müssen für die Anwendung nur noch gewichtet werden. Wenn Kriterien außerhalb der vorgegebenen Paramter optimiert werden sollen, versagen jedoch die Suchtechniken.

5.2.2.6. Globale Suchtechniken

Anstelle des Versuchs, lokal zu verbessern, gehen neuere Suchverfahren dazu über, durch einige zufällige Schritte dem globalen Optimum näher zu kommen. Das Verfahren der simulierten Abkühlung erinnert mit seinem Namen an den physikalischen Prozeß der Temperaturreduktion eines Materials auf sein thermisches Gleichgewicht hin. Wie beim Abkühlen aus der Schmelze die Temperatur schrittweise gesenkt wird, so wird der Wert einer Zielfunktion, die o.B.d.A. zu minimieren ist, gesenkt durch verbessernde Schritte. Im physikalischen Vorbild darf die Temperatursenkung besonders zu Beginn nicht zu rasch voranschreiten, weil sonst lokal suboptimale Konfigurationen erstarren und der Idealzustand nicht erreicht wird. Entsprechend werden bei der Lösungssuche anfangs auch Schritte zugelassen, die die Lösung nicht verbessern. Die Akzeptanz nicht-verbessernder Schritte hängt von einer Wahrscheinlichkeit ab, die mit der Zeit ständig sinkt. Falls d eine Verbesserung der Zielfunktion T (Temperatur) bedeutet mit $d > 0$ (sonst trägt der Schritt zur Minimierung bei und ist automatisch aktzeptiert) und k eine Konstante, dann ist die Wahrscheinlichkeit für die Akzeptanz des Schrittes

$$P(d) = e^{-d/kT}.$$

Durch derartige Schritte hofft man, dem globalen Optimum näher zu kommen. Glover weist allerdings darauf hin, daß im Gegensatz zur thermischen Abkühlung bei kombinatorischen Problemen auch in einem sehr fortgeschrittenen Lösungsstadium durch einen einzigen Schritt in eine andere Richtung das globale Optimum erreicht werden kann [Glover 89, S. 123]. Die simulierte Abkühlung spiegelt deswegen nicht die Natur kombinatorischer Probleme wieder, führt aber zu sehr guten Ergebnissen.

Genetische Algorithmen übertragen Prinzipien aus der biologischen Genetik auf die Kombinatorik. Neue Lösungen werden aus Elternlösungen erzeugt, wobei stets die besten überleben. Entsprechend dem Grad ihrer Stärke, d.h. der erreichten Verbesserung

des Zielfunktionswertes, werden Lösungen reproduziert. Lösungen werden gepaart, indem man Teile einer Lösung mit Teilen einer anderen Lösung zusammenstellt. Dabei ist bei kombinatorischen Problemen die Zulässigkeit der Lösung nicht trivial. Gegebenenfalls müssen zusätzliche Verfahren integriert werden, die harte Konsistenzbedingungen zusichern. Schließlich erlaubt Mutation, eine Lösung zufällig zu modifizieren. Oliver, Smith und Holland geben ein Beispiel für die Anwendung eines genetischen Algorithmus auf das klassische TSP [Oliver 87]. Um die Konsistenz zu erhalten, führen sie einen speziellen Formalismus ein.

Als Speicher für partielle Lösungen von TSPen dienen genetische Algorithmen bei Kadaba [Kadaba 91]. Genetische Algorithmen bestimmen die Parameter einer heuristischen Suchprozedur sowie die Steuerregeln. Die Algorithmen sind in ein hybrides System von Regeln und Neuronalen Netzen eingebettet, die ein Blackboard steuert. Die Autoren berichten von überdurchschnittlich guten Ergebnissen.

Bei der Tabusuche handelt es sich um ein Metaverfahren, das Operationen eines Unterverfahrens, das lokale Optima erreicht, steuert [Glover 89, S. 124], indem dessen Suchschritte organisiert werden. Eine Tabuliste speichert die letzten k Schritte des lokalen Algorithmus. Der nächste Schritt ist derjenige, der am meisten den Zielfunktionswert verbessert und nicht Element der Tabuliste ist. Dazu überprüft man für jedes Element der Tabuliste, wie es die Welt verändern würde, käme es jetzt zum Zug. Sollte ein Schritt der Tabuliste einen Schwellwert überschreiten, kann er dennoch zum Zug kommen. Der neue Schritt wird anschließend an den Anfang der Tabuliste gesetzt, der k-te Schritt wird entfernt und vergessen. Glover gibt anschließend ein Beispiel für die Anwendung von Tabusuche auf das TSP.

Kritisch bei genetischen Algorithmen für das TSP ist die Darstellung des Suchraums. J. Grefenstette, R. Gopal, B. Rosmaita und D. Van Gucht vergleichen mehrere Repräsentationen, die aber alle zu wenig befriedigenden Ergebnissen führen [Grefenstette 85]. Die Autoren verbinden die Crossover-Operation mit einer lokalen Optimierung: Von zwei Wegestücken, die einen Knoten in den beiden Zyklen der Eltern verlassen, wird der kürzeste ausgewählt. Führt der kürzere Weg in einen Kurzzyklus, dann wird eine zufällig erzeugte Kante gewählt. Die Autoren erzielen mit dem Verfahren, das sie "heuristic Crossover" nennen, bessere Ergebnisse.

Ein Verfahren zur Maschinenbelegungsplanung mit genetischen Algorithmen gibt Davis [Davis 85]. Wie Glover beim TSP, so sieht sich Davis mit dem Problem konfrontiert, daß durch Paarung ("crossover") zulässiger Lösungen wahrscheinlich eine unzulässige Lösung entsteht. Um das zu vermeiden, kodiert er die Repräsentation in eine Zwischenform. Lösungen einer Generation werden in die Zwischenform gebracht und dort gekreuzt. Die neuen Lösungen werden dekodiert und sind garantiert zulässig. Eine Zwischenform besteht aus einer Präferenzliste von Aktivitäten, die jede Maschine für sich erhält. Beispiel: (60, Auftrag1, Auftrag2, Warten, Leerzeit) bedeutet, daß ab Minute 60 gearbeitet wird, daß bevorzugt Operationen von Auftrag1 bearbeitet werden, dann von Auftrag2 usw..

Hilliard, Liepins und Palmer lösen Job Shop-Probleme mit einem vereinfachten Classifier System [Hilliard 88]. Ein Classifier ist ein Produktionssystem. Ihr Planer fügt dem klassischen Ablauf eines Produktionssystems einen Schritt hinzu, der jeden neuen Zustand bewertet und die aktiven Regeln entsprechend verstärkt. Die Regeln sind vergleichbar mit den Regeln im STRIPS-Planer, aber mit einer zusätzlichen Aussage über die Stärke. Die Stärke der Regeln - und damit die Wahrscheinlichkeit, daß sie zur Ausführung kommen - wird von einem genetischen Algorithmus modifiziert. Hilliards Ansatz läßt sich so prinzipiell auf jedes Reihenfolgeproblem übertragen, sofern man regelbasiert arbeiten will.

Lernfähige Suchtechniken wie genetische Algorithmen, Tabusuche, simulierte Abkühlung und - unter gewissen Umständen - auch Neuronale Netze sind den einfachen heuristischen Techniken in der Planungsgüte oft überlegen, kosten jedoch erheblich mehr Rechenzeit. F. Davis und F. Ritter empfehlen für Reihenfolgeprobleme Simulated Annealing. Genetische Algorithmen sollen bevorzugt die Technik des Simulated Annealing an einer Stelle unterstützen [Davis 87], auf keinen Fall aber selbst Reihenfolgen generieren. Leider fehlen hierzu noch Erfahrungen. Alternativen zur heuristischen Suche existieren zwar nicht, es sei denn, man hat sehr viel Zeit und schnelle Rechner, aber es gibt auch keine Technik, welche die anderen Suchtechniken dominiert.

5.2.2.7. Mehrstufige Planungsansätze

R. Winter untersuchte mehrstufige Ansätze zur Produktionsplanung in Abstraktionshierarchien auf der Basis relationaler Informationsstrukturen [Winter 91]. Die Ansätze sind sehr unterschiedlich und kommen aus dem Bereich des Management Science/Operations Research, der Künstlichen Intelligenz und aus PPS Ansätzen. Zu den Vorteilen mehrstufiger Planung zählt Winter [Winter 91, S. 3ff]:

- Geringere Komplexität durch Aufteilung eines großen Problems in mehrere, leicht beherrschbare Teilprobleme und durch verbessertes Problemverständnis.
- Bei fehlendem Wissen über die Zukunft können Entscheidungen aufgeschoben werden, um genauere Informationen aus anderen Teilen der Planung abzuwarten.
- Eine Problemzerlegung kann eine vorgegebene Organisationsstruktur wiederspiegeln. Daraus können sich Vorteile wie besseres Verständnis des Konzepts, höhere Akzeptanz und leichtere Umsetzbarkeit der Planung ergeben.
- Daten können in abstrakter Form verarbeitet werden. Abstrakte Daten sind oft in der Handhabung einfacher.
- Größere Flexibilität und Zuverlässigkeit und die Möglichkeit, das Problem als entscheidungsunterstützendes System zu realisieren.
- Aufgeteilte Probleme können dezentral gelöst werden.
- Abstrakte Problemlösungen lassen sich wiederverwenden.

Dem stehen einige Nachteile entgegen:

- Wenn mehrstufige Planungsansätze eine optimale Lösung liefern, dann ist das meist Zufall. Durch Abstraktionsfehler und Interaktionsprobleme zwischen Teilplänen sind nur suboptimale Lösungen erzielbar.
- Die Konsistenz zwischen Teilplänen kann außer Kontrolle geraten. Der Gesamtplan wird dann widersprüchlich. Der Aufwand zur Konsistenzwahrung kann erheblich sein.
- Bisher existiert kein Standard für den Entwurf von Hierarchien.
- Konventionelle Verfahren gehen von fest vorgegebenen Planungsebenen aus, was die Flexibilität, die sich aus der Mehrstufigkeit ergibt, wieder einschränkt.

Von der Suboptimalität einmal abgesehen zeigt Winter, daß insbesondere durch KI-Ansätze die Nachteile der mehrstufigen Planung bewältigt werden können. Ihre Suboptimalität ist aber nicht so relevant, daß dafür die Vorteile aufgegeben werden sollten. Vielmehr geht es in der Praxis stets darum, eine gute Lösung zu finden. Das wiegt umso schwerer, als oft unklar ist, was eigentlich ein Optimum ist: "Wenn man keine Prognose über die Auftragseingänge besitzt, muß man einen Kompromiß finden zwischen den Alternativen *minimale Lagerkosten durch kürzeste Durchlaufzeiten der Aufträge* und

früheste Freistellung der Maschinen durch maximale Auslastung (geringste Brachzeiten). Beide Planungsmaximen widersprechen sich." ([Müller-Merbach 70, S. 176] Formatierungen wie im Quelltext).

Ein Werkzeug zur Reihenfolgeplanung sollte deswegen mit Blick auf die Praxis mehrstufige Planung ermöglichen.

5.2.2.8. Konzepte zur mehrstufigen Planung

Quellen für Konzepte zur mehrstufigen Planung sind der Bereich Management Science/Operations Research und die künstliche Intelligenz. Zwei wichtige mehrstufige Ansätze aus dem Bereich des OR sind nach Winter die Ansätze des Massachusetts Institute of Technology (MIT) und des Linköping Institute of Technology (LIT).

Der MIT-Ansatz definiert die Planungsebenen (z.B. Grob-, Mittel- und Feinplanung) entlang der Abstraktionshierarchie der Entscheidungsvariablen (Produktionsvolumen, Kapazitätszuteilung etc.). Der Ansatz ist auf Entscheidungsvariablen hin orientiert und stellt Ziele und Teilziele für die Planungsebenen auf. Restriktionen werden nicht abstrahiert und treten unabhängig von Planungsebenen auf. Zwischen Teilproblemen, die er durch einseitige Vorgaben koordiniert, besteht nur ein schwacher Zusammenhang. Inkonsistenzen sollen dabei nicht auftreten, weshalb er auch der Beseitigung von Inkonsistenzen wenig Beachtung schenkt. Winter bezeichnet den MIT-Ansatz als "klassischen Prototyp mehrstufiger Planung" [Winter 91, S. 125].

Die Planungsebenen stehen beim Ansatz des LIT in direktem Zusammenhang mit der Weite des zugehörigen Planungshorizontes. Der LIT-Ansatz ist planungshorizont-orientiert. Deswegen fehlen Abstraktionshierarchien für die Entscheidungsvariablen. Das Planungsverfahren wird entsprechend der Planungsebene ausgesucht. Die Koordination erfolgt nachträglich, indem man parallel entwickelte Pläne nachträglich aufeinander abstimmt.

5.2.3. Anwendungen in der betrieblichen Praxis

Um Lösungen schneller implementieren zu können, gibt es inzwischen Programmbibliotheken, die OR-Verfahren zur Optimierung in großer Bandbreite zur Verfügung stellen (z.B. [Lamatsch 86]).

Bei Realisierungen für industrielle Umgebungen greift man gern auf Heuristiken zur Reihenfolgeplanung zurück. Sie sind leicht implementierbar und einfach zu beherrschen. Die Anwendungsdomäne ist für die Auswahl einer Lösungsstrategie weniger entscheidend, weil sich die Aufgabenstellungen oft sehr ähnlich sind. Doulgeri und Hibberd simulieren für ein flexibles Fertigungssystem verschiedene Prioritätsregeln und ordnen dann bestimmte Situationsparametern den Heuristiken zu. Die Regel, die die passendste Heuristik auswählt, nennen sie die "Looking-Ahead"-Regel [Doulgeri 87]. Damit soll zum Ausdruck gebracht werden, daß die Regel nach vorn schaut, also ihre Konsequenzen bei der Sortierung mitberücksichtigt. In eine ähnliche Richtung geht die Arbeit von Moser, der Prioritätsregeln für ein Fertigungsleitsystem entwickelt [Moser 91]. Warnecke und Geitner prüfen eine ganze Reihe von Prioritätsregeln, die temporäre Engpässe vorausschauend behandeln [Warnecke 83]. Für Naßprozesse benutzen Engelke, Hirn, Viehweger und Spur verschiedene Prioritätsregeln und vergleichen sie mit zufällig erzeugten Reihenfolgen, bezogen auf unterschiedliche Auftragszusammensetzungen. Bemerkenswert ist, daß sie sogar die gleichen Heuristiken wie Doulgeri und Hibberd betrachten [Engelke 88].

Eine Fertigungsinsel, die der Leitstandsidee nahekommt, beschreibt Kreimeier. Ein vorhandener Auftragspool kann nach folgenden Strategien abgearbeitet werden [Kreimeier 86a]:

- Kapazitätsabgleich: Verlagern von Aufträgen auf andere Maschinen, Sonderschichten.
- Um Termine zu gewähren, können Aufträge nach Fristen sortiert werden.
- Rüstzeitminimierung.
- Kürzeste Operationszeit, um die Kapazität zu minimieren.
- Störungsstrategien nach Ausfall einer Maschine.
- Manuelle Planung.

Obwohl es hier nur um den Teil eines ganzen Fertigungssystems geht, verwendet Kreimeier die gleichen Strategien wie bei der Planung eines vollständigen flexiblen Fertigungssystems. Eine bewährte Strategie ist die Optimierung der Engpaßmaschine. Die Engpaßmaschine ist die knappste Maschine bezüglich der Nachfrage durch Fertigungsaufträge. Eine Vergrößerung ihrer Kapazität trägt mehr zur Erfüllung der Zielkriterien bei als die Kapazitätsvergrößerung jeder anderen Maschine. Die Aufträge bringt man zuerst auf der Engpaßmaschine in eine möglichst optimale Reihenfolge. Daraus ergibt sich dann die Belegung der übrigen Maschinen. Ein Beispiel dafür gibt D. Kreimeier für die Fertigung in autonomen Arbeitsgruppen [Kreimeier 86b].

Für die "Belastungsorientierte Fertigungssteuerung" (vgl. 4.2.2) schlägt Wiendahl vor, eine Prioritätskennzahl zu errechnen, um zu entscheiden, ob ein Auftrag eingeplant wird. Ihre Parameter sind Auftragsgröße, Frist etc. [Wiendahl 84, S. 330].

Auch Heuristiken, die keine Prioritätsregeln sind, finden Anwendungen. Groh beschreibt ein Verfahren, das er "lokale Enumeration" nennt. Er unterteilt das Problem heuristisch in beherrschbare Teilprobleme, die er enumerativ optimiert. Die Summe lokaler Optima soll an das globale Optimum möglichst nahe herankommen [Groh 68]. Ein Flow Shop Problem mit n Maschinen, das den Produktionsprozeß einer Automobilfabrik abbildet, beschreiben Heinrich und Schneeweiß. Ein globaler Branch-and-Bound Algorithmus ruft eine Heuristik auf. Übersteigt der Wert eines bestimmten Parameters (hier die Leerzeit im Brennofen der Lackiererei) eine Schwelle, wird der Lösungszweig nicht weiter verfolgt und in einem anderen Zweig weitergesucht [Heinrich 80].

Warteschlangen vor Maschinen optimieren Halevi und Weil. Sie generieren parallel zur Maschinenbelegung Arbeitspläne mithilfe des Bellman Algorithmus [Halevi 80]. Bemerkenswert an diesem Verfahren ist die Integration von Arbeitsplanung und Maschinenbelegungsplanung. Beide Planungen erfolgen nicht auf einmal sondern in stetem Wechsel. Das Verfahren führt einen Auftrag so durch die Fertigung, daß die Warteschlangen vor den Maschinen optimal sind. Dann belegt es die Maschinen mit den Aufträgen.

Ein spezielles Problem mit zwei Robotern, die sich ein Werkzeugmagazin teilen, beschreiben Coupez und Delchambre. Die Roboter sind nicht identisch. Es gibt jedoch Operationen, die beide Roboter ausgeführen können. Die Roboter können aber nicht gleichzeitig dasselbe Werkzeug benutzen. Sie können von einer Palette zur anderen fahren, zwischen denen es zu Kollisionen kommen kann. Neben Heuristiken zur Lösungsfindung in Echtzeit wird für prediktive Zwecke ein Branch-and-Bound Algorithmus verwendet.

Bei einem Vergleich der Reihenfolgeplanungssysteme in der Praxis dominieren rein zahlenmäßig Systeme, die die Aufträge einfach mit Prioritätsregeln sortieren und dann zuweisen. Andere Ansätze sind bisher kaum verbreitet.

5.3. Zeitplanung

Ein Modell soll für zeitliche Beziehungen Schlußfolgerungen ermöglichen, um Reihenfolgen zu erstellen. Zeitliche Beziehungen ergeben sich sowohl aus hierarchischen, als auch aus kausalen Zusammenhängen. Folgende Probleme treten in der Planungsrealität auf, die als Kriterien für die Eignung eines Zeitplaners dienen sollen:

- A priori sind Zeitdauern nicht immer bekannt.
- Für einige Vorgänge sind die Start- oder Endzeitpunkte z.B. in Form von Freigabezeiten und Fristen vorgegeben.
- In einigen Fällen fehlen quantitative Zeitangaben.
- Nicht alle Zeitbeziehungen sind bekannt.

Auch ungenaue Informationen treten auf, die das System handhaben soll: ungenaue Zeitbeziehungen, z.B. bei zwei Intervallen, deren zeitliches Verhältnis (vorher, nacher, ...) nicht oder nur unvollständig spezifiziert ist. Für Start-, Endzeitpunkte oder Dauern können nur Grenzen vorgegeben sein. Eine Inferenz ist im Idealfall in der Lage, aufgrund zusätzlicher, möglicherweise deduzierter Information derartige Ungenauigkeiten und Unvollständigkeiten soweit einzuschränken, daß sich entweder am Schluß zwangsläufig eine Zeitbeziehung oder eine Zeitgröße ergibt oder es wird der Suchraum für eine anschließend aufzurufende Heuristik dargestellt.

Das Modell zur temporalen Inferenz soll mit unvollständigen Informationen zurechtkommen und die Möglichkeit bieten, nicht nur mit qualitativen, sondern auch mit quantitativen Informationen umzugehen. Gesucht wird ein Zeitplan, der Vorgängen feste Anfangs- und Endpunkte zuweist.

Planungsstrategien sind mit konventioneller linearer Planung schwieriger zu realisieren, weil Teilpläne aufgrund zeitlicher Beschränkungen zu Interaktionen neigen. Die Planung läuft dann in eine Sackgasse. Wer beispielsweise Vorgänge einplanen will, die nach kürzester Operationszeit sortiert sind, aber ständig mit Freigabezeiten und Fristen in Konflikt gerät, weil ausgerechnet die größte Operation (die als letzte eingeplant wird) die engsten Fristen hat, wird die gesamte Planung von vorn beginnen müssen. Vor der Anwendung einer Planungsstrategie wie der Kürzeste-Operationszeit-Regel[13] ist es sicher sinnvoll, die zeitlichen Beschränkungen zu berechnen. Das leistet die nichlineare Planung. Anschließend kann man auch mit wechselnden Strategien planen.

5.3.1. *Netzplantechnik*

Sehr verbreitet für Probleme, die in Form von Netzen dargestellt werden, ist eine Netzplantechnik wie z.B. CP/M oder PERT (vgl. [Neumann 75b S. 190ff]). Sie verknüpft Vorgänge über Vorher- und Nachherbeziehungen zu einem Netzwerk und kann mehrere Vorgänge zu einem Sammelvorgang zusammenfassen. Dauern und einige Start- oder Endzeitpunkte sind a priori bekannt. Mit der CP/M-Planung ist es dann möglich, für jeden Vorgang früheste und späteste Anfangs- und Endzeiten zu berechnen.

Als Zeitplaner zur Unterstützung eines allgemeinen Reihenfolgeplanungswerkzeugs hat CP/M einige schwerwiegende Nachteile:

- Nicht immer steht in der Planungsrealität a priori fest, ob zwischen zwei Vorgängen eine Vorher- oder Nachherbeziehung besteht. Das kann auch ein gesuchtes Ergebnis sein. In einem Fall können sich Reihenfolgen allein durch zeitliche Beschränkungen ergeben, in einem anderen Fall kann die Beziehung zwischen zwei Vorgängen un-

[13] Die Kürzeste Operationszeitregel sortiert Operationen aufsteigend nach ihrer Dauer und plant sie in dieser Reihenfolge ein.

vollständig spezifiziert sein. In einem dritten Fall kann sie sich aus anderen Zeitbeziehungen ergeben. In keinem der Fälle ist die Netzplantechnik hilfreich.

- Die Beschränkung auf im wesentlichen zwei qualitative Zeitbeziehungen ist wenig befriedigend, denn es können bis zu dreizehn Zeitbeziehungen auftreten.
- Die Netzplantechnik ist nur schwer anwendbar, wenn quantitative Zeitinformationen für einen Vorgang fehlen, z.B. die Dauer. Auch sie könnte aber das gesuchte Ergebnis sein. Einschränkungen für die Dauer in Form von oberer und unterer Schranke sind nicht vorgesehen.

Die Netzplantechnik ist nur für spezielle Reihenfolgeprobleme anwendbar, etwa einfache Projekt- oder Maschinenbelegungsplanung. Bei komplizierteren Zeitverhältnissen oder unvollständiger Information, wie sie durchaus in Reihenfolgeproblemen auftreten können, stößt die Netzplantechnik schnell an ihre Grenzen.

5.3.2. Zeitlogiken

Zeitlogiken sind entweder zeitpunkt- oder intervallbasiert. Zeitlogiken verfolgen einen modallogischen oder einen prädikatenlogischen Ansatz. Modallogische Ansätze erweitern die Prädikatenlogik der ersten Stufe um Operatoren, mit denen einer Aussage zusätzlich eine Zeit zugewiesen wird, zu der sie gelten soll, z.B. bei Templog [Baudinet 89]. Bei prädikatenlogischen Ansätzen legen Axiome die Semantik des Zeitkalküls fest. Die Primitive von Zeitlogiken sind entweder Zeitpunkte oder Zeitintervalle. Ein wichtiger Ansatz sind die Allen'schen 13 Zeitkalküle [Allen 83] (s.u.), auf denen z.B. die Sprache Temporal Prolog [Hrycej 88a] basiert.

Ein Implementierungsbeispiel für den modallogischen Ansatz gibt Guerrieri [Guerrieri 88, S. 200]. Jede Regel erhält einen Zeitparameter, der Aufschluß über die zeitliche Anwendbarkeit der Regel gibt. Eine Funktion *valid_time_guard(Time)* überprüft, ob die Regel zu einem gegebenen Zeitpunkt anwendbar ist.

5.3.3. Allens Intervallogik

J. Allens Ansatz basiert auf 13 disjunkten qualitativen Zeitrelationen zwischen Intervallen. Ein Zeitpunkt kann als sehr kurzes Intervall dargestellt werden. In einem Netzwerk mit Intervallen als Knoten werden Beziehungen als Pfeile zwischen Knoten dargestellt. Eine Beziehung zwischen zwei Intervallen kann sich aus mehreren disjunkten Basisrelationen zusammensetzen, jedes denkbare Verhältnis ist direkt darstellbar.

Soll in ein vorgegebenes Netzwerk eine zusätzliche Relation eingefügt werden, können alle notwendigen Konsequenzen berechnet werden, indem man die transitive Hülle aller Intervallrelationen berechnet. J. Allen gibt transitive Schlußregeln an, die zur Berechnung nötig sind. Bei drei gegebenen Intervallen A, B, C und zwei Intervallrelationen R_1, R_2 mit A R_1 B und B R_2 C läßt sich R_3 mit A R_3 C folgern. Das (nicht-lineare) Verfahren erzeugt keine Inkonsistenzen, kann aber nur zwischen jeweils drei Knoten Inkonsistenzen entdecken. Globale Konsistenztests sind NP-vollständig. Die Anzahl der Intervallrelationen wächst quadratisch mit jedem Einfügeschritt, bei n Intervallen sind es

$$\frac{(n-1)(n-2)}{2}.$$

Die Darstellung wird dadurch wenig transparent. Andererseits ist das Verfahren für komplexe Probleme der Zeitplanung von Prozessen, wie z.B. in der Maschinenbelegungsplanung, sehr geeignet. Das gilt auch für Zeitverhältnisse in Arbeitsplänen, die mit CPM-Netzwerken oder Petri-Netzen schwer darstellbar sind.

Zur Komplexitätsreduktion machte man sich Gedanken über Vereinfachungen. Allen selbst schlägt sog. Referenzintervalle vor. Eine baumähnliche Hierarchie von Intervallen

wird aufgebaut, bei der sich mehrere Intervalle auf ein Referenzintervall beziehen. Die Referenzintervalle ihrerseits stehen wiederum in Relation zueinander, nicht aber Intervalle, die unterschiedlichen Referenzintervallen zugeordnet sind. Damit verringert sich die Anzahl der Intervallrelationen, was einerseits die Komplexität reduziert, andererseits die Ausdruckskraft erheblich einschränkt. Ebenfalls problematisch ist, daß die Effizienz davon abhängt, wie geschickt der Benutzer die Referenzintervalle auswählt. Eine bessere Effizienz und größere Klarheit der Darstellung wird erkauft mit Einschränkungen in den Möglichkeiten.

Eine andere Möglichkeit zur Komplexitätsreduktion der Allenschen Zeitintervalle besteht darin, Intervalle über ihre Anfangs- und Endpunkte zu beschreiben und wieder mit einer zeitpunktorientierten Logik zu arbeiten. Über die Logik von Zeitpunkten gibt es bereits eine Menge Erfahrungen, die sich dann auf diese Art der Intervallogik übertragen lassen. Zwischen den vier Punkten zweier Intervalle existieren insgesamt acht Relationen der Art {vorher/nachher/gleichzeitig} [Günther 84], zur Darstellung reichen aber sechs Relationen aus. Mit der Komplexität verliert das Verfahren an Ausdruckskraft: Nur konvexe Zeitbeziehungen zwischen Intervallen sind darstellbar.

In den bisherigen Ansätzen wurden nur unbestimmte Intervalle berücksichtigt. Vilain [Vilain 82] und [Vilain 86] erweitert Allens Intervallogik in zweierlei Hinsicht. Anstatt als Intervalle von sehr kurzer Dauer stellt Vilain Zeitpunkte und Intervalle explizit dar. Es gibt einen Datierungsmechanismus, der Zeitpunkten und Intervallen einen Zeitstempel oder eine Standardzeiteinheit (z.B. eine Woche oder ein Monat) zuordnet. Vilain stellt Relationen mittels Vektoren dar. Ein Vektor verknüpft zwei Intervalle. Jedes Vektorelement ist eine Zeitrelation, ein Vektor ist eine Disjunktion über die Vektorelemente. Ähnlich einer Tabelle mit Schlußfolgerungsregeln gibt Vilain 169 Regeln an. Durch Vektoraddition stellt er konjunktiv Intervallrelationen dar. Wenn zwei Vektoren die Relation zwischen zwei Intervallen beschreiben, so läßt sie sich als Konjunktion der Intervallrelationen beschreiben. Transitive Schlußfolgerungen entstehen durch Vektormultiplikation. Wenn A, B, C Intervalle sind, R1, R2 Vektoren und gilt: A R1 B, B R2 C, dann folgt mit R3 := R1 * R2: A R3 C. Das Verfahren von Vilain ist in der Größenordnung der Laufzeit genauso komplex wie das von Allen, die Vereinfachung schlägt sich also nicht in einem Gewinn an Rechenzeit nieder. Für Zwecke der Reihenfolgeplanung ist eine Erweiterung auf die explizite Darstellung von Zeitpunkten nicht unbedingt notwendig. An Vilains Verfahren ist die Erweiterung hin zu quantitativen Zeitwerten wichtig, die als Ergebnis einer Zeitplanung gewöhnlich verlangt werden.

Zwischen zeitlichem und kausalem Wissen unterscheidet Dorn [Dorn 89]. Er bildet die Zeit auf einer nach rechts offenen Zeitgeraden ab, deren Elemente Granularitätsintervalle sind. Eine Variable "Jetzt" stellt die Gegenwart dar. Zeitschranken beschränken ein nicht vollständig bestimmtes Intervall, indem sie seinen Wertebereich begrenzen. Den Anfangs- und Endpunkt eines Intervalls beschreibt jeweils einer Zeitschranke. Dorn fügt den Allenschen Relationen Ordnungsrelationen hinzu, die Zeitschranken miteinander verknüpfen. Das sind Beschränkungen von Zeitschranken, die dazu dienen, den maximalen Wertebereich eines Intervalls aufgrund nicht-zeitlichen Wissens einzuschränken. Über die zeitliche Schlußfolgerung hinaus kann Dorn Intervalle auch aufgrund der Intervallattribute beschränken. Dazu sind aber zusätzliche Berechnungsschritte notwendig. Eine Berechnung wird dadurch noch langsamer, aber man darf nicht vergessen, daß nicht-zeitliche Constraints früher oder später sowieso berechnet werden müssen. Dorns Verdienst ist, beide Ansätze zu integrieren.

Hrycej verminderte Allens Darstellungskomplexität durch Einführung sog. transitiver Ketten [Hrycej 88a, b]. Sein Modell verbindet den zeitachsenorientierten Ansatz (s.u. Kahn/Gorry) mit dem intervallorientierten. Er klassifiziert Zeitrelationen in drei Gruppen, die man als Nachfolgerelationen, Einschlußrelationen und sonstige Relationen be-

zeichnen könnte. Nachfolgerelationen (*before, after, meets, met-by*) werden zu Vorranggraphen - den transitiven Ketten - zusammengestellt. Darüberhinaus gibt es Einschlußrelationen (*during, contains, starts/ends-with, started/ended-by*), die als Bäume strukturierbar sind, und sonstige Relationen (*overlaps, overlapped-by)* sowie Disjunktionen von Relationen). Um einen neuen Constraints einzufügen, greift er auf Allens Propagierungsalgorithmus zurück. Danach löscht er alle redundanten Verbindungen. Redundant ist eine Verbindung, die sich durch eine transitive Kette oder durch eine Kette von Verbindungen in einem Einschlußbaum darstellen läßt. Das zeitliche Netzwerk ist also stets ein Minimalgerüst[14] von Beziehungen. Das erhöht die Transparenz der Darstellung und verringert die Komplexität. Deswegen sind Hrycejs transitive Ketten für zeitliche Schlußfolgerungen besonders interessant.

5.3.4. Smiths constraintbasierter Ansatz

Grundsätzlich gibt es zwei Arten zeitlichen Wissens: absolutes zeitliches Wissen, das einer bestimmten Zeitperiode zugeordnet ist, und relatives zeitliches Wissen, bei dem Zeitobjekte ohne Beziehung zu einer bestimmten Zeit aufeinander bezogen werden [Smith 83]. Mit einem allgemeinen "time-line" Objekt wird ein zeitliches Koordinatensystem für absolutes zeitliches Wissen definiert. Es umfaßt die zeitliche Granularität (z.B. die Definition von Stunden, Tagen und Monaten und die Beziehungen zwischen den Einheiten) und eine Zeitarithmetik. Für relatives zeitliches Wissen greift Smith auf Allens 13 Zeitbeziehungen zurück.

Zeitbedingungen, beispielsweise an die Belegung einer Maschine, faßt Smith als Constraints auf, die gleichberechtigt neben anderen existieren. Zeitliches Schlußfolgern ist demzufolge ein Constraintsatisfizierungsproblem (s.o.). Die Constraints sind zeitvariant, d.h. einem Objekt können verschiedene alternative Constraints zugeordnet sein, und man muß sich für eines entscheiden. Die Alternativen nennt Smith "Variants" und organisiert sie in Baumform. Welche Variante eines Constraints anwendbar ist, hängt davon ab, welches Zeitintervall des Lösungsraumes gerade untersucht wird. Constraints entstehen durch Vorgaben oder als Ergebnis von Schlußfolgerungen. Smith unterscheidet zwischen Zeitobjekten mit absoluten Zeitangaben und mit relativen Zeitbeziehungen. Im Projekt ISIS fand Smith's Darstellung erstmals Verwendung (s.o.).

5.3.5. McDermotts verzweigtes Zeitmodell

D. McDermott [McDermott 82] entwickelte ein verzweigtes Zeitmodell, das auf einer Zeitpunktlogik basiert. Es erlaubt, Zeit kontinuierlich darzustellen und die Unsicherheit über zukünftige Entwicklungen auszudrücken. Die Menge der Zeitpunkte ist totalgeordnet und entspricht einem Kontinuum, d.h. zwischen zwei Zeitpunkten existiert stets mindestens ein dritter Zeitpunkt. Weil es unendlich viele Zeitpunkte gibt, wird eine Implementierung stets ineffizienter sein als eine, die mit diskreten Zeitpunkten arbeitet. Die Menge der Zustände ist über "vorher"-Beziehungen teilgeordnet. Jedem Zustand wird ein Zeitpunkt zugeordnet. Die Wurzel eines Zustandsbaums ist die Gegenwart, von der aus sich verschiedene Zustände, die unterschiedliche Zukunften ausdrücken, verzweigen. Zustände werden in sog. Chroniken zusammengefaßt, in geordneten Mengen von Zuständen, die eine Verzweigung darstellen. Wohl aufgrund der Komplexität fand McDermotts Zeitmodell in Reihenfolgeplanern bisher keine Verwendung.

Ebenfalls ein Zeitkontinuum verwenden Kahn und Gorry [Kahn 77], ein Ansatz, der auf die Analyse von Krankengeschichten zurückgeht. Zeit kann in drei Erscheinungsformen dargestellt werden. 1. Aussagen mit Referenz auf ein Datum werden an der entsprechenden Stelle auf der Zeitgeraden abgespeichert. 2. Der Benutzer kann Referenzereignisse festlegen, auf die andere Ereignisse Bezug nehmen. 3. Ereignisse können

[14] Def. Minimalgerüst vgl. [Neumann 75b, S. 72]

sequentielle Vorher-Nachher-Ketten bilden. Der "Zeitspezialist" besitzt über die Anordnung von Ereignissen hinaus keine speziellen Inferenzmechanismen. Vorteile dieses Ansatzes sind die Transparenz der Zeitdarstellung und die Möglichkeit, mit zeitlich vagem Wissen umgehen zu können. Außer dem Fehlen eines mächtigen Inferenzmechanismus ist für Anforderungen der Reihenfolgeplanung von Nachteil, daß Ereignisse nicht parallel sondern nur sequentiell angeordnet werden können. Schließlich ist es nicht möglich, Ereignisse als Intervall darzustellen. Vielmehr müssen stets die Intervallgrenzen beschrieben werden. Damit ist der Ansatz für Planungszwecke nur sehr begrenzt geeignet.

5.3.6. Zeitfenster

Hauptanliegen von Veres DEVISER war, eine Technik zum Schlußfolgern über fixe Zeitpunkte und Zeitdauern zu entwickeln [Vere 83], um daraus einen Planer zu bauen. DEVISER berücksichtigt, wann bestimmte Ziele erreicht werden und wie lange Zielzustände gelten sollen. Aktionen und Zielen sind Fenster zugeordnet, Aktionen besitzen Dauern. Ein Fenster besteht aus den Attributen "Frühester Startzeitpunkt", "Idealer Startzeitpunkt", "Spätester Startzeitpunkt". Dauern können durch absolute Werte oder durch Variablen dargestellt sein; in diesem Fall ist die Dauer unbekannt. Wenn DEVISER den Wert einer Variablen ermittelt hat, propagiert er ihn durch den Plan.

Über Fenster sind syntaktische Operationen erklärt. Kernoperationen sind Linking, Knotenerweiterung und Konfliktentdeckung. Für ein bereits realisiertes Ziel G und einen Knoten E (Aktivität) im Netzwerk bedeutet Linking, daß G mit E verknüpft wird. Falls die Aktivität noch nicht existiert, muß das Netzwerk um den Knoten erweitert werden. Dazu wird eine Aktivität gesucht, die G erfüllt. Die Vorbedingung dieser Aktivität ergibt ein neues Teilziel. Aus den Kernoperationen entsteht eine sequentielle Teilordnung der Aktionen. Für jedes Fenster sind mindestens der früheste und der späteste Startzeitpunkt bekannt. Ist darüberhinaus eine Dauer bekannt, kann das Fenster "komprimiert" werden, d.h. durch Propagation der Dauer auf Vor- und Nachaktivitäten wird das Intervall von frühestem und spätestem Startzeitpunkt eingeschränkt. Gerät dabei der ideale Startzeitpunkt außerhalb des Intervalls, wird er an die nächstgelegene Grenze verschoben. Das Ergebnis ist ein PERT-Plan. Für die zeitliche Schlußfolgerung ist DEVISER deswegen interessant, weil er über teilbestimmte, sequentiell angeordnete Intervalle inferiert, nämlich solche, die bereits Dauern besitzen. Von Dauern ausgehend erhält man am Ende Start- und Endzeitpunkte für Vorgänge. Dabei spielt es keine Rolle, an welchem Knoten des Netzwerks man beginnt. Durch die Möglichkeit, auch mit vorerst unbekannten Dauern zu arbeiten und für Startzeitpunkte einen Wertebereich vorzugeben, besitzt DEVISER ein hohes Maß an Flexibilität. Durch das Prinzip der ständigen Konsistenzsicherung im ganzen Netzwerk hängt die Effizienz von DEVISER von der Zahl der Knoten und der Zahl der Verbindungen zwischen ihnen ab.

Auch DEVISER plant nicht-linear. Weil er auch mit vorerst unbekannten Dauern arbeitet und für Startzeitpunkte einen Wertebereich vorgeben kann, besitzt er ein hohes Maß an Flexibilität.

5.3.7. Zusammenfassung der Zeitplanung

Zur Darstellung von Zeiten in Reihenfolgeproblemen eignet sich besonders eine intervallbasierte Zeitlogik, weil sie Prozesse angemessener abbildet als eine zeitpunktbasierte Logik. Der wichtigsten Ansatz zur Schlußfolgerung über unbestimmte Intervalle stammt von Allen, dessen Netzwerk allerdings sehr komplex wird. Referenzintervalle garantieren keine wesentliche Komplexitätsreduktion, schränken aber die Ausdrucksfähigkeit ein. Ein vielversprechender Ansatz sind Hrycejs transitive Ketten, die die Darstellungskraft nicht einschränken, die Transparenz erhöhen und die Komplexität erheblich reduzieren. Smiths Idee, zeitliche Propagierung auf ein Constraintsatisfizierungs-

problem zurückzuführen, schlägt eine Brücke zu Inferenzen über nicht-zeitliche Probleme. Dagegen erscheint McDermotts Ansatz zwar als derjenige, der Zeitverhältnisse am genauesten darstellt, aber bisher gibt es für ihn weder starke Schlußfolgerungsmechanismen, noch eine Darstellung und Schlußfolgerung, die in ihrer Komplexität vertretbar sind. Hat man sich für einen intervallorientierten Ansatz entschieden, dann sind Veres Zeitfenster eine geeignete Brücke zu festen Zeiten.

Konventionelle Zeitplaner erfüllen weder die Kriterien an Ausdrucks- noch an Planungsfähigkeiten, die für ein allgemeines Werkzeug zur Unterstützung der Reihenfolgeplanung gestellt werden müssen. Wünschenswert ist, Hrycejs und Allens Ansatz auch für quantitative Zeitplanung zu öffnen, etwa im Sinne von Veres Zeitfenstern. Dazu ist jedoch einiges an Forschung notwendig.

5.4. Konsequenzen

Im Rahmen dieses Kapitels wurden weitere Gemeinsamkeiten von Reihenfolgeproblemen aufgezeigt:

- In Planungsumgebungen fallen auch Entscheidungsprozesse über Randbedingungen (vgl. S. 19). Sie können in einigen Fällen Gegenstand der Planung werden.
- Die Planungsumgebung wird oft hierarchisch strukturiert (vgl. S. 19f).
- Unterschiedliche temporale Informationen sind für einige Reihenfolgeprobleme von Bedeutung (vgl. S. 37f).

Zur Darstellung werden fast immer Prädikaten-/Transitionsnetze und Aktivitäts-/Zustandsnetze gewählt. Dazu kommen spezielle, oft graphisch orientierte Werkzeuge zur Modellierung von Fabrikumgebungen. Einen Vergleich enthält Tabelle 5.1.:

Kriterium	Petri-Netze	Aktivitäts/Zustands-Netze	Modellierungswerkzeuge
Darstellung von Zustandsräumen und UND/ODER-Graphen und Unterstützung der Suche	Petri-Netze wurden zur Simulation geschaffen (siehe S. 15). Durch Hinzufügen oder Entfernen von Marken findet kein Suchprozeß statt.	Volle Unterstützung von UND/ODER-Graphen. Zustandsräume darzustellen wird nicht unterstützt.	Ähnlich wie Petri-Netze. Bisher ist dem Verfasser kein Modellierungstool bekannt, das auch nur annähernd dieser Anforderung genügt.
Integration der Konzepte Hierarchie, Kausalität, Zeit	Sehr gute Darstellung von Kausalitäten. Für Hierarchien werden Erweiterungen benötigt. Trotz zahlreicher Ansätze ist die Zeitdarstellung nicht sehr befriedigend.	Unterstützung aller drei Konzepte.	Es gibt Tools, welche die Modellierung von Hierarchien unterstützen. Zeitliche Informationen werden sehr einseitig unterstützt, z.B. a priori bekannte Dauern.
Modellierung von Nebenbedingungen, z.B. einer Fabrikumgebung	Seit Jahren bewährt. Es gibt graphische Modellierungstools, die die Arbeit sehr gut unterstützen.	Gut, aber wenig verbreitet. Es gibt außerdem keine Tools dazu.	Hervorragend, oft mit graphischer Unterstützung, aber sehr anwendungsspezifisch.

Tab. 5.1.: Vergleich von Darstellungsformen für Zwecke der Reihenfolgeplanung

Keine Darstellungsform erfüllt alle Erwartungen. Petri-Netze und gängige Modellierungswerkzeuge sind für Planungszwecke nur begrenzt geeignet, weil sie keine Möglichkeit bieten, den Suchprozeß darzustellen. Aktivitäts-/Zustandsnetzwerke unterstützen ein Darstellungskonzept. Zeitsparende Tools zur Modellierung einer Umgebung fehlen. Der Erweiterungsbedarf ist bei Aktivitäts/Zustandsnetzen jedoch geringer als bei anderen Darstellungsformen.

Beide Darstellungen, Petri-Netze und A/S-Netze erlauben präzise Beschreibungen. Weil jedoch bei Petri-Netzen mächtigere Bausteine fehlen, kann es zu Inkonsistenzen in der Darstellung kommen, etwa wenn hierarchische Beziehungen einmal als Stellenverfeinerungen, ein weiteres Mal als Transitionsverfeinerungen dargestellt werden. Bei A/S-Netzen lassen sich unterschiedliche Hierarchien leicht zusammenfügen. Für die Planung werden nach Sathi A/S-Netze den Kriterien Vollständigkeit, Präzision und Klarheit besser gerecht als andere Darstellungen (vgl. [Sathi 85, S. 551]). Deswegen bauen die Konzepte für den im folgenden Kapitel entwickelten Reihenfolgeplaner auf Aktivitäts-/Zustandsnetzwerken auf.

Welche Vorteile hat ein einziges Werkzeug gegenüber einer Reihe spezialisierter Werkzeuge? Puppe [Puppe 89, S. 144f] unterscheidet zwischen

1. Programmiersprachen,
2. allgemeinen Werkzeugen für die Wissensrepräsentation,
3. problemspezifischen Werkzeugen (Shells) und
4. Shells mit Basiswissen.

Von 1. nach 4. nimmt die Zeitersparnis bei der Entwicklung zu, während sich die Breite des Anwendungsgebietes verringert. Jedes dieser Werkzeuge hat seine Berechtigung.

Für den Spezialfall eines Werkzeugs zur Planung von Reihenfolgen gilt: Zwar gibt es Werkzeuge zur Modellierung einer Fabrikumgebung, beispielsweise für die Maschinenbelegungsplanung, aber sie sind fast immer auf bestimmte Anwendungen beschränkt. Für viele speziellere Probleme sind keine Lösungen erhältlich. Eine Problemlösung mit weniger mächtigen Werkzeugen ist sehr aufwendig. In diese Lücke stößt das in dieser Arbeit anvisierte Werkzeug. Weil ähnliche Konzepte zugrunde gelegt werden können, hält sich der Mehraufwand zur Entwicklung eines allgemeineren Werkzeugs in Grenzen.

Was ist als nächstes zu tun? A/S-Netze sind so zu erweitern, daß sie beide Formen zur Repräsentation von Reihenfolgeproblemen unterstützen - und mit ihnen die Konzepte Zeit, Hierarchie und Kausalität. Aus den am Schluß von Kapitel 4 gefundenen Bausteinen für Lösungsverfahren entsteht ein allgemeines Lösungsschema, das konkreten Problemen angepaßt werden kann. Schließlich ist die Zeitdarstellung und -berechnung zu erweitern, um einem allgemeinen Werkzeug zu genügen. Eine Ausgangsbasis bilden dabei die Arbeiten von Allen, Vere und Hrycej.

Weil Reihenfolgeprobleme oft zeitbehaftet sind und Zeitplanung nur schlecht mit diskreter Propagierung erfolgt (vgl. die Absätze ab S. 37), soll die zeitliche Propagation getrennt entwickelt werden. Diese Auffassung wurde auch dort vertreten, wo A/S-Netze zum erstenmal entwickelt wurden, etwa im Projekt ISIS (vgl. [Fox 84]). Gesucht ist ein durchgängiges Konzept zur Planung unterschiedlicher zeitlicher Informationen, das mit der Darstellung in A/S-Netzen zusammenpaßt.

6. Ein Reihenfolgeplanungsansatz

Die wesentlichen Darstellungs- und Lösungskonzepte sind Gegenstand dieses Kapitels. Es zeigt einen formalen Ansatz auf, um Reihenfolgeprobleme entlang der ermittelten Kriterien einheitlich darzustellen. Um Heuristiken leichter erstellen zu können, entwikkelt es einen Entwicklungsrahmen. Der zeitlichen Planung ist ein gesonderter Ansatz gewidmet. Abschließend zeigt es die Flexibilität der Konzepte am Beispiel des opportunistischen Schließens.

6.1. Ein formaler Ansatz

Das vorige Kapitel zeigte, warum die Aktivitäts-/Zustandsdarstellung als Ausgangspunkt für Reihenfolgeprobleme am besten geeignet ist. Sie motiviert zu einem Formalismus, der die wichtigsten Ideen graphentheoretisch festlegt, um Zustandsräume und Problemreduktionsgraphen zu modellieren. Mit Zustandsräumen und Problemreduktionsgraphen lassen sich nach Pearl so gut wie alle Lösungsverfahren angemessen darstellen. Ein allgemeines Werkzeug zur Reihenfolgeplanung sollte deswegen beide Darstellungsformen bereithalten. Es läßt sich beweisen, daß die hier gefundenen Darstellung zu Pearls Konzepten äquivalent ist. Beispiele aus der Anwendung zeigen schließlich, wie in der Praxis mit der gefundenen Darstellung modelliert werden kann.

6.1.1. Graphentheoretische Grundlagen

Analog zur Formalisierung von Reduktionsgraphen sollen A/S-Netze formalisiert werden. V gibt im Folgenden stets eine Menge von Knoten, E eine Menge von Pfeilen zwischen Knoten an. Die Elemente von V werden oft mit v, die von E oft mit e bezeichnet. Alle Objekte des A/S-Netzes werden auf eine endliche Knotenmenge V_{AS} abgebildet:

Definition 6.1: Die Knotenmenge V_{AS} repräsentiere die Menge aller Objekte O, für die gilt: O ist ein für ein gegebenes Reihenfolgeproblem relevantes Objekt.

Objekte werde auf drei Arten disjunkt aufgeteilt, so daß für jeden Knoten $v \in V$ gilt:

1. v ist entweder *UND*- oder *ODER*-Knoten:
   $v \in UND \vee v \in ODER$,
   mit $UND \cap ODER = \emptyset$ *und* $UND \cup ODER = V$
2. v ist entweder Zustand Z, Aktivität A oder Objekt einer Nebenbedingung NB:
   $v \in Z \vee v \in A \vee v \in NB$, *mit* A, Z, NB *disjunkt und* $A \cup Z \cup NB = V$.
3. v ist entweder Prototyp *PROT* oder Instanz *INST*:
   $v \in PROT \vee v \in INST$,
   mit $PROT \cap INST = \emptyset$ *und*
   $PROT \cup INST = V$
   Instanzen dürfen keine Objekte der Nebenbedingung sein: $INST \cap NB = \emptyset$.

Sind zwei Knoten v und w mit $v, w \in PROT$ oder $v, w \in INST$, dann sind sie von der gleichen Individuationsebene.

Prototypen dienen zum Aufbau des A/S-Digraphen. Objekte der Nebenbedingungen dienen zum Aufbau der taxonomischen Hierarchie (s.u.). Prototypen werden instanziiert, um in die Instanzen zu planen. Eine zusammengesetzte Aktivität $a \in A \cap UND$ besteht aus keiner, einer oder mehreren Einzelaktivitäten, die alle gelten müssen. Eine Einzelaktivität $a \in A \cap ODER$ besteht aus keiner, einer oder mehreren zusammengesetzten

Aktivitäten, von denen mindestens eine ausgeführt sein muß. Für Zustände gilt das analog.

Die Menge der Zustände wird disjunkt unterteilt in Vor- und Nachbedingungen *VORB* und *NACHB*. Außerdem wird unterschieden zwischen Besitzprädikaten *BP* und Zustandsprädikaten *ZP*. Charakteristika sind: Ein Besitzprädikat belegt eine Ressource (Objekt der Nebenbedingung) über einen Zeitabschnitt. Ein Zustandsprädikat prüft in der Vorbedingung Eigenschaften eines Objektes aus NB, in der Nachbedingung werden dem Objekt Eigenschaften zugesichert. Für jeden Zustand $z \in Z$ gilt:

1. z ist entweder Vor- oder Nachbedingung: $z \in VORB \vee z \in NACHB,$ *mit* $VORB \cap NACHB = \emptyset$ *und* . $VORB \cup NACHB = Z$
2. z ist entweder Zustands- oder Besitzprädikat: $z \in ZP \vee z \in BP,$ *mit* $ZP \cap BP = \emptyset$ *und* $ZP \cup BP = Z$.

Für jeden Knoten $v \in INST$ sollen zwei Funktionen w und u existieren, die einem Knoten v eine Bewertung $w(v)$ bzw. $u(v)$ zuordnen. Dabei seien w und u wie folgt definiert: w: v $\rightarrow$ {wahr, falsch, unbekannt}und $u: v \rightarrow IR$. $w(v)$ stellt einen Wahrheitswert, und $u(v)$ einen Nützlichkeitswert dar. Die Abbildungen dienen zur Markierung von Knoten für die Suche im Digraphen. Die Nützlichkeitsbewertung dient zum Vergleich von Knoten, wenn Auswahl besteht. Darüberhinaus kann jedem Knoten eine Menge von Attributen zugeordnet werden. Ein Attribut besteht aus einem Attributnamen und einem Attributwert. Als Attributwert ist für Vorbedingungen ein Prädikat erlaubt.

Ein Knoten heißt *verifizierbar*, wenn für ihn entschieden werden kann, ob er falsch oder wahr ist. Eine Menge von Relationen R_{AS} werde in paarweise disjunkte Teilmengen der

hierarchischen Relationen ={*has-elaboration, has-sub-state, has-sub-activity*},

kausalen Relationen ={*enabled-by, cause, cause-enable*}

taxonomischen Relationen ={*is-a, has-instance*} und

sonstigen Relationen {*next-activity, has-revision, required-by, possessed-by, has-goal*}

aufgeteilt. Dabei gilt: R_{AS} = hierarchischen Relationen ∪ kausalen Relationen∪taxonomischen Relationen ∪ sonstigen Relationen.

Hierarchische Relationen dienen der hierarchischen Verfeinerung. Ausarbeitungen verfeinern ODER-Knoten mit *has-elaboration*, *has-sub-state* verfeinert UND-Zustände und *has-sub-activity* UND-Aktivitäten. Kausale Relationen stellen Verbindungen nach diesem Schema her: Vorbedingung$_1$ *enable*[1] Aktivität *cause* Nachbedingung *cause-enable* Vorbedingung$_2$. *Cause-enable* bedeutet, daß die Nachbedingung einer Aktivität Vorbedingung einer anderen ist. Das wird in zwei getrennten Knoten dargestellt.

R_{AS} heißt Menge der A/S-Relationen. Außerdem erklären wir die

Clusterrelationen = {hierarchische Relationen ∪ kausale Relationen \ *cause-enable*}

Graphisch lassen sich die Knoten und Relationen, die zwischen ihnen bestehen, durch einen bewerteten Digraphen im weiteren Sinne darstellen.

Definition 6.2: Es sei D = *[V, E]* ein Digraph, *B* eine beliebige Menge und c_B: $E \circledS B$ eine Abbildung, die jedem Pfeil $e \in E$ eine Beschreibung oder Bewer-

[1] Gegenbeziehung zu enabled-by

tung $c_B(e)$ zuordnet. Dann heißt das Tripel $D_B = [V, E, c_B]$ ein *beschriebener Digraph* oder *bewerteter Digraph im weiteren Sinne.*

Zur Erinnerung: Ein bewerteter Digraph ist ein Digraph mit einer Abbildung $c: E \rightarrow \mathbb{R} \cup \infty$, d.h. jeder Kante werden bestimmte Kosten zugewiesen. Ein bewerteter Digraph im weiteren Sinne erlaubt, zusätzliche Inhalte in die Darstellung mitaufzunehmen. Diese Erweiterung ist nötig, um Relationen zwischen A/S-Objekten explizit aufzuführen. Wir definieren daher weiter:

Definition 6.3: Es sei $D_{AS} = [V_{AS}, E, c_B]$ ein beschriebener Digraph mit $c_B = E \mapsto R_{AS}$, dann heißt D_{AS} auch A/S-Digraph[2].

Die Clusterrelationen der Tabelle 6.1 zwischen zwei Knoten v, w einer Individuationsebene sind zulässig unter der Bedingung, daß es keinen Weg von w nach v gibt, der mindestens eine Clusterrelation enthält. Eine in der Spalte "Relationsname" aufgeführte Relation kann zwei Knoten v und w verknüpfen, wenn sie den Bedingungen in der entsprechenden Zeile genügen.

von Knoten v	**nach Knoten w**	**Relationsname**
$v \in A \cap UND$	$w \in A \cap ODER$	has-sub-activity
$v \in VORBED \cap UND$	$w \in VORBED \cap ODER$	has-sub-state
$v \in NACHBED \cap UND$	$w \in NACHBED \cap ODER$	has-sub-state
$v \in A \cap ODER$	$w \in A \cap UND$	has-elaboration
$v \in NACHBED \cap ODER$	$w \in NACHBED \cap U$	has-elaboration
$v \in VORBED \cap ODER$	$w \in VORBED \cap U$	has-elaboration
$v \in A$	$w \in VORBED$	enabled-by
$v \in A$	$w \in NACHBED$	cause
$v \in A$	$w \in VORB \wedge w \notin BP$	has-goal

Tab. 6.1: Clusterrelationen zwischen zwei Knoten einer Individuationsebene

Definition 6.4: In einem A/S-Digraphen D = [V, E, R] heißt ein Knoten $s \in V$ *Startknoten*, wenn kein $v \in V$ existiert mit v *rel* s, und *rel* als hierarchischer Relation.

Ein Startknoten ist also ein Knoten ohne Vorgänger. In einem A/S-Digraphen darf es hiervon mehrere geben.

Definition 6.5: A/S-Cluster: Gegeben sei ein A/S-Digraph $D = [V_{AS}, E_{AS}, R_{AS}]$ für den gilt:

Er enthält einen Startknoten $s \in A$.

Seien $V_{Cluster}$ die Menge aller Knoten v, die ausgehend von s ausschließlich über Clusterbeziehungen erreichbar sind, $E_{Cluster} \subseteq E_{A/S}$ die Verbindungen zwischen den Knoten von $V_{Cluster}$ und $R_{Cluster} \subseteq R_{A/S}$ die Bewertungen der Verbindungen. Dann heißt der Teilgraph $D_{Cluster} \subseteq D$ mit $D_{Cluster} = [V_{Cluster}, E_{Cluster}, R_{Cluster}]$ auch A/S-Cluster.

Satz 6.1: Ein A/S-Cluster ist zyklenfrei bezüglich R_{AS}.

[2]Man könnte den A/S-Digraphen auch als vollständigen Graphen definieren, indem man zusätzlich die Inversen der Relation zuläßt. Durch die Doppelbindung wird jedoch keine zusätzliche Information dargestellt. In KnowledgeCraft werden die Inversen automatisch erzeugt.

Beweis: Laut Definition: Eine Verbindung zwischen zwei Knoten darf nur dann etabliert werden, wenn es in Gegenrichtung keinen Weg gibt.

In einem A/S-Cluster gibt es nur einen Startknoten. Das Zustandscluster stellt eine Hierarchie von Zuständen dar:

Definition 6.6: Gegeben sei ein A/S-Digraph $D = [V_{AS}, E_{AS}, R_{AS}]$ für den gilt:

Er enthält einen Startknoten $s \in Z$.

Seien $V_{Cluster}$ die Menge aller Knoten v, die ausgehend von s ausschließlich über hierarchische Beziehungen erreichbar sind, $E_{Cluster} \subseteq E_{A/S}$ die Verbindungen zwischen den Knoten von $V_{Cluster}$ und $R_{Cluster} \subseteq R_{A/S}$ die Bewertungen der Verbindungen. Dann heißt der Teilgraph $D_{Cluster} \subseteq D$ mit $D_{ZCluster} = [V_{ZCluster}, E_{ZCluster}, R_{ZCluster}]$ auch *Zustandscluster*.

Über Clustergrenzen hinweg erlauben wir zwischen zwei Knoten gleicher Individuationsebene v, w Verbindungen mit den Relationen der Tabelle 6.2, den Clusterverbindungsrelationen:

von Knoten v	**nach Knoten w**	**Relationsname**
$v \in ZP \cap VORBED$	$w \in ZP \cap NACHBED$	*cause-enable*
$v \in ZP \cap NACHB$	$w \in ZP \cap NACHB$	*has-revision*
$v \in A$	$w \in A$ und es gibt keine u_1, $u_2 \in A$ mit $u_1 \neq w$ und $u_2 \neq v$ so, daß gilt: v *next-activity* u_1 oder u_2 *next-activity* w. (Verbot von Verzweigungen)	*next-activity*

Tab. 6.2: Clusterverbindungsrelationen für Beziehungen über Clustergrenzen hinweg

Außerdem muß für jede der drei Clusterverbindungsrelationen gelten[3]:

1. $\exists$ zwei Cluster C_v, C_w mit $(V_v, E_v, R_v), (V_w, E_w, R_w)$ und $v \in V_v \wedge w \in V_w \wedge V_v \cap V_w = \varnothing$

2. für jede zwei Cluster C_1, C_2 mit $(V_1, E_1, R_1), (V_2, E_2, R_2)$ und $v \in V_1 \wedge w \in V_2 \wedge V_1 \cap V_2 = \varnothing$ und für zwei beliebige Knoten $u_1 \in V_1, u_2 \in V_2$ gilt stets: Es gibt keinen Weg von u_2 nach u_1, der wenigstens eine Beziehung r $\in$ {*cause-enable*, *has-revision*, *next-activity*} enthält.

Um ein Objekt der Nebenbedingung über Zustandsprädikate von Nachbedingungen entsprechend dem Planungsfortschritt weiterzuentwickeln, dient die Beziehung *has-revision*. Die Eigenschaften des prototypischen Objekts bzw. dessen Revisionen werden in Gegenrichtung der Beziehung voll vererbt. *Next-activity* stellt Vorranggraphen dar, die z.B. aufgrund einer Sortierung entstanden sind.

Für die Suche in einem Digraphen ist wichtig, ob unendlich lange Wege existieren, ob der Digraph also Zyklen enthält. Ist das nicht der Fall, wird die Suche einfacher.

[3]Für Reihenfolgeprobleme, in denen ein Element in einer Sequenz mehrfach vorhandensein darf, ist diese Einschränkung zu relaxieren.

Satz 6.2: Gegeben sei ein A/S-Graph $D_{AS} = [V_{AS}, E_{AS}, R_{AS}]$. Es sei $V_{Inst} = V_{A/S} \cap INST$, D_{Inst} der zugehörige Teilgraph aus D_{AS} mit $[V_{Inst}, E_{Inst}, R_{Inst}]$, dann gilt:

D_{Inst} ist zyklenfrei.

Beweis: Zu zeigen ist, daß der Rang jedes Knotens aus D_{Inst} endlich ist, d.h. daß bei n Knoten die längste Pfeilfolge jedes Knotens < n ist. Knoten von Interesse sind ausschließlich Aktivitäten oder Zustände. Clusterverbindungen müssen nicht untersucht werden, weil Cluster zyklenfrei sind (s.o.).

Annahme: Es existiert ein Zyklus, so daß für zwei Knoten v, w gilt: ∃ ein Weg von v nach w und ∃ ein Weg von w nach v.

Fall 1: v, w in einem Cluster, d.h. es gibt keine zwei Cluster C_v, C_w mit $(V_v, E_v, R_v), (V_w, E_w, R_w)$ und $v \in V_v \wedge w \in V_w \wedge V_v \cap V_w = \varnothing$. Daraus folgt: Bedingung 1 der Bedingungen für Cluster*verbindungs*relationen ist nicht erfüllt. Zwischen v und w können nur Clusterrelationen existieren und ein Cluster ist bezüglich Clusterrelationen zyklenfrei.

Fall 2: v, w nicht in einem Cluster, d.h. die erste Bedingungen für Cluster*verbindungs*relationen ist erfüllt. Dann gibt es keinen Weg zwischen v und w bzw. zwischen w und v, der nur Clusterrelationen[4] enthält. Jeder Weg muß also mindestens eine Cluster*verbindungs*relation enthalten. Das ist aber nach der zweiten Bedingung für Cluster*verbindungs*relationen unzulässig.

Wahrheitswerte sollen nur für Instanzen bestimmt werden. Knoten im A/S-Digraphen können unter folgenden Bedingungen, die alle erfüllt sein müssen, verifiziert werden:

Ein Knoten $v \in$ Menge der Instanzen ist *wahr*, wenn

1. jeder Nachfolgeknoten w wahr ist, mit v *has-sub-activity* w, falls von v mindestens eine *has-sub-activity* Relation ausgeht
2. jeder Nachfolgeknoten w wahr ist, mit v *has-sub-state* w, falls von v mindestens eine *has-sub-state* Relation ausgeht
3. mindestens ein Nachfolgeknoten w wahr ist, mit v *has-elaboration* w, falls von v mindestens eine *has-elaboration* Relation ausgeht
4. mindestens ein Nachfolgeknoten w wahr ist, mit v *enable* w, falls von v mindestens eine *enable* Relation ausgeht
5. mindestens ein direkter Vorgängerknoten w *wahr* ist, falls v ∈ NACHBED und von w nach v eine hierarchische Beziehung besteht
6. falls $v \in ZP$: ∃ $w \in ZP$ mit w *cause-enable* v gilt: w ist *wahr*

Schließlich benötigen wir taxonomische und revisionale Relationen zwischen zwei Knoten v, w, die hauptsächlich Beziehungen zwischen Prototypen und Instanzen sind. *Is-a* dient zur Klassenbildung und vererbt in Gegenrichtung. Vererbung erfolgt über *has-instance* von Prototypen auf Instanzen. *Possessed-by* verbindet ein Objekt mit dem Besitzprädikat, das es belegt. Dieses wurde vorher mit *required-by* spezifiziert.

[4]Clusterrelationen: hierarchische und kausale ohne cause-enable. Nicht zu verwechseln mit Clusterverbindungsrelationen: cause-enable, has-revision, next-activity.

von Knoten v	nach Knoten w	Relationsname
$v \in PROT$	$w \in PROT$ und es gilt: w und v sind nur über is-a oder nur hierarchisch miteinander verbunden.	is-a
$v \in PROT$ und es gilt: $(v \in A \wedge w \in A) \vee (v \in Z \wedge w \in Z)$	$w \in INST$	has-instance
$v \in NB$	$w \in ZP \cap NACHB \cap INST$ und es gilt: Es gibt kein u $\in V_{A/S}$ mit u *cause-enable* w. (w darf nicht von einem anderen Knoten abhängen).	has-revision
$v \in NB$	$w \in VORB \cap INST$	required-by
$v \in NB$	$w \in BP \cap VORB \cap INST$	possessed-by

Tab. 6.3: Revisionale Beziehungen

Ein prototypisches Objekt wird von Zustandsprädikaten revidiert, die ihrerseits von Zustandsprädikaten weiter revidiert werden können. Für die Verifikation haben required-by und has-revision unterschiedliche Aufgaben. Werkzeug required-by Zustandsprädikat einer Aktivität legt fest, daß für die Aktivität auf den Zustand irgendeines Werkzeugs Bezug nimmt. Stehen Werkzeug-1 und Werkzeug-2 zur Verfügung, Werkzeug-2 werde ausgewählt und seine letzte Revision sei z.B. ein Zustand Reststandzeit = 10 min, dann kann die Nachbedingung der Aktivität diesen Zustand mit z.B. Reststandzeit = 8 min revidieren.

Satz 6.3: Der so entstandene Digraph ist zyklenfrei für alle Relationen außer *is-a*.

Beweis: Außer *is-a* beschreiben die dazugekommenen Beziehungen nur Verbindungen zwischen zwei Individuationsebenen. Jeder Teildigraph, der ausschließlich aus Instanzen besteht, ist zyklenfrei (s.o.). Das gleiche gilt für Teilgraphen aus Prototypen, die außer *is-a* die gleichen Beziehungen wie Instanzen haben. Bleibt die Frage, ob Zyklen zwischen den Teildigraphen entstehen können. Das ist aber unmöglich, weil es keine Relation und damit keinen Weg von einer Instanz zu einem Prototypen gibt.

Mit folgenden zusätzlichen Regeln kann ein Knoten im A/S-Digraphen verifiziert werden:

$v \in ZP \cap VORB$ ist *wahr*, falls es ein $u \in PROT$ gibt mit u *required-by* v und falls u die in den Attributen von v spezifizierten Prädikate erfüllt.

$v \in BP$ ist wahr, falls es ein w gibt mit w *possessed-by* v und die Einplanung von v auf w zulässig ist.

Bei der Verifizierung sind folgende Regeln zusätzlich zu beachten:

Die letzte Revision eines Knotens $w \in NB$ ermittelt sich wie folgt: Für einen Knoten u_1 gilt: 1. Es gibt einen Weg ausschließlich über *has-revision* Verbindungen von w nach u_1 oder u_1 ist identisch mit w. 2. u_1 hat keinen Nachfolger u_2 mit u_1 *has-revision* u_2.

Ist ein $v \in NACHB \cap ZP$ *wahr,* dann ist die letzte Revision w eines durch w *required-by* v spezifizierten Objekts zu revidieren, d.h. es gilt: w *has-revision* v.

6.1.2. Formalisierung des Zustandsraumdigraphen

Zuerst werden die Konzepte Zustandsraumcluster und Problemreduktionsgraph mit A/S-Digraphen beschrieben. Der Klassifizierung von Reihenfolgeprobleme von Müller-Merbach (vgl. Kapitel 4) in die Klassen A und B entsprechend werden Problemklassen individuell dargestellt. Um den Vergleich mit Pearls Konzepten von Zustandsraum und Problemreduktionsgraph zu vereinfachen, erklären wir Probleme, die ohne Zustände verplant werden können, für "klassisch":

Definition 6.7: Ein A/S-Digraph D = [V, E, R] heißt *klassisch*, wenn für jeden Knoten $v \in V$ gilt: $v \in A$.

6.1.2.1. Zustandsraumcluster

Definition 6.8: Ein Teildigraph eines A/S-Cluster D=[V, E, R] mit einem Startknoten s heißt *Zustandsraumcluster*, wenn für zwei Knoten v, w ∈ V gilt: wenn v *rel* w mit *rel* als hierachischer Relation, dann ist v der Startknoten.

Ein Zustandsraumcluster besteht also aus zwei Hierarchieebenen: Ein Startknoten, der hauptsächlich Verwaltungszwecken dient und seinen hierarchischen Nachfolgern. Jeder von ihnen repräsentiert einen Schritt in einem Plan. Der Startknoten ist notwendig, um die Elemente eines Zustandsraums gegen andere abzugrenzen. Vor einer Planung besteht zwischen ihnen nicht notwendigerweise eine direkte Beziehung. Ein Reihenfolgeproblem vom Typ[5] A plant alle vorkommenden Elemente in die Reihenfolge ein:

Definition 6.9: Ein Zustandsraumcluster heißt *Zustandsraumcluster vom Typ A*, wenn für den Startknoten s gilt: $s \in UND$.

Den Zusammenhang zwischen den Elementen e eines Zustandsraums und dem Zustandsraumcluster stellt eine Funktion her:

Definition 6.10: Gegeben sei ein Zustandsraum ZR mit den Elementarproblemen e und ein Zustandsraumcluster D = [V, E, R] und Startknoten s. Eine eineindeutige Funktion f heißt *Zustandsraumabbildung* von ZR nach D, wenn sie jedes Elementarproblem e auf einen Knoten $v \in V$ abbildet, so daß gilt:

$f: e \rightarrow V \backslash s$ und $g := f^{-1}: v \rightarrow ZR$ mit g(f(e)) = e.

6.1.2.2. Sequenzen im Zustandsraumcluster

Was ist die Lösung eines Problems im Zustandsraumgraphen vom Typ A? Im klassischen Fall ist sie ein Weg, der die Blätter des Graphen zu einem Weg verbindet. Zur Darstellung der Verbindung wählen wir die Beziehung *next-activity*. Von einem beliebigen Blattknoten aus wird ein Nachfolger gesucht, dafür wieder ein Nachfolger usw. bis alle Knoten einen Nachfolger gefunden haben.

Definition 6.11: In einem Zustandsraumcluster D = [V, E, R] mit Startknoten s heißt ein Weg ausschließlich über *next-activity* zwischen zwei Knoten v_1 und v_2 *gewählte Sequenz seq* mit Anfangsknoten v_1 und Endknoten v_2; $v_1, v_2 \in V \backslash s$, wenn gilt:

1. Es gibt kein $u_1 \in V$ mit u_1 *next-activity* v_1.

[5] vgl. die in Kapitel 4.2.1 beschriebene Klassifikation von H. Müller-Merbach

2. Es gibt kein $u_2 \in V$ mit v_2 *next-activity* u_2.

3. für jeden Knoten $v \in seq$ gilt: v ist *wahr*.

Eine gewählte Sequenz ist also ein Teildigraph eines Zustandsraumclusters, dessen Anfangsknoten keinen Vorgänger über *next-activity* und dessen Endknoten keinen Nachfolger über *next-activity* hat. Sollte kein Knotenpaar mit *next-activity* verknüpft sein, kann ein Knoten v ∈ V\s als einelementige gewählte Sequenz betrachtet werden.

Next-activity legt lediglich eine Reihenfolge für die Erweiterung von Teillösungen fest. Die Beziehung gehorcht einer kausalen Halbordnung der Elemente des Suchraums, weil sie laut Definition nicht in Gegenrichtung zu den Beziehungen *cause-enable* und *revision-of* stehen darf. Die Beziehung impliziert dagegen keine zeitliche *nachher*-Beziehung! Das bedeutet, daß in einem Zustandsraumcluster D = [V, E, R] mit zwei Knoten v, w ∈ V gelten kann: v *next-activity* w ∧ v *after* w.

Ein Beispiel dazu aus der Maschinenbelegungsplanung: v ist ein Vorgang auf Maschine M1, die erst ab 10.00 h bereit steht, w ein Vorgang auf Maschine M2, die bereits ab 8.00 h frei ist. Dauern v und w jeweils eine Stunde und sind unabhängig voneinander, dann ist eine Planung v: 10.00 h bis 11.00 h auf M1 und w: 8.00 h bis 9.00 h auf M2 zulässig. Die Teillösung kann dabei lauten: (v) -> (v, w), d.h. die Teillösung (v) wird mit w erweitert.

Die praktische Konsequenz dieser Überlegung ist, daß durch kausale Beziehungen der Suchraum beschränkt wird, weil *next-activity* in Gegenrichtung zu kausalen Beziehungen nicht eingefügt werden kann.

Für nicht-klassische Anwendungen bei praktischen Problemen vom Typ A gilt zusätzlich, daß die Planung jedes Knotens an zusätzliche Bedingungen geknüpft ist. Diese werden überprüft, indem die Vor- und Nachbedingungen jedes Knotens abgefragt werden. Ist das der Fall, dann markiert man den Knoten mit wahr. Damit können wir eine zulässige Lösung erklären:

Satz 6.4: Ein Reihenfolgeproblem, in dem kein Element mehrfach in eine Reihenfolge aufgenommen werden darf, sei als Zustandsraumcluster D = [V, E, R] mit Startknoten s dargestellt. TL sei eine zulässige Teillösung eines Zustandsraums ZR, der mit einer Zustandsraumfunktion f auf D abgebildet wird. Die Reihenfolge der Elementarprobleme von TL kann dann auf einen Weg abgebildet werden, der eine Untermenge der hierarchischen Nachfolger von s über die Relation *next-activity* verbindet, wenn jedem von ihnen der Wert *wahr* zugeordnet ist.[6]

Beweis: Reihenfolgeprobleme werden durch eine kombinatorische Anordnung ihrer Elemente beschrieben. Jedes Element der Lösung wurde auf eine Aktivität unterhalb von s abgebildet.

Fall 1: das klassische Problem: die einzige Relation, die Cluster verbindet, ist die *next-activity* Beziehung. Sie ist immer dann zwischen zwei Knoten möglich, wenn es keinen Weg in Gegenrichtung gibt und Zyklen entstehen. Jede Lösung muß diese Bedingung erfüllen.

Fall 2: das Reihenfolgeproblem ist nicht klassisch. *Next-Activity* kann nicht eingefügt werden, wenn eine *cause-enable* oder *has-revision* Beziehung in Gegenrichtung verläuft. Eine Lösung würde mit kausalen Restriktionen in Konflikt kommen.

[6] Der umgekehrte Weg gilt nicht. An die Zulässigkeit können zusätzliche Anforderung, z.B. das Erreichen eines bestimmten Zielfunktionswertes, gestellt werden.

Satz 6.5: Bei klassischen Reihenfolgeproblemen vom Typ A kann der Startknoten nur verifiziert werden, wenn die Lösung zulässig und vollständig ist.

Beweis: Reihenfolgeprobleme vom Typ A werden mit einem UND-Knoten als Startknoten dargestellt. Er kann nur dann verifiziert werden, wenn jeder seiner hierarchischen Nachfolger wahr ist. Bei der Verifizierung wurden die hierarchischen Nachfolger zusätzlich paarweise über *next-activity* Relationen zu einer Sequenz verknüpft.

Die bisherigen Bedingungen zur Zulässigkeit sind notwendig, aber nicht hinreichend. Um die Gültigkeit einer Sequenz einschränken zu können, muß ein Ziel erreicht sein. Es wird als Zustandscluster von Vorbedingungen formuliert:

Definition 6.12: Ein Ziel ist ein Zustandscluster D = [V, E, R] mit einem Startknoten $s \in VORB$.

Um ein Ziel einem Zustandsraumclusters mit Startknoten s zuzuordnen, wird es mit s des Zustandsraumclusters über die sonstige Beziehung *has-goal* zugeordnet. Im folgenden wird unterstellt, daß jeder Zielzustand einer Sequenz damit darstellbar ist.

6.1.2.3. Lösungen im Zustandsraum

Nun können wir den Begriff "Lösung" definieren:

Definition 6.13: In einem Zustandsraumcluster D heißt eine gewählte Sequenz seq *Lösung*, wenn nach der Verifizierung aller Knoten der Sequenz das Ziel von D verifizierbar ist. Fehlt ein Ziel, dann ist jede gewählte Sequenz eine Lösung.

Definition 6.14: In einem Zustandsraumcluster D heißt eine gewählte Sequenz *seq* mit Anfangsknoten v_1 und Endknoten v_2 *Teillösung*, wenn es eine Lösung gibt mit Anfangsknoten v_1 und Endknoten v_3 und gilt:

1. Es gibt einen einen Weg von v_2 nach v_3 ausschließlich über *next-activity* Beziehungen oder es gilt: $v_2 = v_3$.

2. Jeder Knoten in *seq* wurde verifiziert.

Daraus folgt sofort, daß jede Lösung gleichzeitig Teillösung ist. Der folgende Satz zeigt, daß eine Teillösung eines Zustandsraumclusters und die Knoten eines Problems in der Zustandsraumdarstellung äquivalent sind.

Satz 6.6: Wurde ein Zustandsraum ZR eines Reihenfolgeproblems, in dem kein Element mehrfach vorkommt, mit einer Zustandsraumabbildung f auf ein Zustandsraumcluster D abgebildet, dann ist jede Teillösung des Zustandsraumclusters äquivalent zu einem Knoten in ZR und umgekehrt.

Beweis: Der umgekehrte Weg wurde bereits in Satz 6.4 gezeigt. Die Äquivalenz der Darstellungen in beide Richtungen ergibt sich aus der Definition der Abbildung. Bleibt zu zeigen, daß jede Teillösung in D mit Zielzustand einem Knoten in ZR entspricht.

Annahme: Es gibt eine Teillösung in D, die kein Äquivalent in ZR hat. Weil jede Teillösung laut Definition zu einer Lösung in D vervollständigt werden kann, folgt daraus: Es gibt eine Sequenz *seq* in D, die das Ziel erfüllt und jeden Knoten der Sequenz verifiziert hat, die nicht auf einen Knoten in ZR abgebildet wird. Eine zulässige und vollständige

Anordnung der Elementarprobleme ist also nicht in ZR dargestellt. Das ist aber ein Widerspruch zur Grundannahme über den ZR, der alle zulässigen Lösungen des Reihenfolgeproblems darstellen soll.

Anmerkung: Daraus folgt sofort, daß die Menge aller Teillösungen in D zur Menge aller Knoten in ZR äquivalent ist.

Definition 6.15: B = [W, E] mit Knotenmenge W und Kantenmenge F sei ein Baum für ein Zustandsraumcluster D = [V, E, R]. Jeder Knoten aus W entspreche einer Teillösung TL von D. Die Knoten aus W werden so angeordnet, daß für zwei Knoten v, w ∈ W gilt: Wenn w die Teillösung von v um genau einen Knoten erweitert, dann ist w ein Sohn von v. B heißt dann Lösungsbaum von D.

Damit läßt sich nun die Entsprechung beider Darstellungsformen leicht zeigen:

Satz 6.7: Wird ein Zustandsraum ZR = [V, E] mit einer Zustandsraumabbildung f auf ein Zustandsraumcluster D mit dem Lösungsbaum B = [W, F] abgebildet, dann sind die Bäume B und ZR gleich.

Beweis: Die Knotenmengen beider Bäume sind äquivalent aufgrund Satz 6.6. Die Anordnung der Teillösungen aus D ist in B genauso angeordnet wie die Teilprobleme in ZR, d.h. die Anordnung der Knoten über die Kanten aus E bzw. F sind gleich. Damit sind die Bäume gleich.

Jeder Zustandsraum entspricht einem Lösungsbaum eines Zustandsraumclusters. Daraus folgt, daß sich Lösungen in beiden Formen der Darstellung mit dem gleichen Aufwand ermitteln lassen. Die hier vorgelegte Darstellung ist also in der Lage, Reihenfolgeprobleme, die als Zustandsräume dargestellt werden sollen, genauso angemessen darzustellen und zu lösen.

6.1.2.4. Probleme für die Zustandsraumdarstellung

Ein Travelling Salesman Problem besitzt meist mehr Verknüpfungen als Knoten. Weil man den Weg durch den Digraphen optimieren will, müssen die Beziehungen zwischen den Knoten betrachtet werden, nicht die Knoten selbst. Unter diesem Blickwinkel trifft die Klassifizierung des Travelling Salesman Problem mit A (vgl. [Müller-Merbach 70]) nicht zu. Wie läßt sich das Travelling Salesman Problem darstellen? Nach Morlock [Morlock 88, S. 72] benötigt allein die Problemdarstellung mit Hilfe der linearen Programmierung eine kombinatorisch explodierende Anzahl von Ungleichungen zur Verhinderung von Kurzzyklen. Bei neun Knoten sind fast eintausend Nebenbedingungen aufzustellen! Die Darstellung mit Hilfe eines Zustandsraumcluster ist wesentlich kürzer:

M = Teillösung (eine Sequenz von Knoten, die Wege darstellen und mit *next-activity* verknüpft sind)

Zielfunktion Zf:

$\sum_{m \in M} x_m \rightarrow Min$ mit x_m = Nützlichkeitswert jedes Knotens aus M

Zielzustand:

Suche eine Lösungssequenz mit

Menge der besuchten Städte S = {A, B, C, D, E, F, G, H, I}

Position p = A, o.B.d.A. beginne und ende die Rundreise bei Stadt A

Objekte der Nebenbedingung:

Menge der besuchten Städte S; Initialwert = { }

Position p; Initialwert = A

Ein Weg von U nach V mit U, V ∈S ist eine Aktivität mit Nützlichkeitswert = x_{UV} und den Vorbedingungen

1. $zp_1 \in ZP$ mit S *required-by* zp_1 und V∉S (Wegende V darf noch nicht in der Menge der besuchten Städte enthalten sein)

2. $zp_2 \in ZP$ p *required-by* zp_2 mit p = U (die letzte Revision der Position muß Weganfang U sein)

sowie den Nachbedingungen

3. $zp_3 \in ZP$ p *required-by* zp_3 mit p:= V (Revidiere die Position durch das Ende des Weges)

4. $zp_4 \in ZP$ S *required-by* zp_4 mit S := S∪V (nimm V in die Menge der besuchten Städte S auf)

Um einen Weg von U nach V einzuplanen, darf V noch nicht besucht sein und der Handlungsreisende muß bei U angekommen sein. Das Zustandsprädikat zp_1 untersucht, ob das Wegende V bereits in der Menge S der besuchten Städte enthalten ist. Es fragt den Wert von S ab. Falls V noch nicht in der Menge S ist, wird zp_1 verifiziert. Das Zustandsprädikat zp_2 untersucht, ob der Handlungsreisende gerade bei U angekommen ist. Die letzte Revision seiner Position p muß U lauten. Ist das der Fall, wird zp_2 verifiziert.

Nun hat der Handlungsreisende den Weg von U nach V zurückgelegt. In der Nachbedingung zp_3 sichert das Bahnelement zu, daß der Wert der aktuellen Position auf V gesetzt wird. In der Nachbedingung zp_4 wird V der Menge der besuchten Städte S zugefügt.

Je Weg ergeben sich vier Bedingungen, bei neun Wegen sind es sechsunddreißig. Die Anzahl der Nebenbedingungen explodiert nicht kombinatorisch. Eine zulässige Lösung ist gefunden, wenn s alle Städte von A bis I enthält und die Position p gleich A ist. Kurzzyklen können nicht auftreten, weil stets nur nach einer Lösungssequenz gesucht wird.

Wie in der Darstellungsform mittels linearer Programmierung wird auch hier kein Lösungsverfahren impliziert. Im Vergleich ist jedoch die Darstellungsform hier wesentlich übersichtlicher und Lösungsverfahren zugänglicher als die der linearen Programmierung.

Wie läßt sich das Bahnelementeproblem damit darstellen? Elemente des Problems sind die Luftbahnen, Werkzeugwechsel und Bearbeitungsbahnen. In einem Zustandsraumdigraph $D_{Bahnelemente}$ = [V, E, R] mit Startknoten $s \in ODER \cap A$ wird jede der Bahnen auf einen Knoten v abgebildet mit s *has-sub-activity* v. Außerdem gibt es drei Objekte der Nebenbedingung: Eines, "Position" beschreibt die Position der Roboterhand, ein anderes "Werkzeug" den Namen des aktuellen Werkzeugs und ein drittes heißt "abgefahrene Bahnen". Um sicherzustellen, daß alle Bearbeitungsbahnen abgefahren werden, gibt es ein Ziel mit *required-by* "abgefahrene Bahnen" und einem Prädikat, das überprüft, ob n Bahnen abgefahren wurden, mit n = Anzahl der Bearbeitungsbahnen.

Jeder Bearbeitungsbahn werden zwei Zustandsprädikate als Vorbedingungen zugeordnet: Vorbedingung-1 *required-by* einem Werkzeug, Vorbedingung-2 *required-by* einer Position. Ein Zustandsprädikat in der Nachbedingung revidiert die aktuelle Position, d.h. die Position der Roboterhand wird auf die Endposition einer Werkzeugbewegung gesetzt. Ein anderes Zustandsprädikat erhöht inkrementell die Anzahl abgefahrener Bahnen. Luftbahnen haben nur eine Vor- und eine Nachbedingung, die sich auf die aktuelle

Position beziehen, Werkzeugwechsel sichern in der Nachbedingung ein neues Werkzeug zu. Jede der Bahnen erhält als Eintrag im Attribut *utility* die zum Abfahren erforderliche Zeitdauer.

6.1.3. Problemreduktionsgraphen

Zuerst wird der klassische Problemreduktionsgraph modelliert, anschließend das Maschinenbelegungsproblem. Ein Problemreduktionsgraph oder UND/ODER-Graph ist per definitionem eine Spezialform des A/S-Clusters:

Definition 6.16: Gegeben sei ein A/S-Cluster D = [V, E, R] mit Startknoten s. Es heißt Problemreduktionsgraph oder UND/ODER-Baum, wenn gilt

1. Jeder Knoten hat genau einen Vorgänger bezüglich der Cluster-Relationen.

2. Für zwei Knoten v, w∈V mit v rel w, rel ∈ {hierarchische Relationen}: $(v \in UND \wedge w \in ODER) \vee (v \in ODER \wedge w \in UND)$.

Die zweite Bedingung sichert auf dem Weg von der Wurzel zum Blatt zu, daß sich UND-Knoten ständig mit ODER-Knoten abwechseln. Problemreduktionsgraphen sind gerichtete Bäume[7]. Das vereinfacht die Suche, weil die Suche in zyklenfreien Digraphen und die Suche in Bäumen beide gut beherrscht werden.

Satz 6.8: Gegeben sei ein Teildigraph D = [V, E, R], bei dem für jedes $r \in R$ gilt: r ist eine Clusterrelation. Ein Problemreduktionsgraph in D ist ein gerichteter Baum mit dem Startknoten als Wurzel.

Beweis: Zu zeigen ist, daß der zugeordnete Graph G eines Problemreduktionsgraphen D=[V, E, R] ein Baum ist und, daß der Startknoten s Basis des Baums ist.[8]

G ist ein Baum, wenn G ein zusammenhängender Graph ist, der keine Zyklen enthält: Von s aus ist jeder Knoten gemäß der Definition des A/S-Clusters erreichbar. Daraus folgt, daß alle Knoten von G miteinander verbunden sind. Sei v∈V\s, dann gilt: v hat laut Definition genau einen Vorgänger bezüglich Clusterrelationen. Je zwei Knoten aus v∈V\s sind miteinander verbunden; G ist also zusammenhängend. Für A/S-Cluster wurde außerdem gezeigt, daß sie zyklenfrei sind (vgl. Satz 6.1). Daraus folgt, daß G ein Baum ist.

Nun wird die Basiseigenschaft von s gezeigt. Dazu ist zu zeigen, daß 1. von s aus alle Knoten erreichbar sind und daß 2. dies für keine Untermenge von s gilt. Punkt 2 ist einfach zu zeigen, weil s elementar ist, d.h., s hat keine Untermenge. Punkt 1 ist laut Definition des Startknotens eines A/S-Clusters so: Die Clustermenge der Knoten ist genau die von s aus erreichbare Menge.

Bei einem klassischen Problemreduktionsgraphen wird die Knotenmenge auf Aktivitäten begrenzt. Er ist damit äquivalent zu Pearls Beschreibung eines Problemreduktionsgraphen (vgl. Kapitel 4).

Als Beispiel für einen Problemreduktionsgraphen soll das vereinfachte Maschinenbelegungsproblem dienen: Das Problem ist ein ODER-Knoten mit zwei Ausarbeitungen: manuelles und automatisches Gußputzen. Manuelles Gußputzen ist eine Aktivität ohne

[7] "Ein Digraph D = [V, $] heißt gerichteter Baum mit der Wurzel r∈V, wenn der D zugeordnete Graph G_D ein Baum und {r} Basis von D ist." [Neumann 75, S. 40]

[8] für die Definitionen vgl. [Neumann 75, Kapitel 1]

Söhne. Automatisches Gußputzen ist ein UND-Knoten bestehend aus den Aktivitäten Trennen, Messen, Putzen, die nicht weiter verfeinert werden. Der Problemreduktionsgraph ist in Abbildung 6.1 dargestellt:

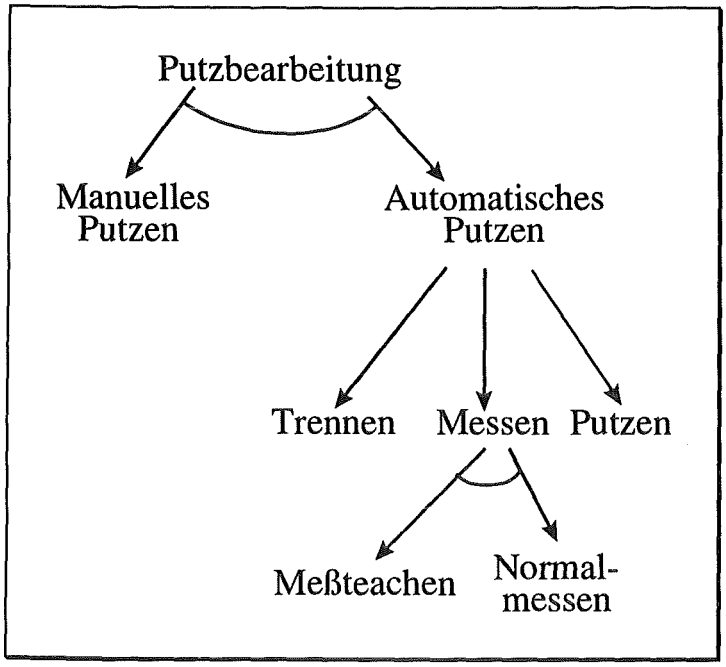


Abb. 6.1: Darstellung der Aktivitäten des Maschinenbelegungsproblems und ihre hierarchischen Beziehungen in Form eines Problemreduktionsgraphen. ODER-Knoten wurden durch eine Verbindung der Pfeile nach unten angedeutet.

Hinzu kommen entsprechende Vor- und Nachbedingungen, Objekte der Nebenbedingung usw., die analog dem Bahnelementeproblem modelliert werden. Eine Besonderheit ist hier, daß die Vorgänge Trennen, Messen, Putzen stets in dieser Reihenfolge stehen. Zwischen einer Nachbedingung von Trennen kann eine *cause-enable* Beziehung zur Vorbedingung von Messen etabliert werden. In Gegenrichtung kann von vornherein keine *next-activity* Beziehung eingefügt werden, was den Problemraum verkleinert.

Das Ergebnis ist eine klare Darstellung für Reihenfolgeprobleme, die die beiden wichtigsten Darstellungsformen durch graphentheoretische Modellierung integriert. Weil sie zyklenfrei ist, sind Standardsuch- und -sortierverfahren auf sie anwendbar. Im Gegensatz zu anderen Darstellungsformen stellt sie eine Fülle von Objektarten und Relationen bereit, um Probleme direkt zu modellieren.

6.2. Objekte des Netzwerks

Die Reihenfolgeprobleme der Anwendung sind einander in den Objektstrukturen sehr ähnlich. Ressourcen können knapp sein: Roboter, Paletten, Bedienpersonal und Werkzeuge bei der Maschinenbelegung, Werkzeuge bei der Bahnelementeoptimierung und die Fahrzeugkapazität bei der Wegeoptimierung des autonomen Fahrzeugs. Aufträge besitzen Nebenbedingungen physikalischer, technischer oder organisatorischer Natur, die an ihre Ausführung geknüpft sind. Es war die Beobachtung dieser Gemeinsamkeiten, die den Verfasser dazu motivierte, für Reihenfolgeprobleme einen einheitlichen Ansatz zu suchen. Folgende Ordnung der Objekte kann getroffen werden:

- Physikalische Objekte: Ressourcen, Zeitpunkte, Zeitintervalle, Bahnpunkte usw.
- Organisatorische Objekte: Aufträge, Operationen und Fristen
- Wissensobjekte: Methoden und Zielfunktionen

Physikalische Objekte können in Ressourcen und abstrakte, meßbare Objekte wie Zeit oder Raum eingeteilt werden. Abstrakte physikalische Objekte sind Begriffe wie Zeit, Punkt, Linie, Bahn. Ressourcen sind das Bedienpersonal, Maschinen, Paletten, Lager, Spannvorrichtungen, Roh- Hilfs- und Betriebsstoffe, Werkstücke und Werkzeuge. Ressourcen werden belegt oder durchlaufen Zustände. Ein Werkstück wechselt z.B. aus dem Zustand *unbearbeitet* auf *bearbeitet.*

Aufträge und Operationen sind organisatorische Objekte. Ein Auftrag besteht aus Operationen. Einem Auftrag oder einer Operation kann höchstens ein Zeitfenster zugeordnet werden, weil keinem organisatorischen Objekt mehrere Start- und Endzeitpunkte sinnvoll zugeordnet werden können.

Die Bausteine für Heuristiken werden explizit als Wissensobjekte dargestellt. Sie lassen sich gliedern in Such- und Sortierverfahren und Methoden zur Bewertung einschließlich der Zielfunktionen. Wissen kann hierarchisch strukturiert werden, indem es einem Knoten im A/S-Digraphen als Methode[9] zugeordnet wird.

Die Gliederung bildet eine leicht erweiterbare taxonomische (begriffliche) Hierarchie. Objekte besitzen Attribute, die Werte enthalten und vererbt werden sollen. Deswegen wird zur Objektdarstellung ein framebasierter Ansatz herangezogen.

Hinsichtlich der Bedeutung[10] für die Problemlösung kann man Objekte gliedern in

- den Gegenstand der Problemlösung (stets als Aktivität dargestellt),
- Zustände und
- Objekte von Nebenbedingungen.

Dazu ein Beispiel: Die Operation *Messen* soll geplant werden. *Messen* ist an die Vorbedingung *Meßroboter-vorhanden* gebunden. Für den Planungsprozeß ist die Operation eine Aktivität, *Meßroboter-vorhanden* ein Zustand, der sich an einem Objekt *Meßroboter* orientiert. Seine Existenz wird zur Nebenbedingung. Der *Meßroboter* kann aber auch selbst zum Gegenstand der Problemlösung werden, z.B. wenn ein Meßroboter unter mehreren auszuwählen ist. Ein Objekt kann also im Problemlösungsprozeß verschiedene Funktionen einnehmen.

Über die formale Beschreibung von A/S-Digraphen hinaus erhalten Aktivitäten und Zustände Attribute, die sie z.T. von einem "allgemeinen Objekt" erben. Einen Überblick gibt die folgende Tabelle:

allgemeines Objekt		
Attribut	Wertebereich	Beschreibung
is-a+INV	Einzelaktivität, zusammengesetzte Aktivität	
Typ	Prototyp, Instanz	Default ist Prototyp
Wert	wahr, falsch, unbekannt	Zur Dokumentation des Planungsfortschritts
utility	eine reelle Zahl	Ausdruck der Präferenz für dieses Objekt
Hat-Zeitintervall	Ein oder kein Zeitintervall	

Tab. 6.4: Attribute zur Speicherung von Werten für die Problemlösung

Die Zuordnung von Relationen zu Objekten wurde bereits im formalen Teil beschrieben.

[9]Methoden werden in KnowledgeCraft einem Schema wie Attribute zugeordnet und können auch vererbt werden.

[10]Man spricht auch von der Rolle eines Objekts im Problemlösungsprozeß.

Welche Konsequenz hat die Trennung von Rolle und taxonomischer Hierarchie? Neue Objekte werden spezifiziert, indem sie taxonomisch und hinsichtlich ihrer Rolle festgelegt werden. Ohne die Trennung gäbe es eine einzige große Hierarchie, die alle Kombinationen von Rolle und begrifflicher Zuordnung explizit enthielte.

6.3. Intervallrepräsentation

Reihenfolgeprobleme können als Probleme mit zeitbehafteten Vorgängen dargestellt werden. Im allgemeinen Fall eines Vorranggraphen[11] sind Zeitbeziehungen nicht notwendig mit festen Zeiten verknüpft, aber für die Maschinenbelegung und die Projektplanung müssen qualitative und quantitative Zeitinformationen dargestellt werden. Dazu zählen die Verfügbarkeit physikalischer Objekte über die Zeit, z.B. Arbeitszeiten, Wartungszeiten, die bereits vorgegebenen Reihenfolgebeziehungen zwischen Aktivitäten und die Veränderlichkeit der Umwelt über die Zeit hinweg.

6.3.1. Zeitfenster

Ein Zeitfenster stellt ein Intervall dar. Die Lage des Intervalls auf dem Zeitstrahl kann durch Bedingungen an Dauer, Start- oder Endzeitpunkt eingeschränkt werden. Dauern und Zeitpunkten können feste Werte zugeordnet werden, z.B. *Vorgang Einlernen dauert 4 Stunden.* Alternativ können untere oder obere Grenzen vorgegeben werden wie *Das Einlernen muß zwischen 9 h und 11 h morgens beginnen.* Hier ist der früheste Anfangszeitpunkt 9 h, der späteste 11 h.

Qualitative Beziehungen zwischen Zeitfenstern werden mit Zeitrelationen ausgedrückt. Zwischen zwei Knoten ist grundsätzlich jede Zeitbeziehung erlaubt. Einträge müssen lediglich konsistent sein. Konsistenzprobleme ergeben sich zwischen Start, Dauer und Ende und zwischen frühesten und spätesten Zeitpunkten.

6.3.2. Schnittstellen zu A/S-Netzen

Die Zuordnung von Zeitfenstern zu A/S-Objekten ist mehrdeutig: Ein Zeitfenster kann mehreren A/S-Objekten zugeordnet werden, wenn diese gleichzeitig gelten. Anstatt zwei zeitgleiche A/S-Objekte auf zwei Zeitfenster zu beziehen, die dann mit *time-equal* verbunden werden, bildet man die zeitgleichen A/S-Objekte auf nur ein Zeitfenster und damit nur eine Zeit ab. Die Größe des Netzwerks der Zeitfenster kann bei A/S-Clustern dadurch erheblich eingeschränkt werden.

Die kausalen Beziehungen *enable*, *cause* und *cause-enable* (s.o.) implizieren Zeitgleichheit. *Enable* verbindet eine Aktivität mit der Wurzel des Baums ihrer Vorbedingungen, *cause* mit der Wurzel des Baums ihrer Nachbedingungen. Weil alle Bedingungen letztlich auf die Aktivität bezogen werden müssen, muß die Aktivität zeitgleich mit den Wurzeln der Zustandscluster sein. Bei jeder anderen Zeitbeziehung wäre eine eindeutige Beziehung zwischen den Blättern der Zustandscluster und der Aktivität nicht mehr möglich. Die *time-equal* Beziehung ist als einzige Beziehung neutral bei der Hintereinanderabbildung von Zeitbeziehungen, weil sie als einzige Relation transitiv, reflexiv und symmetrisch ist. Die *cause-enable* Beziehung schließlich verbindet zwei Zustandsobjekte, die in unterschiedlichen A/S-Clustern den gleichen Zustand repräsentieren. Offensichtlich ist damit ebenfalls eine *time-equal* Beziehung impliziert.

Elaboration Beziehungen verbinden eine Ganzheit mit ihren Verfeinerungen. Verfeinerungen betrachten den gleichen Gegenstand, man hat sich nur auf eine Strukturierung festgelegt. Deswegen besteht zu Ausarbeitungen Zeitgleichheit. Wird eine Aktivität dagegen dekomponiert, soll die Oberaktivität sich über einen Zeitraum erstrecken, der vom Startzeitpunkt der ersten Unteraktivität bis zum Endzeitpunkt der letzten Unteraktivität

11 Vorranggraph: Ein Graph, zwischen dessen Knoten nur Vorrangbeziehungen bestehen: vorher, nachher.

reicht. Unteraktivitäten finden während der Oberaktivität statt. Das impliziert drei mögliche Zeitbeziehungen: *starts-with, during, ends-with.* Ein A/S-Cluster läßt sich für die Propagation von Zeiten mit diesen Implikationen erheblich reduzieren.

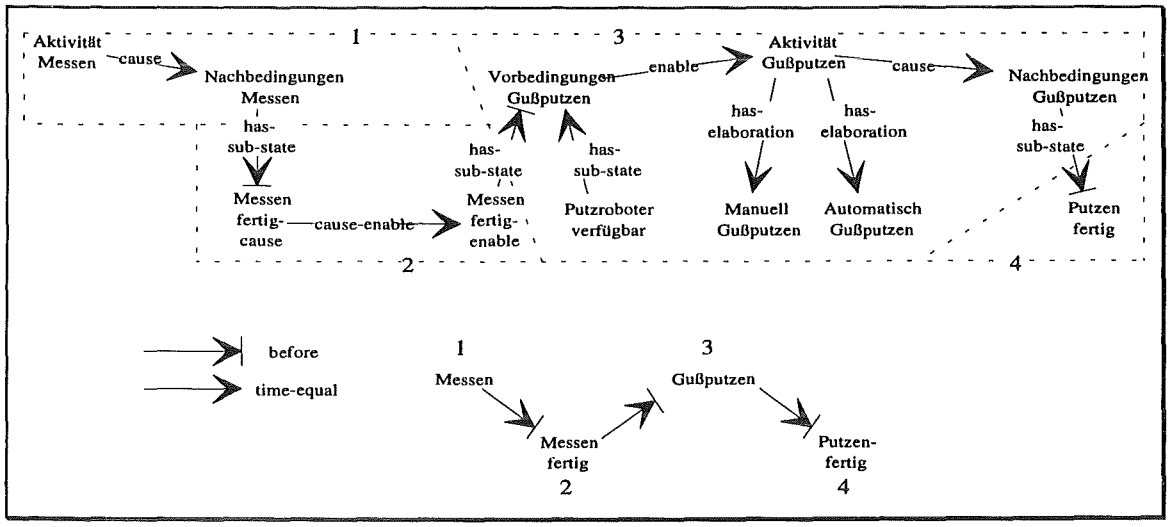


Abb. 6.2: Ein A/S-Cluster (oben) und sein zeitliches Gegenstück (unten). Zeitgleiche Beziehungen (time-equal) sind durch einen einfachen Pfeil, vorher Beziehungen (before) durch einen Pfeil mit Abschluß gekennzeichnet. Daraus ergab sich eine Einteilung in vier verschiedene Bereiche, die mit gestrichelten Linien umrahmt sind; die Objekte jedes Bereichs sind zeitgleich und bilden auf das selbe Zeitfenster ab. Die Beziehungen zwischen den Zeitfenstern sind darunter dargestellt. Ein A/S-Digraph von 11 Knoten wurde auf nur 4 Zeitfenster abgebildet.

Neben A/S-Objekten besitzen physikalische Objekte Zeitbeziehungen. Jede Ressource hat ein Attribut *belegt-von.* Es enthält eine Liste der Besitzprädikate, deren Aktivitäten die Ressource belegen. Im Normalfall kann eine Ressource nicht mehrere Operationen gleichzeitig ausführen. Deswegen bestehen zwischen den Zeitfenstern einer Ressource die Beziehungen *before, after, during* oder *contains.*

6.3.3. Diskussion der Darstellung

Der A/S-Digraph vereinigt die beiden wesentlichen Darstellungsformen von Reihenfolgeproblemen in einem Ansatz. Probleme aus der Praxis sind oft Mischprobleme. Gerade hier ist das Einsatzfeld von A/S-Digraphen zu sehen. Die vorgelegte Darstellungsform (Abschnitte 6.1 - 6.3) ist

- vollständig, weil alle relevanten Aspekte der Problemdomäne dargestellt wurden. Sie erlaubt die explizite Darstellung von zeitlichem, hierarchischem und kausalem Wissen. Die beiden für die Reihenfolgeplanung wesentlichen Darstellungsformen werden äquivalent repräsentiert.
- präzise, weil Wissen mit angemessener Granularität beschrieben wird. Wissen kann hierarchisch abstrahiert werden. Durch unterschiedliche Beziehungsarten zwischen Knoten müssen Relationen nicht auf ein einziges Relationenkonzept zurückgeführt werden.
- klar, weil Wissen konsistent abgebildet wird, d.h. zusätzliches Wissen kann leicht integriert werden. Stehen die Operationen Trennen - Messen - Putzen z.B. in wahl-

freier Reihenfolge, kann Planungswissen aus der Zustandsraumdarstellung leicht übertragen werden, indem entsprechende Suchverfahren appliziert werden. Der Digraph unterscheidet bei Objekten zwischen der Klassifizierung und der Rolle für den Problemlösungsprozeß. Das macht die Struktur für Erweiterungen offen.

Ein weiterer Vorteil ist, daß sie auch als Basis für eine Problemlösung geeignet ist, also zur Verarbeitung von Planungswissen. Die Knoten des A/S-Digraphen spannen den Lösungsraum auf, wohingegen die Nebenbedingungen in der konventionellen Darstellung mit Zielfunktion den Lösungsraum nur einschränken. Eine Lösung ist nicht eine Kombination der Zielvariablen, welche die Nebenbedingungen erfüllt, sondern eine Anordnung von verifizierten Knoten, die einer Zielbedingung genügt. Es ist also nur notwendig, eine entsprechende Anordnung zu suchen; die Lösung ergibt sich aus der Entscheidung für ein Suchverfahren.

Am Beispiel des Travelling Salesman Problem konnte gezeigt werden, daß die Darstellung mit weniger Nebenbedingungen auskommt als herkömmliche Darstellungen.

6.4. Schlußfolgern in einem A/S-Netzwerk

Für die Planung mit A/S-Digraphen ist ein Verfahren gesucht, das auf den vier gefundenen Bausteinen von Heuristiken aufbaut.

6.4.1. Propagationsschema

Das Propagationsschema dient als Schablone für speziellere Verfahren. Es ähnelt dem Zweiphasenmodell zur Vorwärtsverkettung von Regeln (vgl. [Puppe 89, S. 22f]) oder der Steuerstrategie eines Prologinterpreters, der ebenfalls die Suche nach anwendbaren Regeln steuern muß. Der wesentliche Unterschied ist, daß die Verfahren zur Suche, Sortierung, Verifikation und Bewertung anwendungsabhängig zusammengestellt werden sollen. Die Schritte sind:

1. Vorsortieren: Sortiere die Objekte in eine vorläufige, nicht über *next-activity* verbundene Sequenz.
2. Suchen: Wähle nächstes Objekt für die anfangs leere Sequenz und plane es mit *next-activity* ein.
3. Einplanen: Berechne die Konsequenzen des Suchschritts für den A/S-Digraphen.
4. Bewerten: Bewerte den Suchschritt anhand der Konsequenzen.

Zu 1.: Sind die Elemente einer Sequenz z.B. für das Travelling Salesman Problem bereits in der richtigen Reihenfolge sortiert, erübrigt sich ein komplexes Suchverfahren, die Planung wird einfacher. Die Sortierkriterien sind anwendungsabhängig. Infrage kommen einschlägige Verfahren wie Kürzeste Operationszeit, Best-first usw.

Zu 2.: Die Auswahl des nächsten Planungschritts (Suche) entspricht einer Bewegung durch den Zustandsraum. Gesucht wird nach einem Element das die Teillösung erweitert. Mögliche Kriterien für die Auswahl sind

- die Geschichte der Planung, z.B. welche Suchschritte sind bereits erfolgt; Zielfunktionswert der bisher besten (Teil-)sequenz.
- ob ein gestecktes Ziel erreicht wurde (Beziehung has-goal des Vaterknotens)
- Eigenschaften des Knotens, die in Form von Attributwerten abprüfbar sind. Z.B. sein Nützlichkeitswert oder ob der Knoten eine Luftbahn ist.

Für A/S-Cluster läßt sich der Begriff der Suche spezifizieren:

Definition 6.17: Ein Programm S, das in einem A/S-Cluster D = [V, E, R] für eine Teillösung t_k, die keine Lösung ist, einen Knoten v festlegt, mit dem t_k zu einer neuen Teillösung t_{k+1} erweiterbar ist, heißt Suchprogramm.

Die Wahl des Suchverfahrens hängt vom Knoten- und Problemtyp ab. So ist bei UND-Knoten sicherzustellen, daß alle Söhne verifiziert werden. Im einfachsten Fall geschieht das in beliebiger Reihenfolge. Der Aufwand ist dann linear. Im schwierigsten Fall müssen alle sequentiellen Anordnungen der Söhne untersucht werden, ggf. sind Sequenzen miteinander zu vergleichen. Derartige Suchverfahren besitzen exponentiellen Aufwand. In KnowledgeCraft läßt sich der Sequenzvergleich recht elegant über Kontexte realisieren. Nachteil ist, daß ihre Verwaltung aufwendig ist. Ab einer Tiefe von etwa 15 Kontexten ist ein effizienter Gebrauch nicht mehr möglich.

Zu 4.: Stehen die (Teil-)sequenzen zueinander im Wettbewerb, so sind sie zu bewerten. Zuerst wird überprüft, ob ein evt. vorhandenes Ziel erreicht wurde. Dann wird die Sequenz einer Zielfunktion zugeführt, um die Zielbeiträge zu vergleichen.

6.4.2. Einplanen

Mit dem Schritt 3, Einplanen, wird das Restproblem nach Erweiterung der Teillösung bestimmt. Die Konsequenzen der Aufnahme eines Elements in die Sequenz werden simuliert. Interaktionen zwischen Objekten werfen Konsistenzprobleme auf: Durch Abfahren einer Bahn ändert der Roboterarm seine Position, die für die Einplanung der nächsten Bahn relevant ist. Für die Propagierung bedeutet das:

> Ein Objekt in einem Netzwerk kann genau dann in eine Lösungssequenz aufgenommen werden, wenn es verifiziert wurde. Aufgabe der Planung ist, die Vorbedingungen des Objekts zu überprüfen und die Konsequenzen der Aufnahme darzustellen.

Dazu ein Beispiel für eine Aktivität *Gußputzen*. *Gußputzen* besitze zwei Vorbedingungen: 1. Es steht ein Putzroboter zur Verfügung. 2. Es gibt ein bereits vermessenes Werkstück. Die Nachbedingung sichere für das Werkstück den Zustand *geputzt* zu. Gußputzen bestehe aus den Unteraktivitäten *Aufspannen, Putzen, Abspannen*.

Jede Aktivität ist Startknoten eines A/S-Clusters, das ggf. aus Vorbedingungen, Unteraktivitäten und Nachbedingungen besteht (vgl. S. 58). Im vorliegenden Ansatz propagieren alle Aktivitäten mit der gleichen Strategie:

1. Überprüfe die mit *enabled-by* zugewiesenen Vorbedingungen.
2. Rufe das Propagationsschema für die Unteraktivitäten. Das Suchverfahren variiert mit den verschiedenen Klassen von Aktivitäten und Anwendungsdomänen, z.B. UND- bzw. ODER-Aktivität.
3. Setze den Wert der Aktivität auf wahr und propagiere die Nachbedingungen, die über *cause* mit der Aktivität verbunden sind.

Der Wahrheitswert eines Zustands hängt ab

1. vom Wahrheitswert seiner Unterzustände,
2. vom Wahrheitswert des über *enable-cause* assoziierten Zustands und
3. von einer konsistenten Beziehung zu assoziierten Objekten der Nebenbedingung.

Im Fall 2 können alle weiteren Überprüfungen entfallen, weil über *cause-enable* verknüpfte Zustände identisch sind.

Besitzprädikate belegen eine Ressource über eine gewisse Dauer. Mit dem Attribut Zeitintervall kann ein *Besitzprädikat* feststellen, ob die zeitliche Belegung erfolgreich war. Die Belegung erfolgt mit der Zeitpropagation (s.u.). *Zustandsprädikate* revidieren physikalische Objekte: Nach einer Bearbeitung erhöht ein Zustandsprädikat z.B. die

Eingriffszeit eines Werkstücks um die Dauer der Bearbeitung. Dabei entstehen Revisionsketten.

Wurde für Zustands- oder Besitzprädikate mit der require-Beziehung ein Objekt der Nebenbedingung festgelegt, das noch Ausarbeitungen besitzt, ist eine Ausarbeitung auszuwählen, z.B. ein bestimmter Roboter unter mehreren verfügbaren. Dazu wird das A/S-Cluster des Objekts instanziiert und als Instanz der Propagation propagiert. Für die genauen Regeln zur Verifizierung vgl. S. 49f.

6.4.3. Komplexitätsreduktion durch Individuation

Die Komplexität einer Planung läßt sich oft durch individuelle Zuordnung von Such- und Sortierverfahren erheblich verringern. Der Umfang der Einsparung ergibt sich aus der Komplexität der Suchverfahren jeder einzelnen Hierarchiestufe. Der Aufwand wird durch Multiplikation der Suchschritte auf jeder Hierarchieebene eines Suchbaums ermittelt. An einem Beispiel soll das verdeutlicht werden:

In einem binären UND/ODER-Baum besitze jeder Knoten genau zwei Nachfolger, sofern er kein Blatt ist. Zusätzlich gelte, daß alle Blätter gleich weit von der Wurzel entfernt sind. Söhne von UND-Knoten müssen ohne Ausnahme überprüft werden. Zur Bestimmung der ODER-Knoten gebe es zwei Strategien. Strategie 1 wählt von zwei Knoten den besten, Strategie 2 wählt den nächstbesten Knoten, untersucht also nur einen von zwei Knoten. Außerdem sei die Wurzel des Baums vom Typ ODER. Mit der ersten Strategie sind in einem Baum mit einer Tiefe n 2^{n-1} Sequenzen zu bilden. Dabei werden $2^n - 1$ Knoten untersucht. Strategie 2 bildet nur $2^{\left\lceil \frac{n}{2} \right\rceil - 1}$ Sequenzen und untersucht dabei $2\left(2^{\left\lceil \frac{n}{2} \right\rceil} - 1\right) - 2^{\left\lceil \frac{n}{2} \right\rceil - 1}\left(\left\lceil \frac{n}{2} \right\rceil - \left\lfloor \frac{n}{2} \right\rfloor\right)$ Knoten. Die etwas komplexere Formel kommt zustande, weil nur bei ODER-Knoten die Suche vereinfacht wird. Der hintere Teil der Formel subtrahiert Knoten bei ungeradem n. Im vorderen Teil wurde die Komplexität der Suche um einen Faktor $2^{\frac{n}{2}}$ verringert, also eine beträchtliche Einsparung.

Ein Anwender entscheide sich, nur die letzten m Schritte der Suche mit der schnelleren Strategie 2 durchzuführen. Bis zur Tiefe (n-m-1) wird mit Strategie 1 gesucht. Dabei werden $2^{n-m} - 1$ Knoten durchsucht. Auf Stufe (n-m) werden 2^{n-m-1} Teilbäume mit einem Aufwand von je $2\left(2^{\left\lceil \frac{m}{2} \right\rceil} - 1\right) - 2^{\left\lceil \frac{m}{2} \right\rceil - 1}\left(\left\lceil \frac{m}{2} \right\rceil - \left\lfloor \frac{m}{2} \right\rfloor\right)$ durchsucht. Zusammen gibt das:

(1) $$2^{n-m-1} - 1 + 2^{n-m-1}\left(2\left(2^{\left\lceil \frac{n}{2} \right\rceil} - 1\right) - 2^{\left\lceil \frac{n}{2} \right\rceil - 1}\left(\left\lceil \frac{n}{2} \right\rceil - \left\lfloor \frac{n}{2} \right\rfloor\right)\right)$$ oder

(2) $$2^{n-m-1} - 1 + 2^{n-\left\lfloor \frac{m}{2} \right\rfloor} - 2^{n-m} - 2^{n-\frac{m}{2}-2}\left(\left\lceil \frac{m}{2} \right\rceil - \left\lfloor \frac{m}{2} \right\rfloor\right)$$

durchsuchte Knoten. In (2) besitzen der erste Term 2^{n-m-1} und der dritte Term 2^{n-m} eine vergleichbare Größenordnung, der vierte Term spielt nur bei ungeradem m eine Rolle. Wesentlich wird die Größenordnung der Suche vom zweiten Term $2^{n-\left\lfloor \frac{m}{2} \right\rfloor}$ bestimmt. Aus ihm geht hervor, daß die Größe von m auf die Komplexität der Suche exponentiellen Einfluß besitzt, wobei m die Tiefe ist, ab der Strategie 2 zum ersten Mal angewandt wird. Durch die Wahl von m kann also erheblich Laufzeit gespart werden.

6.4.4. Zusammenfassung der Propagation in A/S-Digraphen

Der Ansatz besteht aus einem rekursiven Propagationsschema, das individualisiert werden soll, um die Planung effizienter zu machen. Die Verfahren wurden gemäß der identifizierten Bausteine von Heuristiken aufgeteilt. Neue Heuristiken können durch Kombination bestehender Verfahren schneller erstellt werden als durch Neuprogrammierung. Die Verfahrensauswahl ist problemabhängig: Ein UND-Knoten sucht unter den Söhnen anders als ein ODER-Knoten. Der Ansatz besitzt eine Reihe von Vorteilen:

1. Er kann, besonders mit klassenspezifischen Ausarbeitungen, für die auftretenden Reihenfolgeprobleme verwendet werden, wie die Beispiele belegen.
2. Neues Wissen, neue Heuristiken können bei Bedarf leicht hinzugefügt werden.
3. Durch Spezialisierungen können Anforderungen an die Laufzeit oder die Planungsqualität erfüllt werden. Durch Spezialisierungen kann man problemspezifische Parameter zum Vorteil der Planung ausnutzen.

Die Komplexität hängt von den gewählten Such- und Sortierverfahren ab und ergibt sich aus einem Abwägen zwischen Güte und Laufzeit. Grundsätzlich kann jeder bekanntere Ansatz ohne großen Aufwand mit den Schritten des Schemas realisiert werden. Nachteil ist, daß man bei der Entwicklungsumgebung rasch an Grenzen stößt, weil im Falle umfangreicherer Suchen im Zustandsraum Kontexte in KnowledgeCraft tief geschachtelt werden.

6.5. Zeitliches Schlußfolgern[12]

Die Analyse in Kapitel 4 und 5 ergab, das herkömmliche temporale Ansätze nicht allen Anforderungen genügen. Der Verfasser hat auf den Ansätzen von Allen, Hrycej und Vere aufgebaut und einen eigenen nicht-linearen Zeitplaner entwickelt. Zu dessen Vorteilen zählt neben der Kompatibilität mit A/S-Netzen die Durchgängigkeit der Zeitpropagierung. Angefangen von unbestimmten Ereignissen bis hin zu festen Zeitpunkten kann man eine große Bandbreite zeitlicher Informationen verarbeiten.

6.5.1. Beschreibung des Modells

Das hier entwickelte Modell basiert auf Allens Zeitmodell in der Erweiterung von Hrycej. Grundelemente sind Intervalle und Relationen zwischen ihnen. Mit Relationen sind im Folgenden stets Zeitrelationen gemeint. Es existieren 13 Basisrelationen: *before, after, meets, met-by, starts, started-by, during, contains, ends, ended-by overlaps, overlapped-by* und *time-equal*. Die Relationen umfassen alle eindeutigen Beziehungen zwischen zwei Intervallen. Im weiteren wurden die Abkürzungen von Allen *(< > m mi s si d di f fi o oi =)* verwendet: <: *before*, vorher; >: *after*, nachher; m: *meets*, unmittelbar vorher; mi: *meets-inverse*, after, unmittelbar nachher; s: *starts*, beginnt; si: *starts-inverse*, started-by, begonnen von; d: *during*, während; di: *during-inverse*, contains, enthält; f: *finishes*, ends, endet; fi: *finishes-inverse*, finished-by, ended-by, beendet; o: *overlaps*, überlappt; oi: *overlaps-inverse*, overlapped-by, überlappt von; =: *time-equal*, zeitgleich.

6.5.2. Löschen von Redundanzen im Ereignisnetz

Intervalle und ihre Relationen bilden einen gerichteten und zusammenhängenden Digraphen $D = [V, E, c]$, dessen Knoten V Intervalle repräsentieren, die Zeitfenster von Ob-

[12]Die hier verwendete Notation von Zeitgraphen lehnt sich an die von [Neumann 75] an. Im Kapitel 1 seines Buches werden auch graphentheoretische Definitionen vorgestellt, die hier nicht wiederholt werden.

jekten im A/S-Netz sind. Er wird so initialisiert, daß alle Relationen zwischen den Intervallen möglich sind: Alle 13 Beziehungen zwischen zwei Knoten werden auf *wahr* gesetzt. Anfangs setzt sich also die Relation zwischen zwei Knoten aus allen dreizehn Basisrelationen zusammen. Damit wird ausgedrückt, daß die Beziehung zwischen ihnen unbekannt ist. Einschränkungen der Relationen können ins Netz eingefügt werden. Die Auswirkungen auf die übrigen Zeitfenster und -relationen werden mit Allens Propagierungs-Algorithmus[13] berechnet. Dieser Algorithmus erzeugt stets konsistente Netze, ein Konsistenztest erübrigt sich. Da die Darstellung noch sehr komplex ist (jedes Intervall ist mit jedem anderen Intervall verbunden), wird eine Reduktion gesucht, bei der einerseits möglichst wenig an Information verloren geht, andererseits aber eine Strukturierung sichtbar wird. Hrycej nutzt die Transitivität der Vorher/nachher-Relationen und der Einschlußrelationen (s, si, d, di, f, fi). Beziehungen zwischen zwei Intervallen, die sich aus der Transitivität über ein drittes Intervall herleiten lassen, sind redundant und können entfernt werden. So entstehen Ketten transitiver Beziehungen, die Hrycej *transitive Ketten* nannte.

Durch Löschen redundanter Relationen erhält der Digraph eine Struktur, die viele Verbindungen nicht mehr explizit darstellt. Es werden aber nicht alle redundanten Verbindungen, die durch Kreuzen von Ketten entstehen, entfernt.

6.5.3. Variable Zeitfenster

Um mit teils unbestimmten Dauern, Start- und Endpunkten umgehen zu können, erhält jedes Intervall ein fixes und ein variables Zeitfenster. Das fixe Zeitfenster gibt den Anfangs- und Endpunkt sowie die Dauer eines Vorgangs an. Das variable Zeitfenster gibt für den Anfangs- und Endpunkt und die Dauer eines Vorgangs jeweils ein Intervall an, das den Wertebereich beschränkt.

Veres Propagation kann, sobald Dauern bekannt sind, Anfangs- und Endzeitpunkte von Intervallen berechnen, die sequentiell verbunden sind. Dieser sequentiellen Anordnung der Intervalle entspricht in dem hier verwendeten Modell die Verknüpfung von Intervallen über eine Vorher/Nachher-Beziehung. Das bedeutet, daß bei Vere nur in einem Spezialfall eine Propagierung von festen Zeiten möglich ist. Der hier vorgestellte Algorithmus propagiert feste Dauern und Zeitpunkte durch den gesamten oben erwähnten Digraphen. Er geht über Vere weit hinaus, weil er 1. durch ein Netz von Zeitbeziehungen propagieren kann und 2. auch über andere Zeitbeziehungen als Vorher/Nachher-Beziehungen folgert.

Einige Annahmen werden dabei getroffen: Sind von außen keine weiteren Dauern oder Zeitpunkte gegeben, wird die unbekannte Dauer an die minimale Dauer gebunden im Sinne einer vorläufigen Bindung von Variablen. Start- und Endzeitpunkte werden möglichst nahe an den Zeitpunkt, von dem aus die Propagierung startete, gelegt.

6.5.4. Formale Darstellung des relationalen Netzwerkes

Graphisch lassen sich die Intervalle und Relationen, die zwischen ihnen bestehen, durch einen bewerteten Digraphen im weiteren Sinne $D = [V, E, c]$ darstellen.

Definition 6.18: Es sei $D = [V, E]$ ein Digraph, B eine beliebige Menge und c_B: $E \rightarrow B$ eine Abbildung, die jedem Pfeil $e \in E$ eine Beschreibung oder Bewertung $c_B(e)$ zuordnet. Dann heißt das Tripel $D_B = [V, E, c_B]$ ein beschriebener Digraph oder bewerteter Digraph im weiteren Sinne.

Zur Erinnerung: Ein bewerteter Digraph ist ein Digraph mit einer Abbildung $c: E \rightarrow IR \cup \infty$, d.h. jeder Kante werden bestimmte Kosten zugewiesen. Ein bewerteter Digraph im weiteren Sinne ermöglicht es, statt Gewichten auch andere Inhalte in die

[13] wegen Allens Propagierungsalgorithmus und Hrycejs transitive Ketten vgl. Kapitel 5

Darstellung mitaufzunehmen. Diese Erweiterung ist nötig, da man hier nicht kardinal bewertet, sondern Relationen zwischen Intervallen explizit aufführen will. Wir definieren daher weiter:

Definition 6.19: Es sei $D_B = [V, E, c_B]$ ein beschriebener Digraph oder bewerteter Digraph im weiteren Sinne, c_B: $E \rightarrow B$ die Beschreibungsfunktion, R eine Menge von Relationen und $B = \mathbf{P}(R)$, wobei $\mathbf{P}$ für die Potenzmenge steht. Dann nennt man den Digraphen D_B auch relationalen Digraphen D_R.

Definition 6.20: Es sei $D_R = [I, E, c]$ ein relationaler Digraph, I die Menge der Intervalle, die paarweise durch einen Pfeil verbunden sind, E die Menge der Pfeile, $c: E \rightarrow \mathbf{P}(R)$ die Beschreibungsfunktion, die jeder Kante eine Menge von Zeitrelation zuordnet mit $R = \{<, >, m, mi, o, oi, d, di, s, si, f, fi, =\}$. Dann nennt man den Digraph D_R auch *Zeitgraph Z.*

Bemerkung: Man könnte auch den Zeitgraphen als vollständigen Graphen definieren. In diesem Fall wäre sowohl der Knoten v mit dem Knoten w, als auch w mit v durch je einen beschriebenen Pfeil (siehe Definition 6.18) verbunden. Weil die Inversen der Zeitrelationen bekannt sind, stellt eine Doppelbindung keine zusätzliche Information dar.

Um den Zeitgraphen zu strukturieren und anschließend vereinfachen zu können, faßt man 10 der 13 Relationen zu 4 Relationen *vor, nach, während* und *enthält* wie folgt zusammen:

Definition 6.21: Die Relationen *vor, nach, während* und *enthält* werden wie folgt definiert:
$vor := <' \subseteq \{<, m\}$
$nach := >' \subseteq \{>, mi\}$
$enthält := di' \subseteq \{si\ di\ fi\}$
$während := d' \subseteq \{s\ d\ f\}$

Wie man sofort sieht, sind diese Relationen transitiv. Um diese Eigenschaft im Graphen auszunutzen, definieren wir zwei zusätzliche Pfeilmengen:

Definition 6.22: Sei $Z = [I, E, c]$ ein Zeitgraph, dann werden die zwei Pfeilmengen E_1 und E_2 wie folgt definiert:

$E_1 := \{e \in E: c(e) = <'\}$

$E_2 := \{e \in E: c(e) = di'\}$

$E_3 := \{e \in E: c(e) = >'\}$

$E_4 := \{e \in E: c(e) = d'\}$

Um eine übersichtliche Darstellung zu erhalten, soll ein Pfeil $e = [w, v] \in E_3$ gedreht werden, d.h. der Pfeil e wird durch einen Pfeil $e' = [w, v] \in E_1$ ersetzt. Außerdem werden Pfeile $e = [v, w] \in E_4$ durch einen Pfeil $e' = [w, v] \in E_2$ ersetzt. Man erhält einen Graphen, der neben den üblichen Pfeilen noch Einfachpfeile $e_1 (\rightarrow)$, e_1 aus E_1, und Doppelpfeile $e_2 (\Rightarrow)$, e_2 aus E_2, enthält. Übliche Pfeile repräsentieren Relationen, die nicht zu E_1, E_2, E_3 oder E_4 gehören. Weil solche Pfeile eher die Ausnahme sind, werden sie mit dem nicht ganz so einfachen Symbol $\mapsto$ dargestellt. Man kann den Graphen weiter in streng transitive Untergraphen aufteilen, die man in einem weiteren Schritt reduzieren kann. Teilgraphen, die nur Kanten der Art E_1 und E_2 besitzen, haben noch eine charakteristische Eigenschaft:

Definition 6.23: Ein beschriebener Digraph $D_B = [V, E, c_B]$ mit $c_B: E \rightarrow B$ wird streng transitiv bzgl. $B' \subset B$ *genannt, falls er transitiv ist und für alle Knoten u, v, w* gilt: $c_B[u, v] \in B' \wedge c_B[v, w] \in B' \Rightarrow c_B[u, w] \in B'$. *Falls die Beschreibungsfunktion konstant ist, nennt man den Digraphen streng transitiv. Der Digraph D = [V, E]* heißt *der zu D_B gehörige Digraph.*

Satz 6.9: Sei $Z = [I, E, c]$ ein Zeitgraph, $I_1, I_2 \subseteq I$. Dann gilt:

(1) der Zeitgraph $Z_1 := [I_1, E_1, c]$ ist streng transitiv

(2) der Zeitgraph $Z_2 := [I_2, E_2, c]$ ist streng transitiv.

(3) der Zeitgraph $Z_3 := [I_3, E_3, c]$ mit $E_3 \subseteq E_1 \cup E_2$ ist streng transitiv bzgl. {<', di'}, falls für alle $u, v, w \in E_3$ gilt: aus $[u, v] \in E_3$ und $[v, w] \in E_3$ folgt $[u, w] \in E_3$.

Beweis: (1) Seien $u, v, w \in I_1$, $[u, v], [v, w] \in E_1$. Dann gilt für die zugehörigen Intervalle $u <' v$, $v <' w$. Nach Allens Schlußfolgerungstabelle erhält man $u <' w$, also $c_B[u, w] = <'$ (Beachte: <' ist unsere neu definierte Relation vorher).
$\Rightarrow [u, w] \in E_1 \Rightarrow Z_1 = [I_1, E_1, c]$ ist streng transitiv.

(2) analog wie (1).

(3) nach Definition ist das so. q.e.d.

Um der Transitivität einiger Zeitrelationen Rechnung zu tragen, führen wir den Begriff der transitiven Kette ein. Zur Erinnerung: "Eine Kette ist eine offene Kantenfolge $(v_{i_0}, v_{i_1}, ..., v_{i_s})$ mit lauter verschiedenen Knoten v_{i_σ} $(\sigma = 0(1)s)$ (...)" [Neumann 75b, S. 26].

Definition 6.24: Eine *transitive Kette* ist eine Kette in einem transitiven Graphen oder in einem streng transitiven beschriebenen Graphen.

In unserem Zeitgraphen kann man drei transitive Ketten unterscheiden.

Definition 6.25: Sei $Z = [I, E, c]$ ein Zeitgraph.

(1) eine *vorher/nachher-Kette* ist eine transitive Kette in dem Teilgraphen $Z_1 = [I_1, E_1, c]$

(2) Eine *während/enthält-Kette* ist eine transitive Kette in dem Teilgraphen $Z_2 = [I_2, E_2, c]$

(3) Eine *gemischte transitive Kette* ist eine transitive Kette in dem Teilgraphen $Z_3 = [I_3, E_3, c]$ (definiert wie oben).

6.5.5. Redundante Verbindungen im Zeitgraphen

Mithilfe der Reduktion sollen zwischen Knoten diejenigen Pfeile gelöscht werden, die sich auch über eine transitive Kette verbinden lassen. Eine Reduktion findet nur in streng transitiven Teilgraphen statt. Wenn sich zwei Knoten *v, w* über eine transitive Kette verbinden lassen, nennt man ihre gerichtete Kante *[v, w]* redundant. Redundante Verbindungen im Zeitgraphen zeigen die folgenden Beispiele:

Beispiele redundanter Verbindungen: A, B, C: Intervalle;

1. $A <' B, B <' C \Rightarrow A <' C$, <' ist eine redundante Verbindung

2. $A\ di'\ B, B\ di'\ C \Rightarrow A\ di'\ C$, *di'* ist eine redundante Verbindung

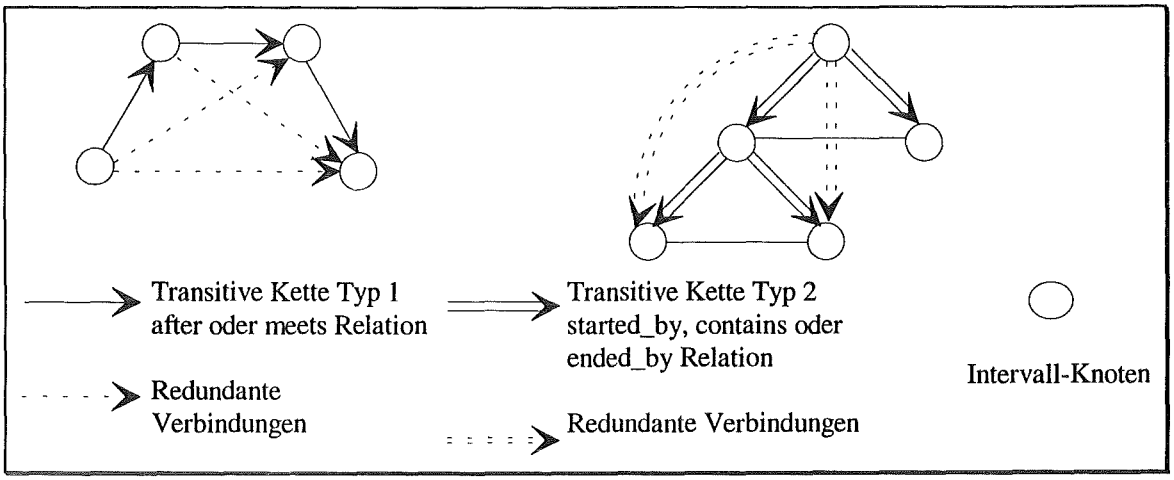


Abb 6.3: Redundante Relationen vom Typ 1 (links zu Beispiel 1) und Typ 2 (rechts zu Beispiel 2) innerhalb einer tranisitiven Kette können gelöscht werden.

3. $A <' B, B\ di'\ C \Rightarrow\ A <' C$, $<'$ ist eine redundante Verbindung

4. $A\ di'\ B, B <' C, C <' D, A\ di'\ D \Rightarrow\ A\ di'\ C$, di' ist eine redundante Verbindung

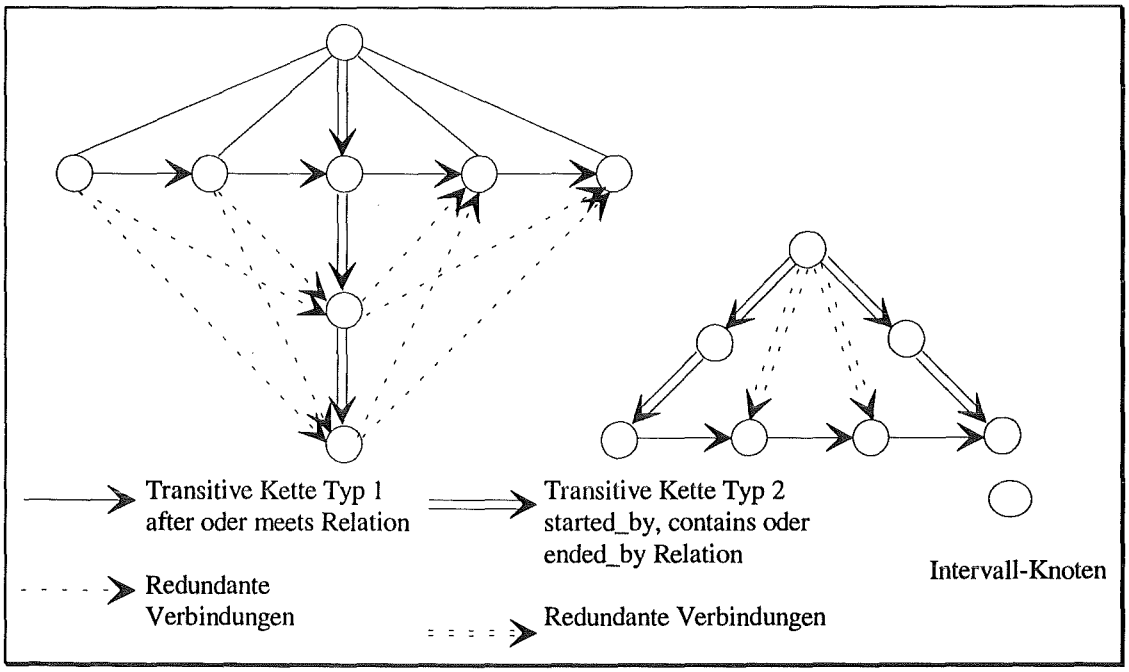


Abb 6.4: Redundante Relationen durch Kreuzen zweier (links zu Beispiel 3) bzw. dreier transitiver Ketten (rechts zu Beispiel 4) unterschiedlichen Typs.

Hrycej bietet eine Lösung zum Löschen der Relationen wie sie im Beispiel 1 und 2 auftreten an. In seinem Beispiel zeigt er jedoch nur das Löschen von vorher/nachher-Ketten. Als Ergänzung fügen wir eine Methode hinzu, die solche Redundanzen, wie sie im 3.und 4. Beispiel vorgestellt werden, findet und löscht.

Formal suchen wir eine Art Gerüst unseres Zeitgraphen, das alle Informationen enthält und die kleinstmögliche Anzahl an Pfeilen hat. Die übliche Begriffsbezeichnung *gerichtetes Gerüst* kann man hier nicht verwenden, da hier ein transitiver Teilgraph nicht

notwendigerweise einen gerichteten Baum darstellt. Deshalb führen wir den Begriff *Basisgraph* ein.

Definition 6.26: Sei $D = [V, E]$ ein zusammenhängender transitiver Digraph. Ein Untergraph $B = [V, E_B]$ von D heißt *Basisgraph von D*, falls
(1) die transitive Hülle $[V, E_B'] = [V, E]$
(2) keine Menge $E_{B'} \subset E_B$ existiert mit $[V, E_{B'}'] = [V, E_B]$

Definition 6.27: Sei $D_B = [V, E, const]$ ein zusammenhängender, streng transitiver beschriebener Digraph. Ein Untergraph $B = [V, E_B, c_B]$ von D heißt Basisgraph von D_B, falls $B' = [V, E_B]$ der Basisgraph des zu D_B gehörenden Digraphs $D = [V, E]$ ist.

Jetzt können wir formal das Gerüst beschreiben, das wir für unseren Zeitgraphen suchen.

Definition 6.28: Sei $Z = [I, E, c]$ ein Zeitgraph. Dann nennt man einen Zeitgraph $Z = [I, E', c]$ einen *minimalen Zeitgraphen* oder *Gerüst des Zeitgraphen Z*, falls alle bzgl. $\{<', di'\}$ streng transitiven Teilgraphen durch ihre Basisgraphen ersetzt wurden.

Definition 6.29: Sei $D_B = [V, E, c_B]$ ein beschriebener Digraph mit $n \in IN$ paarweise disjunkten Pfeilmengen $E_1,..E_n$ so, daß die Untergraphen $D_i = [V_i, E_i, const.]$ mit $i = 1,..n$ streng transitiv sind. Dann nennt man einen Pfeilzug $[v_{i0}, v_{i1},...v_{is}]$, $s \geq 2$, mit $e_{k_s} = \left[v_{i_{s-1}}, v_{i_s}\right] \in \bigcup_{i=1}^{n} E_i$ (s=0(1)s) eine gemischte transitive Kette, falls $e = \left[v_{i_0}, v_{i_s}\right] \in \bigcup_{i=1}^{n} E_i$.

Wir suchen also einen Zeitgraph, dessen bzgl. $\{<' di'\}$ streng transitive Teilgraphen durch Basisgraphen ersetzt werden. Im Weiteren wird die Vorgehensweise der Reduktion beschrieben. Sie schließt sich in der Abfolge direkt an die Constraintpropagierung nach Allen an: Sobald eine neue Relation der Art <', di' eingefügt wird, untersucht man das Netz auf Reduktionsmöglichkeiten und entfernt überflüssige Verbindungen.

Definition 6.30: Sei $Z = [I, E, c]$ ein Zeitgraph, $v \in I$. Dann definiert man:
$VOR_1(v) := \{w \in I: c[w, v] = <'\}$
$NACH_1(v) := \{w \in I: c[v, w] = <'\}$
$VOR_2(v) := \{w \in I: c[w, v] = di'\}$
$NACH_2(v) := \{w \in I: c[v, w] = di'\}$
$ZWISCHEN(v, w) := \{u \in I: c[v, u] = <' \wedge c[u, w] = <'\} = NACH_1(w) \setminus NACH_2(v)$

Knoten aus $VOR_1(v)$ und $VOR_2(v)$ sind Vorgänger von v im weiteren Sinne[14], Knoten aus $NACH_1(v)$ und $NACH_2(v)$ sind Nachfolger von v im weiteren Sinne.

6.5.6. Reduktion des Zeitgraphen

Reduziert wird direkt nach Einfügen einer neuen Bewertung $c[v, w] \in \{<', >', d' di'\}$. O.B.d.A. betrachten wir nur die Bewertungen $<', di'$. ($>', d'$ werden ja durch "Drehung" auf $<, di'$ zurückgeführt.)

[14]Zum Begriff des Vorgängers bzw. Nachfolgers im weiteren Sinne vgl. [Neumann 75, Bemerkung 1.19, S. 28].

Es seien also $v, w \in I$ und $c[v, w] = <'$. Dann werden folgende Pfeile aus dem Zeitgraphen gestrichen.

(1) $[v, u_1]$ mit $u_1 \in NACH_1(w)$ (korrekt: $u_1 \in NACH_1(v) \setminus \{w\}$)

(2) $[u_2, w]$ mit $u_2 \in VOR_1(v)$ (korrekt: $u_2 \in VOR_1(w) \setminus \{v\}$)

(3) $[u_2, u_1]$ mit $u_2 \in VOR_1(v)$ und $u_1 \in NACH_1(w)$ (dto.)

(4) $[u_2, u_3]$ mit $u_2 \in VOR_1(w)$ und $u_3 \in NACH_2(w)$ (dto.)

(5) $[u_3, u_1]$ mit $u_3 \in NACH_2(v)$ und $u_1 \in NACH_1(v)$

Ähnlich verfährt man, falls $c[v, w] = di'$. Man entfernt hier folgende gerichtete Kanten.

(6) $[v, u_3]$ mit $u_3 \in NACH_2(w)$

(7) $[u_4, w]$ mit $u_4 \in VOR_2(v)$

(8) $[u_4, u_3]$ mit $u_4 \in VOR_2(v)$ und $u_3 \in NACH_2(w)$

(9) $[u_2, u_3]$ mit $u_2 \in VOR_1(v)$ und $u_3 \in NACH_2(v)$

(10) $[u_2, u_1]$ mit $u_2 \in VOR_1(v)$ und $u_1 \in NACH_1(w)$

(11) $[u_3, u_1]$ mit $u_3 \in NACH_2(v)$ und $u_1 \in NACH_1(v)$

(12) $[u_2, u_1]$ mit $u_2 \in VOR_1(w)$ und $u_1 \in NACH_1(v)$

(1) - (3) beschreibt redundante Kanten, wie sie im Beispiel (1) aufgezeigt werden. Die Kanten (7) - (9) sind die überflüssigen Verbindungen aus Beispiel (2). Die in den verbleibenden Punkten (4) - (6) und (10) - (12) genannten Pfeile sind überflüssig, weil die Knoten, die sie verbinden, auch durch eine transitive Kette verknüpft werden können. Als Ergebnis erhält man einen teilweise reduzierten Zeitgraphen, keine Redundanzen der Beispiel 1. - 3. enthält.

Das Problem, das im vierten Beispiel angesprochen wird, bedarf einer aufwendigeren Lösung. Man kann redundante Kanten dieses Typs nicht unmittelbar nach dem Einfügen einer neuen Relation löschen. Sie treten meist später auf, nachdem andere Relationen eingefügt wurden, weil sie sich durch eine Relationenkombination ergeben, die sich nicht auf ein Relationenpaar beschränkt. Deshalb werden redundante Kanten dieses Typs erst entfernt, wenn keine weiteren Kanten hinzugefügt werden sollen. Im letzten Schritt der Reduzierung des Zeitgraphen wird geprüft, ob sich solche überflüssigen Verbindungen im Graphen befinden. Ist das der Fall, dann werden sie gelöscht. Bei allen gemischten transitiven Ketten, deren erster und letzter Pfeil aus E_1 ist, werden jeweils der Anfangsknoten v_0 und der Endknoten v_n nach gemeinsamen Vorgängern $w \in VOR_2(v_0) \cap VOR_2(v_n)$ überprüft. Falls solche Vorgänger existieren, werden - last but not least - folgende gerichtete Kanten gelöscht:

(13) $[w, v]$ mit $v \in ZWISCHEN(v_0, v_n)$

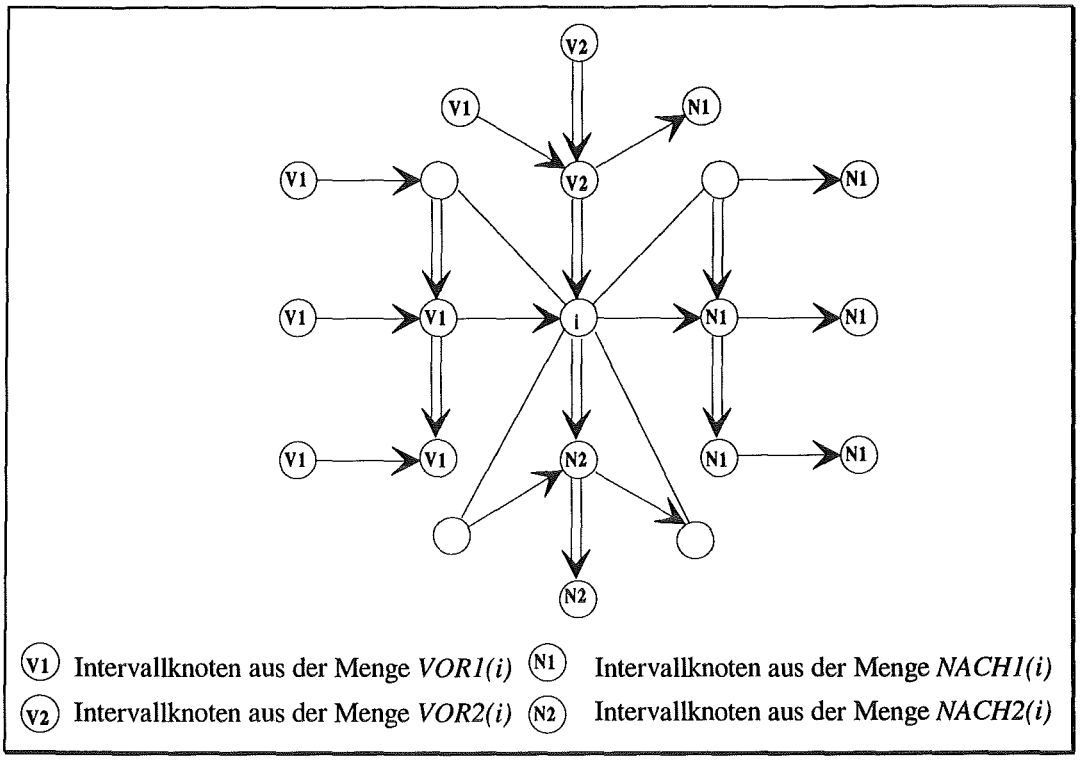


Abb. 6.5: Zeitnetz ohne redundante Verbindungen in Bezug auf Knoten i

Als Ergebnis erhält man das gesuchte Gerüst des Zeitgraphen. Er besteht hauptsächlich aus vorher/nachher-Ketten und während/enthält-Ketten.

6.5.7. Propagierung von Dauern und Zeitpunkten

In der Praxis sind oft Dauern von Operationen bekannt und die Start- und Endzeitpunkte gesucht. Im Fall der COMETOS-Zelle sind für Meß- und Bearbeitungsoperationen die Dauern nach dem ersten Durchlauf bekannt. Für das Einlernen kann man Dauern schätzen. Die Bearbeitung einer Achsbrücke kann z.B. 3 Minuten dauern. Gesucht ist der Bearbeitungsbeginn.

Zusätzlich kann der Anwender Bedingungen äußern: "Mit dem Einlernen will ich zwischen 10.00 h und 11.00 h anfangen." Für den Start wird ein Intervall aus frühestem und spätestem Anfangszeitpunkt vorgegeben. Das Verfahren kann all diese Fälle handhaben, die Folgen für alle andere Operationen und den Plan als Ganzes berechnen und jeder Operation anschließend den maximalen Freiraum zur Festlegung zuordnen, also nicht-linear planen. Abschließend werden die Präferenzen soweit wie möglich realisiert.

Jedem Intervall I ordnet man ein fixes und ein variables Zeitfenster zu. Das fixe Zeitfenster gibt feste Anfangs-, Endzeiten und Dauern an. Das variable Zeitfenster beschreibt Metaintervalle, innerhalb derer Anfangs-, Endzeiten und Dauern von I liegen.

Definition 6.31: Ein festes Zeitfenster *Zeitfenster-fix* beschreibt ein Intervall I mit einem Tripel (Anfang, Ende, Dauer), wobei sich Anfang, Ende und Dauer auf die zugehörigen Werte des Intervalls I beziehen.

Definition 6.32: Ein variables Zeitfenster Zeitfenster-variabel beschreibt ein Intervall I mit einem Sechstupel (frühester Anfang, spätetster Anfang, frühestes Ende, spätestes Ende, minimale Dauer, maximale Dauer) oder kürzer (FA SA FE SE MIN_D MAX_D). FA und SA sind feste Zeitpunkte, die den Wert für den Anfangszeitpunkt von I eingrenzen. FE und SE sind feste Zeitpunkte, die den Wert für den Endzeitpunkt von I ein-

grenzen. MIN_D und MAX_D sind feste Dauern, die den Wert für die Dauer von I eingrenzen.

Als Defaultwerte setzt man:

Zeitfenster_fix = (nil nil nil)

Zeitfenster_variabel = (-inf inf -inf inf 0 inf)

Inf ist die rechnerinterne größte Zahl, im Idealfall wäre *inf* gleich unendlich. Das Intervall, von dem aus die Propagierung startet, bekommt den Anfangspunkt 0 zugewiesen. Treten am Schluß negative Startzeiten auf, so werden einfach die Start- und Endzeiten um den Betrag der maximalen negativen Startzeit verschoben. Eine andere Möglichkeit wäre, die Defaultwerte des variablen Zeitfenster auf *(0 inf 0 inf 0 inf)* zu setzen und dem ersten Intervall den unbestimmten Startpunkt t_0 zuzuweisen. t_0 wird am Schluß so gewählt, daß keine negativen Anfangszeiten auftreten.

Folgende Beziehungen müssen während der gesamten Propagation eingehalten werden:

$$0 \le FA \le \left\{ \begin{matrix} SA \\ FE \end{matrix} \right\} \le SE$$

$$0 \le MIN\_D \le MAX\_D$$

$$FE - SA \le MIN\_D \le \left\{ \begin{matrix} SE - SA \\ FE - FA \end{matrix} \right\}$$

$$MAX\_D \le SE - FA$$

Falls *ANFANG, ENDE, DAUER* $\neq$ *nil* muß auch noch gelten:

$ANFANG \le ENDE$

$FA \le ANFANG \le SA$

$FE \le ENDE \le SE$

Wenn von außerhalb Dauern eingegeben werden, können folgende neue Zeitpunkte berechnet werden.

Fall 1: Ein Wert für *DAUER* werde vorgegeben:
MIN_D := DAUER
MAX_D := DAUER
Falls *ANFANG = nil* und *ENDE* $\neq$ *nil*:
ANFANG = ENDE - DAUER;
Falls *ENDE = nil* und *ANFANG* $\neq$ *nil*:
ENDE = ANFANG + DAUER;

Fall 2: Ein Wert für die maximale Dauer *MAX_D* werde vorgegeben:

SA_NEU := min{SA, FA + MAX_D}

Dieser Wert muß zuerst ermittelt werden, da sonst Widersprüche zur oben aufgeführten Ungleichung $SA \le SE$ auftreten können. Es folgt dann:
SE_NEU := max{FA + MAX_D, FE}
FA_NEU := SE_NEU - MAX_D

Fall 3: Ein Wert für die minimale Dauer MIN_D werde vorgegeben:
SA_NEU := max{FE - MIN_D, FA}
FE_NEU := SA_NEU + MIN_D

Falls für ein Intervall I eine Dauer bekannt ist, werden zuerst also Änderungen der Anfangs- und Endwerte für I und anschließend die Änderungen der Anfangs- und Endwerte der mit I verbundenen anderen Intervalle berechnet.

Ändert sich ein Zeitpunkt bei der Propagation, so werden Dauern nach folgenden Gleichungen berechnet:

Gleichung 6.1: Falls *FA* oder *SE* sich ändern: *MAX_D_NEU := SE - FA*;

Gleichung 6.2: Falls *SA* oder *FE* sich ändern: *MIN_D_NEU := max{0,FE - SA};*

Gleichung 6.3: Falls *ANFANG* (bzw. *ENDE*) ein Wert $\neq$ *nil* zugewiesen wird und *ENDE* (bzw. *ANFANG*) $\neq$ *nil*: *DAUER := ANFANG - ENDE*

Sobald bei einem Intervall Zeitpunkte oder Dauern eingeschränkt werden, wird es in eine Warteschlange geschrieben, um auf die weitere Berechnung zu warten. Für alle anderen Intervalle, die mit einem Intervall aus der Warteschlange direkt verbunden sind, werden über Schlußfolgerungstabelle 1 (s.u.) Einschränkungen der Anfangs- und Endgrenzen berechnet. Sind mehrere Relationen zulässig, wird bei frühesten Start- bzw. Endzeitpunkten die minimale Zeit gewählt, bei den spätesten Start- und Endzeitpunkten die maximale. Bezeichnet man das Intervall aus den Grenzen *FA*, *SA* als *Anfangs_Intervall:=[FA, SA]* und entsprechend das *Ende_Intervall:=[FE, SE],* so werden die neuen Grenzen über eine Schnittoperation berechnet.

Gleichung 6.4: Anfangs_Intervall_NEU:=Anfangs_Intervall_ALT $\cap$ Anfangs_Intervall_EINSCHRÄNKUNG

Gleichung 6.5: ENDE_Intervall_NEU:=ENDE_Intervall_ALT $\cap$ ENDE_Intervall-_EINSCHRÄNKUNG

Am Ende erhält man ein Netzwerk, in dem alle Informationen verarbeitet worden sind. Man kann nun unbestimmte Start- und Endzeitpunkte sowie Dauern beliebig festsetzen. Strategie hierbei ist, Dauern möglichst minimal zu setzen und die Anfangs- und Endzeitpunkte möglichst nahe an den Startpunkt 0 (bzw. t_0) zu legen.

Schlußfolgerungstabelle 1:
A, B Intervalle;
FAA := frühester Anfang von Intervall A;
SAA := spätester Anfang von Intervall A;
FEA := frühestes Ende von Intervall A;
SEA := spätestes Ende von Intervall A;

A r B	*FAB*	*SAB*	*FEB*	*SEB*
<	*FEA*	*inf*	*FEA*	*inf*
>	*-inf*	*SAA*	*-inf*	*SAA*
m	*FEA*	*SEA*	*FEA*	*inf*
mi	*inf*	*SAA*	*FAA*	*SAA*
o	*FAA*	*SEA*	*FEA*	*SEA*
oi	*-inf*	*SAA*	*FAA*	*SEA*
d	*-inf*	*SAA*	*FEA*	*inf*
di	*FAA*	*SEA*	*FAA*	*SEA*
s	*FAA*	*SAA*	*FEA*	*inf*
si	*FAA*	*SAA*	*FAA*	*SEA*
f	*-inf*	*SAA*	*FEA*	*SEA*
fi	*FAA*	*SEA*	*FEA*	*SEA*
=	*FAA*	*SEA*	*FEA*	*SEA*

Tab. 6.5: Tabelle zur quantitativen Folgerung aus qualitativer Information

6.5.8. *Verfahrensaufbau und Komplexität*

Zum Aufbau des Zeitplaners werden einige Verfahren benötigt: Eine Funktion *Zeitrelation_existiert(i, j)* → {wahr, falsch} überprüft, ob im minimalen Zeitgraphen zwischen den Knoten (= Zeitfenster) i und j eine qualitative Zeitbeziehung explizit existiert. *Zeitrelation_existiert(i, i)* ist stets falsch.

Eine Funktion *Schneide_Intervalle(i, j)* → {wahr, falsch} berechnet die Schnittoperationen der Gleichungen 6.4 und 6.5 für das Zeitfenster i. Anfangs_Intervall_Einschränkung (i, j) und Ende_Intervall_Einschränkung (i, j) werden über die Schlußfolgerungstabelle 1 berechnet. *Schneide_Intervalle* ist wahr, falls Zeitfenster i dabei verändert wurde, sonst falsch:

```
Schneide_Intervalle(VAR i : Zeitfenster; j : Zeitfenster)
Anfangs_Intervall_NEU:=Anfangs_Intervall(i) ∩ Anfangs_In-
tervall_EINSCHRÄNKUNG(i, j)
ENDE_Intervall_NEU:=ENDE_Intervall(i) ∩ ENDE_Intervall-
_EINSCHRÄNKUNG(i, j)
falls (Anfangs_Intervall_NEU ≠ Anfangs_Intervall(i)) oder
  (ENDE_Intervall_NEU ≠ ENDE_Intervall(i))
  dann
    Anfangs_Intervall(i) := Anfangs_Intervall_NEU
    ENDE_Intervall(i) := ENDE_Intervall_NEU
    return wahr
  sonst
    return falsch
```

Die Komplexität von *Schneide_Intervalle* ist konstant und unabhängig von der Anzahl der Zeitfenster n. Zwischen i und j werden nur die dreizehn Zeitbeziehungen überprüft. Die Prozedur *Dauern_und_Zeitpunkte_prop(i)* berechnet Dauern und Zeitpunkte eines Zeitfensters i über jede seiner Beziehungen. Neue Dauern werden nach den Gleichungen 6.1 bis 6.3 ermittelt. Zwei Warteschlangen V, W nehmen von Änderungen betroffene

Zeitfenster zur weiteren Berechnung auf (s.u.). Ein Zeitfenster wird nur dann in V bzw. W angefügt, wenn es dort fehlt.

```
Dauern_und_Zeitpunkte_prop(VAR i : zeitfenster)
für jedes Zeitfenster j mit Zeitrelation_existiert(i,j)
  falls Schneide_Intervalle(j, i)
    dann
      führe Gleichungen 6.1 bis 6.3 für j aus
      hänge j ans Ende von W an
      hänge i ans Ende von V an
```

Dauern_und_Zeitpunkte_prop prüft im schlechtesten Fall n-1 Zeitfenster, ein Aufruf benötigt O(n) Zeit. Aufgrund quantitativer Festlegungen können Zeitbeziehungen obsolet werden. Die Funktion *Relation_gültig(i, j)* → {wahr, Zeitvektor} errechnet den aufgrund der Werte der Zeitfenster i und j zulässigen Zeitvektor $(i, j)_{neu}$. Dazu verwendet sie die Schlußfolgerungstabelle 2 (siehe Anhang). Ist $(i, j)_{neu}$ eine Einschränkung gegenüber dem Vektor (i, j) des minimalen Zeitgraphen, muß die Einschränkung durch diesen propagiert werden. Sie wird von der Prozedur *Qualitative_Änderungen(i)* in die Warteschlange ToDo[16] hineingeschrieben. Wurden alle Knoten überprüft, wird ToDo von der qualitativen Zeitpropagation (s.o.) abgearbeitet. Weil die neue qualitative Einschränkung wiederum quantitative Konsequenzen haben kann, werden alle Zeitfenster hinter W angefügt.

```
Qualitative_Änderungen(i: Zeitfenster)
für jedes Zeitfenster j ≠ i
  falls nicht (vektor := Relation_gültig(i, j))
    dann
      hänge vektor an das Ende von ToDo
      setze Änderungsmerker
falls Änderungsmerker gesetzt
  dann
    hänge alle Zeitfenster an das Ende von W
    solange ToDo nicht leer
      Hole nächsten Vektor aus ToDo
      qualitative Propagation von vektor
```

Der Aufwand von *Relation_gültig* ist konstant, weil zwischen zwei Zeitfenstern die dreizehn Zeitbeziehungen überprüft werden. Die erste Schleife besitzt einen linearen O(n) Aufwand. Im schlechtesten Fall werden alle n Knoten in die Warteschlange ToDo geschrieben. Alle neuen qualitativen Einschränkungen können demzufolge mit einem Aufwand $O(n^2)$ propagiert werden, wie in [Allen 83, S. 837, rechte Spalte] gezeigt. Der Algorithmus für die quantitative Zeitplanung arbeitet wie folgt:

[16] Allen nannte diese Warteschlange ebenfalls ToDo (vgl. [Allen 83]).

```
Quantitative_Zeitplanung
hänge alle Zeitfenster an das Ende von W
Solange W nicht leer ist
  Solange W nicht leer ist
    hole nächstes Zeitfenster i aus W
    Dauern_und_Zeitpunkte_prop(i)
  Solange V nicht leer ist
    hole nächstes Zeitfenster j aus V
    Qualitative_Änderungen(j)
```

Das Programm terminiert, sobald weder qualitative noch quantitative Änderungen möglich sind. Unter der Voraussetzung, daß es für einen minimalen Zeitgraphen aufgerufen wurde, gilt: W und V enthalten nie mehr als n Knoten. Ein Durchlauf der ersten inneren Schleife überprüft maximal n Zeitfenster im Verhältnis zu jeweils (n-1) Zeitfenstern. Der Aufwand liegt bei $O(n^2)$. Die zweite Schleife ruft höchstens n mal die Prozedur *Qualitative_Änderungen* auf, die ihrerseits einen Aufwand $O(n^2)$ hat, braucht also $O(n^3)$. Danach besitzt W wiederum höchstens n Elemente. Im schlechtesten Fall muß *Qualitative_Änderungen* für alle $13 * \frac{(n-1)*(n-2)}{2}$ Beziehungen aufgerufen werden. Der Aufwand für die quantitative Propagation liegt bei $O(n^4)$ und ist damit in der Praxis realisierbar.

6.5.9. Zusammenfassung der Zeitpropagation

Unterschiedliche Zeitinformationen werden in einem einzigen Verfahren verarbeitet. Der Zeitgraph erlaubt, mit relativen und unbestimmten Zeiten umzugehen. Es ist eine Darstellung, die jederzeit Informationen über Zeitbeziehungen liefert und außerdem jede neue Information durch den Graphen propagiert. So können auch feste Zeitpunkte und Dauern durch den Graphen berechnet werden.

Trotz der Reduktion des Zeitgraphen ist diese Form der Propagation rechenintensiv, weil Allens Verfahren einen Aufwand von $O(n^3)$ besitzt, der in der Größenordnung unverändert bleibt. Einen Aufwand von $O(n^4)$ hat die quantitative Propagation. Das Verfahren kann deswegen für mittelgroße Probleme gut eingesetzt werden.

Die Zeitpropagation kann Teil des dritten Schritts "Planen" sein. Sie wird Bestandteil der Verifikation, etwa für eine Aktivität: "Wenn die Vorbedingungen erfüllt sind, die Unterobjekte gelten und das Zeitfenster der Aktivität in das Netz eingefügt werden kann, dann sichere die Nachbedingungen zu." Ist eine kontinuierliche zeitliche Überprüfung nicht notwendig, kann sie auch am Ende der Planung erfolgen.

Ist eine ganze Sequenz verplant worden, kommt es vor, daß Zeiten immer noch nicht eindeutig festgelegt sind. Vorgänge könnten anstelle eines Anfangszeitpunktes nur Werte für FA und SA haben. Mit einer beliebigen Planungsstrategie, z.B. der Kürzeste-Operationszeit-Regel, wird der Anfang jedes Vorgangs so früh (oder so spät usw.) wie möglich gelegt. Im Gegensatz zur Planung mit naiven Ansätzen kommt es dabei auf keinen Fall zu Sackgassen.

6.6. Opportunistisch Planen

Mit den vorgestellten Verfahren lassen sich weitere Formen der Planung realisieren. In der Maschinenbelegungsplanung kann es interessant sein, opportunistisch zwischen maschinen- und auftragsorientierter Planung zu wechseln. Maschinenorientiert planen bedeutet, zuerst den Engpaßmaschinen Operationen zuzuweisen und dann weniger knappe Ressourcen anzupassen. In der COMETOS-Zelle kann z.B. der Meßroboter bei sehr viel Einlernarbeit zur knappen Ressource werden. Bei mehreren Bearbeitungsrobotern ist es

sinnvoll, ihn zuerst zu verplanen. Für die übrigen Roboter und die manuellen Putzzellen geht man anschließend zur auftragsorientierten Planung über.

Für eine Menge von Aufträgen fand eine Planung mit dem Propagationsschema (vgl. S. 61) und der Zeitplanung statt. Die Zeitplanung legt nicht immer eine Totalordnung der Operationen fest und gibt für Start- und Endzeitpunkt oft nur Intervalle an. Zuerst wählt man die Zeitpunkte von Operationen der kritischen Maschine. Danach werden die verbleibenden Operationen jedes Auftrags angepaßt: Vorgängeroperationen starten so spät wie möglich, Nachfolgeoperationen so früh wie möglich. Sie werden nach der Kürzeste-Operationszeit-Regel eingeplant. Ergebnis ist eine Belegungsplanung, die völlig auf die kritische Maschine hin organisiert ist. Durch zusätzliche Anwendung der KOZ-Regel kann ein Ergebnis nahe dem Optimum erwartet werden. Im Unterschied zur linearen Planung können keine Sackgassen auftreten. Deswegen kann jede beliebige Planungsstrategie leicht realisiert werden.

6.7. Zusammenfassung der Konzepte

Bisher fehlte ein Ansatz, um Reihenfolgeprobleme einheitlich darzustellen. Die Darstellung ist eine Erweiterung der Aktivitäten und Zuständen von Sathi und Fox. Der Digraph von Aktivitäten und Zuständen besitzt eine graphentheoretische Fundierung. Daraus wurden Darstellungen für Zustandsraumgraphen und Problemreduktionsgraphen entwickelt. Das Modell repräsentiert also explizit kausales, hierarchisches und temporales Wissen und erlaubt die vollständige, präzise und klare Darstellung bisher nicht formalisierter Reihenfolgeprobleme. Um die Anwendungsumgebung zu modellieren, stehen zwei Objekthierarchien zur Verfügung, die zusätzliches Wissen leicht integrieren.

Der skelletale Planungsrahmen soll die Entwicklung und den Einbau von Planungsverfahren unterstützen. Das Planungsschema baut auf den vier identifizierten Grundelementen von Heuristiken auf, damit ein Anwender neue Verfahren rascher erstellen kann. Er kann es für Objekte individuell anpassen, um problemspezifische Parameter zur Komplexitätsreduktion auszunutzen.

Für Zeitprobleme entstand ein nichtlineares, durchgängiges Verfahren, das qualitative und quantitative Informationen verarbeitet. Mit einem Reduktionsverfahren werden Redundanzen radikaler entfernt als das bisher möglich war. Die Zeitplanung berücksichtigt auch quantitative und qualitative Bedingungen an Zeitpunkte und Dauern.

Die Kombination zeitlicher und kausaler/hiearchischer Planung erlaubt, Strategien ohne Rücksicht auf Interaktionen zu implementieren, insbesondere bei komplizierten Zeitverhältnissen. Eine Planungsstrategie kann man also opportunistisch wählen. Die wichtigsten Neuerungen sind:

- Eine formalisierte Darstellung, die wichtige Konzepte der Reihenfolgeplanung unterstützt,
- Ein geschlossener Ansatz für wichtige Repräsentationsformen von Reihenfolgeproblemen und deren Mischformen,
- Quantitative Zeitinformation wird über alle qualitativen Zeitbeziehungen propagiert..
- Ein durchgängiges Verfahren zur Verarbeitung qualitativer und quantitativer Zeitinformationen.

7. Implementierung

Wie kann man die Konzepte von Kapitel 6 in einem Reihenfolgeplaner realisieren? Der erste Abschnitt zeigt, wie der Planungsansatz in die Umgebung von COMETOS eingebunden ist. Eine funktionale Analyse beschreibt im zweiten Abschnitt wichtige Prozesse und ihr Zusammenspiel. Die objektorientierte Analyse stellt Objekte zusammen, verfeinert sie und weist ihnen Attribute zu. Daraus entsteht die Wissensrepräsentation. Jedes Objekt erhält Methoden, die im dritten Abschnitt beschrieben werden. Der Zeitplanung ist ein eigener Absatz gewidmet, weil sie sich mit einer prozeduralen Darstellung leichter erschließt.

7.1. Integration und Schnittstellen

Zu den Funktionen des Planers zählen das Auffinden schnellster Wege für ein autonomes Fahrzeug, die Maschinenbelegungsplanung und die zeitoptimale Anordnung von Bahnelementen. Es gibt zwei Anwendertypen: 1. Der Wissensmanager fügt der Wissensbasis anwendungspezifisches Wissen hinzu, um eine Planungsumgebung zu modellieren. 2. Der Endanwender gibt Aufträge ein und erwartet eine Lösung.

Bahn- und Wegelemente stellt das COMETOS-System in externen Datenbanken bereit. Der Planer liest die Bahnelemente und schreibt das Planungsergebnis in die Datenbank zurück. Für die Maschinenbelegung gibt es noch keine Datenbank. Abbildung 7.1 beschreibt den Reihenfolgeplaner in seiner Umgebung.

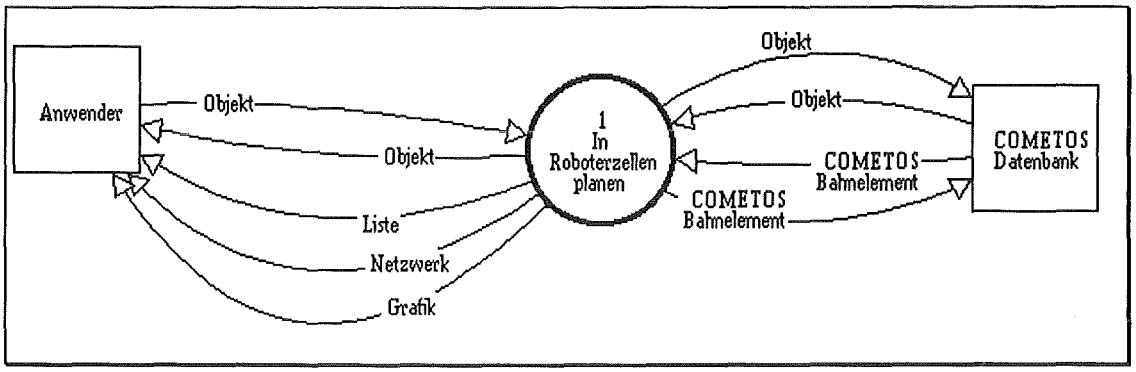


Abb. 7.1: Außenansicht des Reihenfolgeplaners. Links die Interaktionen mit dem Anwender, rechts die Interaktionen mit COMETOS.

7.1.2. Benutzeroberflächen

Eine standardisierte grafische Darstellungsform für Reihenfolgeprobleme hat sich bis heute nicht durchgesetzt. Der Reihenfolgeplaner bietet dem Anwender problemspezifisch drei Darstellungsformen an. Der Wissensingenieur erhält eine Netz- und Listendarstellung, die auf dem "Schema Network Editor" von KnowledgeCraft™ aufbaut. Mit Microsoft Project™ werden Projekte wie auch Maschinenbelegungsprobleme dargestellt. Die Darstellung des Bahnelementeproblems schließlich entstand in Eigenentwicklung.

7.1.2.1. Wissensingenieur

Der Wissensingenieur modelliert Reihenfolgeprobleme. Folgende Tätigkeiten fallen dabei an:

- Erstellen von Objekten und ihren Beziehungen
- Auswahl von Planungswissen

- Schreiben kurzer Stücke Lispcode als Testbedingung.

Der Planer ist auf einer SUN unter KnowledgeCraft implementiert. Der Wissensingenieur kann dort auf die volle Funktionalität von KnowledgeCraft zurückgreifen.

Die Editoren von KnowledgeCraft sind Maus-Menü orientiert und bauen auf der Fenstertechnik auf. Im Gegensatz zum professionellen Open Look™ wirken sie jedoch bescheiden. Andererseits stellt KnowledgeCraft bereits eine Reihe von Werkzeugen zur Verfügung, mit deren Hilfe sich der Reihenfolgeplaner leichter anpassen läßt als mit einer importierten Oberfläche.

Der "Palm Network Editor" ist ein solches Werkzeug, das die Handhabung von Objekthierarchien grafisch unterstützt. Daraus entstand ein Editor, der die Modellierung von A/S-Clustern unterstützt und hilft Zeitbeziehungen aufzubauen sowie einzelne Objekte zu bearbeiten. Ausgabefunktionen stellen die Planungsergebnisse dar. Dazu gehört eine Liste der Ressourcen, ihre Belegungen und die Belegungszeiten, eine sortierte Liste der Bahnelemente in der Reihenfolge des Abfahrens und eine Liste der Operationszeiten. Methoden werden Objekten einfach zugewiesen oder vererbt. Speziellere Objekte können die Methoden allgemeinerer Objekte erben, spezifizieren oder ganz überschreiben. Eigene Objektklassen kann der Wissensingenieur beliebig mit weiteren Methoden ausstatten.

7.1.2.2. Planungsanwender

Der Endanwender kann von seinem COMETOS-PC aus planen. Seine Arbeit beschränkt sich auf Aufgaben wie Dateneingabe, Modifikation, Aufruf des Planers und Ausgabe der Lösung. Einige Funktionen unterstützen das Editieren und die grafische Darstellung, andere die Ausgabe. Ein einziger Befehl löst die Planung aus. Die Oberflächen für den Endanwender konzentrieren sich auf wenige Funktionen. Aufwendige Grafiken unterstützen die Arbeit des Bedieners, verringern die Einlernzeit und steigern die Systemakzeptanz.

Der Planer besitzt zwei Umgebungen für den Endanwender: Die Umgebung zur Planung von Bahnelementen entstand im Rahmen einer Diplomarbeit (vgl. [Lotter 92]) und dient als Preprozessor, der aus geometrischen Daten einen A/S-Digraphen erzeugt. Microsoft Project™ wurde für die Maschinenbelegungsplanung erweitert. Es stellt GANTT- und PERT-Diagramme dar.

Für die Optimierung von Bahnelementen läßt sich das Werkstück nicht darstellen, weil seine genaue Form umbekannt ist. Vom Laserscanner eingelesenen Grate beschreiben es. Sind alle mit COMETOS zu bearbeitenden Grate eingelernt und nötige Zusatzinformationen [vgl. Kapitel 3.4] eingegeben, kann man das Werkstück mit Quadern umhüllen. Abbildung 7.2 veranschaulicht das am Beispiel der Seitenwand einer Druckmaschine.

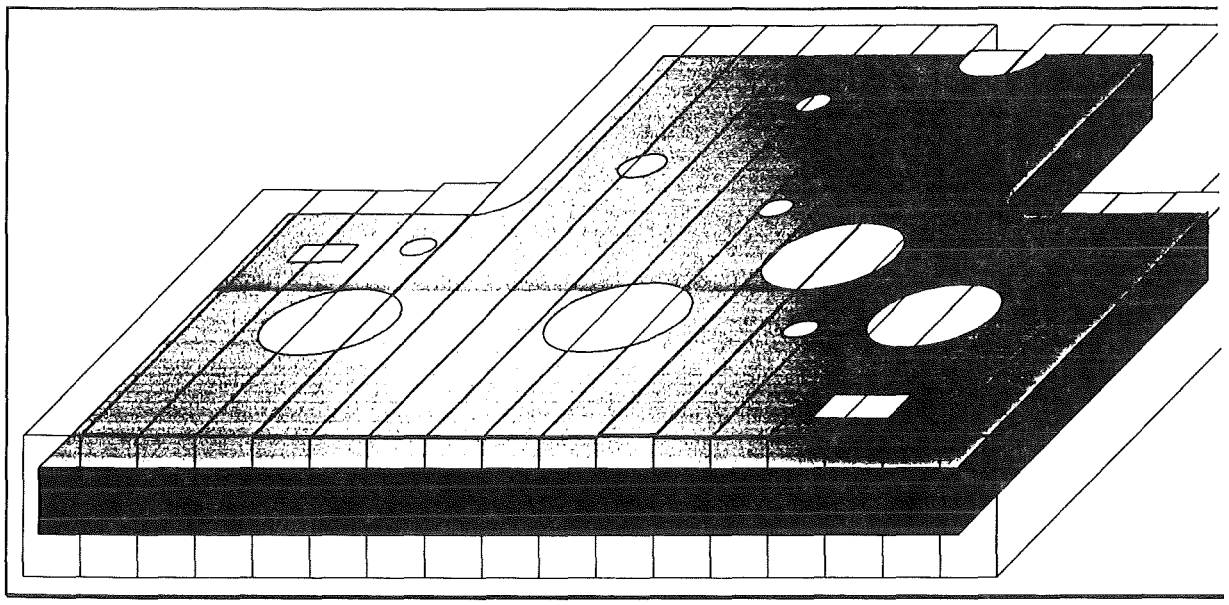


Abb. 7.2: Quader um die Seitenwand einer Druckmaschine

Das Programm liest die COMETOS-Bahnelemente ein, übergibt sie dem Reihenfolgeplaner und stellt sie grafisch dar. Man kann es aus dem Ablaufplan oder der DOS-Kommandoebene heraus starten. Ein Aufruf enthält zwei Parameter. Der erste Parameter ist der Pfad mit Namen der bestehenden COMETOS-Bahnelementedatei *p_split.use*. Der zweite Parameter nennt den Namen der entstehenden Bahnelementedatei. Die Angabe des zweiten Parameters ist notwendig, um die durch die Optimierung festgelegte Reihenfolge rückgängig zu machen.

Die Bedienung des Programmes kann mit der Maus und/oder der Tastatur erfolgen. Zum Funktionsumfang gehören Vergrößern, Verkleinern, Verschieben des Ausschnitts, Rotation der Grafik und Ausdrucken.

Zur leichteren Unterscheidung erhalten Luft- und Bearbeitungsbahnen unterschiedliche Farben. Jede Bahn wird beschriftet, um ihre Position in der Reihenfolge zu verdeutlichen. Abbildung 7.3 zeigt, wie Bahnelemente auf Quadern dargestellt werden.

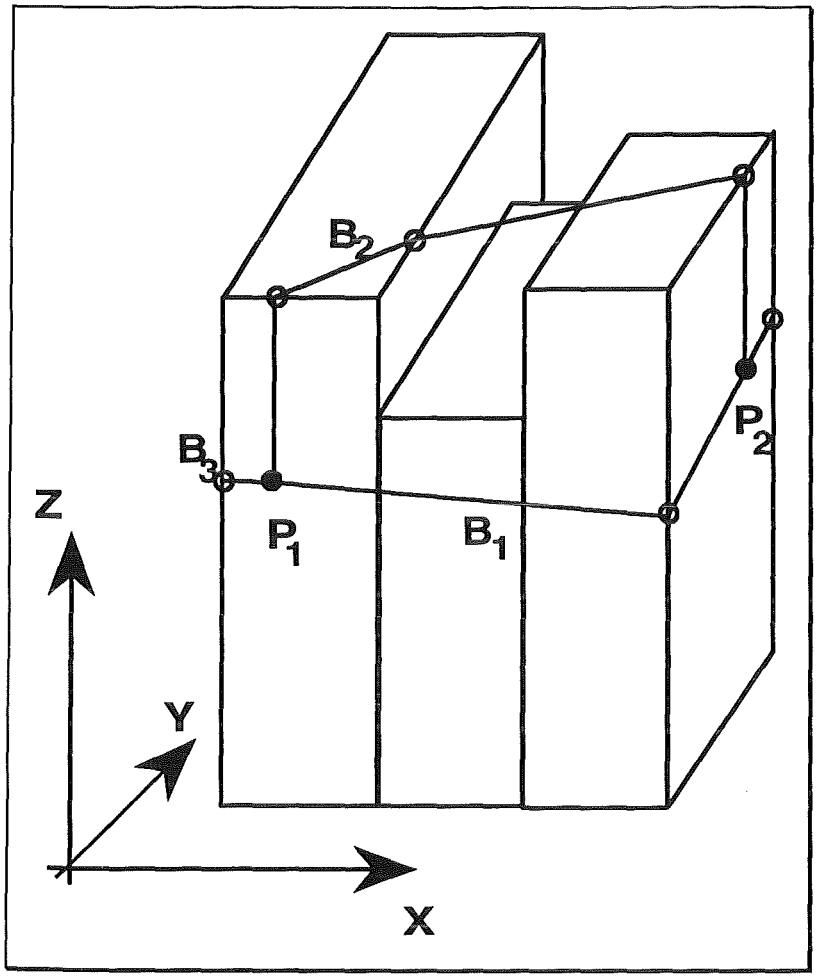


Abb. 7.3: Bahnelemente auf Quadern, die ein nur durch Grate beschriebenes Werkstück umhüllen. P_1, P_2 sind Endpunkte von Bearbeitungsbahnen. B_1, B_2, B_3 sind mögliche Luftbahnen von P_1 nach P_2. B3 geht hinten um das Werkstück herum.

In Microsoft Project, der Oberfläche für die Maschinenbelegungsplanung, gibt der Anwender zuerst seine Aufträge ein. Er kann über die Tastatur Daten in Tabellen schreiben, Masken ausfüllen und mit der Maus Vorgänge anordnen. Eine Sonderfunktion startet die Planung. Das Ergebnis kann frei manipuliert, verworfen, gespeichert oder ausgedruckt werden.[1]

7.1.3. Softwareumgebung

Die Software zu COMETOS wurde in Turbo-Pascal 5.0 unter DOS 3.0 realisiert. Drei 386 ATs sind über ein 3COM LAN verbunden. Eine V24 DNC Schnittstellenkarte verbindet sie außerdem mit den r500 Steuerungen der Roboter. Auf dem PC liegt ein Ablaufplan im ASCII-Format, dem eine Palette und ein Werkstück zugeordnet werden. Er ruft PC-Programme auf, die Daten hin zur Robotersteuerung r500 und von dort zurück übertragen, Roboterprogramme erzeugen und deren Ergebnisse verarbeiten. Auf der r500 laufen die vom PC erzeugten Roboterprogramme ab.

Die Daten eines Werkstücktyps werden permanent gespeichert, insbesondere die Bahnelementedatei und das Meßprogramm. Temporäre Daten sind einer Palette, auf der ein Werkstück liegt, zugeordnet. Die Bahnelementedateien besitzen ASCII-Format. Sie werden mit Nummer und Herkunft des Bahnelements abgespeichert: Der Dateiname

[1] Weitere Informationen zu MS Project stehen im Benutzerhandbuch.

verweist auf die Nummer, die Extension auf die Herkunft. Die Binärdatei split.use enthält die Reihenfolge der Bahnelemente, die durch das Messen entstanden ist. Die Datei p_split.use ordnet Bahnelementen ein Werkzeug zu. Sie enthält die Bearbeitungsreihenfolge einschließlich der zugeordneten Werkzeuge. Für den Zugriff auf die Bahnelementedateien stellt COMETOS in den Units split_pu und u_pkte Funktionen und Prozeduren zur Verfügung.

7.1.4. Schnittstellen

Der Präprozessor zur Planung der Bahnelemente ist auf einem PC in Turbo-Pascal™ realisiert. Er ist in sich abgeschlossen; externe Funktionen stellt er nicht zur Verfügung. Ein Aufruf im Ablaufplan startet ihn und verändert damit den Inhalt der Bahnelementedateien. Das Programm generiert zuerst die Hüllquader (vgl. Abbildung 7.2). Von der Datei *p_split.use* liest es die Dateinamen der Bahnelemente, welche die Punkte enthalten. Bahnelemente liegen mit ihrer Orientierung in kartesischen Koordinaten vor (vgl. [Lawo 90a, S. 82]). Das Programm wertet die Punktedateien aus, um den Bereich der X-Achse, in dem das Werkstück liegt, zu bestimmen. Es zerlegt ihn in n äquidistante Teilstücke und weist ihnen die lokal maximalen Y- und Z-Koordinaten zu. So entstehen Quader i, die durch die Koordinaten $xmin_i$, $xmax_i$, $ymin_i$, $ymax_i$, $zmin_i$ und $zmax_i$ gegeben sind. Ihre Daten speichert eine temporäre Datei.

Das Programm erzeugt eine Arbeitskopie der Bahnelementedateien und der Datei p_split.use, um dem Anwender die Möglichkeit zu geben, das Ergebnis zu verwerfen. Sie enthält am Ende die optimierte Reihenfolge. Das Programm erzeugt Quader um das Werkstück und teilt die Bahnelemente an Stellen gleicher Orientierung. Zwischen allen Anfangs- und Endpunkten der Bearbeitungsbahnen generiert es mit Hilfe der Quader kollisionsfreie Luftbahnen. Aus den roboterspezifischen Daten ermittelt das Programm für jede Luftbahn die Dauer, die nötig ist, um sie abzufahren. Es berücksichtigt dabei die Zeiten für Orientierungsänderungen der Roboterhand. Die Ergebnisse schreibt es in die Datei p_split.use und die Punktedateien. Es sucht Punkte mit gleicher Orientierung auf verschiedenen Bearbeitungsbahnen und teilt die Punktefiles an diesen Stelle. Die Datei *p_split.use* erhält nun die aktuellen Daten (vgl. [Lotter 92]). Ebenfalls unter Zugriff auf p_split.use werden die Bahnelemente nach Werkzeugen zur Bearbeitung des Werkstücks sortiert. Dabei bleibt die Zuordnung von Bearbeitungsbahnelementen zu Werkzeugen bestehen. Zwischen Gratgeometrie, Werkzeugwahl und Bearbeitungsqualität besteht ein Zusammenhang, der bisher nicht untersucht wurde. Das System kann deswegen nicht selbständig ein Werkzeug ersetzen. Andererseits kommen nicht mehr als fünf Werkzeuge zum Einsatz. Der Anwender kann die Werkzeugwahl also noch überblicken. Abbildung 7.4 veranschaulicht den Ablauf.

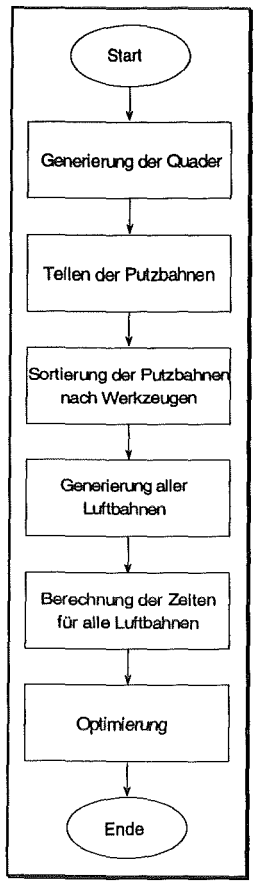


Abb. 7.4: Ablaufplan der Aufbereitung und Optimierung (vgl. [Lotter 92, S. 56]). Der Reihenfolgeplaner entspricht dem Kasten Optimierung, die Schritte davor bereiten die Daten für die Planung auf.

7.2. Funktionale Analyse

Der Reihenfolgeplaner besteht aus drei Prozessen: Der Modellierung von A/S-Digraphen (Prozeß 1.1), dem Propagationsschema (Prozeß 1.2) und der Zeitplanung (Prozeß 1.3). Den Datenaustausch zwischen ihnen stellt Abbildung 7.5 dar.

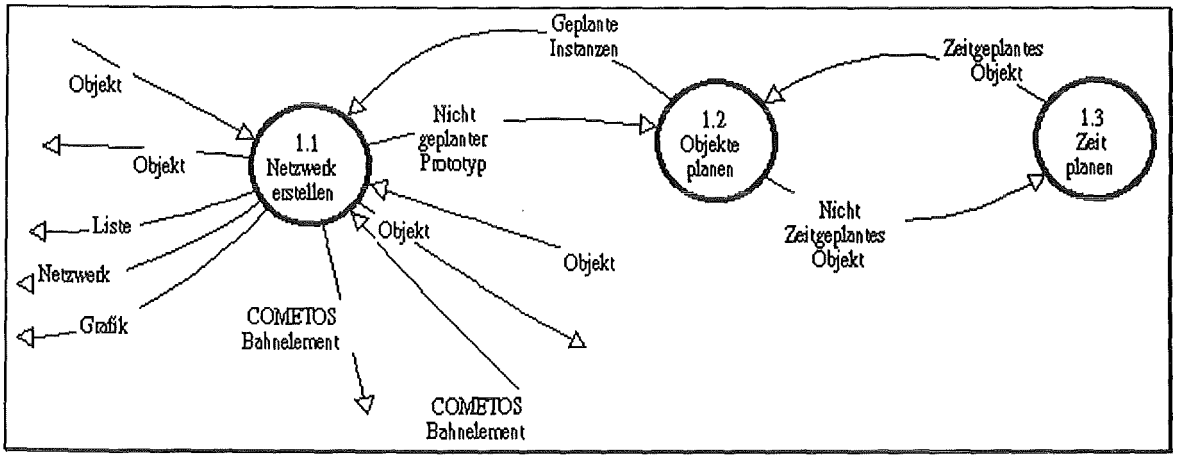


Abb. 7.5: Der Reihenfolgeplaner gliedert sich funktional in drei Prozesse. 1.1 dient der Arbeit an der Wissensrepräsentation, die anderen beiden planen.

Im Prozeß 1.1 *Netzwerk erstellen* modelliert der Anwender ein Reihenfolgeproblem in einen A/S-Digraphen. Alle dabei erstellten Objekte sind Prototypen. Der Prozeß enthält Ein- und Ausgabefunktionen sowie Konsistenzprüfungen. Er wandelt COMETOS-Da-

ten in A/S-Digraphen um und übergibt sie der Planung. Die Ergebnisse transformiert er zurück und schickt sie an das COMETOS-System. Der Prozeß 1.2 *Objekte planen* instanziiert Prototypen und ordnet sie mit dem Propagationsschema zu Sequenzen. Bei Bedarf übergibt er Instanzen an den Prozeß 1.3 *Zeit planen*, wo ihnen Zeitfenster zugeordnet werden, die qualitativ und quantitativ verarbeitet werden.

Prototypische Teildigraphen dienen als Arbeitspläne. In *Objekte planen* werden sie für die Planung instanziiert. Der Prozeß enthält die vier Bausteine Suchen, Sortieren, Einplanen und Bewerten (vgl. Kapitel 6).

Zur Zeitplanung gehören die qualitative Zeitplanung und quantitative Zeitplanung. Zuvor muß der Zeitplaner die Daten für die zeitliche Propagation aufbereiten: Zeitliches Schlußfolgern erfolgt nicht über A/S-Objekte, sondern über deren Zeitfenster. Zwischen Vorgängen, die die gleiche Ressource in Anspruch nehmen, werden Beziehungen vom Typ *before/after/meets/met-by* geschaffen, falls nichts anderes festgelegt ist. Damit wird vermieden, daß mehrere Vorgänge eine Ressource zur gleichen Zeit belegen.

7.3. Objektorientierte Analyse

Eine objektorientierte Analyse besteht aus fünf Schritten:

1. Klassen und Objekte finden
2. Strukturen identifizieren
3. Subjekte identifizieren
4. Attribute definieren
5. Dienste definieren.

Ein Teil der Analyse ist das Ergebnis von Kapitel 6: Klassen und Objekte entsprechen den Objekten der Wissensrepräsentation, Strukturen den Hierarchien, Attribute den Relationen und Dienste den Regeln zur Verifizierung. Die Dienste werden im folgenden Objekten zugeordnet und genauer spezifiziert.

Es wird folgende Notation verwendet: Um Objekte von Beziehungen, Attributen und Attributwerten optisch zu unterscheiden, werden Objekte stets in Kapitälchen geschrieben, z.B. ODER-AKTIVITÄT. Beziehungen und Attribute werden kursiv geschrieben, z.B. *has-elaboration*. Das Attribut oder die Beziehung eines gegebenen Objekts wird wie ein Record in PASCAL mit einem Punkt dazwischen geschrieben: PUTZEN.*utility* meint das Attribut *utility* des Objekts PUTZEN. Ein Doppelpunkt ":" steht für Vergleiche, das Definitionszeichen ":=" für Wertzuweisungen.

7.3.1. Allgemeine Klassen, Objekte und Strukturen finden

Die Struktur (Beziehung *is-a*) der Klassen und Objekte für den Problemlösungsprozeß stellt Tabelle 7.1 dar.

```
1. Objekt
  1.1. Aktivität
    1.1.1. UND-Aktivität
      1.1.1.1. Gesamtproduktion
      1.1.1.2. Tagesproduktion
    1.1.2. ODER-Aktivität
      1.1.2.1. Bahngruppe
  1.2. Vorbedingung
    1.2.1. UND-Vorbedingung
    1.2.2. ODER-Vorbedingung
      1.2.2.1. Besitzprädikat der Vorbedingung
      1.2.2.2. Zustandsprädikat der Vorbedingung
  1.3 Nachbedingung
    1.3.1. UND-Nachbedingung
    1.3.2. Zustandsprädikat der ODER-Nachbedingung
```

Tab. 7.1: Hierarchie der Objekte für den Problemlösungsprozeß

1. OBJEKT faßt Attribute und Dienste zusammen, die Aktivitäten und Zustände gemeinsam brauchen. Objekte der Nebenbedingung sind keine eigene Klasse, weil sie im Gegensatz zu Aktivitäten und Zuständen keine speziellen Eigenschaften besitzen. Für die Maschinenbelegungsplanung wurde die UND-AKTIVITÄT zu GESAMTPRODUKTION und TAGESPRODUKTION spezialisiert, die allen Einzeloperationen übergeordnet sind und die COMETOS-Aufträge nach eigenen Kriterien sortieren. BAHNGRUPPE ist eine Spezialisierung der ODER-AKTIVITÄT. Sie faßt mehrere Bahnelemente zu einer Gruppe zusammen. Den Objekten der taxonomischen Hierarchie wurde ein Planungsobjekt vorangestellt. Die Hierarchie zeigt Tabelle 7.2.

```
2. Planungsobjekt
  2.1. Physikalisches Objekt
    2.1.1. Ressource
      2.1.1.1. Roboter
        2.1.1.1.1. Putzroboter
        2.1.1.1.2. Meßroboter
        2.1.1.1.3. Trennroboter
      2.1.1.2. Palette
      2.1.1.3. Mensch
        2.1.1.3.1. Aufspannbediener
        2.1.1.3.2. Einlernbediener
        2.1.1.3.3. Gußputzer
      2.1.1.4. Werkzeug
        2.1.1.4.1. Fräsmeißel
        2.1.1.4.2. Schruppscheibe
        2.1.1.4.3. Fräser
        2.1.1.4.4. Trennscheibe
        2.1.1.4.5. Meißel
      2.1.1.5. Werkstück
        2.1.1.5.1. Achsbrücke
        2.1.1.5.2. Druckmaschinenseitenwand
        2.1.1.5.3. Zylinder
      2.1.1.6. Roboterhand
    2.1.2. Bahnelement
      2.1.2.1. Bearbeitungsbahn
      2.1.2.2. Luftbahn
      2.1.2.3. Werkzeugwechsel
    2.1.3. Zeitfenster
  2.2. Organisatorisches Objekt
    2.2.1. Bahnelemente-Optimierungsauftrag
    2.2.2. Maschinenbelegungsauftrag
    2.2.3. Unterauftrag
  2.3.Wissen
    2.3.1. Sortierverfahren
      2.3.1.1. Tagesproduktion-Sortierung
      2.3.1.2. Bahnelemente-Sortierung
      2.3.1.3. Bahnelemente-Verbesserung
    2.3.2. Suchverfahren
      2.3.2.1. ODER-Suche
      2.3.2.2. UND-Suche
      2.3.2.3.  Job-Suche
    2.3.3. Zielfunktion
      2.3.3.1. Bahnelemente-Zielfunktion
```

Tab. 7.2: Taxonomische Hierarchie der Anwendungsdomäne

7.3.2. *Attribute und Dienste für A/S-Objekte*

Dienste sind Prozeduren, die man einem Objekt zuordnet, um sein Verhalten zu beschreiben. Sie werden wie Attribute in Gegenrichtung zu den Beziehungen *is-a* und *instance* vererbt. Coad und Yourdon unterscheiden zwischen einfachen Diensten und komplexen Diensten [Coad 91, S. 146ff]. Einfache Dienste verbinden Objekte, schaffen Unterklassen, löschen ein Objekt und geben Attributwerte zurück. Im folgenden werden nur die komplexen Dienste beschrieben.

Über die in Kapitel 6 beschriebenen Relationen hinaus brauchen die Anwendungen die folgenden Beziehungen:

- *unbestimmte-Zeitrelation* verbindet zwei ZEITFENSTER (2.1.3) über eine Permutation der dreizehn qualitativen Beziehungen.
- *Hat-Unteraufträge* weist UNTERAUFTRÄGEN (2.2.3) einen AUFTRAG (2.2.2) zu.
- *Operation-von* weist Aktivitätsinstanzen ein ORGANISATORISCHES OBJEKT (2.2) zu.
- *kann-bearbeiten* verbindet ein WERKZEUG (2.1.1.5) mit der BEARBEITUNGSBAHN (2.1.2.1), die es bearbeiten kann.
- *hat-Bahnelemente* verbindet einen BAHNELEMENTE-OPTIMIERUNGSAUFTRAG (2.2.1) mit einem BAHNELEMENT (2.1.2).

Ein besonderer Attributtyp heißt *Anwenderslot*. Der Wissensingenieur kann beliebige Anwenderslots definieren und sie einem PLANUNGSOBJEKT (2.) zuweisen. Ein ZUSTANDSPRÄDIKAT DER VORBEDINGUNG (1.2.2.2) fragt sie zu seiner Verifizierung mit Prüfprädikaten ab; ein ZUSTANDSPRÄDIKAT DER NACHBEDINGUNG (1.3.2) revidiert mit Anwenderslots ein Planungsobjekt über *has-revision*.

Die Wurzel des Baums der A/S-Objekte (vgl. Abbildung 7.5.), OBJEKT, besitzt zwei Attribute. *Wert* speichert das Ergebnis der Verifizierung, *Typ* ist entweder *Prototyp* oder *Instanz*. Prototypen dienen zum Aufbau von A/S-Digraphen, Instanzen zur Planung. Um auf Sortier-, Such- oder Bewertungsverfahren zuzugreifen, kann jedes Objekt die Methode eines Wissensobjekts aufrufen. Zeitbehaftete Objekte erhalten mit *hat-Zeitfenster* ein ZEITFENSTER zugewiesen. Einen evtl. vorhandenen Wert für die im Zeitfenster eingetragene *Dauer* ermittelt der Dienst *Monitor-Dauer*. Der Dienst *prüfe-Zeitbelegung* stößt die Zeitpropagierung für alle verifizierten Objekte an. Abbildung 7.6 faßt die wichtigsten Eigenschaften zusammen.

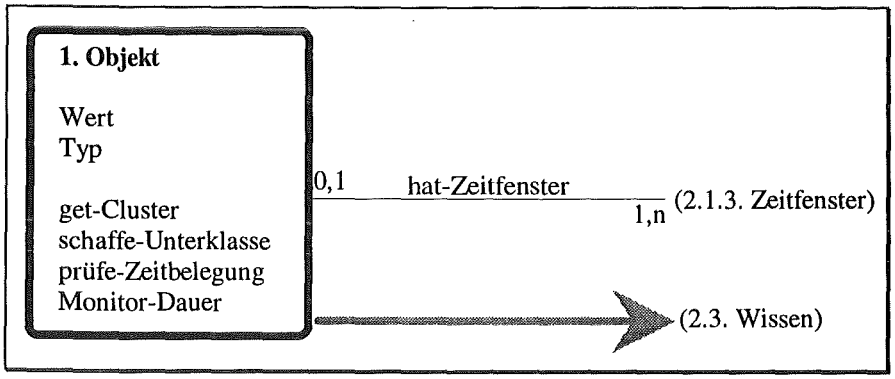


Abb. 7.6: Die Wurzel des Baums heißt OBJEKT.[2]

Aktivitäten und Vorbedingungen erhalten zusätzlich die Attribute *Utility* zur Aufnahme von Präferenzen und *Context* zur Unterstützung bei der Suche. Defaultwert in *Utility* ist 1. Die Wahrheitserhaltung wird durch Kontexte unterstützt, wenn Alternativen durchgespielt werden sollen. Wird eine Alternative abgelehnt, löscht man einfach die Contexte aller abgelehnten Alternativen. Einen Überblick über Attribute und Dienste für Aktivitäten gibt Abbildung 7.7.

[2]In dieser und den folgenden Abbildungen stellt ein schwarzer Kasten mit abgerundeten Ecken eine Objektklasse dar. Der Objektname steht in der ersten Zeile. Unter einer Leerzeile stehen die Attribute, unter einer zweiten Leerzeile die Methoden. Ein grauer Pfeil bedeutet, daß Dienste eines anderen Objekts in Anspruch genommen werden. Eine einfache Linie ist eine Relation, ihr Name steht auf der Linie. Die Zahlen begrenzen die zulässigen Einträge. Um die Richtung der Relation anzudeuten, wurden sie über (bei der Quelle) und unter (bei der Senke) die Linie geschrieben. Nicht immer konnten alle Beziehungen gezeichnet werden.

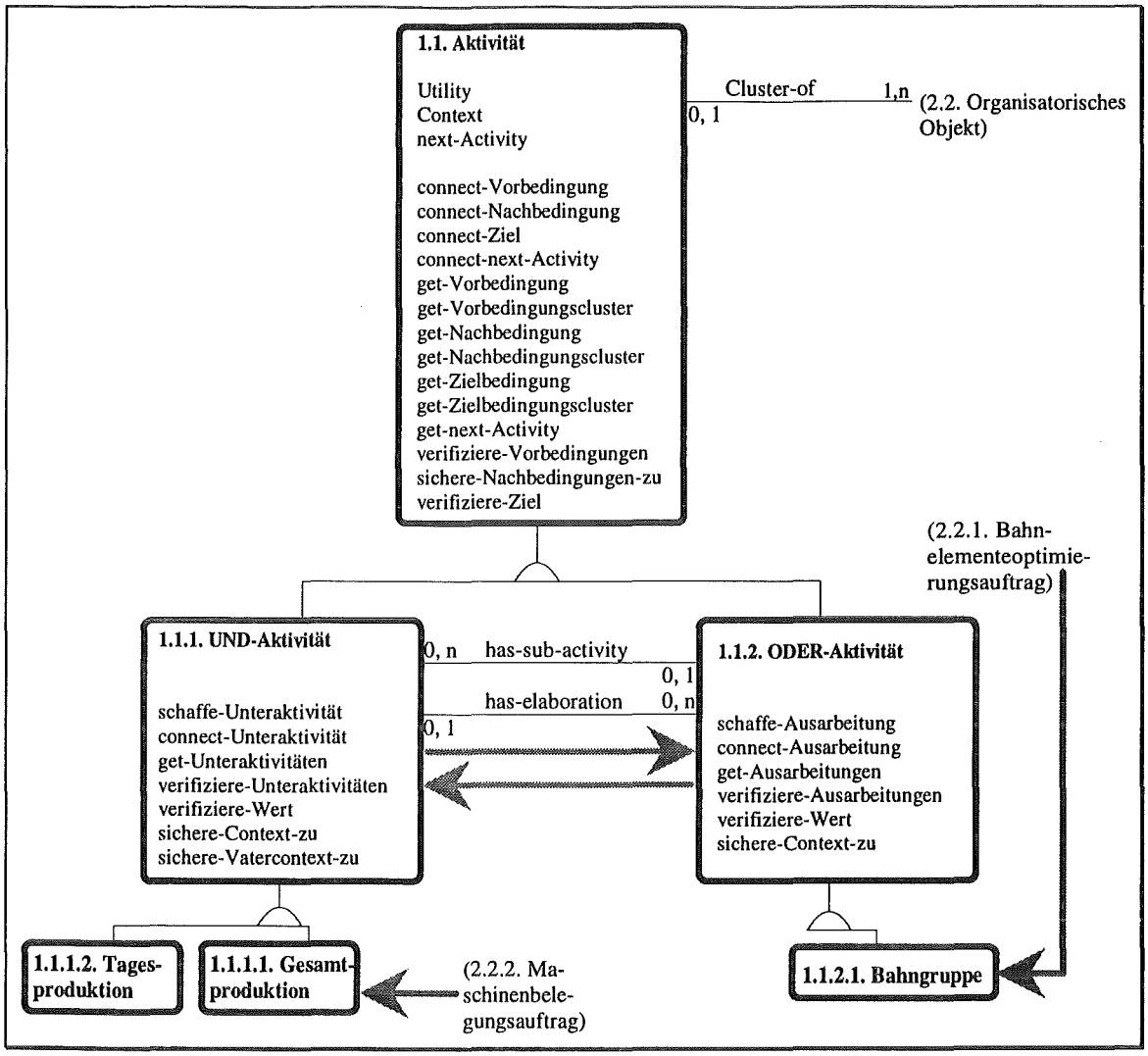


Abb. 7.7: Die Klasse der Aktivitäten und ihre internen Beziehungen. Die is-a Beziehung wird durch Halbkreise angedeutet.

Gemäß den Verifizierungsregeln von Kapitel 6 muß eine Aktivität ihre Vorbedingungen und ihr Ziel verifizieren und ihre Nachbedingungen zusichern können. *Verifiziere-Vorbedingungen* schickt eine Nachricht an das über *enabled-by* verbundene Objekt und ruft den dortigen Dienst *verifiziere-Wert* auf. Analog verfahren die Dienste *verifiziere-Ziel* und *sichere-Nachbedingungen-zu.*

Außerdem muß eine Aktivität ihre Verfeinerungen verifizieren können. UND-Aktivitäten verifizieren alle Unteraktivitäten, ODER-Aktivitäten mindestens eine Ausarbeitung. Das allgemeine Schema zur Verifizierung einer Aktivität hat folgende Gestalt:

```
verifiziere-Wert(A : Aktivität)
falls verifiziere-Wert der Vorbedingung erfolgreich
dann
  (1) ermittle, ob A aufgrund von Verfeinerungen
      und Ziel verifizierbar
  Falls (1) erfolgreich
  dann
    setze Wert von A auf wahr
    sichere-Nachbedingungen-zu
  sonst
    setze Wert von A auf falsch
```

Schritt (1) hängt von der Aktivitätsklasse ab. Bei UND-Aktivitäten sind sämtliche Unteraktivitäten zu verifizieren. Bestehen zwischen ihnen keine Interaktionen, ist das in wahlfreier Folge möglich. Andernfalls müssen Folgen mit Sortier-, Such- und Bewertungsverfahren erzeugt und verglichen werden. Je nach Typ muß ein ODER-Knoten mehr als nur eine Ausarbeitung verifizieren. Beim Bahnelementeproblem müssen zumindest alle Bearbeitungsbahnen in die Sequenz aufgenommen werden. Tabelle 7.3 gibt einen Überblick über gängige Verfahren.

Oberknotentyp	Problem	Suchverfahren
UND-Aktivität	Überprüfen, ob alle Unteraktivitäten verifizierbar sind. Zwischen ihnen wird Unabhängigkeit angenommen.	Verifiziere die Unteraktivitäten in beliebiger Reihenfolge. Dient als Default für UND-Aktivitäten.
UND-Aktivität	wie oben, aber zwischen den Unteraktiviäten können Abhängigkeiten bestehen.	Schreibe alle Unteraktivitäten in eine Liste L1. (*) Gehe L1 in beliebiger Reihenfolge durch. Ist ein Unterknoten nicht verifizierbar, dann schreibe ihn in eine Liste L2. Falls L1 gleich L2, dann gehe zu (**), sonst kopiere L2 nach L1 und lösche L2. Gehe zu Schritt (*) (**) Falls L2 nicht leer, dann ist die UND-Aktivität falsch.
ODER-Aktivität	Überprüfen, ob wenigstens eine Ausarbeitung verifizierbar ist.	Gehe alle Ausarbeitungen in beliebiger Reihenfolge durch, bis ein Knoten verifizierbar ist. Findet sich keiner, ist die ODER-Aktivität falsch. Das Verfahren dient als Default für ODER-Aktivitäten.
ODER-Aktivität	Überprüfen, ob genau/mindestens/höchstens n Ausarbeitungen verifizierbar sind	Gehe alle Ausarbeitungen in beliebiger Reihenfolge durch. Sind nicht genau/mindestens/höchstens n wahr, dann ist die ODER-Aktivität falsch. Anmerkung: Der wichtigste Spezialfall ist hier wohl n = 1, die X-ODER-Verknüpfung.
UND-Aktivität	Eine zulässige Sequenz finden, in der jede Unteraktivität genau einmal vorkommt	Depth-First-Suche: Bilde solange Sequenzen, bis eine gefunden wurde, bei der in der generierten Reihenfolge alle Aktivitäten propagiert werden können und dem Ziel genügen.
UND-Aktivität	Die beste Sequenz finden, in der jeder UnterAktivität genau einmal vorkommt	Depth-First-Suche: Bilde alle möglichen Sequenzen. Speichere alle zulässigen Sequenzen und vergleiche sie anhand einer Zielfunktion.
ODER-Aktivität	Die kürzeste zulässige Sequenz finden	Breadth-First-Suche: Beginne mit je einem Startelement. Erweitere jede Seqenz solange, bis das über has-goal spezifizierte Ziel erreicht wurde.

Tab. 7.3: Gängige Suchverfahren für UND- und ODER-Knoten

Die drei letzten Zeilen beschreiben enumerative Suchverfahren, die durch Branch-and-Bound o.ä. abgekürzt werden können. Ihre Komplexität ist exponentiell. In den letzten beiden Verfahren werden so gebildete Sequenzen miteinander verglichen.

7.3.2.1. Planung der Bahnelemente

Wie kann man den gefundenen Ansatz auf das Bahnelementeproblem umsetzen? Die Objektklassen 1.1.1.1. GESAMTPRODUKTION und 1.1.2.1. BAHNGRUPPE verifizieren ihre Ausarbeitungen mit speziellen Verfahren. Beide dienen ausschließlich als Startknoten für eine Propagation. GESAMTPRODUKTION besitzt als Unteraktivitäten eine Anzahl von Tagesproduktionen, die ihrerseits die Bearbeitungen eines Tages[3] zusammenfassen. Die BAHNGRUPPE faßt Luft- und Bearbeitungsbahnen zusammen.

Für das Bahnelementeproblem ist eine Sequenz gesucht, die alle Bearbeitungsbahnen und die notwendige Teilmenge der Luftbahnen enthält, deren gesamte Länge möglichst kurz sein soll. Die Bearbeitungsbahnen werden dazu nach dem verwendeten Werkzeug gruppiert. Jede Gruppe wird von einem Objekt BAHNGRUPPE zusammengefaßt und separat optimiert. Das ist sinnvoll, weil jede optimale Lösung die Werkzeugwechsel minimiert (vgl. [Lotter 92, S. 28]). Das Verfahren des sukzessiven Einfügens erzeugt je Bahngruppe eine Anfangslösung und verbessert sie mit einem 2- und einem 3-optimalen Verfahren, die z.B. von Domschke beschrieben werden [Domschke 85]. Die Optimierungshierarchie spiegelt sich in der Objekthierarchie wieder, wie Abbildung 7.8 zeigt.

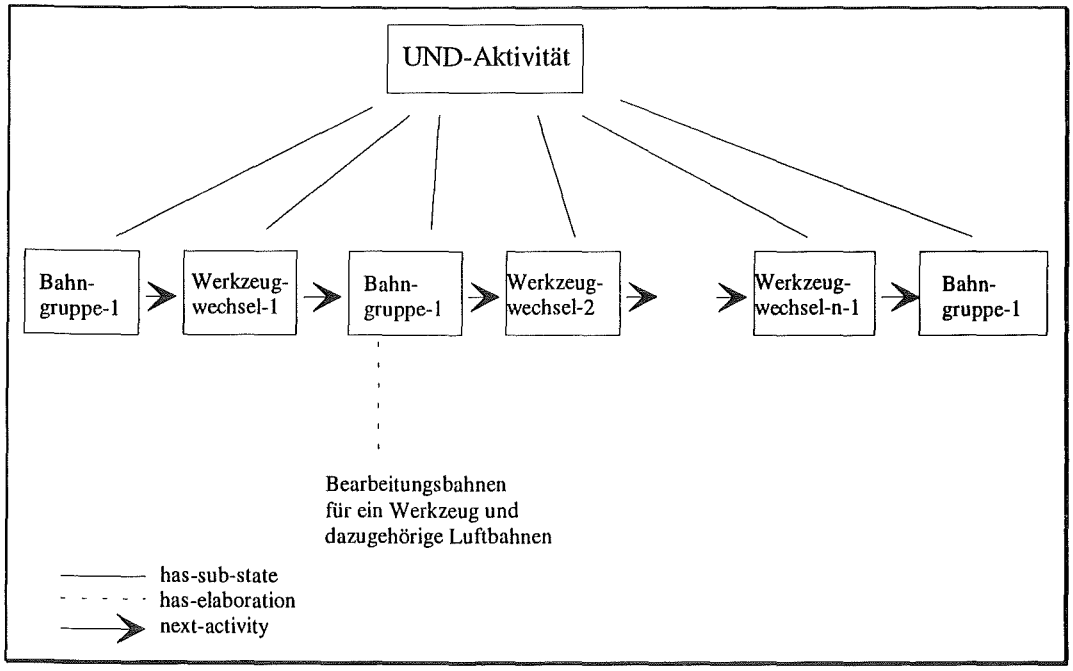


Abb. 7.8: Strukturierung des Bahnelementeproblems auf 3 Ebenen: Oberste Ebene ist eine UND-AKTIVITÄT, die die *Bahngruppen* konjunktiv verknüpft.

Pro verwendetes Werkzeug wird eine Instanz von 1.1.2.1. BAHNGRUPPE geschaffen. Sie hat als Ausarbeitungen (*has-elaboration*) Bearbeitungsbahnen für ein Werkzeug sowie alle Luftbahnen zwischen ihnen. Außerdem wird ihr mit *has-goal* ein Ziel zugeordnet. Es prüft, ob alle Bearbeitungsbahnen der Bahngruppe in die Sequenz S eingeplant wurden. S ist die Menge, die die Elemente der Lösungssequenz enthält. Zu Beginn ist S = ∅. Wurde eine Bearbeitungsbahn B verifiziert, erweitert man in der Nachbedingung die Sequenz S: S:= S ∪ B.

Gegeben sei eine Instanz BG von 1.1.2.1 BAHNGRUPPE und die ROBOTERHAND 2.1.1.6 mit einem Anwenderslot: *S* ist die Sequenz der eingeplanten Bearbeitungsbahnen und wird mit { } initialisiert. Außerdem sei ein Ziel Z_1 gegeben, das mit B über *has-goal* verknüpft ist. Z_1 ist ein Zustandsprädikat und besitzt ebenfalls den Anwenderslot

[3]Eine Tagesproduktion soll lediglich die Gesamtproduktion unterteilen. Ob dabei genau Tage herauskommen, ist unwesentlich.

S. Es gilt: ROBOTERHAND 2.1.1.6 *required-by* Z_1, d.h. Z_1 bezieht sich in seinen Abfragen auf den Anwenderslot *S* von ROBOTERHAND. Z_1 prüft, ob ROBOTERHAND.*S* bereits alle Bearbeitungsbahnen von B enthält. Etwas formaler kann man nun ein Bahngruppencluster definieren:

Definition 7.1: Ein Zustandsraumcluster $D_Z = [V, E, R]$ mit Startknoten s heißt genau dann *Bahngruppencluster* D_{BG}, wenn gilt:

1. s ist eine BAHNGRUPPE.

2. D_{BG} hat ein Ziel $Z_1 \in ZP \cap VORB$ und ROBOTERHAND *required-by* Z_1. Es hat den Anwenderslot *S* mit dem Prüfprädikat ROBOTERHAND.*S* = BB sowie dem Initialwert ROBOTERHAND.*S* $\cap$ BB = $\varnothing$.

3. V besteht aus Aktivitäten und Zuständen, die disjunktiv auf die folgenden Mengen aufgeteilt werden können:

3.1. Aktivitäten, die eine Menge Bearbeitungsbahnen BB darstellen, so daß für jede Bearbeitungsbahn $BB_i \in BB$ gilt: Es gibt genau eine Unterklasse W von WERKZEUG mit W *kann-bearbeiten* BB_i.

3.2. Aktivitäten, die eine Menge Luftbahnen LB_A darstellen, so daß für ein beliebiges Paar Bearbeitungsbahnen $BB_1, BB_2 \in BB$, $BB_1 \neq BB_2$ genau eine Luftbahn $LB_i \in LB_A$ existiert mit $LB_i.von = BB_1.nach$ und $LB_i.nach = BB_2.von$.

3.3. Aktivitäten, die eine Menge Luftbahnen LB_B darstellen, so daß für eine beliebige Bearbeitungsbahn $BB_1 \in BB$ und den Werkzeugwechselpunkt WW genau zwei Luftbahnen LB_i, $LB_k \in LB_B$ existieren mit $LB_i.von = BB_1.nach$ und $LB_i.nach = WW$ sowie $LB_k.von = WW$ und $LB_k.nach = BB_1.von$.

3.4. Nachbedingungen $NBB \subseteq NACHB \cap ZP$: Jede Bearbeitungsbahn $BB_i \in BB$ hat genau eine Nachbedingung $NBB_i \in NBB$ mit ROBOTERHAND *required-by* NBB_i und der Zusicherung ROBOTERHAND.*S* := ROBOTERHAND.*S* $\cup$ BB_i.

4. D_{BG} benutzt zur Sortierung seiner Ausarbeitungen die in BAHNELEMENTE-SORTIERUNG und BAHNELEMENTE-VERBESSERUNG spezifizierten Methoden.

Erläuterung: Bedingung 1 verlangt, daß die Bahnelemente von einer Unterklasse des Objekts BAHNGRUPPE zusammengefaßt werden. Das Ziel der Propagierung erklärt Bedingung 2: Eine Bahngruppe ist dann fertig geplant, wenn eine Lösungssequenz entstand, die alle Bearbeitungsbahnen der Bahngruppe enthält. Am Anfang ist die Lösungssequenz leer. Was für Bearbeitungsbahnen werden in einer Bahngruppe zusammengefaßt? Bedingung 3.1. verlangt, daß sie alle vom gleichen Werkzeug bearbeitet werden. Die Bearbeitungsbahnen werden in Bedingung 3.2. paarweise mit Luftbahnen verbunden. In Bedingung 3.3. erhält jede Bearbeitungsbahn eine Luftbahn zum Werkzeugwechselpunkt hin und zurück. Die Vorgehensweise schließlich spezifiziert Bedingung 4.

Im nächsten Schritt kann man nun das gesamte Lösungsverfahren formalisieren, indem man die Bahngruppen über eine UND-Aktivität zusammenfaßt:

Definition 7.2: Ein A/S-Cluster $D_S = [V, E, R]$ mit Startknoten s heißt genau dann *Bahnelementelösungscluster* D_{EL}, wenn gilt:

1. s ist eine UND-Aktivität und verifiziert seine Unteraktivitäten wahlfrei.

2. Für jede v ∈ V mit s *has-sub-activity* v gilt: entweder v *is-a* WERKZEUGWECHSEL oder v *is-a* BAHNGRUPPE. Ist n die Anzahl der Bahngruppen, dann ist (n-1) die Anzahl der Werkzeugwechsel.

3. Repräsentieren zwei Aktivitäten zwei Bearbeitungsbahnen BB_1, BB_2 aus nicht identischen Bahngruppen, dann gilt: es gibt kein WERKZEUG W mit W *kann-bearbeiten* BB_1 und W *kann-bearbeiten* BB_2.

Erläuterung: Bedingung 1 verlangt, daß alle Bahngruppen von einer UND-Aktivität zusammengefaßt werden, die ein sehr einfaches Suchverfahren verwendet. Die UND-Aktivität gewährleistet, daß keine Bahngruppe vergessen wird. Bahngruppen und Werkzeugwechsel werden in beliebiger Reihenfolge verifiziert. Das ist bei n Bahngruppen in höchstens 2n-1 Schritten getan. Die Unteraktivitäten dieser UND-Aktivität sind gemäß Bedingung 2 entweder Werkzeugwechsel oder Bahngruppen. Bedingung 3 spezifiziert, daß eine Bahngruppe alle Bearbeitungselemente, die das gleiche Werkzeug nutzen, enthalten muß. Ist das der Fall, reichen n-1 Werkzeugwechsel bei n Bahngruppen aus. Dieses Beispiel belegt, daß sich ein recht komplexes Reihenfolgeproblem mit dem in Kapitel 6 entwickelten Konzepten präzise und kurz formalisieren läßt.

Das Verfahren des sukzessiven Einfügens von Bahnelementen beginnt mit einer beliebigen Bearbeitungsbahn B. Es wählt die Luftbahnen vom Werkzeugwechselpunkt WW zum Start von B, LB_1, und vom Ende von B zurück zum Werkzeugwechselpunkt WW, LB_2. Man beginnt mit dem Weg Werkzeugwechsel, Luftbahn LB_1, Bearbeitungsbahn B und zurück über Luftbahn LB_2. Die erste Teillösung ist also die Sequenz LB_1 - B - LB_2, mit S = {B}. Die weiteren Bearbeitungsbahnen werden nun sukzessive eingefügt, bis das Ziel verifiziert werden kann, d.h. bis S alle Bearbeitungsbahnelemente der Gruppe enthält. Das Verfahren ersetzt beim Einfügen eine Luftbahn LB_i der jeweils letzten Teillösung TL durch ein Bahnelement und zwei Luftbahnen LB_j und LB_k[4]. Es wählt in TL das LB_i, für das gilt:

$LB_j.Dauer + LB_k.Dauer - LB_i.Dauer = \text{Minimum}$[5].

Verifiziere-Ausarbeitungen beschreibt das Verfahren.

```
Verifiziere-Ausarbeitungen(A1 : Bahngruppe)
H := aktueller Kontext
schaffe Unterkontexte U1 und U2 von H mit U1, U2 := H
gehe in Kontext U1
B1 := beliebige Bearbeitungsbahn von A1
LB1 := Luftbahn von WW nach B1.Startpunkt
LB2:= Luftbahn von B1.Endpunkt nach WW
verifiziere die Teillösung LB1-B1-LB2
Solange Ziel nicht erfüllt do
  kopiere U1 nach H
  kopiere U2 nach U1
  gehe in Kontext (1
  sukzessives-Einfügen(A1)
```

Kontexte (H, U1, U2) dienen zur Speicherung von Systemzuständen. Damit können Schlußfolgerungen zurückgezogen werden. *Sukzessives-Einfügen* wählt eine beliebige, noch nicht verifizierte Bearbeitungsbahn und bestimmt die Stelle, an der es bei minimalem Zuwachs eingefügt werden kann.

---

[4]Zwei Bearbeitungsbahnen sind stets durch eine Luftbahn getrennt. Andernfalls würden sie von COMETOS beim sog. Glätten zusammengefaßt.

[5]Die Notation von Objekten und Attributen ist an die Record-Notation in PASCAL angelehnt: $LB_j.Dauer$ heißt "Attribut *Dauer* des Objekts LB_j".

```
Sukzessives-Einfügen(A1 : Aktivität)
Zuwachs := ∞
Out-Luftbahn:= nil
B := beliebige Bearbeitungsbahn von A1, die in Kontext H
nicht verifiziert wurde
für jede Luftbahn L der Teillösung in Kontext H do
  L1(L,B) := suche-Luftbahn(L.Startpunkt, B.Startpunkt)
  L2(B,L) := suche-Luftbahn(B.Endpunkt, L.Endpunkt)
  falls ((L1.Dauer + L2.Dauer - L.Dauer) > hilf)
    dann
      Zuwachs := L1.Dauer + L2.Dauer - L.Dauer
      Out-Luftbahn := L
Ersetze L durch L1(L,B) - B - L2(B,L) und verifiziere die
neue Teillösung
```

7.3.2.2. Belegung von Maschinen

Die Ziele bei der Belegung von Maschinen sind konträr: Einerseits soll die Kapazität maximiert, andererseits sollen Fristüberschreitungen minimiert werden. In der Anwendungsumgebung von COMETOS sind Fristen nie nach Stunden, selten nach Tagen und häufiger nach Wochen vorgegeben[6]. In der Putzerei kann die Tagesproduktion in wahlfreier Folge abgearbeitet werden. Um Terminüberschreitungen zu vermeiden, muß man mittelfristig planen.

Die Auslastung wird besser, wenn man die Einlernoperationen geschickt über eine Tagesproduktion verteilt. Während das Putzeinlernen[7] beide Roboter in Anspruch nimmt, reicht für das Meßeinlernen[8] der Meßroboter allein aus. Der Putzroboter hat währenddessen nichts zu tun. Wurde ein Vorrat an Werkstücken vermessen, kann der Putzroboter ihn während des Meßeinlernens abarbeiten. Auf der anderen Seite dauert es länger, ein Werkstück automatisch zu putzen, als es automatisch zu vermessen. Würde pausenlos gemessen, entstünde vor dem Putzroboter eine stetig wachsende Warteschlange von Werkstücken.

Die hier entwickelte Heuristik verknüpft den Wunsch nach Minimierung der Fristen mit der Maximierung der Kapazität. Ein "Arbeitsplan" ist ein A/S-Cluster, das genau die Aktivitäten $a_i \in$ PROT enthält, die zur Bearbeitung eines einzigen, bestimmten Werkstücks notwendig sind. Ein A/S-Cluster heißt "Job", wenn es ein instanziierter Arbeitsplan ist. Ein UNTERAUFTRAG besteht demzufolge aus einem oder mehreren Jobs über den gleichen Werkstücktyp. Eine TAGESPRODUKTION ist eine Menge von Jobs mit ähnlichen Fristen, die innerhalb eines Zeitabschnitts, z.B. eines Tages, von der Roboterzelle bearbeitet werden kann. Die GESAMTPRODUKTION ist die Summe aller Tagesproduktionen.

Zur Minimierung der Verspätung sortiert man alle Jobs aufsteigend nach der Frist. Die sortierte Folge wird in Tagesproduktionen eingeteilt. Die Ordnung zwischen den Tagesproduktionen wird nicht verändert, aber Jobs innerhalb einer Tagesproduktion werden unter Kapazitätsgesichtspunkten umsortiert. Beispiel: Die Jobs A, B, C, D, E, F, G, H, I seien bereits nach der Frist sortiert. Eine Tagesproduktion umfasse genau 3 Jobs. Die Tagesproduktionen sind dann (A, B, C), (D, E, F), (G, H, I). Nun werden die Jobs jeder Tagesproduktion umsortiert zu (B, A, C), (F, E, D), (H, I, G). Die Reihenfolge der Einplanung lautet dann entsprechend B, A, C, F, E, D, H, I G.

[6] Auskunft der Gießerei Schubert & Salzer in Ingolstadt
[7] Teachen des Bearbeitungsprogramms
[8] Teachen des Meßprogramms

Wieviel ist eine Tagesproduktion? Bei 8 Stunden Arbeitszeit je Schicht und zwei Schichten täglich stehen bei zwei Robotern 32 Maschinenstunden zur Verfügung[9]. Eine Tagesproduktion kann z.B. 32 Stunden betragen.[10] Bei sehr engen Fristen wird man nur 8 Stunden ansetzen, bei starker Unterkapazität möglicherweise 80 Stunden. Je größer die Tagesproduktion, desto stärker wird unter Kapazitätsgesichtspunkten geplant, je kleiner sie ist, desto stärker kommen Fristen zum Tragen. Eine Tagesproduktion ist eine beliebig große Gruppe von Jobs, auch wenn man beim Begriff Tag sich eine Grenze denkt, z.B. maximal ein Arbeitstag.

Grundidee für die Kapazitätsmaximierung ist, ein Werkstück erst dann einzulernen, wenn genügend vermessene Werkstücke auf eine Bearbeitung durch den Putzroboter warten. Bei einer Zeit zum Meßeinlernen $t_{Meßeinlern}$ wartet man mit dem Einlernen, bis in der Warteschlange M soviele Werkstücke auf die automatische Putzbearbeitung t_{Putz} stehen, daß gilt:

$$(7.1)\quad t_{Meßeinlern} \approx \sum_{k \in M} t_{k,Putz}$$

Das ist unter zwei Voraussetzungen sinnvoll: 1. die Zeit zum automatischen Messen $t_{Meß}$ ist kleiner als die Zeit zum automatischen Putzen t_{Putz}, d.h. Abgangsrate(M) < Zugangsrate(M). 2. Weil sich vor dem Putzroboter eine Warteschlange M von Werkstücken bildet, müssen genügend Paletten zu ihrer Aufnahme bereitstehen.

Ein Verfahren zur Planung einer Tagesproduktion habe Jobs zur automatischen Bearbeitung J_{auto} und Jobs, die ein Meßeinlernen verlangen $J_{Meßeinlern}$. Es plant solange Jobs aus J_{auto} ein, bis Gleichung 7.1 erfüllt ist oder J_{auto} leer ist. Der nächste Job ist dann aus $J_{Meßeinlern}$. Nach seiner Einplanung werden die Mengen J_{auto} und $J_{Meßeinlern}$ erneut berechnet. Durch das Einlernen eines Werkstücks W ändern sich nämlich die Mengen $J_{Meßeinlern}$ und J_{auto}. Alle Jobs zur Bearbeitung von Werkstück W, die vorher in $J_{Meßeinlern}$ waren, gehören nun zu J_{auto}: ein Werkstücktyp muß nur einmal eingelernt werden.

Die GESAMTPRODUKTION (Objekt 1.1.1.1) hat als Unteraktivitäten Tagesproduktionen, die wiederum Jobs als Unteraktivitäten haben, die durch einen Auftrag und seine Unteraufträge (Objekte 2.2.1 und 2.2.3) erzeugt wurden. GESAMTPRODUKTION nutzt das Sortierverfahren von Objekt 2.3.1.1. *Tagesproduktion-Sortierung*, das nach folgendem Algorithmus sortiert:

```
Tagesproduktion-Sortierung(A : Gesamtproduktion)
Jobs := Unteraktivitäten der Tagesproduktionen von A
Folge_1 := Sortiere Jobs nach der Frist
Weise jeder Tagesproduktion ein Element aus Folge_1 zu.
```

Die Tagesproduktionen bleiben genau in der Reihenfolge, in der ihnen die Folgen zugewiesen wurden, also nach der Frist sortiert. Im nächsten Schritt plant jede TAGESPRODUKTION ihre Jobs gemäß Formel 7.1. ein. Das Verfahren dazu heißt *Job-Suche*. Es sortiert die Jobs nicht um, sondern sucht nach jedem eingeplanten Job den nächsten.

[9] Andere Ressourcen wie Bediener oder Spannstationen sind hier nicht berücksichtigt, weil ihr Einfluß auf die Optimierung vergleichsweise gering ist.

[10] Die COMETOS-Zelle ist sogar für zweieinhalb Schichten mit reduzierter Mannschaft in der dritten Schicht konzipiert.

```
Job-Suche(T : Tagesproduktion)
Jobmenge := Unteraktivitäten von T
solange Jobmenge ≠∅ do
  falls Jobmenge Einlern- und Automatikjobs enthält
  dann
    Meßroboter-fertig := berechne-Belegungszeit(Meßroboter)
    Putzroboter-fertig := berechne-Belegungs-
zeit(Putzroboter)
    falls (Meßroboter-fertig + 3 Stunden) > Putzroboter-
fertig
    dann
      plane einen Job mit Einlernen ein
    sonst
      plane einen Job ohne Einlernen ein
  sonst
    plane einen beliebigen Job ein
```

"Plane ein" bedeutet, daß der entsprechende Job verifiziert wird. "3 Stunden" ist die für das Meßeinlernen angesetzte Zeit. Bei Bedarf kann man hier einen Wert abfragen. Am Ende hat das Suchverfahren eine gesamte Tagesproduktion verifiziert und damit eingeplant.

7.3.2.3. Vorbedingungen

UND-Vorbedingungen und ODER-Vorbedingungen verifizieren ihre Verfeinerungen wie UND- bzw. ODER-Aktivitäten. Verfeinerungen innerhalb eines Zustandsclusters sind im allgemeinen voneinander unabhängig. Aufwendige Such- oder Sortierverfahren sind unnötig. Der Wert einer Vorbedingung wird wahr, wenn ihre Verfeinerungen konjunktiv bzw. disjunktiv verifiziert wurden.

Für Zustands- und Besitzprädikate der Vorbedingung wurden in Kapitel 6 zusätzliche Verifizierungsregeln ausgearbeitet. Zur Erinnerung: Ein ZUSTANDSPRÄDIKAT DER VORBEDINGUNG w ist wahr, wenn ein ZUSTANDSPRÄDIKAT DER NACHBEDINGUNG v existiert mit v *cause-enable* w und gilt: v ist wahr. Der Dienst *prüfe-cause-enable* nimmt diese Überprüfung vor.

Ein Zustandsprädikat w ist außerdem wahr, wenn ein Objekt der Nebenbedingungen v existiert mit v *required-by* w und w die von v spezifizierten Prädikate erfüllt. Ist v eine Objektklasse, muß diese Bedingung für ein Objekt v' aus v gelten. v' ist unter den Klassenmitgliedern von v mit der Methode *wähle-Objekt* zu bestimmen. *Wähle-Objekt* instanziiert die Objektklasse: v wird zu einer ODER-Aktivität v-inst, die Klassenmitglieder von v zu Ausarbeitungen von v-inst. v-inst kann nun verifiziert werden, eine Nebenbedingung wird also zum Gegenstand der Planung. Anschließend werden die verifizierten Klassenmitglieder von v in absteigender Reihenfolge der Nützlichkeit mit der Prozedur *prüfe-Objekt-Anwenderslots* überprüft. Die Methode prüft die Anwenderslots jedes Klassenmitglieds mit den im Zustandsprädikat w erklärten Prädikaten. Sie sind Lisp-Ausdrücke des Wissensingenieurs. Für die Komplexität der Prädikate und die Anzahl der Anwenderslots pro Zustandsprädikat gibt es keine Begrenzungen. Anwenderslots können beliebige Namen tragen[11]. Das erste Objekt v', das alle Prädikate erfüllt, ist das gewählte Objekt. Der folgende Algorithmus beschreibt die Prozedur *wähle-Objekt*.

[11] Schlüsselwörter von Lisp, KnowledgeCraft und Begriffe aus der Welt dieses Reihenfolgeplaners sind wegen der Gefahr von Ambiguitäten zu vermeiden.

```
wähle-Objekt(w : Zustandsprädikat der Vorbedingung)
v : Objekt der Nebenbedingung mit v required-by w
gewähltes-Objekt := nil
Falls v ohne Unterklassen
  und prüfe-Objekt-Anwenderslots(v) = wahr
  dann
    gewähltes-Objekt := v
  sonst
    Instanziiere v mit Unterklassen zu v-inst
    verifiziere v-inst
    kritische-Ausarbeitungen :=
      Liste der verifizierten Ausarbeitungen von
      v-inst
    sortiere kritische-Ausarbeitungen absteigend nach uti-
lity
    gehe Objekte von kritische-Ausarbeitungen durch bis
      Prüfe-Objekt-Anwenderslots(Objekt) = wahr
      gewähltes-Objekt := Objekt
```

Die Methode *Prüfe-Objekt-Anwenderslots* prüft für ein Zustandsprädikat w und ein prototypisches Objekt prot, ob die Bedingungen von w erfüllt sind. w und prot besitzen Anwenderslots gleichen Namens. Anwenderslot i von w enthält dazu ein Prädikat p(x) → {wahr, falsch}. x wird mit dem Wert von Anwenderslot i von prot gebunden. Beispiel: Es sei eine Roboterhand w von einem Zustandsprädikat v eines Bahnelements zu überprüfen. v und w besitzen einen Anwenderslot *Position*. Das Bahnelement geht von P1 nach P2. w soll prüfen, ob die Position der Roboterhand gleich P1 ist. Das Prädikat in *Position* von w lautet: *(equal x P1)*, wobei *x* mit dem Wert von *Position* der Roboterhand w gebunden wird.

Ein Zustandsprädikat verifiziert seinen Wert wie folgt:

```
verifiziere-Wert(w : Zustandsprädikat der Vorbedingung)
falls verifiziere-cause-enable
  dann Wert := wahr
  sonst
    falls o = wähle-Objekt und verifiziere-Ausarbeitungen
      dann
        w.belegt := o
        Wert := wahr
      sonst
        Wert := falsch
```

Ein BESITZPRÄDIKAT DER VORBEDINGUNG muß eine Ressource auswählen und belegen, um wahr zu sein. Ist nur eine Ressourcenklasse vorgegeben, bestimmt *wähle-Ressource* das geeignetste Klassenmitglied analog zu *wähle-Objekt* bei den Zustandsprädikaten. Für die Belegung wird die Ressource mit dem höchsten Wert in *utility* gewählt. Der Wert *utility* einer Ressource ist umgekehrt proportional zu ihrer bisherigen Auslastung. *Belege-Ressource* stellt eine possessed-by Beziehung her und ruft die Zeitpropagierung auf. Einen Überblick über Vorbedingungen gibt Abbildung 7.9.

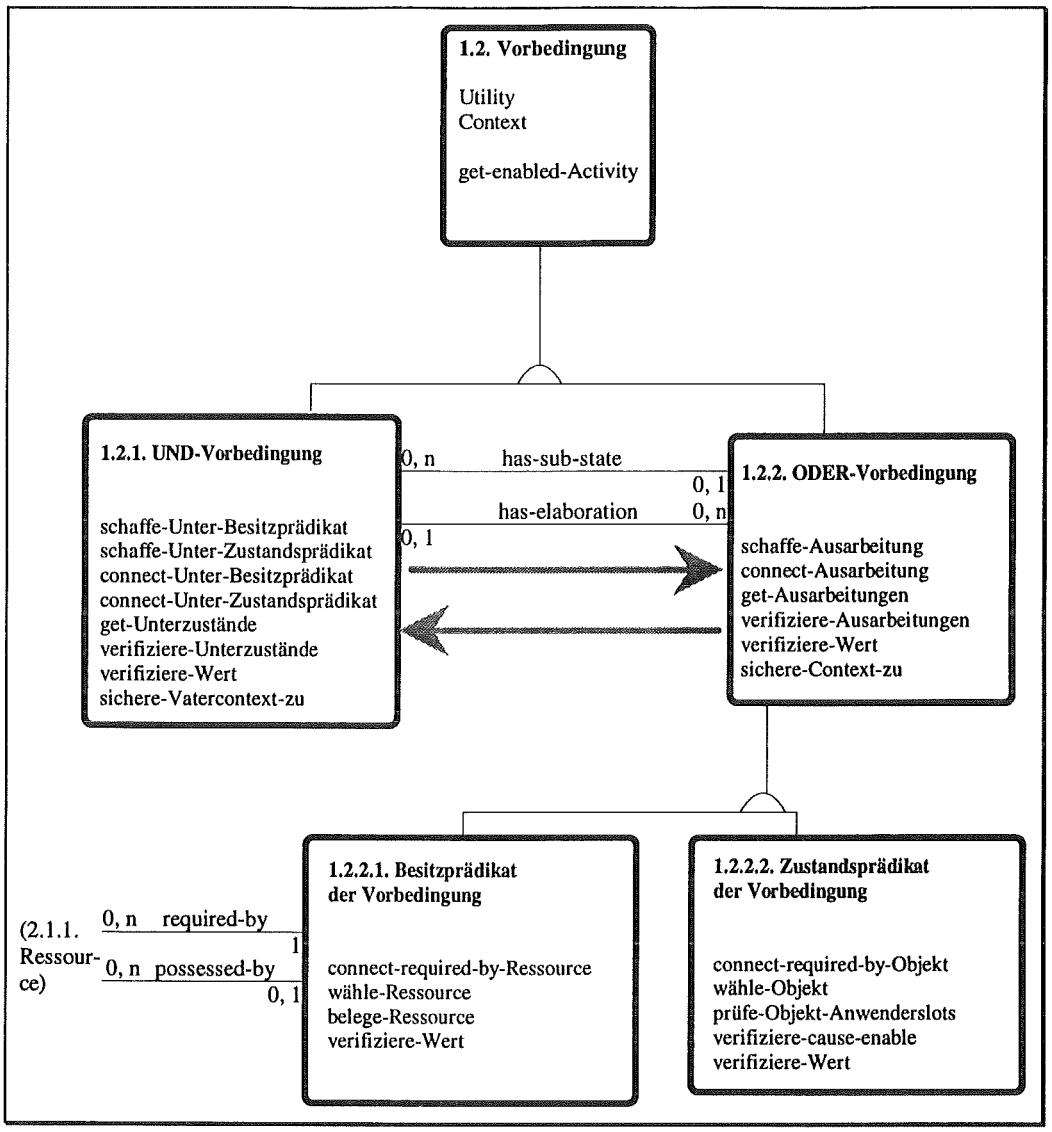


Abb. 7.9: Klasse der Vorbedingungen

7.3.2.4. Nachbedingungen

Vorbedingungen werden überprüft, Nachbedingungen zugesichert. Eine UND-Nachbedingung setzt dazu ihren eigenen Wert auf wahr und ruft die Zusicherung für alle ihre Unterzustände auf. Die Unterzustände sind Zustandsprädikate der Nachbedingung, die ein Objekt der Nebenbedingung revidieren.

Oft wird ein Objekt der Nebenbedingung revidiert, dessen Wert bereits in der Vorbedingung des gleichen Clusters abgefragt wurde. Das Einfügen einer Bearbeitungsbahn etwa revidiert die Position des Roboterarms. Im Beispiel von Abbildung 7.10 soll ZUSTANDSPRÄDIKAT-2 den FRÄSER hinsichtlich der *Reststandzeit* revidieren, indem es den Wert um 3 Minuten vermindert. *Bestimme-letzte-Revision* ermittelt die letzte Revision des Objekts der Nebenbedingung. Im Beispiel ist ZUSTANDSPRÄDIKAT-1 die letzte Revision. Es wird mit ZUSTANDSPRÄDIKAT-2 über *has-revision* verbunden. ZUSTANDSPRÄDIKAT-2 ist damit die neue letzte Revision.

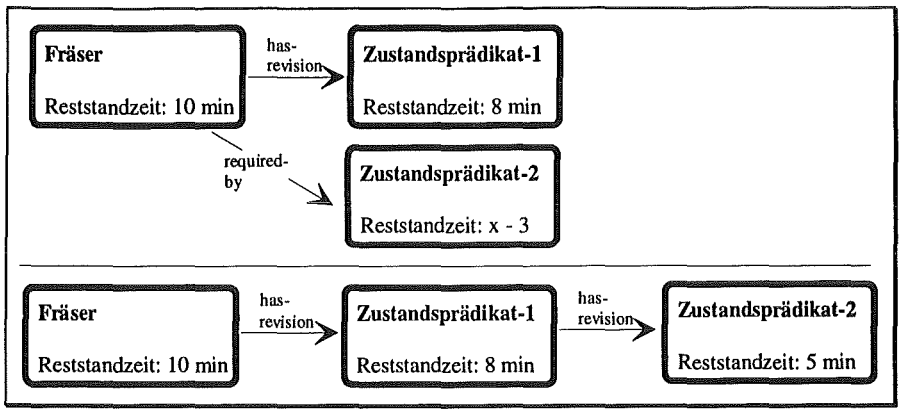


Abb. 7.10: Anfügen einer weiteren Revision an eine Revisionskette

7.3.3. Planungsobjekte

Das Objekt an der Wurzel der taxonomischen Hierarchie stellt allen Objekten vier Relationen und Dienste zur Verfügung. Die Relation *required-by* verbindet ein PLANUNGSOBJEKT mit Zustandsprädikaten der Vorbedingung (1.2.2.1), um eine zulässige Verwendung durch eine Aktivität darzustellen. Ein ZUSTANDSPRÄDIKAT DER NACHBEDINGUNG (1.3.2) revidiert über *has-revision* ein Planungsobjekt oder ein anderes Zustandsprädikat (vgl. Abbildung 7.11).

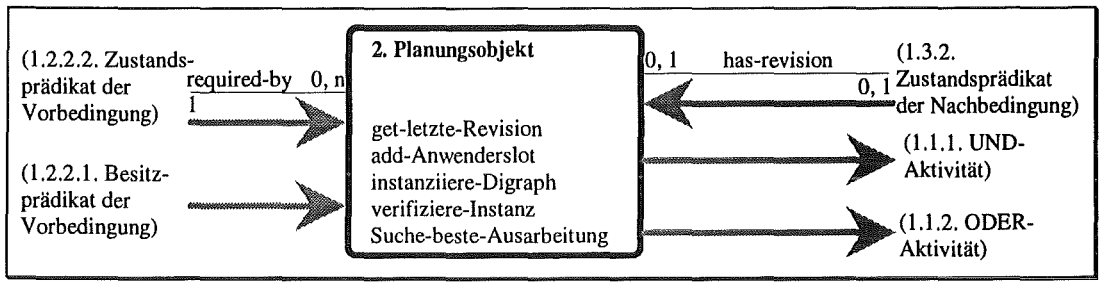


Abb. 7.11: Beziehungen des Planungsobjekts (ohne Verfeinerungen)

Mit *Add-Anwenderslot* schafft der Wissensingenieur einen Anwenderslot beliebigen Namens. Er wird dem Planungsobjekt hinzugefügt und steht für Überprüfungen durch Zustandsprädikate zur Verfügung. *Instanziiere-Digraph* instanziiert Objekte der Nebenbedingungen mit evtl. vorhandenen Vorbedingungen und Unterobjekten zu einem A/S-Cluster. *Verifiziere-Instanz* ruft für den Startknoten dieses Clusters die Methode *verifiziere-Wert* auf. *Suche-beste-Ausarbeitung* findet unter den verifizierten Söhnen des Startknotens den Knoten mit dem größten Wert im Attribut *utility*.

7.3.3.1. Physikalische Objekte

Physikalische Objekte gliedern die Fabrikumgebung in Ressourcen, Bahnelemente und Zeitfenster. Ressourcen können über einen Zeitraum belegt werden. Die Relation *required-by* verbindet eine Ressource mit Besitzprädikaten, um darzustellen, welche Aktivitäten die Ressource verwenden können. Wird die RESSOURCE tatsächlich belegt, dann wird zwischen ihr und dem Besitzprädikat die Relation *possessed-by* etabliert (vgl. Abbildung 7.12).

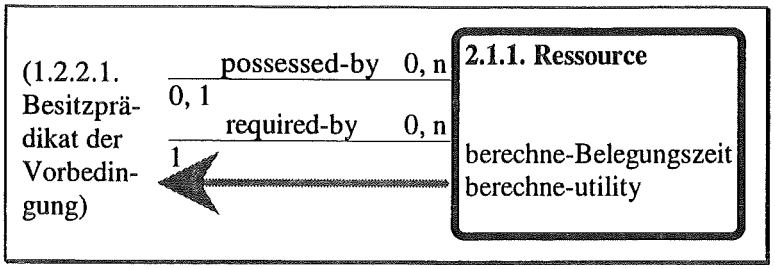


Abb. 7.12: Das Planungsobjekt *Ressource*

Die Methode *Berechne-Belegungszeit* ermittelt, ab wann eine Ressource keine Aufträge mehr hat. Das entspricht dem Wert Ende bzw. FE vom Zeitfenster des letzten Jobs auf der Ressource. Jedes Besitzprädikat, das über *possessed-by* mit der Ressource verbunden ist, gehört zu einer Aktivität, deren Ende über die Beziehung *hat-Zeitfenster* ermittelt wird[12]. Je früher eine Maschine leersteht, desto eher kommt sie für eine Belegung infrage. Die Priorität, ausgedrückt im Attribut *utility* berücksichtigt die Leerzeit und den vorgegebenen Wert nach der Formel Priorität = vorgegebener Wert in *utility* * Berechne-Belegungszeit^{-1}. Je länger eine Ressource ausgelastet ist, desto niedriger wird ihre Priorität. Damit lastet man die Ressourcen gleichmäßiger aus und gleicht die Kapazitäten ab.

Ressourcen werden für die COMETOS-Anwendung weiter unterteilt. Zu den Robotern gehört der PUTZ-, TRENN- und MEßROBOTER. Es gibt eine bestimmte Zahl PALETTEN. MENSCHEN haben drei Unterklassen. Der AUFSPANNBEDIENER spannt Werkstücke auf die Paletten und spannt sie wieder ab. Der EINLERNBEDIENER programmiert die Roboter. Der GUßPUTZER bearbeitet ungeplanten Putzaufwand, Innenputzarbeiten oder bearbeitet bei Bedarf ein Werkstück vollständig.

Ein WERKZEUG (2.1.1.4) hat einen Anwenderslot *Reststandzeit*, um den Verschleiß zu erfassen. Reststandzeit ist die Differenz zwischen Standzeit und Eingriffszeit. Es besitzt eine eigene Methode *berechne-utility*, daß die ererbte Methode überschreibt. Bei Ressourcen wie Maschinen oder Menschen möchte man die Belastung oft möglichst gleichmäßig auf alle verteilen. Hat man dagegen mehrere gleiche Werkzeuge, braucht man erst eins vollständig auf, bevor man ein neues einsetzt. *Berechne-utility* setzt den Attributwert für *utility* umgekehrt proportional zur Reststandzeit: Je weniger Standzeit ein Werkzeug hat, desto größer ist seine Chance, aufgebraucht zu werden. Unterklassen von Werkzeug sind der Fräsmeißel, der Fräser, die Trennscheibe, der Meißel und die dicke Schruppscheibe. Eine mögliche Verwendung durch Bearbeitungsbahnelemente stellt die Beziehung *kann-bearbeiten* dar.

Jedes WERKSTÜCK (2.1.1.5) besitzt Attribute zur Speicherung der Bearbeitungszeiten. Dazu zählen die Dauern zum Einlernen der Einlern-, Meß- und Trennroboter, die Dauern zum automatischen Messen, Trennen und Putzen, die Dauer zum manuellen Gußputzen und die Nachbearbeitungsdauer. Die Zeit zum Auf- und Abspannen ist bei allen Werkstücken annähernd gleich. Deshalb ist sie kein Attribut. Der Bearbeitungszustand eines Werkstücks ist ein Anwenderslot, der von Zustandsprädikaten abgefragt wird. Werkstücke sind die ACHSBRÜCKE und die DRUCKMASCHINENSEITENWAND.

BAHNELEMENTE besitzen Attribute für den *Start-* und *Endpunkt* und für die *Dauer*. Sie werden weiter unterteilt in LUFT- und BEARBEITUNGSBAHNEN. Bearbeitungsbahnen wird über die Beziehung *kann-bearbeiten* ein Werkzeug zugeordnet. Der Dienst *Schaffe-Bahnelemente-Cluster* generiert aus einem Bahnelement ein A/S-Cluster. Das Verfahren lautet für Bearbeitungsbahnelemente wie folgt:

[12]Bei nicht-zeitbehafteten Planungsproblemen erübrigt sich diese Methode.

```
Schaffe-Bahnelemente-Cluster (Bearbeitungsbahnelement BE)
A := neue Instanz von UND-Aktivität
VB := neue Instanz von "Zustandsprädikat der Vorbedingung"
mit dem Anwenderslot "Position" wobei
  VB.Position := BE.Startpunkt
NB := neue Instanz von "Zustandsprädikat der Nachbedingung"
mit dem Anwenderslot "Position" und "abgefahrene-Bahnen"
wobei
  NB.Position := BE.Endpunkt
  NB.abgefahrene-Bahnen := BE
  NB.Dauer := (+ x BE.Dauer)
Etabliere die folgenden Verbindungen
  A enabled-by VB
  A cause NB
  Roboterhand required-by VB
  Roboterhand required-by NB
```

Bei LUFTBAHNEN (2.1.2.2) generiert *Schaffe-Bahnelemente-Cluster* keinen Anwenderslot für "abgefahrene-Bahnen". Ein WERKZEUGWECHSEL (2.1.2.3) kann von jeder Position aus erfolgen. In seiner Vorbedingung wird auf die Abfrage der *Position* verzichtet.

ZEITFENSTER besitzen die bereits in Kapitel 6 beschriebenen Attribute für den frühesten Anfang *FA*, spätesten Anfang *SA*, das früheste Ende FE, späteste Ende *SE*, die minimale Dauer *Min_Dauer*, maximale Dauer *Max_Dauer*, den *Anfang*, das *Ende* und die *Dauer*. In einer Matrix stehen die Zeitfenster in der *linken Spalte* (Beziehung von einem Knoten weg) und in der *Kopfzeile* (Beziehung zu einem Knoten hin). Die Matrixelemente sind qualitative Zeitrelationen, dargestellt durch einen dreizehnstelligen Binärvektor. Objekten werden Zeitfenster über die Relation *hat-Zeitfenster* zugeordnet.

Der ROBOTERARM ist zur Optimierung der Bahnelemente wichtig. Er besitzt die Anwenderslots *Position*, *angeflanschtes-Werkzeug* und *abgefahrene-Bahnelemente*.

7.3.3.2. Organisatorische Objekte

Organisatorische Objekte dienen zur Vor- und Nachbereitung der Planung. Mit dem Startknoten eines A/S-Clusters können sie über die Beziehung *Operation-von* verbunden werden. Das A/S-Cluster ist dann ein Arbeitsplan. Ein BAHNELEMENTE-OPTIMIERUNGSAUFTRAG ist ein Auftrag zur Lösung des Bahnelementeoptimierungsproblems für ein Werkstück, dessen Bahnelemente ihm mit der Beziehung *hat-Bahnelemente* zugeordnet sind. Die Methoden dazu werden vom Dienst *plane-Bahnelemente* koordiniert: Zuerst ruft er die Methode *schaffe-Bahngruppe* auf. Sie generiert Instanz von Objekt 1.1.2.1 *Bahngruppe*, verbindet sie über eine UND-AKTIVITÄT und gibt deren Namen zurück. Die von 2. PLANUNGSOBJEKT geerbte Methode *Instanziiere-Digraph* erzeugt aus den Bahnelementen einen A/S-Digraphen auf der Ebene der Instanzen. Zuvor werden notwendige Zustände und ihre Verbindungen zu Werkzeugen erzeugt. Das Propagationsschema ruft zur Planung die Methode *verifiziere-Bahngruppe* auf, indem von der Instanz von Objekt 1.1.2.1. der Dienst *verifiziere-Wert* aufgerufen wird. Der Dienst *plane-Bahnelemente* ist im Folgenden dargestellt.

```
plane-Bahnelemente
Top := Schaffe-Bahngruppe
Für jedes zugeordnete Bahnelement do
  schaffe Bahnelemente-Cluster
Top-Instanz := instanziiere-Digraph(Top)
verifiziere Top-Instanz
```

Der Dienst *gib-Optimierungsresultate-aus* dient zur strukturierten Ausgabe der Ergebnisse auf der SUN, könnte aber auch als Schnittstelle zu anderen Programmen dienen. Abbildung 7.13 zeigt die Zusammenhänge zwischen organisatorischen Objekten.

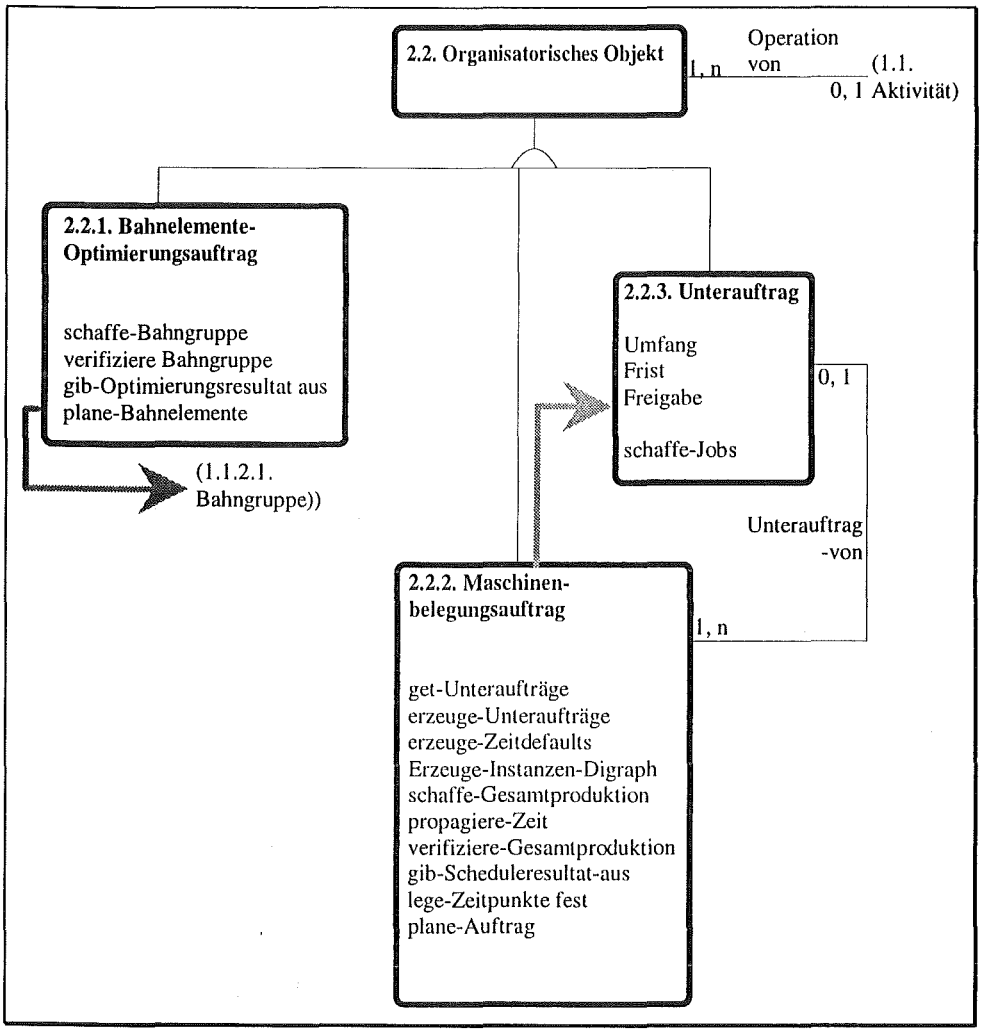


Abb. 7.13: Organisatorische Objekte

Ein MASCHINENBELEGUNGSAUFTRAG (2.2.2) besteht aus mehreren Unteraufträgen. Ein UNTERAUFTRAG bearbeitet ein oder mehrere Werkstücke gleichen Typs: Sollen z.B. Achsbrücken und Druckmaschinenseitenwände bearbeitet werden, sind zwei Unteraufträge notwendig. Aufträge sind mit Unteraufträgen über die Beziehung *hat-Unterauftag* verbunden. Unteraufträge schafft der Dienst *erzeuge-Unteraufträge*, der für jeden Unterauftrag dessen Methode *schaffe-Jobs* aufruft (Objekt 2.2.3). *Erzeuge-Instanzen-Digraph* instanziiert einen prototypischen Digraph, der als Arbeitsplan diente. *Erzeuge-Zeitdefault* fügt zwischen Zeitfenstern, zwischen denen keine qualitativen Zeitbeziehungen erklärt wurden, Defaultbeziehungen ein, wie in Kapitel 6.4 beschrieben. Der Dienst *Verifiziere-Gesamtproduktion* stößt die Planung mit dem Propagationsschema für eine Instanz von Objekt 1.1.1.1 an, indem sie deren Wert abfragt. Zur Zeitplanung stehen die

Verfahren *propagiere-Zeit* und *lege-Zeitpunkte-fest* zur Verfügung. Oft bleiben am Ende der Zeitplanung noch Zeitpunkte unbestimmt, d.h. der früheste und der späteste Anfangs- oder Endzeitpunkt sind ungleich. *Lege-Zeitpunkte-fest* legt nach einer Planungsstrategie Zeitpunkte fest, z.B. in der Reihenfolge der Tagesproduktionen. *Gib-Schedule-Resultate-aus* stellt die Planung strukturiert auf dem Bildschirm dar oder übergibt sie einer anderen Ausgabeprozedur. Die gesamte Planung wird von der Methode *plane-Auftrag* koordiniert:

```
plane-Auftrag
Operationen := erzeuge-Unteraufträge
Top-Auftrag := schaffe-Gesamtproduktion
Top-Auftrag.has-sub-activities := Operationen
erzeuge-Zeitdefaults(Top-Auftrag)
Top-Instanz := erzeuge-Instanzendigraph(Top-Auftrag)
verifiziere-Gesamtproduktion(Top-Instanz)
sortiere Unteraktivitäten von Top-Instanz nach gewähltem
Sortierverfahren
solange nicht alle Zeitpunkte festgelegt sind do
  lege-Zeitpunkte-fest(nächste Unteraktivität)
  propagiere-Zeit(Top-Instanz)
gib-Scheduleresultate-aus
```

UNTERAUFTRAG besitzt die Attribute *Umfang, Freigabe*[13] und *Frist.* Mit dem Arbeitsplan ist es durch die Beziehung *Unterauftrag-von* verbunden. Der Arbeitsplan eines Jobs wird vom Dienst *schaffe-Unterklassen* so oft instanziiert, wie im *Umfang* festlegt wurde.

7.3.3.3. Wissen

Wissen gliedert sich im Sinne der Bausteine von Heuristiken in Sortier-, Such- und Bewertungsverfahren. Implementierte SORTIERVERFAHREN sind die TAGESPRODUKTIONEN-SORTIERUNG, die BAHNELEMENTESORTIERUNG und die BAHNELEMENTE-VERBESSERUNG. Implementierte SUCHVERFAHREN sind die JOB-SUCHE, die ODER-SUCHE und die UND-SUCHE. Zur Bewertung der Optimierung von Bahnelementen dient die Zielfunktion.

7.4. Aufbau des Zeitplaners

Den Zeitplaner kann man auch separat benutzen. Ein aufrufendes Programm muß lediglich zur Dateneingabe einige Matrizen generieren. Er kann in einer C-Bibliothek verwendet werden, um andere Planer zu ersetzen oder zu ergänzen. Bei n Zeitfenstern oder Knoten ruft man ihn mit fünf Parametern auf:

1. eine 13 x n x n Matrix M1 für qualitative Zeitbeziehungen. Die erste Dimension vom Rang 13 bezieht sich auf die 13 Zeitrelationen. Ihre Reihenfolge von 0 bis 12 ist <, >, d, di, o, oi, m, mi, s, si, f, fi, =[14]. Die zweite Dimension ist Quelle, die dritte Senke der Zeitrelation. M1(0, 2, 3) bedeutet: Nullte Zeitrelation (*before*) von Knoten 2 nach Knoten 3. Zulässige Werte für die Matrixelemente sind 0 und 1 vom Typ int mit 1:= Beziehung gilt und 0:= Beziehung gilt nicht. Ist zwischen zwei Knoten keine Relation bekannt, wird ein 13-stelliger Vektor aus Einsen zugewiesen.
2. eine 13 x n x n Matrix M2 mit den reduzierten Zeitrelationen. Zu Anfang enthält sie normalerweise nur Einsen, weil die Reduktionen erst später gemacht werden. M2 ist genauso aufgebaut wie M1.

[13] Zeitpunkt, ab dem der Unterauftrag bearbeitet werden darf
[14] Die Symbole sind im Abkürzungsverzeichnis erklärt.

3. eine 9 x n Matrix mit den Daten der ZEITFENSTER: *FA, SA, FE, SE, Anfang, Ende, Minimale Dauer, Maximale Dauer, Dauer.* Die Matrixelemente sind vom C-Typ *double-float* und sollten positiv sein. Unbekannte Werte werden mit -1 markiert.
4. ein sogenannter Knotenvektor vom Rang n. Er markiert jeden Knoten mit 1, der bereits propagiert wurde. Das Netz muß dann lediglich um die unbekannten Knoten erweitert werden, Reduktionen müssen nicht zum zweiten Mal berechnet werden. Der Knotenvektor konserviert die Ergebnisse einer Propagation für den nächsten Aufruf.
5. ein *int*-Wert für die Dimension n.

Der Planer besteht aus zwei Teilen, die rückgekoppelt sind (vgl. Abbildung 7.14):

1. Der qualitative Planer berechnet und reduziert das Netzwerk. Anschließend ruft er den quantitativen Planer auf.
2. Der quantitative Planer bestimmt Zeitpunkte und Dauern. Wird eine bisher zulässige qualitative Relation unzulässig, wird erneut der qualitative Planer aufgerufen.

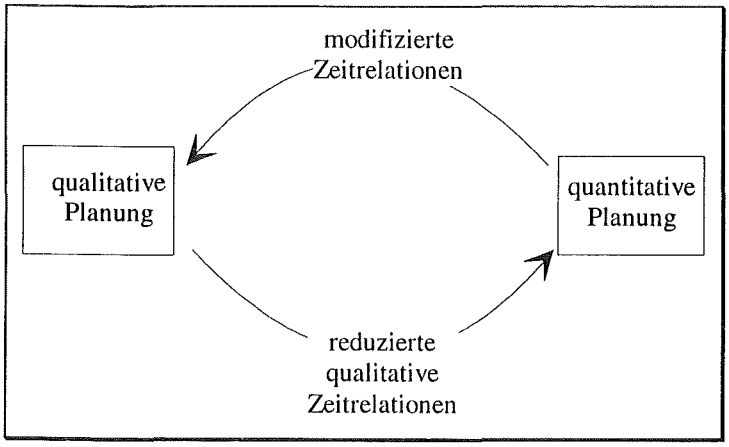


Abb. 7.14: Aufrufstruktur zwischen quantitativer und qualitativer Zeitplanung

Beispiel: Es gebe zwei Vorgänge A und B. Vorgang B endet spätestens um 10.00 h. Außerdem errechnet der qualitative Planer, daß Vorgang A zeitlich *vor oder nach* B[15] liegt. Die quantitative Planung errechnet: A beginnt um 11.00. Daraus folgt, daß B auf keinen Fall *nach* A beginnen darf. Die quantitative Planung reduziert das Spektrum der qualitativen Relationen zwischen den Vorgängen A und B, die betroffene Beziehung wird gelöscht und die qualitative Planung startet erneut.

Die qualitative Planung arbeitet mit dem reduzierten Netz. Damit wird Zeit gespart, weil weniger Relationen zu untersuchen sind.

7.4.1. Qualitative Zeitplanung

Der Allen'sche Ansatz wurde um zwei Verfahren zum Aufbau eines reduzierten Netzes erweitert. Es war notwendig[16], den Aufbau des reduzierten Netzes zu teilen, weil ein Teil der Reduktion erst am Schluß der Propagierung möglich ist, ein anderer dagegen parallel zur Propagation erfolgt.

Transitive Ketten vom Typ *vorher/nachher* und *enthält/während* können parallel zur Propagation (Prozedur *Transitivitäten folgern*) gebildet werden. Die Prozedur *Eindeutige-Transitivitäten-löschen* entdeckt und entfernt dabei entstehende Redundanzen. Die

[15]Eine Relation vorher oder nachher ist typisch für Vorgänge, die eine gemeinsame Ressource belegen. Sie dürfen sich niemals überschneiden. Sie belegen dann die Ressource entweder vorher oder nachher.

[16]vgl. die Ausführungen zur qualitativen Propagation in Kapitel 6.

Prozedur *Restliche-Transitivitäten-löschen* findet und löscht Redundanzen, die sich aus der Kreuzung transitiver Ketten ergeben. Abbildung 7.15 verdeutlicht die Aufrufstruktur.

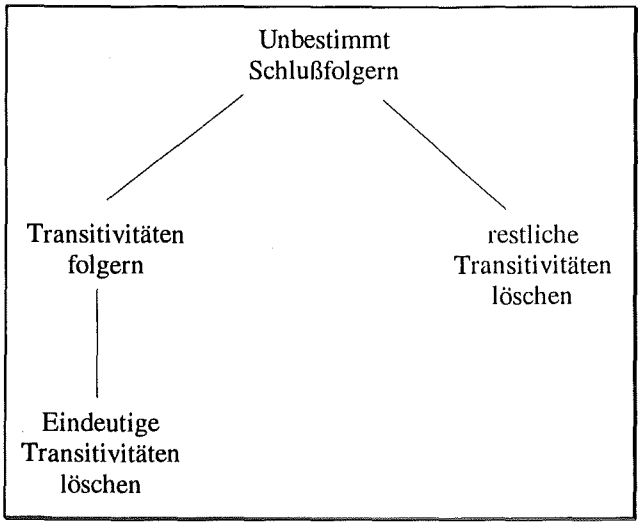


Abb. 7.15: Struktur der qualitativen Zeitplanung.

Eindeutige-Transitivitäten-löschen untersucht Beziehungen im Dreieck: Eine zu untersuchende Relation besteht zwischen zwei Knoten k_1 und k_2. Jeder der Knoten hat weitere Nachbarn, die herangezogen werden. Eine transitive Beziehung *tr* zwischen zwei Knoten k_1 und k_2 ist entweder vom Typ *vorher, nachher, während* oder *enthält*. Es gilt: k_1 *tr* k_2. k_1 steht auch zu anderen Knoten außer k_2 in Beziehung. Bei einem solchen Knoten k_i wird untersucht, ob er mit k_1 und k_2 in der transitiven Relation steht. Es muß dann gelten k_i *tr* $k_1 \wedge k_i$ *tr* k_2 . Man löscht dann die Relation k_i *tr* k_2. Analog verfährt man für k_2. Ist *tr* vom Typ *während* oder *enthält*, können weitere Relationen gelöscht werden.

```
EINDEUTIGE-TRANSITIVITÄTEN-LÖSCHEN(k1, k2)
tr: transitive Relation zwischen k1 und k2
für alle Nachbarn kn von k1 do
  falls (kn <> k2) und (kn tr k1) und (kn tr k2)
  dann
    lösche (kn tr k2).
Verfahre analog für die Nachbarn von k2.
```

Die restlichen Transitivitäten werden entdeckt, indem man für alle Anfangs- und Endknoten transitiver *vorher/nachher*-Ketten, k_A und k_E, nach gemeinsamen Knoten k_W sucht. k_W steht zu k_A und k_E in einer *während*-Relation. Falls es ein k_W gibt, werden alle Relationen zwischen ihm und den Knoten der transitiven *vorher/nachher* Kette mit Ausnahme von k_A und k_E gelöscht.

```
RESTLICHE-TRANSITIVITÄTEN-LÖSCHEN(kA, kE)
für alle Nachbarn kW von kA do
  falls (kA während kW) und (kE während kW)
  dann
    für jeden Knoten kT der transitiven Kette von kA nach
kE do
      falls (kT <> kA) und (kT <> kE)
      dann
        lösche (kT während kW)
```

Das so erhaltene Netz ist maximal reduziert und wird der quantitativen Planung übergeben.

7.4.2. Quantitative Zeitplanung

Die quantitative Planung propagiert feste Zeitpunkte und Dauern im reduzierten Zeitnetz. Nach der Initialisierung werden in einer Schleife über Dauern und Zeitpunkte Schlüsse gezogen und die Auswirkungen auf die qualitativen Beziehungen überprüft. Wird eine qualitive Beziehung unzulässig, ruft der Planer die qualitative Planung auf. Abbildung 7.16 verdeutlicht die Aufrufstruktur.

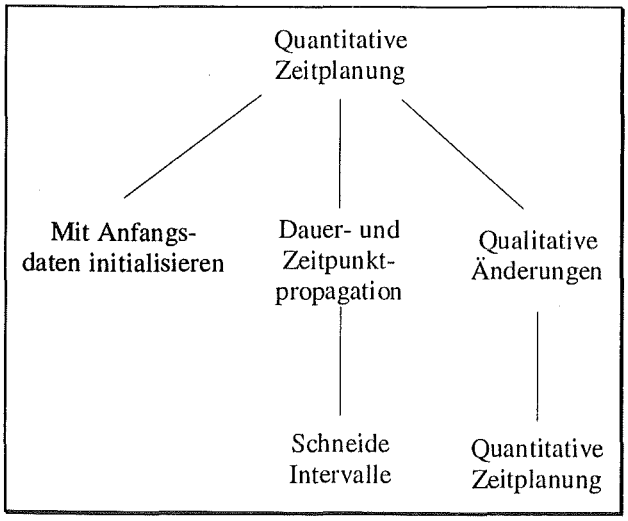


Abb. 7.16: Struktur der quantitativen Zeitplanung. In der Initialisierung wird vor allem die Netzkonsistenz mit den vorgegebenen Zeitdaten überprüft.

Die Prozeduren sind im Abschnitt über Verfahrensaufbau und Komplexität des Kapitels 6 beschrieben. Zeitpunkte und Dauern sind vom Datentyp *double-float*. Er akzeptiert so gut wie alle numerischen Werte und macht umständliches Transformieren an der Schnittstelle überflüssig. Einziger reservierter Wert ist -1 für unbekannte Werte. Ein Anwender sollte nach Möglichkeit auf die Verwendung negativer Zahlen verzichten.

7.4.3. Verhalten des Zeitplaners

Der Zeitplaner liefert stets eine nichtlineare Lösung. Sie kann beliebig ausgearbeitet werden. Insbesondere können Strategien wie kürzeste Operationszeit, maschinen- oder auftragsorientierte Planung realisiert werden. Dazu sortiert man die Operationen in die gewünschte Reihenfolge und beginnt, die noch unbestimmten Zeiten der Operationen sukzessive festzulegen. Um beispielsweise semiaktive Ablaufpläne[17] zu erzeugen, wählt

[17] In einem semiaktiven Ablaufplan kann keine Operation ohne Umorganisation der Reihenfolge zeitlich nach vorn geschoben werden.

man die frühesten Anfangs- und Endzeiten als Start- bzw. Endzeitpunkte. Dann stößt man die Zeitplanung erneut an, die die Konsequenzen der Festlegungen berechnet.

7.5. Zusammenfassung

Der Reihenfolgeplaner besteht aus drei Teilen: den Benutzerschnittstellen, dem Planer, der auf dem Propagationsschema aufbaut, und dem Zeitplaner. Zu den Anwendern zählen der Wissensingenieur und der Endanwender . Wissensingenieure erhalten einen erweiterten Netzwerkeditor und weitere Editoren, die durch Weiterentwicklung der KnowledgeCraft™ Workbench entstanden. Für Reihenfolgeprobleme gibt es keine allgemeine grafische Darstellung. Deswegen wurden zwei Benutzerschnittstellen entwikkelt: Die Benutzeroberfläche für die Bahnelementeoptimierung entstand eigens für den Reihenfolgeplaner. Sie stellt das Werkstück und die Bahnelemente grafisch dar und besitzt Funktionen, die man vom CAD her kennt. Die Oberfläche ist maus- und menüorientiert, die Bedienung leicht erlernbar. Zur Maschinenbelegungsplanung wurde ein kommerzielles Programm angepaßt, das auf Microsoft Windows™ aufbaut.

Ein Preprozessor wandelt COMETOS-Bahnelemente in Objekte für den Planer um. Er erzeugt die Luftbahnen unter Berücksichtigung der Orientierung der Roboterhand. Bei der Planung spielen nur noch Luftbahnzeiten und Werkzeugwechsel eine Rolle.

Die gesamte Wissensrepräsentation kann leicht erweitert werden. An bestehende Klassen werden einfach neue Unterklassen angefügt, die bei Bedarf eigene Attribute und Methoden erhalten. Um zu erklären, wie ein Objekt plant, ordnet man ihm die gewünschten Methoden zu. Sie sind in Wissensobjekten gespeichert. Der Wissensingenieur stellt ein Lösungsverfahren zusammen, indem er Such-, Sortier- und Bewertungsverfahren kombiniert. Der Anwender kann sog. Anwenderslots definieren und Werte sehr individuell abfragen.

Bahnelemente werden in drei Ebenen organisiert. Ein Startknoten faßt alle Bahngruppen eines Problems zusammen. Sie bestehen aus den Luft- und Bearbeitungsbahnen, die ein einzelnes Werkzeug abfährt. Jede Bahngruppe optimiert ihre Wege durch "Sukzessives Einfügen" sowie 2- und 3-optimale Verfahren. In der Maschinenbelegung gibt es einen Konflikt zwischen Fristen und der Kapazitätsmaximierung. Jobs werden deshalb zuerst nach der Frist sortiert und in Tagesproduktionen eingeteilt. Die Jobs jeder Tagesproduktion werden umsortiert, um die Kapazität zu maximieren: Beim Einlernen auf dem Meßroboter steht der Bearbeitungsroboter leer, wenn nicht vorher für einen Puffer Werkstücke vermessen wurden. Deswegen wartet man mit dem Einlernen, bis genügend Jobs vor dem Bearbeitungsroboter warten, die seine Leerzeit füllen. Planungsobjekte besitzen ergänzende Methoden. Sie instanziieren Prototypen, generieren aus Aufträgen einzelne Jobs, stoßen die Planung an und bereiten die Ergebnisse auf.

Der Zeitplaner ist auch separat als Teil einer C-Bibliothek nützlich. Wird er mehrmals für ein Problem aufgerufen, kann man die Ergebnisse der Netzwerkreduktion speichern und verkürzt so beim nächsten Mal den Netzwerkaufbau.

8. Ergebnisse

Der vorliegende Ansatz sucht kein schnelleres Verfahren. Er soll vielmehr helfen, eine große Bandbreite unterschiedlicher Reihenfolgeprobleme zu formalisieren und zu lösen. Zwei Probleme aus der COMETOS-Roboterzelle und ein Modellbeispiel einer Projektplanung werden zeigen, ob das gelungen ist. Wie weit das Ziel erreicht wurde, kann man an drei Kriterien messen: 1. Wie gut kann man mit dem Ansatz die Reihenfolgeprobleme und -lösungen darstellen? Als Gütemaßstab nennt Sathi Klarheit, Präzision und Vollständigkeit (vgl. Kapitel 4). 2. Wie effizient ist die Lösung? Ein bewährter Qualitätsmaßstab dazu ist die Komplexität. Zum Vergleich kann man eine Implementierung z.B. in Turbo-Pascal danebenhalten und fragen, ob der gleiche Planungsalgorithmus dort effizienter implementiert werden kann. Genauso relevant ist die Zeit für die Implementierung. 3. Hat die Lösung eine neue Qualität? Ein Maß dafür ist, ob das System inhaltlich Ergebnisse erzielt, die von anderen Systemen nicht erzielt werden können und wie relevant diese Ergebnisse sind.

8.1. Maschinenbelegungsplanung

Dem Beispiel zur Maschinenbelegungsplanung liegt die Anwendungsidee von COMETOS zugrunde (vgl. Kapitel 4). Ein Anwender beschreibt seine Aufgabe zunächst informell. Daraus entsteht ein formales Modell, anhand dessen die Arbeitsweise des Planers und die Ergebnisse gezeigt werden.

8.1.1. Informelle Beschreibung

Die COMETOS-Zelle bearbeitet Werkstücke aus zwei Gießereien. Die Firma Heidelberger Druckmaschinen schickte 44 Druckmaschinenseitenwände: 23 vom Typ A und 21 vom Typ B. Eine andere Achsbrücke vom Typ B wurde vor einiger Zeit bereits eingelernt und ist dem System bekannt. Die Gießerei Schubert & Salzer hat 65 Achsbrücken der Typen C, D und E geschickt: 22 vom Typ C, 17 vom Typ D und 26 vom Typ E. Typ E ist ebenfalls dem System bekannt.

Arbeitsbeginn ist der 1. Juni, 8:00 h. Die Werkstücke der Firma Schubert & Salzer müssen bis zum Abend des 4. Juni, die der Firma Heidelberger Druckmaschinen bis zum Abend des 5. Juni bearbeitet sein.

Die COMETOS-Roboterzelle besteht aus drei Robotern: einem Trennroboter zum Abtrennen der Angüsse und Speiser, einem Meß- und einem Putzroboter. Die Werkstücke können auf insgesamt 40 Paletten aufgespannt werden.

Vier Personen arbeiten in und um die Roboterzelle: Ein Spannbediener arbeitet an der Spannstation, ein Einlernbediener lernt den Roboter ein. Zwei Gußputzer können bei Bedarf Werkstücke putzen, sind aber sonst mit anderen Arbeiten wie Innenputzen beschäftigt. Der Einlernbediener kann wegen eines Arztbesuchs am 1. Juni erst ab 11.00 h arbeiten.

Ein Werkstück kann manuell vom Gußputzer oder automatisch von der Roboterzelle bearbeitet werden. Die Roboterzelle soll den Hauptteil der Arbeitslast bewältigen. Der Gußputzer arbeitet nur, wenn Engpässe auftreten. Die automatische Bearbeitung besteht aus sechs Schritten: Aufspannen, Messen für die Trennoperation, Trennen, Messen für die Putzoperation, Putzen, Abspannen. Druckmaschinenseitenwände werden jedoch nicht getrennt. Das gleiche gilt für drei Achsbrücken vom Typ C und eine Achsbrücke vom Typ D, wo die Steiger und Angüsse abgefallen sind. Ist ein Werkstück dem System unbekannt, muß der Einlernbediener das automatische Trennen, Messen und Putzen einlernen. Zum Einlernen der Putzbearbeitung, "Putzeinlernen", braucht er den Putz- und den Meßroboter gleichzeitig. Der Meßroboter wird für das Einlernen des Meßroboters zum Trennen und für das Einlernen des Meßroboters zum Putzen benötigt. Für das

Einlernen des Trennprogramms reicht der Trennroboter. Zum automatischen Messen für das Trennen und Putzen, zum Trennen und Putzen wird der jeweilige Roboter benötigt. Der Spannbediener an der Spannstation besorgt das Auf- und Abspannen. Die Putzerei setzt die Zeiten gemäß Tabelle 8.1 an (in Minuten):

	Seitenwand A	**Seitenwand B**	**Achsbrücke C**	**Achsbrücke D**	**Achsbrücke E**
Auf-/Abspannen	1	1	1	1	1
Einlernen des Trenn-roboters			9	10	8
Einlernen des Meß-roboters für Trennen			20	20	20
Messen für Trennen			1	1	1
Trennen			1	1	1
Einlernen des Meß-roboters für Putzen	150	180	140	180	140
Messen für Putzen	4	6	10	8	12
Einlernen des Putz-roboters	100	130	100	130	100
Putzen	4	4	19	16	22
Manuell Putzen	6	6	50	60	55

Tab. 8.1: Für das Beispiel angenommene Operationszeiten

Die Kapazität der Roboterzelle soll maximiert werden, möglichst ohne Fristen zu verletzen. Genau bestimmt sind nur die Zeiten der Werkstücke B und E, weil sie dem System bekannt sind. Alle anderen Zeiten sind Schätzzeiten.

8.1.2. Modellierung des Problems

Am einfachsten formalisiert man das Problem, wenn man die Aufgabenstellung Schritt für Schritt durchgeht. Als Faustregel gilt: Jedes Substantiv ist Kandidat für ein Planungsobjekt, jedes Adjektiv ist Kandidat für ein Attribut des Planungsobjekts, auf das ein Zustandsprädikat zugreifen kann, jedes Verb ist Kandidat für eine Aktivität.

8.1.2.1. Modellierung der Planungsobjekte

Zur Modellierung greift man auf vorhandene Klassen (vgl. Kapitel 7) zurück:

- "Die COMETOS-Roboterzelle besteht aus drei Robotern: einem Trennroboter zum Abtrennen der Angüsse und Speiser, einem Meß- und einem Putzroboter." Die Putz-, Meß- und Trennroboter der Hierarchie von PLANUNGSOBJEKT (Objekte 2.1.1.1 - 2.1.1.3) werden dafür unverändert verwendet.
- "Ein Spannbediener arbeitet an der Spannstation, ein Einlernbediener lernt den Roboter ein. Zwei Gußputzer können bei Bedarf Werkstücke putzen, sind aber sonst mit anderen Arbeiten wie Innenputzen beschäftigt." Menschen sind bereits als Ob-

jekte vorhanden (vgl. Kapitel 7). Von GUßPUTZER schafft man zwei Unterklassen, GUßPUTZER-1 und GUßPUTZER-2. "Der qualifizierte Arbeiter ist am 1. Juni bis 11.00 h verhindert. Um das darzustellen, schafft man ein Zeitfenster Z1 mit Anfang := 8.00 h, Ende = 11.00 h und verbindet es: QUALIFIZIERTER ARBEITER *hat-Zeitfenster* Z1

- "Die Werkstücke können auf insgesamt 40 Paletten aufgespannt werden." 40 Unterklassen von PALETTE generiert man mit Dienst *schaffe-Unterklasse*
- Werkstücke: Mit dem Dienst *schaffe-Unterklasse* werden drei Unterklassen von Objekt 2.1.1.5.1 ACHSBRÜCKE und zwei von Objekt 2.1.1.5.2 HDM-SEITENWAND geschaffen. Sie heißen SEITENWAND-A, SEITENWAND-B, ACHSBRÜCKE-C, ACHSBRÜCKE-D, ACHSBRÜCKE-E. Von ihnen schafft man weitere Unterklassen: 23 von SEITENWAND-A, 21 von SEITENWAND-B, 22 von ACHSBRÜCKE-C, 17 von ACHSBRÜCKE-D, 26 von ACHSBRÜCKE-E.
- "Die COMETOS-Zelle bearbeitet Werkstücke aus zwei Gießereien. (...)" Aufträge werden mit *schaffe-Unterklasse* von Objekt 2.2.2 MASCHINENBELEGUNGSAUFTRAG erzeugt: einer für AUFTRAG-1 von Schubert & Salzer und AUFTRAG-2 von Heidelberger Druckmaschinen. AUFTRAG-1 hat als Unteraufträge AUFTRAG-A, AUFTRAG-B, mit den Werten für *Umfang* 23 bzw. 21 und der *Frist* 3. Juni. AUFTRAG-2 hat die Unteraufträge AUFTRAG-C, AUFTRAG-D, AUFTRAG-E mit den Werten für *Umfang* 22, 17 bzw. 26 und der *Frist* 4. Juni.
- "Die Kapazität der Roboterzelle soll maximiert werden, möglichst ohne Fristen zu verletzen." Als Sortierverfahren für Objekt 1.1.1.1 GESAMTPRODUKTION ist die Methode von Objekt 2.3.1.1. TAGESPRODUKTIONEN-SORTIERUNG geeignet. Das Suchverfahren 2.3.2.3. JOB-SUCHE wählt für eine TAGESPRODUKTION den nächsten Job aus. Alle anderen Objekte verwenden ihre Standardsuch- und -sortierverfahren (vgl. Kapitel 7).

Um einem Werkstück Attribute zuzuweisen, orientiert man sich an Eigenschaften, die für die Bearbeitung interessant sind. In diesem Beispiel gibt es Werkstücke, die nicht getrennt werden. "Getrennt" ist daher eine wesentliche Eigenschaft. Die gewählten Anwenderslots sind:

- *Trennprogramm-eingelernt*
- *Meßprogramm-T-eingelernt*
- *Meßprogramm-P-eingelernt*
- *Putzprogramm-eingelernt*
- *aufgespannt*
- *getrennt*

Standardwert jedes Attributs ist *falsch.*

8.1.2.2. Modellierung der Arbeitspläne

Zur Notation: Jedem Objekt ist in den folgenden Tabellen eine eigene Zeile gewidmet. Die Spalte "Beziehung" nennt die Relationen zu anderen Objekten. Beispiel: Darzustellen ist die Beziehung MY-OBJECT *enabled-by* MY-CONDITION in der Zeile von MY-OBJECT. In der Spalte Beziehung steht dann: *enabled-by* MY-CONDITION. In der Zeile von MY-CONDITION kann analog stehen: MY-OBJECT *enabled-by*. Das Objekt der jeweiligen Zeile wird nicht wiederholt.

Arbeitspläne sind A/S-Cluster, deren Startknoten mit einem organisatorischen Objekt wie 2.2.3 UNTERAUFTRAG über die Beziehung *Operation-von* verbunden sind (vgl. Kapitel 7).

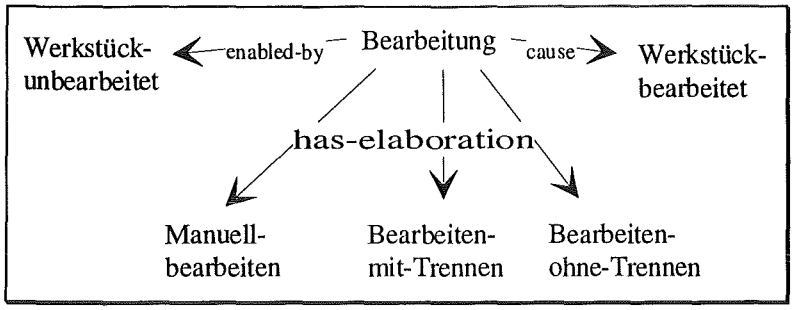


Abb. 8.1: Direkte Nachfolger der Aktivität BEARBEITUNG

"Ein Werkstück kann manuell vom Gußputzer oder automatisch von der Roboterzelle bearbeitet werden." "Druckmaschinenseitenwände werden jedoch nicht getrennt. Das gleiche gilt für drei Achsbrücken vom Typ B und eine Achsbrücke vom Typ C, wo die Steiger und Angüsse abgefallen sind." Die Aktivität BEARBEITUNG faßt zwei Bearbeitungsformen , die manuelle und die automatische zusammen. TRENNEN ist ein Sonderfall, weil nicht alle Werkstücke getrennt werden müssen. Deswegen wird BEARBEITUNG dreifach ausgearbeitet: MANUELL-BEARBEITEN, BEARBEITEN-MIT-TRENNEN und BEARBEITEN-OHNE-TRENNEN (vgl. Abbildung 8.1). Jede Bearbeitung braucht ein unbearbeitetes Werkstück. Ein Zustandsprädikat in der Vorbedingung sucht es heraus. In der Nachbedingung sichert man seine Bearbeitung zu, wie Tabelle 8.2 darstellt. Der Anwenderslot *bearbeitet* des ausgewählten Werkstücks wird mit dem Wert *wahr* revidiert.

Knotenname	**Beschreibung**	**ist ein (is-a)**	**Beziehungen**	**Sloteinträge**
BEARBEITUNG	Startknoten des Clusters Es gibt drei Wege für eine Bearbeitung: entweder MANUELL-BEARBEITEN, BEARBEITEN-MIT-TRENNEN oder BEARBEITEN-OHNE-TRENNEN; läßt mit der Vorbedingung WERKSTÜCK-UNBEARBEITET prüfen, ob es noch ein unbearbeitetes Werkstück gibt. Sichert in seiner Nachbedingung WERKSTÜCK-BEARBEITET die Fertigstellung des Werkstücks zu	ODER-AKTIVITÄT	*has-elaboration* MANUELL-BEARBEITEN *has-elaboration* BEARBEITEN-MIT-TRENNEN *has-elaboration* BEARBEITEN-OHNE-TRENNEN *enabled-by* WERKSTÜCK-UNBEARBEITET *cause* WERKSTÜCK-BEARBEITET	
WERKSTÜCK-UNBEARBEITET	Prüft, ob noch ein unbearbeitetes Werkstück W existiert. Falls ja, wird es ausgewählt.	ZUSTANDSPRÄDIKAT DER VORBEDINGUNG	W *required-by*	*bearbeitet*: falsch
WERKSTÜCK-BEARBEITET	Sichert für das Werkstück W die Bearbeitung zu: Revision des Werkstücks	ZUSTANDSPRÄDIKAT DER NACHBEDINGUNG	W *required-by*	W.*bearbeitet*:= wahr

Tab. 8.2: Die Aktivität Bearbeiten mit ihrer Vor- und Nachbedingung

Für die Aktivität MANUELL-BEARBEITEN braucht man einen Gußputzer, der die Arbeit macht. Er wird mithilfe eines Besitzprädikats für die Dauer der Bearbeitung belegt, wie Tabelle 8.3 zeigt.

MANUELL-BEAR-BEITEN	Ein Gußputzer bearbeitet das Werkstück manuell Ein Gußputzer muß für den Bearbeitungszeitraum zur Verfügung stehen. Die Vorbedingung Gußputzer-belegt reserviert ihn.	UND-AK-TIVITÄT	*enabled-by* Gußputzer-belegt	
GUßPUTZER -BELEGT	prüft, ob ein Gußputzer G zur Verfügung steht und reserviert ihn.	BESITZ-PRÄDIKAT DER VOR-BEDIN-GUNG	G *required-by*	

Tab. 8.3: Das Cluster MANUELL-BEARBEITEN

"Die automatische Bearbeitung besteht aus sechs Schritten: Aufspannen, Messen für die Trennoperation, Trennen, Messen für die Putzoperation, Putzen, Abspannen." BEARBEITEN-MIT-TRENNEN umfaßt die Aktivitäten AUFSPANNEN, MESSEN-T, TRENNEN, MESSEN-P, PUTZEN und ABSPANNEN. Für die Dauer der automatischen Bearbeitung liegt das Werkstück auf einer Palette, die in der Vorbedingung belegt wird. Die Struktur von BEARBEITEN-MIT-TRENNEN zeigt Abbildung 8.2.

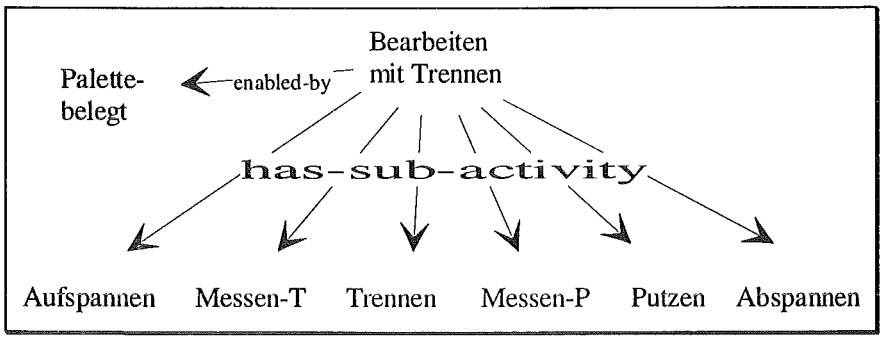


Abb. 8.2: Die Aktivität BEARBEITEN-MIT-TRENNEN mit Vorbedingung und Unteraktivitäten

Zwischen den Unteraktivitäten ist noch keine Reihenfolge festgelegt. Sie ist vielmehr das Ergebnis von Kausalitäten zwischen ihnen. "Die Roboterzelle soll den Hauptteil der Arbeitslast bewältigen. Der Gußputzer arbeitet nur, wenn Engpässe auftreten." Der Automatikbetrieb soll also Normalfall sein, die manuelle Bearbeitung die Ausnahme. Falls kein Trennen erforderlich ist, soll darauf verzichtet werden. Deswegen erhält BEARBEITEN-MIT-TRENNEN im Slot *utility* einen höheren Wert als den Default (= 1). Damit ist eine Rangfolge vorgegeben (vgl. Tabelle 8.4).

BEARBEITEN-MIT-TRENNEN	Die COMETOS-Zelle bearbeitet das Werkstück automatisch. Für den gesamten Zeitraum wird eine Palette gebraucht. Die Vorbedingung PALETTE-BELEGT reserviert sie. Die Aktivität besteht aus fünf Teilschritten: AUFSPANNEN, TRENNEN, MESSEN-T, MESSEN-P, PUTZEN, ABSPANNEN.	UND-AKTIVITÄT	*enabled-by* PALETTE-BELEGT *has-sub-activity* AUFSPANNEN *has-sub-activity* TRENNEN *has-sub-activity* MESSEN-T *has-sub-activity* MESSEN-P *has-sub-activity* PUTZEN *has-sub-activity* ABSPANNEN	*utility* := 2
PALETTE-BELEGT	prüft, ob eine Palette P zur Verfügung steht und reserviert sie.	BESITZ-PRÄDIKAT DER VORBEDINGUNG	P *required-by*	
BEARBEITEN-OHNE-TRENNEN	Die COMETOS-Zelle bearbeitet das Werkstück automatisch und ohne Trennen. Die Vorbedingungen faßt BoT-Vorbedingung zusammen. Die Aktivität besteht aus vier Teilschritten: Aufspannen, Messen für Putzen, Putzen, Abspannen.	UND-AKTIVITÄT	*enabled-by* BOT-VORBEDINGUNG *has-sub-activity* AUFSPANNEN *has-sub-activity* MESSEN-P *has-sub-activity* PUTZEN *has-sub-activity* ABSPANNEN	*utility* := 3
BOT-VORBEDINGUNG	Für den gesamten Zeitraum wird eine Palette gebraucht. Die Vorbedingung Palette-belegt reserviert sie. Außerdem darf das Werkstück kein Trennen benötigen.	ODER-VORBEDINGUNG	*has-sub-state* PALETTE-BELEGT *has-sub-state* TRENNEN-UNNÖTIG	
TRENNEN-UNNÖTIG	prüft, ob Werkstück W schon getrennt ist.	ZUSTANDS-PRÄDIKAT DER VORBEDINGUNG	W *required-by*	W.*getrennt* : t

Tab. 8.4: BEARBEITEN-MIT-TRENNEN mit Vorbedingung

Den höchsten Wert in *utility* hat BEARBEITEN-OHNE-TRENNEN. Seine Unteraktivitäten sind identisch mit denen von BEARBEITEN-MIT-TRENNEN (vgl. Abbildung 8.3). Sie sind keine Duplikate; der Anwender hat keine doppelte Arbeit. Der Reihenfolgeplaner kann solche Strukturen, die keine echten Bäume sind, ebenfalls verarbeiten. Wer diese Möglichkeit nutzt, sollte aber bedenken, daß dabei in Verbindung mit bestimmten Suchverfahren Zirkularitäten nicht auszuschließen sind.

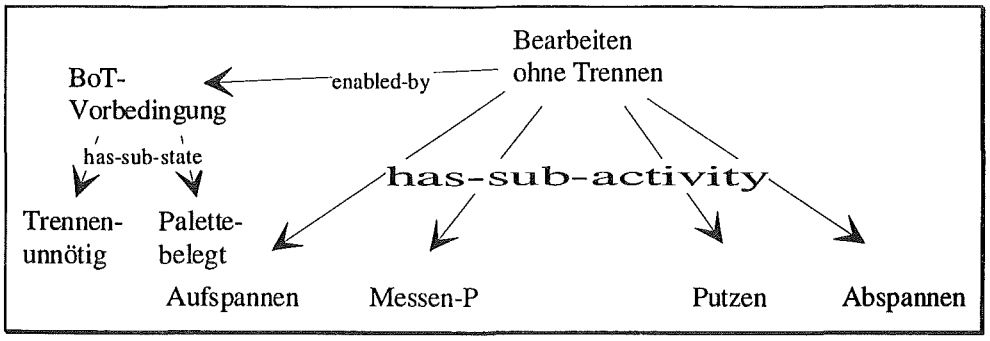


Abb. 8.3: Die Aktivität BEARBEITEN-OHNE-TRENNEN mit Vorbedingung und Unteraktivitäten

"Der Spannbediener an der Spannstation besorgt das Auf- und Abspannen." Die Aktivitäten AUFSPANNEN und ABSPANNEN werden nicht weiter verfeinert. Sie benötigen aber einen Spannbediener an der Spannstation. AUFSPANNEN setzt den Anwenderslot *aufgespannt* des Werkstücks auf wahr, ABSPANNEN setzt ihn auf falsch. Eine Sequenz MESSEN-T - TRENNEN - AUFSPANNEN - MESSEN-P - ABSPANNEN - PUTZEN ist aus zwei Gründen unzulässig: 1. Das Werkstück wird getrennt, bevor es auf die Palette gespannt wurde. 2. Das Werkstück wird geputzt, nachdem es bereits wieder abgespannt wurde. Die Vor- und Nachbedingungen auf den Anwenderslot *aufgespannt* bewirken, daß die Aktivitäten TRENNEN, MESSEN-T, MESSEN-P und PUTZEN nur bei aufgespanntem Werkstück stattfinden. Die Bedingungen für AUFSPANNEN beschreibt Tabelle 8.5, für Abspannen Tabelle 8.6.

AUFSPANNEN	Aufspannen des Werkstücks auf eine Palette. Zum Aufspannen ist ein Spannbediener nötig. Die Vorbedingung SPANNBEDIENER-BELEGT reserviert ihn.	ODER-AKTIVITÄT	*enabled-by* Spannbediener-belegt *cause* Werkstück-aufgespannt	
SPANNBEDIENER-BELEGT	prüft, ob eine Spannbediener S zur Verfügung steht und reserviert ihn.	BESITZPRÄDIKAT DER VORBEDINGUNG	S *required-by*	
WERKSTÜCK-AUFGESPANNT	sichert zu, daß Werkstück W aufgespannt wurde.	ZUSTANDSPRÄDIKAT DER NACHBEDINGUNG	W *required-by*	W.*aufgespannt* := wahr

Tab. 8.5: AUFSPANNEN mit Vor- und Nachbedingung

ABSPANNEN	Abspannen des Werkstücks von einer Palette. Zum Abspannen ist ein Spannbediener nötig. Die Vorbedingung SPANNBEDIENER-BELEGT reserviert ihn.	ODER-AKTIVITÄT	*enabled-by* Abspann-Vorbedingungen *cause* Werkstück-abgespannt	
ABSPANN-VORBEDINGUNGEN	Prüft, ob die Bedingungen WERKSTÜCK-NOCH-AUFGESPANNT und SPANNBEDIENER-BELEGT beide gelten	UND-VORBEDINGUNG	*has-sub-state* Spannbediener-belegt *has-sub-state* Werkstück-abgespannt	
SPANNBEDIENER-BELEGT	prüft, ob ein Spannbediener S zur Verfügung steht und reserviert ihn.	BESITZPRÄDIKAT DER VORBEDINGUNG	S *required-by*	
WERKSTÜCK-NOCH-AUFGESPANNT	prüft, ob Werkstück W noch aufgespannt ist.	ZUSTANDSPRÄDIKAT DER VORBEDINGUNG	W *required-by*	W.*aufgespannt* : wahr
WERKSTÜCK-ABGESPANNT	sichert zu, daß Werkstück W abgespannt wurde.	ZUSTANDSPRÄDIKAT DER NACHBEDINGUNG	W *required-by*	W.*aufgespannt* := falsch

Tab. 8.6: Nachfolger der Aktivität ABSPANNEN

Zum ABSPANNEN müssen zwei Bedingungen gelten. Die ABSPANN-VORBEDINGUNGEN faßt sie zusammen. (vgl. Abbildung 8.4).

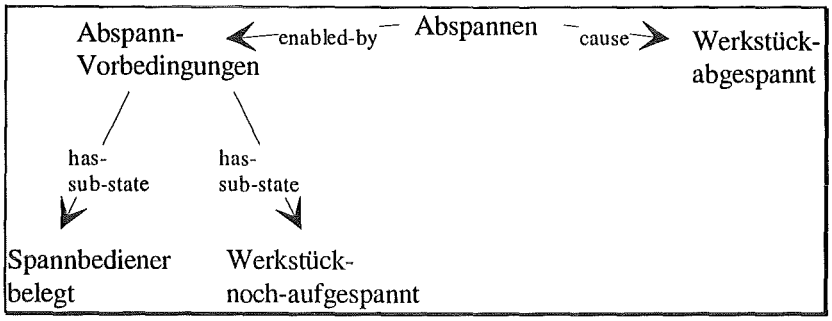


Abb. 8.4: Grafische Darstellung des A/S-Clusters für Abspannen: Zu den Vorbedingungen zählt, daß ein Spannbediener zur Verfügung steht und das Werkstück noch aufgespannt ist. In den Nachbedingungen gilt dann, daß das Werkstück abgespannt ist.

"Ist ein Werkstück dem System unbekannt, muß der Einlernbediener das automatische Trennen, Messen und Putzen einlernen." Diese Operationen lassen sich ausarbeiten in AUTO-TRENNEN, AUTO-MESSEN, AUTO-PUTZEN sowie EINLERN-TRENNEN, EINLERN-MESSEN, EINLERN-PUTZEN. Der Einlernbediener testet am Ende jedes Einlernbetriebs sein Programm am Beispielwerkstück. Man braucht deswegen für dieses Werkstück keinen Automatikbetrieb mehr einzuplanen.

Zum Messen für die Trennoperation muß das Werkstück aufgespannt und getrennt sein und ein Roboter bereitstehen. In Tabelle 8.7 ist MESSEN-T mit seinen Bedingungen formalisiert.

MESSEN-T	Messen des Grates am Werkstück für die Trennoperation. Anschließend sichert man in MESSEN-T-FERTIG den Erfolg zu, um das Putzen zu ermöglichen. Es gibt zwei Ausarbeitungen: 1. Es existiert ein Meßprogramm: AUTO-MESSEN-T, sonst 3. EINLERN-MESSEN-T	ODER-AKTIVITÄT	*enabled-by* MESS-T-VORBEDINGUNG *cause* MESSEN-T-FERTIG *has-elaboration* AUTO-MESSEN-T *has-elaboration* EINLERN-MESSEN-T	
MESS-T-VORBEDINGUNG	MESS-T-VORBEDINGUNG faßt die Vorbedingungen zusammen. Zwei Bedingungen müssen erfüllt sein, um mit dem Messen für die Trennoperation beginnen zu können: 1. Es gibt einen Meßroboter 2. Das Werkstück ist auf eine Palette gespannt.	UND-VORBEDINGUNG	*has-sub-state* AUFGESPANNT-FÜR-MESSEN-T *has-sub-state* ROBOTER-FÜR-MESSEN-T	
AUFGESPANNT-FÜR-MESSEN-T	prüft, ob Werkstück W aufgespannt und gemessen ist. (gemessen über *cause-enable*)	ZUSTANDSPRÄDIKAT DER VORBEDINGUNG	W *required-by*	W.*aufgespannt*: wahr
ROBOTER-FÜR-MESSEN-T	prüft, ob der Meßroboter M zur Verfügung steht und belegt ihn.	BESITZPRÄDIKAT DER VORBEDINGUNG	M.*required-by*	
MESSEN-T-FERTIG	sichert zu, daß Werkstück W gemessen wurde.	ZUSTANDSPRÄDIKAT DER NACHBEDINGUNG	*cause-enable* FÜR-TRENNEN-GEMESSEN	

Tab. 8.7: MESSEN-T mit Vor- und Nachbedingungen

Zwischen den Ausarbeitungen von MESSEN-T gibt es Prioritäten: Das Verfahren soll zuerst prüfen, ob AUTO-MESSEN-T möglich ist. Falls nicht, soll es dafür sorgen, daß jemand das Programm einlernt. Einträge im Slot *utility* reflektieren die Rangfolge. Das automatische Messen für die Trennoperation greift auf ein Meßprogramm zurück. Sein A/S-Cluster zeigt Tabelle 8.8.

AUTO-MESSEN-T	Ein Meßroboter mißt das Werkstück für die Trennoperation automatisch mit dessen Meßprogramm.	UND-AKTIVITÄT	*enabled-by* MESSEN-T-BEREITS-EINGELERNT	*utility* := 2
MESSEN-T-BEREITS-EINGELERNT	prüft, ob für Werkstück W ein Meßprogramm für die Trennoperation eingelernt wurde.	ZUSTANDS-PRÄDIKAT DER VORBEDINGUNG	W *required-by*	W.*Meß-programm-T-eingelernt* : t

Tab. 8.8: Strukur von AUTO-MESSEN

Das Einlernen des Meßroboters erzeugt ein Meßprogramm, das in der Nachbedingung von EINLERN-MESSEN-T zugesichert wird. Der Vorgang benötigt einen Einlern-Bediener (vgl. Tabelle 8.9).

EINLERN-MESSEN-T	Für das Werkstück wird ein Meßprogramm für die Trennoperation erzeugt. Anschließend wird es gemessen.	UND-AKTIVITÄT	*enabled-by* EINLERN-MESSEN-T-VORBEDINGUNG *cause* MESSEN-EINGELERNT	
BEDIENER-FÜR-EINLERN-MESSEN-T	prüft, ob ein Bediener B zur Verfügung steht und belegt ihn.	BESITZ-PRÄDIKAT DER VORBEDINGUNG	B *required-by*	
MESSEN-T-EINGELERNT	Sichert für das Werkstück W das Meßprogramm für die Trennoperation zu: Revision des Werkstücks.	ZUSTANDS-PRÄDIKAT DER NACHBEDINGUNG	W.*required-by*	W.*Meß-programm-T-eingelernt* := wahr

Tab. 8.9: Strukur von EINLERN-MESSEN-T

Stets muß man sicherstellen, daß das Werkstück auf der Palette liegt. Eine Vorbedingung überprüft beim Werkstück jedesmal den Attributwert in *aufgespannt*. Die Vorrangbeziehung zwischen TRENNEN und MESSEN-P wird gewährt, indem MESSEN-P abfragt, ob das Werkstück getrennt ist. Zwischen der Nachbedingung von MESSEN-P und einer Vorbedingung von PUTZEN genügt eine einfache *cause-enable* Beziehung. Sie garantiert, daß MESSEN-P abgeschlossen wurde, bevor PUTZEN beginnt, vgl. Tabelle 8.10.

TRENNEN	Trennen von Steigern und Angüssen Zum Trennen muß das Werkzeug auf der Palette aufgespannt sein. Anschließend wird beim Werkstück das Attribut getrennt zugesichert. Es gibt zwei Möglichkeiten: 1. Es gibt ein Trennprogramm: AUTO-TRENNEN, sonst 2. EINLERN-TRENNEN	ODER-AKTIVITÄT	*enabled-by* AUFGESPANNT-FÜR-TRENNEN *cause* WERKSTÜCK-GETRENNT *has-elaboration* AUTO-TRENNEN *has-elaboration* EINLERN-TRENNEN	
TRENNEN-VORBEDINGUNG	Faßt die Vorbedingungen AUFGESPANNT-FÜR-TRENNEN und ROBOTER-FÜR-AUTO-TRENNEN zusammen.	UND-VORBEDINGUNG	has-sub-state AUFGESPANNT-FÜR-TRENNEN has-sub-state ROBOTER-FÜR-AUTO-TRENNEN	
AUFGESPANNT-FÜR-TRENNEN	prüft, ob Werkstück W aufgespannt ist.	ZUSTANDS-PRÄDIKAT DER VORBEDINGUNG	W *required-by*	W.*aufgespannt* : wahr
ROBOTER-FÜR-TRENNEN	prüft, ob ein Trennroboter T zur Verfügung steht und belegt ihn.	BESITZ-PRÄDIKAT DER VORBEDINGUNG	T *required-by*	
WERKSTÜCK-GETRENNT	sichert zu, daß Werkstück W getrennt wurde.	ZUSTANDS-PRÄDIKAT DER NACHBEDINGUNG	W.*required-by*	W.*getrennt* := wahr

Tab. 8.10: TRENNEN mit Vor- und Nachbedingungen

Zwischen den Ausarbeitungen von TRENNEN gibt es Prioritäten: Das Verfahren soll zuerst prüfen, ob AUTO-TRENNEN möglich ist. Falls nicht, soll es EINLERN-TRENNEN. Einträge im Slot *utility* reflektieren diese Rangfolge. "Zum automatischen Messen, Trennen und Putzen wird der jeweilige Roboter benötigt." Ein Besitzprädikat ROBOTER-FÜR-TRENNEN prüft die Verfügbarkeit des Roboters. Eine weitere Vorbedingung für das automatische Trennen ist ein eingelerntes Trennprogramm, wie Tabelle 8.11 zeigt.

AUTO-TRENNEN	Ein Trennroboter trennt das Werkstück automatisch mit dessen Trennprogramm.	UND-AKTIVITÄT	*enabled-by* TRENNEN-BEREITS-EINGELERNT	*utility* := 2
TRENNEN-BEREITS-EIN-GELERNT	prüft, ob für Werkstück W ein Trennprogramm eingelernt wurde.	ZU-STANDS-PRÄDIKAT DER VOR-BEDIN-GUNG	W *required-by*	W.*Trenn-pro-gramm-eingelernt* : t

Tab. 8.11: Strukur von AUTO-TRENNEN

Der Vorgang EINLERN-TRENNEN benötigt zusätzlich einen Einlern-Bediener. In der Nachbedingung sichert er zu, daß für das Werkstück ein Trennprogramm existiert (vgl. Tabelle 8.12).

EINLERN-TRENNEN	Für das Werkstück wird ein Trennprogramm generiert, anschließend wird es getrennt.	UND-AKTIVITÄT	*enabled-by* BEDIENER-FÜR-EINLERN-TRENNEN *cause* TRENNEN-EINGELERNT	
BEDIENER-FÜR-EINLERN-TRENNEN	prüft, ob ein Bediener B zur Verfügung steht und belegt ihn.	BESITZ-PRÄDIKAT DER VOR-BEDIN-GUNG	B *required-by*	
TRENNEN-EINGE-LERNT	Sichert für das Werkstück W das Einlernprogramm zu: Revision des Werkstücks.	ZU-STANDS-PRÄDIKAT DER NACHBE-DINGUNG	W.*required-by*	W.*Trenn-pro-gramm-eingelernt* := wahr

Tab. 8.12: Strukur von EINLERN-TRENNEN

Die A/S-Cluster für Trennen, Messen und Putzen sind sich ähnlich. Abbildung 8.5 veranschaulicht grafisch das A/S-Cluster TRENNEN.

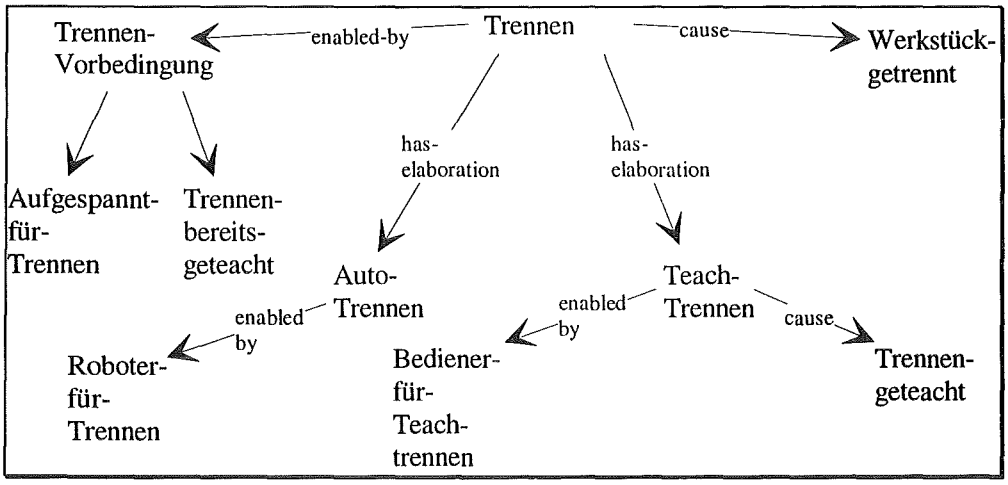


Abb. 8.5: Das A/S-Cluster der Aktivität TRENNEN

Zum Messen für die Putzoperation muß das Werkstück aufgespannt und getrennt sein und ein Roboter bereitstehen. In Tabelle 8.13 ist MESSEN-P mit seinen Bedingungen formalisiert.

MESSEN-P	Messen des Grates am Werkstück für die Putzoperation. Anschließend sichert man in MESSEN-P-FERTIG den Erfolg zu, um das Putzen zu ermöglichen. Es gibt zwei Ausarbeitungen: 1. Es existiert ein Meßprogramm: AUTO-MESSEN-P, sonst 3. EINLERN-MESSEN-P	ODER-AKTIVITÄT	*enabled-by* MESS-P-VORBEDINGUNG *cause* MESSEN-P-FERTIG *has-elaboration* AUTO-MESSEN-P *has-elaboration* EINLERN-MESSEN-P	
MESS-P-VORBEDINGUNG	MESS-P-VORBEDINGUNG faßt die Vorbedingungen zusammen. Drei Bedingungen müssen erfüllt sein, um mit dem Messen für die Putzoperation beginnen zu können: 1. Es gibt einen Meßroboter 2. Das Werkstück braucht nicht mehr getrennt zu werden 3. Das Werkstück ist auf eine Palette gespannt.	UND-VORBEDINGUNG	*has-sub-state* AUFGESPANNT-UND-GETRENNT-FÜR-MESSEN-P *has-sub-state* ROBOTER-FÜR-MESSEN-P	
AUFGESPANNT-UND-GETRENNT-FÜR-MESSEN-P	prüft, ob Werkstück W aufgespannt und getrennt ist.	ZUSTANDS-PRÄDIKAT DER VORBEDINGUNG	W *required-by*	W.*aufgespannt* : wahr W.*getrennt* : wahr
ROBOTER-FÜR-MESSEN-P	prüft, ob der Meßroboter M zur Verfügung steht und belegt ihn.	BESITZ-PRÄDIKAT DER VORBEDINGUNG	M.*required-by*	
MESSEN-P-FERTIG	sichert zu, daß Werkstück W gemessen wurde.	ZUSTANDS-PRÄDIKAT DER NACHBEDINGUNG	*cause-enable* FÜR-PUTZEN-GEMESSEN	

Tab. 8.13: MESSEN-P mit Vor- und Nachbedingungen

Zwischen den Ausarbeitungen von MESSEN-P gibt es Prioritäten: Das Verfahren soll zuerst prüfen, ob AUTO-MESSEN-P möglich ist. Falls nicht, soll es dafür sorgen, daß jemand das Programm einlernt. Einträge im Slot *utility* reflektieren die Rangfolge. Das automatische Messen für die Putzoperation greift auf ein Meßprogramm zurück. Sein A/S-Cluster zeigt Tabelle 8.14.

AUTO-MESSEN-P	Ein Meßroboter mißt das Werkstück für die Putzoperation automatisch mit dessen Meßprogramm.	UND-AKTIVITÄT	*enabled-by* MESSEN-P-BEREITS-EINGELERNT	*utility* := 2
MESSEN-P-BEREITS-EINGELERNT	prüft, ob für Werkstück W ein Meßprogramm für die Putzoperation eingelernt wurde.	ZUSTANDS-PRÄDIKAT DER VORBEDINGUNG	W *required-by*	W.*Meß-programm-P-eingelernt* : t

Tab. 8.14: Strukur von AUTO-MESSEN

Das Einlernen des Meßroboters erzeugt ein Meßprogramm, das in der Nachbedingung von EINLERN-MESSEN-P zugesichert wird. Der Vorgang benötigt einen Einlern-Bediener (vgl. Tabelle 8.15).

EINLERN-MESSEN-P	Für das Werkstück wird ein Meßprogramm für die Putzoperation erzeugt. Anschließend wird es gemessen.	UND-AKTIVITÄT	*enabled-by* EINLERN-MESSEN-P-VORBEDINGUNG *cause* MESSEN-EINGELERNT	
BEDIENER-FÜR-EINLERN-MESSEN-P	prüft, ob ein Bediener B zur Verfügung steht und belegt ihn.	BESITZ-PRÄDIKAT DER VORBEDINGUNG	B *required-by*	
MESSEN-P-EINGELERNT	Sichert für das Werkstück W das Meßprogramm für die Putzoperation zu: Revision des Werkstücks.	ZUSTANDS-PRÄDIKAT DER NACHBEDINGUNG	W.*required-by*	W.*Meß-programm-P-eingelernt* := wahr

Tab. 8.15: Strukur von EINLERN-MESSEN-P

Das Cluster für PUTZEN hat bis auf zwei Ausnahmen die gleiche Gestalt wie MESSEN-P: 1. In der Vorbedingung wird abgefragt, ob das Werkstück gemessen wurde. Die Zusicherung erfolgt über *cause-enable*, was einfacher ist als eine Abfrage mit Prädikaten. 2. Putzen benötigt keine Nachbedingung (vgl. Tabelle 8.16).

PUTZEN	Putzen des Grates am Werkstück Es gibt zwei Ausarbeitungen: 1. Es existiert ein Putzprogramm: AUTO-PUTZEN, sonst 2. EINLERN-PUTZEN	ODER-AKTIVITÄT	*enabled-by* PUTZ-VORBEDINGUNG *has-elaboration* AUTO-PUTZEN *has-elaboration* EINLERN-PUTZEN	
PUTZ-VORBEDINGUNG	PUTZ-VORBEDINGUNG faßt die Vorbedingungen zusammen. Drei Bedingungen müssen erfüllt sein, um mit dem Putzen beginnen zu können: 1. Es gibt einen Putzroboter 2. Das Werkstück ist gemessen 3. Das Werkstück ist auf eine Palette gespannt.	UND-VORBEDINGUNG	*has-sub-state* AUFGESPANNT-UND-GEMESSEN-FÜR-PUTZEN *has-sub-state* ROBOTER-FÜR-PUTZEN	
AUFGESPANNT-FÜR-PUTZEN	prüft, ob Werkstück W aufgespannt und gemessen ist (gemessen über *cause-enable*)	ZUSTANDS-PRÄDIKAT DER VORBEDINGUNG	W *required-by*	W.*aufgespannt* : wahr
ROBOTER-FÜR-PUTZEN	prüft, ob der Putzroboter M zur Verfügung steht und belegt ihn.	BESITZ-PRÄDIKAT DER VORBEDINGUNG	M.*required-by*	

Tab. 8.16: PUTZEN mit Vorbedingungen

Zwischen den Ausarbeitungen gibt es Prioritäten: Das Verfahren soll zuerst prüfen, ob AUTO-PUTZEN möglich ist. Falls nicht, soll es Einlernen. Einträge im Slot *utility* reflektieren die Prioritäten. Das automatische Putzen greift auf ein Putzprogramm zurück. Die Aktivitäten und Zustände enthält Tabelle 8.17.

AUTO-PUTZEN	Ein Putzroboter mißt das Werkstück automatisch mit dessen Putzprogramm.	UND-AKTIVITÄT	*enabled-by* PUTZEN-BEREITS-EINGELERNT	*utility* := 2
PUTZEN-BEREITS-EINGELERNT	prüft, ob für Werkstück W ein Putzprogramm eingelernt wurde.	ZUSTANDS-PRÄDIKAT DER VORBEDINGUNG	W *required-by*	W.*Putzprogramm-eingelernt* : t

Tab. 8.17: Strukur von AUTO-PUTZEN

Putzen mit Einlernen erzeugt ein Putzprogramm. "Zum Einlernen der Putzbearbeitung 'Putzeinlernen' braucht er den Putz- und den Meßroboter." Den zusätzlichen Meßroboter belegt die Vorbedingung MESSROBOTER-FÜR-EINLERN-PUTZEN. EINLERN-PUTZEN benötigt außerdem einen Einlern-Bediener. In der Nachbedingung sichert EINLERN-PUTZEN das neue Putzprogramm zu (vgl. Tabelle 8.18).

EINLERN-PUTZEN	Für das Werkstück wird ein Putzprogramm erzeugt. Anschließend wird es geputzt.	UND-AKTIVITÄT	*enabled-by* EINLERN-PUTZEN-VORBEDINGUNG *cause* PUTZEN-EINGELERNT	
EINLERN-PUTZEN-VORBEDINGUNG	Prüft ob die Vorbedingungen MEßROBOTER-FÜR-EINLERN-PUTZEN und BEDIENER-FÜR-EINLERN-PUTZEN gleichzeitig gelten	UND-VORBEDINGUNG	*has-sub-state* MEßROBOTER-FÜR-EINLERN-PUTZEN *has-sub-state* BEDIENER-FÜR-EINLERN-PUTZEN	
MEßROBOTER-FÜR-EINLERN-PUTZEN	Prüft, ob zusätzlich ein Meßroboter M verfügbar ist.	BESITZ-PRÄDIKAT DER VORBEDINGUNG	M *required-by*	
BEDIENER-FÜR-EINLERN-PUTZEN	prüft, ob ein Bediener B zur Verfügung steht und belegt ihn.	BESITZ-PRÄDIKAT DER VORBEDINGUNG	B *required-by*	
PUTZEN-EINGELERNT	Sichert für das Werkstück W das Einlernprogramm zu: Revision des Werkstücks.	ZUSTANDS-PRÄDIKAT DER NACHBEDINGUNG	W.*required-by*	W.*Putzprogramm-eingelernt* := wahr

Tab. 8.18: Strukur von EINLERN-PUTZEN

Im so entstandenen Cluster mit dem Startknoten BEARBEITEN wurden alle notwendigen Kausalitäten und Alternativen der Aufgabenstellung dargestellt. Abbildung 8.6 faßt die Aktivitäten des A/S-Clusters zusammen. Die Struktur des Problems wird dabei sofort deutlich.

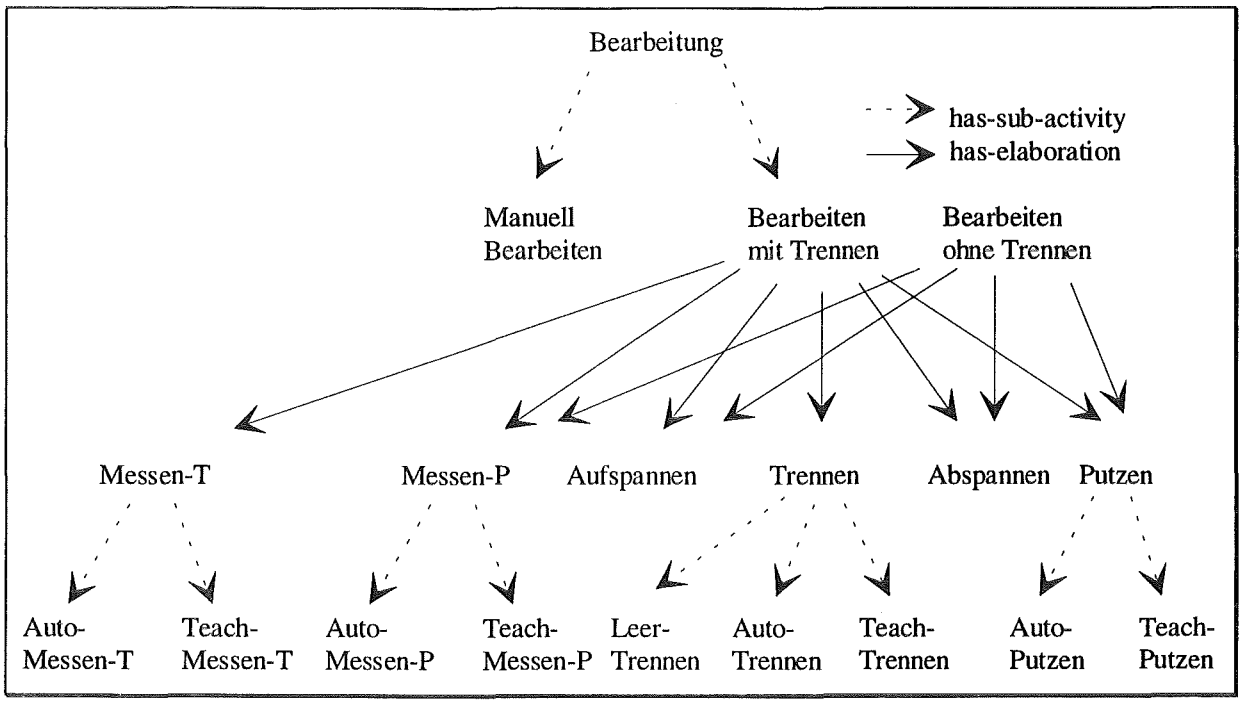


Abb. 8.6: Aktivitäten des Clusters BEARBEITUNG mit Verfeinerungen.

8.1.2.3. Zeitfenster

Ein Zeitfenster wird für jede Aktivität automatisch erzeugt: Ausarbeitungen sind zeitgleich mit ihren ODER-AKTIVITÄTEN. Unteraktivitäten ereignen sich während ihrer UND-AKTIVITÄT, starten oder beenden sie (d, s, f). Diese Beziehungen werden als Defaultbeziehungen von der Methode *erzeuge-Zeitdefaults* generiert, müssen also nicht von Hand eingegeben werden. Tabelle 8.19 zeigt die Beziehungen.

Zeitfenster von	AUFSPANNEN	MESSEN-T	TRENNEN	MESSEN-P	PUTZEN	ABSPANNEN	BEARBEITUNG
AUFSPANNEN		<, m	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	d, s, f
MESSEN-T	<, >, d, di, o, oi, m, mi, s, si, f, fi, =		<, m	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	d, s, f
TRENNEN	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =		<, m	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	d, s, f
MESSEN-P	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =		<, m	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	d, s, f
PUTZEN	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =		<, m	d, s, f
ABSPANNEN	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =		d, s, f
BEARBEITUNG	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	<, >, d, di, o, oi, m, mi, s, si, f, fi, =	

Tab. 8.19: Matrix der Zeitbeziehungen zwischen den Aktivitäten, wenn keine Zeitbeziehung vom Anwender ausdrücklich vorgegeben wird.

Das Matrixelement (AUFSPANNEN, BEARBEITUNG) = "d, s, f" bedeutet, daß zwischen AUFSPANNEN und BEARBEITUNG die Beziehungen *während, startet* oder *beendet* zulässig sind. In der ersten Zeile und der ersten Spalte stehen die Aktivitäten der Zeitfenster. Die Tabelle enthält keine Zeitbeziehungen zwischen einer ODER-AKTIVITÄT und ihren Ausarbeitungen - die sind nämlich einfach *zeitgleich*. Eine Aktivität steht nie zu sich selbst in einer Zeitbeziehung. Deswegen ist die Hauptdiagonale leer. "<, >, d, di, o, oi, m, mi, s, si, f, fi, =" bedeutet, daß alle Zeitbeziehungen möglich sind, d.h. bisher wurde nichts festgelegt.

Die qualitative Zeitpropagation errechnet Zeitbeziehungen zwischen den Zeitfenstern wie in Tabelle 8.20 dargestellt.

Zeitfenster von	AUF-SPANNEN	MESSEN-T	TRENNEN	MESSEN-P	PUTZEN	ABSPAN-NEN	BEARBEI-TUNG
AUF-SPANNEN		<, m					s
MESSEN-T	>, mi		<, m				d
TRENNEN		>, mi		<, m			d
MESSEN-P			>, mi		<, m		d
PUTZEN				>, mi		<, m	d
ABSPAN-NEN					>, mi		f
BEARBEI-TUNG	si	di	di	di	di	fi	

Tab. 8.20: Zeitbeziehung nach der qualitativen Propagation. Im Vergleich zu Tab. 8.19 wurde das Beziehungsnetz nennenswert vereinfacht und ist fast eindeutig bestimmt.

Bereits mit den qualitativen Angaben in Tabelle 8.20 kann die Zeitpropagation das Netz stark vereinfachen. Der Anwender muß das Zeitfenster Z1 für eine Fehlzeit des Einlernbedieners als einziges Zeitfenster selbst eingeben. Solange noch nicht geplant wurde, kann es zu keiner der Aktivitäten in Beziehung gesetzt werden.

Jede Aktivität ohne Verfeinerung hat gemäß Tabelle 8.1 eine *Dauer*, die der Planer aus den Attributen des Werkzeugs in die Zeitfenster überträgt. "Arbeitsbeginn ist der 1. Juni, 8:00 h. Die Werkstücke der Firma Schubert & Salzer müssen bis zum Abend des 4. Juni, die der Firma Heidelberger Druckmaschinen bis zum Abend des 5. Juni bearbeitet sein." Diese Fristen schreibt das System in das Zeitfenster von BEARBEITUNG hinein. Es erhält den frühesten Anfang *FA* und das späteste Ende *SE* von den Attributen *Frist* und *Freigabezeitpunkt* des zugehörigen Unterauftrags. Die Werte sind für die Freigabe der 1. Juni und für die Frist je nach Werkstück der 4. oder der 5. Juni.

8.1.2.4. Modellierung des Werkzeugverschleißes

Wenn sich der Anwender zusätzlich für den Werkzeugverschleiß in der Roboterzelle interessiert, kann er die notwendigen Objekte jederzeit anfügen, ohne die bisherige Arbeit verwerfen zu müssen. Zunächst die zusätzliche Aufgabenstellung:

"Zur Bearbeitung stehen drei Werkzeugtypen zur Verfügung: Ein Fräser mit einer Standzeit von 10 Minuten, eine Trennscheibe mit einer Standzeit von 30 Minuten, ein Fräsmeißel mit einer Standzeit von 5000 Minuten und eine Schruppscheibe mit einer Standzeit von 300 Minuten. Der Werkzeugbestand umfaßt: 5 neue Fräser und einen mit einer Reststandzeit von 6 Minuten, 4 neue Schruppscheiben und eine mit einer Reststandzeit von 20 Minuten sowie einen Fräsmeißel mit einer Reststandzeit von 3000 Minuten. Beim Putzen und Trennen verschleißen die verwendeten Werkzeuge, die bei Bedarf ausgewechselt werden."

Zur Modellierung greift man auf die Klasse WERKZEUG der Planungsobjekte zurück (vgl. Kapitel 7). Von den Klassen FRÄSER, FRÄSMEIßEL und SCHRUPPSCHEIBE schafft man Unterklassen, um jedes einzelne Werkzeug[1] darzustellen. Man erzeugt es mit der Methode *schaffe-Unterklasse*. Im Slot *Reststandzeit* vom Objekt *Fräser* (2.1.1.4.3) trägt man die Standzeit von 10 Minuten ein. Der Wert wird an alle Unterklassen von *Fräser* vererbt. Ein Fräser z.B. Fräser-1 war bereits 4 Minuten im Einsatz. In seinen Slot *Reststandzeit* schreibt man den Wert "6 Minuten", überschreibt also den geerbten Wert.

[1] z.B. Fräser 1 bis 10, falls 10 Fräser vorhanden

Um den Verschleiß darzustellen, kann man den Aktivitäten TRENNEN und BEARBEITEN zusätzliche Vor- und Nachbedingungen zuordnen. Sie erhalten ein Zustandsprädikat der Vorbedingung WERKZEUG-VERFÜGBAR, das ein Werkzeug mit genügend Reststandzeit aussucht. Ein Zustandsprädikat der Nachbedingung revidiert anschließend den Wert *Reststandzeit* des betroffenen Werkzeugs. Die Zustandsprädikate für TRENNEN zeigt Tabelle 8.21 t_1, t_2, t_3,t_4 sind Eingriffszeiten des Werkzeugs.

Knotenname	**Beschreibung**	**ist ein (is-a)**	**Beziehungen**	**Sloteinträge**
TRENN-SCHEIBE-FÜR-TRENNEN	Prüft, ob noch eine Trennscheibe T mit einer Reststandzeit > t_1 existiert. Falls ja, wird sie ausgewählt.	ZU-STANDS-PRÄDIKAT DER VOR-BEDING-UNG	TRENNEN-VORBE-DINGUNG *has-sub-state* T *required-by*	*Reststand-zeit* : > t_1
TRENN-SCHEIBE-FÜR-TRENNEN-VER-SCHLISSEN	Sichert für die Trennscheibe T den Verschleiß zu: Revision der Trennscheibe	ZU-STANDS-PRÄDIKAT DER NACHBE-DINGUNG	TRENNEN-NACHBEDIN-GUNG *has-sub-state*T *required-by*	*Reststand-zeit* := Reststand-zeit - t_1

Tab. 8.21: Erforderliche Vor- und Nachbedingungen zur Modellierung des Werkzeugverschleißes beim AUTO-TRENNEN und EINLERN-TRENNEN

Die entsprechenden Bedingungen beim Putzen zeigt Tabelle 8.22. Jedes verwendete Werkzeug erhält eine eigene Vor- und Nachbedingung.

TRENNSCHEIBE-FÜR-PUTZEN	Prüft, ob noch eine Trennscheibe T mit einer Reststandzeit > t_2 existiert. Falls ja, wird sie ausgewählt.	ZUSTANDSPRÄDIKAT DER VORBEDINGUNG	PUTZEN-VORBEDINGUNG *has-sub-state* T *required-by*	*Reststandzeit* : > t_2
TRENNSCHEIBE-FÜR-PUTZEN-VERSCHLISSEN	Sichert für die Trennscheibe T den Verschleiß zu: Revision der Trennscheibe	ZUSTANDSPRÄDIKAT DER NACHBEDINGUNG	PUTZEN-NACHBEDINGUNG *has-sub-state*T *required-by*	*Reststandzeit* := Reststandzeit - t_2
FRÄSER-FÜR-PUTZEN	Prüft, ob noch ein Fräser F mit einer Reststandzeit > t_3 existiert. Falls ja, wird er ausgewählt.	ZUSTANDSPRÄDIKAT DER VORBEDINGUNG	PUTZEN-VORBEDINGUNG *has-sub-state* F *required-by*	*Reststandzeit* : > t_3
FRÄSER-FÜR-PUTZEN-VERSCHLISSEN	Sichert für den Fräser F den Verschleiß zu: Revision der Trennscheibe	ZUSTANDSPRÄDIKAT DER NACHBEDINGUNG	PUTZEN-NACHBEDINGUNG *has-sub-state* F *required-by*	*Reststandzeit* := Reststandzeit - t_3
FRÄSMEISSEL-FÜR-PUTZEN	Prüft, ob noch ein Fräsmeißel F mit einer Reststandzeit > t_4 existiert. Falls ja, wird er ausgewählt.	ZUSTANDSPRÄDIKAT DER VORBEDINGUNG	PUTZEN-VORBEDINGUNG *has-sub-state* F *required-by*	*Reststandzeit* : > t_4
FRÄSSMEISSEL-FÜR-PUTZEN-VERSCHLISSEN	Sichert für den Fräsmeißel F den Verschleiß zu: Revision der Trennscheibe	ZUSTANDSPRÄDIKAT DER NACHBEDINGUNG	PUTZEN-NACHBEDINGUNG *has-sub-state* F *required-by*	*Reststandzeit* := Reststandzeit - t_4

Tab. 8.22: Zustandsprädikate in Vor- und Nachbedingungen, die den Werkzeugverschleiß beim Putzen modellieren

Für die Modellierung muß der Anwender a priori die Eingriffszeiten t_1, t_2, t_3, t_4 jedes Werkzeugs kennen. Wurde ein Werkstück mit dem Reihenfolgeplaner vorher optimiert, können sie automatisch übertragen werden. Die meisten Werkstücke werden erst innerhalb des Planungshorizonts eingelernt. Wenn der Anwender dabei nicht recht genau die Eingriffszeiten schätzen kann, wird die Planung des Verschleißes Makulatur. Denkbar ist aber, die Zeiten dynamisch zu aktualisieren, nachdem ein Werkstück eingelernt wurde.

8.1.3. Zusammenfassung der Modellierung

Der Arbeitsplan besteht aus 18 Aktivitäten und 42 Zuständen. Wird er für jeden der 109 zu planenden Werkstücke vom System generiert, entstehen 6540 A/S-Instanzen, etwa die Hälfte davon wird in der Propagation auf wahr gesetzt. Sie enthalten alle Informationen über die Planung: Welche Ressource wurde wann von welcher Operation belegt? Welches Werkzeug, welches Werkstück oder welcher Auftrag wurde wann von welcher Operation verändert? Die Objekte speichern nicht nur die Lösung, sondern auch den Lösungsweg. Für einen Wissensingenieur, der die Modellierung optimieren möchte, ist das ein unschätzbarer Vorteil.

Die Anwendungsumgebung konnte mit einer einfachen Faustregel modelliert werden. Zuerst muß der Anwender ein Problem informell beschreiben. Daraus leitet er Planungsobjekte, Aktivitäten, Zustände und die Beziehungen dazwischen ab. Kompliziertere Dinge wie Zeitfenster und Zeitbeziehungen nimmt ihm - im Beispiel bis auf eine Ausnahme - das System ab. Aufgrund von Alternativen ist der Arbeitsplan keineswegs trivial. Mit nur 60 Objekten wurde er formalisiert. Am Beispiel des Werkzeugverschleißes wurde gezeigt, daß zusätzliches Wissen leicht und konsistent integriert werden konnte. Die Darstellung ist damit im Sinne Sathis klar.

Der Arbeitsplan abstrahiert das Problem entlang der Problembeschreibung kompositionell und in Ausarbeitungen. Der Plan beschreibt also Wissen in angemessener Granularität und ist nach Sathi präzise. Er ist schließlich vollständig, weil alle relevanten Aspekte dargestellt wurden.

8.1.4. Beschreibung der Planung

Eine Instanz von Objekt 2.2.2. MASCHINENBELEGUNGSAUFTRAG steuert die gesamte Planung mit seiner Methode *plane-Auftrag*. Für jedes der 109 Werkstücke instanziiert es einen Arbeitsplan. Der Wissensingenieur erzeugt die Instanzen also nicht von Hand. So entstehen u.a. 109 Instanzen von BEARBEITUNG. Sie sind alle Unteraktivitäten einer Instanz G-Inst von Objekt 1.1.1.1. GESAMTPRODUKTION, das die Planung der Jobs steuert. Es arbeitet die Tagesproduktionen mit dem Propagationsschema ab. Um zu sortieren, greift G-Inst auf die Methode von Objekt 2.3.1.1. TAGESPRODUKTIONEN-SORTIERUNG zurück. G-Inst plant nun die Tagesproduktionen ein, indem es der Reihe nach jede Instanz von BEARBEITUNG verifiziert. Jede TAGESPRODUKTION ihrerseits verifiziert ihre Jobs. Dabei sucht sie einen ersten Job mit dem Verfahren JOB-SUCHE, plant ihn ein, sucht den zweiten mit JOB-SUCHE usw. bis alle Jobs verplant sind. Die Instanz von MASCHINENBELEGUNGSAUFTRAG kann nun noch Zeitpunkte festlegen, die von der Zeitplanung nicht vollständig bestimmt wurden. Der Reihe nach wird jeweils der früheste Zeitpunkt gewählt. Damit ist die Planung schon fast fertig. Bei Bedarf werden die Resultate am Bildschirm ausgegeben oder zum PC geschickt.

Wie funktioniert die Sortierung? Die TAGESPRODUKTIONEN-SORTIERUNG sortiert die Jobs zuerst nach Fristigkeit. Weil die Druckmaschinenseitenwände der Firma Heidelberger Druckmaschinen am 5. Juni fertig sein sollen, stehen sie in der Reihenfolge hinter den Achsbrücken von Schubert & Salzer. Das Verfahren sortiert die Werkstücke nach Fristigkeit in folgender Reihenfolge:

22 $Jobs_C$, 17 $Jobs_D$, 26 $Jobs_E$, 23 $Jobs_A$, 17 $Jobs_B$.

22 $Jobs_C$ steht für 22 Bearbeitungen für Werkstücke vom Typ C, eine Abkürzung für Job_{C1}, Job_{C2}, ... , Job_{C22}. Die Aufträge werden anschließend zu Tagesproduktionen à 16 h zusammengefaßt (vgl. Tabelle 8.23).

1. Juni	**2. Juni**	**3. Juni**	**4. Juni**
660 min für 22 $Jobs_C$	400 min für 16 $Jobs_D$	700 min für 18 $Jobs_E$	88 min für 11 $Jobs_A$
460 min zum Einlernen von WERKSTÜCK-D	360 min zum Einlernen von WERKSTÜCK-E	350 min zum Einlernen von WERKSTÜCK-A	210 min für 21 $Jobs_B$
	125 min für 5 $Jobs_E$		

Tab. 8.23: Tagesproduktionen à 16 Maschinenstunden.

Zur Erinnerung: 16 Maschinenstunden sind 960 Maschinenminuten. Klammert man die Tagesproduktionen, lautet die Reihenfolge der Jobs:

(22 $Jobs_C$, *1 Job_D), (16 $Jobs_D$, *6 $Jobs_E$), (20 *$Jobs_E$, *1 Job_A), (11 *$Jobs_A$, 21 $Jobs_B$).

Noch nicht eingelernte Jobs sind mit einem *Sternchen gekennzeichnet. Jede Tagesproduktion wird nun so umsortiert, daß vor dem Putzroboter möglichst eine Warteschlange entsteht, die er abarbeitet, wenn ein Werkstück auf dem Meßroboter eingelernt wird. Die folgende Beschreibung betrachtet nur die beiden kritischen Maschinen, den Meß- und den Putzroboter. Andere Resourcen paßt das Planungsverfahren an.

Zuerst wählt es 22 $Jobs_C$. Ein Job_D müßte zuerst eingelernt werden, was den Putzroboter leer stehen ließe. Um 8.01, nach einer Minute, ist das erste Werkstück-C aufgespannt. Um 12.25 h, nach 264 Minuten hat der Meßroboter 22 Stücke von Werkstück-C gemessen. Die Zeit dafür setzt sich zusammen aus: 1 min Messen für die Trennoperation, 1 min Trennen, 10 min Messen für Putzen = 12 min: 12 min x 22 = 264 min[2]. Die Putzbearbeitung für C dauert 19 Minuten. Der Putzroboter beginnt um 8.13, nachdem das erste Werkstück gemessen wurde. Um 12.25 h wurden deswegen nur 13,3 Werkstücke geputzt, 8,7 Bearbeitungen hat der Putzroboter vor sich. Das entspricht einer Warteschlange von 166 Minuten (= 19 min x 8,7), also etwas weniger als die Dauer, die der Bediener braucht, um das Meßprogramm von Werkstück-D einzulernen. Das geschieht von 12.25 h bis 15.55 h. In dieser Zeit arbeitet der Putzroboter seine 8,7 Werkstücke vom Typ C ab. Anschließend lernt der Bediener auf beiden Robotern 130 Minuten lang bis 10.05 h des 2. Juni das Bearbeitungsprogramm ein. Zwischen 16.00 h des 1. Juni und 8.00 h des 2. Juni waren die Roboter abgeschaltet. Die erste Tagesproduktion ist abgeschlossen (vgl. Tabelle 8.24).

Zeitpunkt bis	1. Juni, 8.01 h	12.25 h	15.55 h	2. Juni, 10.05 h
Δt	-	264 min	210 min	130 min
Meßroboter: bearbeitete Werkstücke	-	22 x Werkstück-C	Einlernen der Meßprogramme von 1 x Werkstück-D	Putzeinlernen von 1 x Werkstück-D
Putzroboter: bearbeitete Werkstücke	-	ab 8.13 h: 13,3 x Werkstück-C	8,7 x Werkstück-E 44 min Leerzeit	Putzeinlernen von 1 x Werkstück-D
Warteschlange vor Putzroboter	-	8,7 x Werkstück-E	-	-
Wartezeit vor Putzroboter	-	166 min	0	0

Tab. 8.24: Erste Tagesproduktion

Der Meßroboter beginnt am 2. Juni um 10.05 mit der zweiten Tagesproduktion. Er mißt bis 12.45 h vom Typ D 16 Werkstücke. Zu diesem Zeitpunkt hat der Putzroboter noch 6,6 Werkstücke vom Typ D in seiner Warteschlange. Das entspricht 106 Minuten Bearbeitungsdauer, also Zeit genug, um am Meßroboter Werkstück E einzulernen. Damit ist

[2]Wegen der Zeiten vgl. Abb. 8.x.

er um 15.33 fertig. Der Putzroboter hat seine Warteschlange 42 Minuten vorher abgearbeitet. Auf beiden Robotern lernt der Bediener 100 Minuten lang bis 9.03 h des 3. Juni das Putzprogramm von Werkstück E ein.

Den Rest der zweiten Tagesproduktion, 5 Werkstücke vom Typ E, mißt der Meßroboter zusammen mit 12 weiteren vom Typ E der dritten Tagesproduktion bis 13.11 h. Zu diesem Zeitpunkt hat der Putzroboter 10,8 Werkstücke vom Typ E abgearbeitet. 6,2 Werkstücke E sind in seiner Warteschlange, das entspricht 136 Minuten Wartezeit (vgl. Tabelle 8.25), 150 Minuten sind zum Meßeinlernen von Werkstück A erforderlich.

Zeitpunkt	2. Juni, 12.45 h	15.33 h	3. Juni, 9.03 h	13.11 h
Δt	160 min	168 min	100 min	238 min
Meßroboter: bearbeitete Werkstücke	16 x Werkstück-D	Einlernen der Meßprogramme von 1 x Werkstück-E	Putzeinlernen von 1 x Werkstück-E	17 x Werkstück-E
Putzroboter: bearbeitete Werkstücke	ab 10.15 h: 9,4 x Werkstück-D	6,6 x Werkstück-D 42 min Leerzeit	Putzeinlernen von 1 x Werkstück-E	ab 9.27 h: 10,8 x Werkstück-E
Warteschlange vor Putzroboter	6,6 x Werkstück-D	-	-	6,2 x Werkstück-E
Wartezeit vor Putzroboter	106 min	0	0	136 min

Tab. 8.25: Zweite Tagesproduktion

Nach den 136 Minuten hat der Putzroboter seine Warteschlange abgearbeitet. Um 15.41 h beginnt der Bediener, auf beiden Robotern das Putzprogramm für A einzulernen. Damit ist er um 9.21 h des 4. Juni fertig. Bis 14.02 h des 4. Juni hat der Meßroboter die 8 verbleibenden Werkstücke E, 11 Werkstücke A und 21 Werkstücke vom Typ B gemessen. Der Putzroboter ist um 14.25 h fertig (vgl. Tabelle 8.26).

Zeitpunkt	15.41 h	4. Juni, 9.21 h	
Δt	15 min	100 min	
Meßroboter: bearbeitete Werkstücke	Meßeinlernen von 1 x Werkstück-A,	Putzeinlernen von 1 x Werkstück-A	bis 14.02 h: Mesen von 8 x Werkstück-E 11 x Werkstück-A 21 x Werkstück-B
Putzroboter: bearbeitete Werkstücke	6,2 x Werkstück-E 14 min Leerzeit	Putzeinlernen von 1 x Werkstück-A	9.35 h - 14.25 h Putzen von 8 x Werkstück-E 11 x Werkstück-A 21 x Werkstück-B
Warteschlange vor Putzroboter	-	-	
Wartezeit vor Putzroboter	0	0	

Tab. 8.26: Dritte und vierte Tagesproduktion

In diesem Beispiel stand der Meßroboter nur während der Trennoperationen leer, der Putzroboter hatte nur zu Beginn und nach jedem Einlernvorgang Leerzeiten. Die Aufträge über die Werkstücke A, B und C konnten am Vormittag des 4. Juni beendet werden, weil sie gleich zu Beginn eingeplant wurden. Das Ziel, die Kapazität unter Wahrung von Fristen maximal zu nutzen wurde gut erreicht. Abbildung 8.7 zeigt am Beispiel des 2. und 3. Juni, wie die Zeiten zum Einlernen der Meßprogramme genutzt wird, um auf dem Putzroboter die vorher aufgebaute Warteschlange abzuarbeiten.

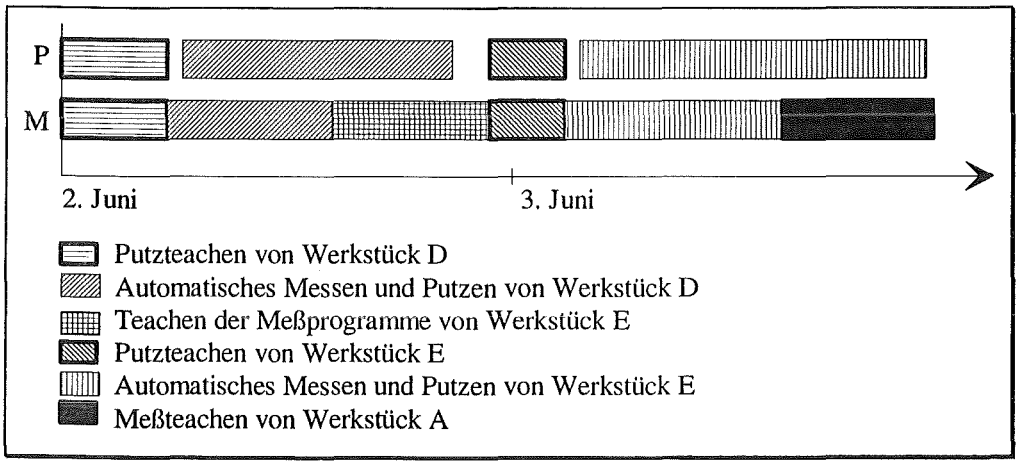


Abb. 8.7: Unterschiedliche Dauern zum Putzen und Bearbeiten werden genutzt, um den Putzroboter während des Einlernens der Meßprogramme zu nutzen. Einlernaufgaben sind mit einem fetten Kasten gekennzeichnet.

Wie wird in diesem Beispiel der Benutzer unterstützt? Der Wissensingenieur entwickelt den Arbeitsplan. Dazu modelliert Varianten eines Arbeitsschritts und nennt Bedingungen für jede Variante. Der hier entwickelte Arbeitsplan deckt mehrere Werkstücktypen ab, es braucht also kein zweiter Arbeitsplan entwickelt werden. Für jeden Werkstücktyp teilt der Anwender dem System mit, welche Tätigkeiten auszuführen sind. Wird nichts mitgeteilt, unterstellt das System, das der Werkstücktyp eingelernt werden muß und im Automatikbetrieb jedes Werkstück nach vorherigem Messen getrennt und geputzt wird.

Für einzelne Werkstücke kann der Anwender Schritte auslassen, hier z.B. das Trennen. Den Arbeitsschritten muß er Zeitdauern zuordnen. Falls eine Ressource nicht verfügbar ist, muß er das ebenfalls dem System mitteilen. Alle weiteren Zeitinformationen entwickelt das System aus der Logik des Arbeitsplans. Die Standardvorgaben kann der Anwender ändern, auch wenn das nur in Ausnahmefällen nötig sein wird. Schließlich muß er die Aufträge eingeben und den Planer starten.

Die Arbeit des Planers ließe sich wesentlich vereinfachen, wenn die Planungsumgebung sich nicht durch Einplanen eines Jobs verändern würde. Das ist aber hier nicht der Fall: Wurde z.B. ein beliebiges Werkstück eines Typs eingelernt, brauchen alle weiteren Werkstücke dieses Typs nicht mehr eingelernt zu werden. Der Reihenfolgeplaner entwickelt deswegen für jedes Werkstück einen eigenen Arbeitsplan. Dabei berücksichtigt er die Auslastung der Ressourcen, Fristen, Eigenheiten des Werkstücktyps, des einzelnen Werkstücks und der Roboterzelle. Der Reihenfolgeplaner ordnet jedes Werkstück seinen Ressourcen und Operationen sachlich und zeitlich zu. Diese Zuordnung heißt Maschinenbelegungsplan.

8.1.4.1. Propagationsschema innerhalb des Arbeitsplans

Wie entwickelt das System einen Arbeitsplan? Bei ODER-Knoten im Arbeitsplan werden die Alternativen nach dem Attributwert von *utility* absteigend sortiert. Innerhalb des Arbeitsplans ist es nicht notwendig, alle Alternativen zu prüfen. Die Suche bricht ab, sobald eine Alternative verifiziert wurde. Beim Putzen besteht z.B. die Möglichkeit, automatisch zu putzen oder das Putzen einzulernen. AUTO-PUTZEN hat die höhere Priorität und wird verifiziert, wenn der Roboter eingelernt ist. EINLERN-PUTZEN zusätzlich zu prüfen wäre überflüsig.

Die UND-Aktivitäten BEARBEITEN-MIT-TRENEN und BEARBEITEN-OHNE-TRENNEN halten ihre Unteraufträge bereits in der sortierten Reihenfolge; eine Sortierung ist unnötig und die Unteraufträge werden in dieser Reihenfolge verifiziert. Ist der Anwender sich nicht über die Reihenfolge im Klaren, wählt er ein Suchverfahren, das für jede Unteraktivität prüft, ob sie zulässig ist. Falls nicht, wird sie für einen weiteren Versuch in eine Warteliste geschrieben (vgl. Abb. 7.9., Zeile 2). Dafür dürfen die Unteraufträge in chaotischer Reihenfolge vorliegen.

8.1.4.2. Propagationsschema bei der Suche nach Planungsobjekten

Das System sucht einerseits aus Operationen aus, andererseits aus physikalischen Objekten, die in der Planung benötigt werden. Die Auswahl von physikalischen Objekten erfolgt analog der Auswahl einer Operation. Mehrmals greift ein Besitz- oder Zustandsprädikat auf ein Planungsobjekt zu, das in diesem Augenblick ein Objekt der Nebenbedingung ist. Dabei spezifiziert das Besitz- oder Zustandsprädikat oft nur eine Klasse mit mehreren Verfeinerungen. Die Vorbedingungen WERKSTÜCK-UNBEARBEITET und GUß-PUTZER-BELEGT spezifizieren gewissermaßen irgendein Werkstück und irgendeinen Gußputzer. Wie wählt das System aus? Zwischen GUßPUTZER und seinen Unterklassen GUßPUTZER-1 und GUßPUTZER-2 besteht zunächst nur eine *is-a* Beziehung. Die Methode *instanziiere-Digraph* fügt eine *has-elaboration*-Beziehung dazu, so daß gilt: GUßPUTZER *has-elaboration* GUßPUTZER-1 und GUßPUTZER *has-elaboration* GUßPUTZER-2. Die drei Objekte werden instanziiert. Die Instanz von Gußputzer wird zusätzlich zu einer ODER-Aktivität, die anderen beiden zu UND-Aktivitäten. Das so entstandene A/S-Cluster kann mit dem Propagationsschema verifiziert werden.

Dabei ist wichtig, daß alle Alternativen untersucht werden, denn es steht durchaus nicht fest, daß die Alternative mit der höchsten *utility* die Prädikate des Zustands- oder Besitzprädikates erfüllt. So hat bei der Auswahl zwischen mehreren Werkzeugen das Werkzeug mit der längsten Standzeit die höchste *utility*. Das ist sinnvoll, weil ein Werkzeug erst ausgewechselt wird, wenn es die Standzeit überschritten hat. Was aber, wenn

die Reststandzeit kleiner ist als die vorgesehene Eingriffszeit? Das Werkzeug mit dem größten *utility*-Wert darf dann nicht genommen werden. Deswegen müssen auch die anderen Werkzeuge verifiziert werden. Die ODER-Aktivität verwendet für solche Fälle ein Suchverfahren, das alle Alternativen untersucht.

8.2. Optimierung der Bahnelemente

Aufgrund fehlender Daten aus der Anwendung baut das hier vorgestellte Beispiel auf Erfahrungswerten auf, die im Rahmen einer Diplomarbeit [Lotter 92, S. 64ff] in der Fertigungszelle gewonnen wurden. Zur Datenerzeugung wurde extra ein Programm geschrieben.

8.2.1. Darstellung

Zur Erinnerung: Das A/S-Cluster für das Bahnelementeproblem hat drei Hierarchiestufen. Eine BAHNGRUPPE faßt alle Bearbeitungsbahnen zusammen, die ein Werkzeug bearbeitet. Zusätzlich enthält die Bahngruppe sämtliche Luftbahnen zwischen ihren Bearbeitungsbahnen einschließlich der Luftbahnen vom und zum Werkzeugwechselpunkt. Alle Bahngruppen sind Unteraktivitäten einer gemeinsamen UND-Aktivität.

8.2.2. In der Zustandsraumrepräsentation suchen

Die gewählte Darstellung ist sinnvoll, weil sie 1. die optimale Lösung enthält und weil 2. zwischen den Bahngruppen keine Interaktionen bestehen. Zu 1.: Jede optimale Lösung muß die Werkzeugwechsel minimieren. Das ergab die Studie von Lotter [Lotter 92, S. 28]. Bei n verwendeten Werkzeugen sind mindestens n-1 Werkzeugwechsel notwendig. Das ist offenbar immer dann der Fall, wenn die Bearbeitungsbahnen nach Werkzeugen sortiert werden. Daraus folgt, daß es keine optimale Lösung gibt, bei der die Bearbeitungsbahnen nicht nach Werkzeugen sortiert werden.

Zu 2.: Einziges Objekt der Nebenbedingung ist die Roboterhand. Sie besitzt zwei Attribute, die abgefahrenen Bahnen und die Position. Weil die Bearbeitungsbahnen disjunkt über die Bahngruppen aufgeteilt sind, interessiert bei der Optimierung einer Bahngruppe B nicht, welche Reihenfolge eine andere Bahngruppe A bestimmte. Die Position der Roboterhand ist zu Beginn und Ende jedes Teilproblems der Werkzeugwechselpunkt. Die Bahngruppen sind damit voneinander unabhängig. Daraus folgt, daß die UND-Aktivität ihre Bahngruppen in beliebiger Reihenfolge verifizieren kann. Ihr Such- und Sortieraufwand ist bei k Bahngruppen gleich k. Ein lineares Suchprogramm reicht dafür aus.

Bleibt die Frage, mit welchem Aufwand das Programm abläuft. Zur Berechnung der Komplexität hilft man sich mit der in Definition 7.1. und 7.2. erarbeiteten Formalisierung des Bahnelementeproblems.

Satz 8.1: Ein Bahnelementelösungscluster D_{EL} habe k Bahngruppencluster mit insgesamt n Bearbeitungsbahnen, $k \leq n$, $k, n \in IN$. Jedes Bahngruppencluster D_{BGi}, $i \in \{1, ..., k\}$ hat b_i Bahnelemente, $b_i \leq (n-k)$. Der Aufwand zur Verifizierung von D_{EL} ist dann $\sum_{m=1}^{k} O(b_i^2)$.

Beweis: D_{BGi} wendet definitionsgemäß die Verfahren des sukzessiven Einfügens sowie der 2- und 3-optimalen Verbesserung an. Ihr Aufwand ist $O(b_i^2)$. Weil das Bahnelementelösungscluster die Bahngruppencluster wahlfrei verifiziert, terminiert das Verfahren nach k Schritten, die k-1 Schritte für die Werkzeugwechsel nicht mitgerechnet. Weil jedes Bahngruppencluster nur einmal verifiziert wird, ist der Gesamtaufwand gleich der Summe dieser Verifikationen.

Die Analyse ist einfacher, wenn alle k Bahngruppen die gleiche Anzahl m an Bearbeitungsbahnen besitzen, daß gilt: n = k * m.

Satz 8.2: Ein Bahnelementelösungscluster D_{EL} habe k Bahngruppencluster mit insgesamt n Bearbeitungsbahnen, $k \leq n$, k, $n \in IN$. Jedes Bahngruppencluster D_{BGi}, $i \in \{1, ..., k\}$ hat genau m Bahnelemente, m = k * n. Der Aufwand zur Verifizierung von D_{EL} ist dann $O(k*m^2)$.

Beweis: Den Aufwand des Verfahrens berechnet man so, wie im Abschnitt 6.4.3. "Komplexitätsreduktion durch Individuation" gezeigt wurde: Man multipliziert die Komplexitäten der Hierarchiestufen. Jedes Bahngruppencluster hat m Bearbeitungsbahnen und besitzt eine Komplexität $O(m^2)$. Das Bahnelementelösungscluster hat k Bahngruppen und eine Komplexität O(k). Es ruft also k mal die Veirifizierung von Bahngruppenclustern auf: $O(k) * O(m^2) = O(k * m^2)$.

Nun kann man drei Fälle unterscheiden. Fall 1: Es findet nur ein Werkzeug Anwendung. Demzufolge gibt es nur eine Bahngruppe, d.h. k = 1. Diese enthält alle n Bearbeitungsbahnen: m = n. Daraus folgt, daß das Verfahren von quadratischer Komplexität $O(n^2)$ ist.

Fall 2: Die Bearbeitungsbahnen verwenden paarweise verschiedene Werkzeuge: m = 1. Die Anzahl der Bahngruppen ist gleich der Anzahl der Bearbeitungsbahnen: n = k. Das Verfahren besitzt einen linearen Suchaufwand O(n).

Fall 3: Das Verhältnis von Bahnelementen je Gruppe m zu Bahngruppen k ist ausgewogen, d.h. $k = m = \sqrt{n}$. Die Komplexität des Verfahrens ist dann $O\left(n\sqrt{n}\right) = O\left(n^{\frac{3}{2}}\right)$.

Wäre ein vergleichbares Programm in Pascal effizienter? Die Überlegungen zur besten Bahnelementekombination gelten für jedes Verfahren. Im Rahmen der Diplomarbeit wurde das Bahnelementeproblem in Turbo-Pascal gelöst. Die Implementierung besitzt die gleiche Komplexität wie der Planer.

Die Realisierung des Programms in Turbo-Pascal nahm einen knappen Monat in Anspruch. Für die hier dargestellte Lösung mit dem Reihenfolgeplaner wurden nur drei Tage benötigt, davon die meiste Zeit für die Anpaßprogrammierung der Sequenzverwaltung. Was folgt daraus?

1. Die Effizienz des Planers hängt entscheidend davon ab, wieviel Wissen über die Domäne zur Verfügung steht.
2. Es konnte für das Bahnelementeproblem experimentell gezeigt werden, daß mit dem Reihenfolgeplaner die Lösungssuche nicht weniger effizient abläuft als mit speziellen Planern. Die Erfahrung zeigt jedoch, daß spezielle Planer nur mit wesentlich größerem Aufwand realisiert werden können.

8.2.3. Vergleich mit bisherigen Zeiten

Das hier verwendete Sortierverfahren basiert auf einer am IAI angefertigten Diplomarbeit von Lotter [Lotter 92]. Es erzeugt stets zulässige Bahnelementesequenzen. Zwischen allen Bahnelementen werden Luftbahnen generiert. Das Verfahren "sukzessives Einfügen" findet eine Anfangslösung, die ein 2- und 3-optimales Verfahren verbessert. Einfachere Suchverfahren führen zu erheblich schlechteren Resultaten. Eine Druckmaschinenseitenwand dient hier als Beispiel.

In der Geometrie des Werkstücks steckt Optimierungspotential: Enge Innenradien verhindern, daß Bahnelement 7 der Abbildung 8.8 von einer Schruppscheibe bearbeitet wird. Für andere Bahnen ist dagegen die Schruppscheibe zulässig und langsameren

Werkzeugen wie dem Fingerfräser vorzuziehen. Bei einer Entscheidung für die Schruppscheibe bei anderen Bahnelementen ist ein Werkzeugwechsel erforderlich.

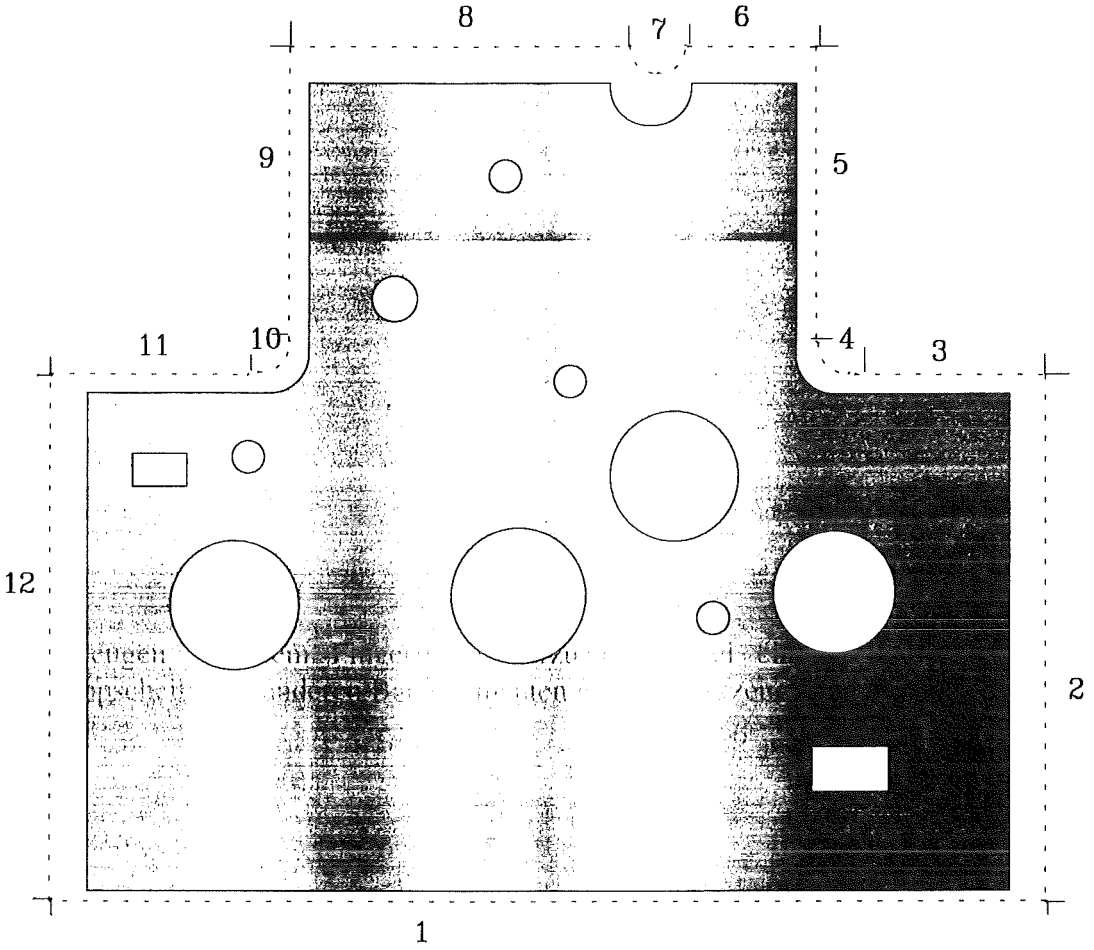


Abb. 8.8: Beispiel für eine eineingelernte Reihenfolge der Bearbeitungsbahnelemente an der HDM-Seitenwand, die beim Einlernen mit einem einzigen Werkzeug entstehen kann.

Wird eine dicke Trennscheibe für die geradlinigen Umfangsgrate 1, 2, 3, 5, 6, 8, 9, 11, 12 und Frässtiften für die konkaven Kurvenzüge 4, 7 und 10 verwendet, dauert eine Bearbeitung in der wahrscheinlich eingelernten Reihenfolge 1-2-3-4-5-6-7-8-9-10-11-12 von 368 Sekunden. Diese Reihenfolge ist für das Einlernen und Messen optimal, nicht aber zum Putzen. Die Bearbeitungszeit bisherige ergibt sich aus:

10s Werkzeug holen

130s Trennen aller geraden Umfangsgrate

120s Werkzeugwechsel (6 mal)

50s Fräsen der konkaven Umfangsgrate

48s Umorientierungen auf Luftbahnen (Drehen der Hand)

10s Werkzeug ablegen

368s Gesamte Bearbeitungszeit

Fünf Werkzeugwechsel können eingespart werden, wenn jedes Werkzeug vor dem Werkzeugwechsel alle Bahnen abfährt, die es bearbeiten kann. So ergibt sich eine Bearbeitungsreihenfolge mit der Trennscheibe 1-2-3-5-6-8-9-11-12-Werkzeugwechsel auf Frässtift 4-7-10. Dabei wurden die Luftbahnen von 3-5, 6-8 und 9-11 eingefügt, die 17

Sekunden kosten. Um die Bahnelemente 4, 7, 10 zu verbinden, wurden weitere Luftbahnen eingefügt, die 17 Sekunden kosteten. Die Summe der Zeiten ergibt sich aus Tabelle 8.27.

Vorgang/Vorgänge	Zeit [s]
Werkzeug holen	10
Trennen der geraden Umfangsgrate	130
Umorientierungen für Trennen (Drehen der Hand)	48
eingefügte Luftbahnen zwischen geraden Umfangsgraten	17
Werkzeugwechsel	20
Fräsen der konkaven Umfangsgrate	50
Luftbahnen zwischen konkaven Umfangsgraten	32
Werkzeug ablegen	10
Gesamte Bearbeitungszeit	**317**

Tab. 8.27: Vorgangszeiten bei optimierter Reihenfolge

Diese Reihenfolge vermeidet Werkzeugwechsel und spart damit 51 Sekunden. Das ursprüngliche Bearbeitungsprogramm dauert 22 % länger als das optimierte.

Bei ausschließlicher Bearbeitung der Teilungsgrate mit dem Fräsmeißel, werden in der vom Benutzer vermutlich eingelernten Reihenfolge 1-2-3-4-5-6-7-8-9-10-11-12 einschließlich der Umorientierungen auf Luftbahnen ca. 128 Sekunden benötigt. Die Zeitdauer unterteilt sich wie in Tabelle 8.28 dargestellt.

Vorgang/Vorgänge	Zeit [s]
Werkzeug holen	10
Fräsmeißeln	60
Umorientierungen auf Luftbahnen (Drehen der Hand)	48
Werkzeug ablegen	10
Bearbeitungszeit für den Teilungsgrat	128

Tab. 8.28: Optimierte Reihenfolge, wenn nur der Fräsmeißel eingesezt wird

Die Reihenfolge der Bahnelemente wurde von der Planung geändert zu 5-7-10-11-3-9-12-6-8-4-2-1. Dafür wurden nur noch 110 Sekunden gebraucht. Mit der einfachen Optimierung wurde die Bearbeitungszeit der HDM-Seitenwand um 18 Sekunden verbessert.

Für weitere Betrachtungen über die Optimierung mit der verwendeten Heuristik, die Zulässigkeit und Anwendbarkeit vgl. [Lotter 92].

8.2.4. Werkzeugwahl bei Bahnelementen

Die bisherigen Überlegungen unterstellen, daß einer Bearbeitungsbahn ein Werkzeug, das sie bearbeitet, eindeutig zugeordnet ist. Der Bediener zum Putzeinlernen hat die Aufgabe, für jede Bahn das Werkzeug auszuwählen. Dabei erhält er vom System keinerlei Unterstützung. Bisher fehlt ein klarer Zusammenhang zwischen den Charakteristika eines Grats und dem Werkzeug, das diesen Grat optimal bearbeiten kann. Optimal bedeutet, daß das Werkzeug nicht mehr Grat stehen läßt als der Kunde erwartet und die Bearbeitung möglichst schnell durchführt. Die Wahl ist nicht einfach, weil grobe Werkzeuge wie der Fräsmeißel zwar sehr schnell arbeiten aber nicht jede Art von Grat und nicht jeden Innenbogen eines Werkstücks zufriedenstellend bearbeiten. Andererseits erzielt ein feines Werkzeug wie der Fingerfräser meist sehr gute Bearbeitungsergebnisse, braucht dafür aber unverhältnismäßig viel Zeit.

Angenommen, es lasse sich an drei Gratcharakteristika und einem Werkstückparameter feststellen, ob ein Werkzeug ein Bahnelement bearbeiten kann. Diese Charakteristika heißen *Parameter-1, Parameter-2, Parameter-3* und *Parameter-4*. Beispiele dafür sind die maximale Grathöhe, Gratdicke, der engste Bogen und das Material des Werkstücks. Sie sind als Anwenderslots der BEARBEITUNGSBAHN, dem WERKSTÜCK und dem WERKZEUG zugeordnet. Beim Messen für die Putzbearbeitung wurde der Grat jeder Bearbeitungsbahn mit dem Scanner eingelesen und entsprechend ausgewertet. Die Charakterstika des Werkstücks sind vorher bekannt. Vor Beginn der Planung hat jede Bearbeitungsbahn mindestens einen Wert in jedem der drei Anwenderslots.

Jedem Werkzeug wird wenigstens ein ZUSTANDSPRÄDIKAT DER VORBEDINGUNG zugeordnet, das überprüft, ob das fragliche Bahnelement die Bedingungen für die Anwendung des Werkzeugs erfüllt. Dazu trägt man vorher in die Anwenderslots Prädikate ein. Im konkreten Fall kann das so aussehen, wie es Abbildung 8.9 darstellt.

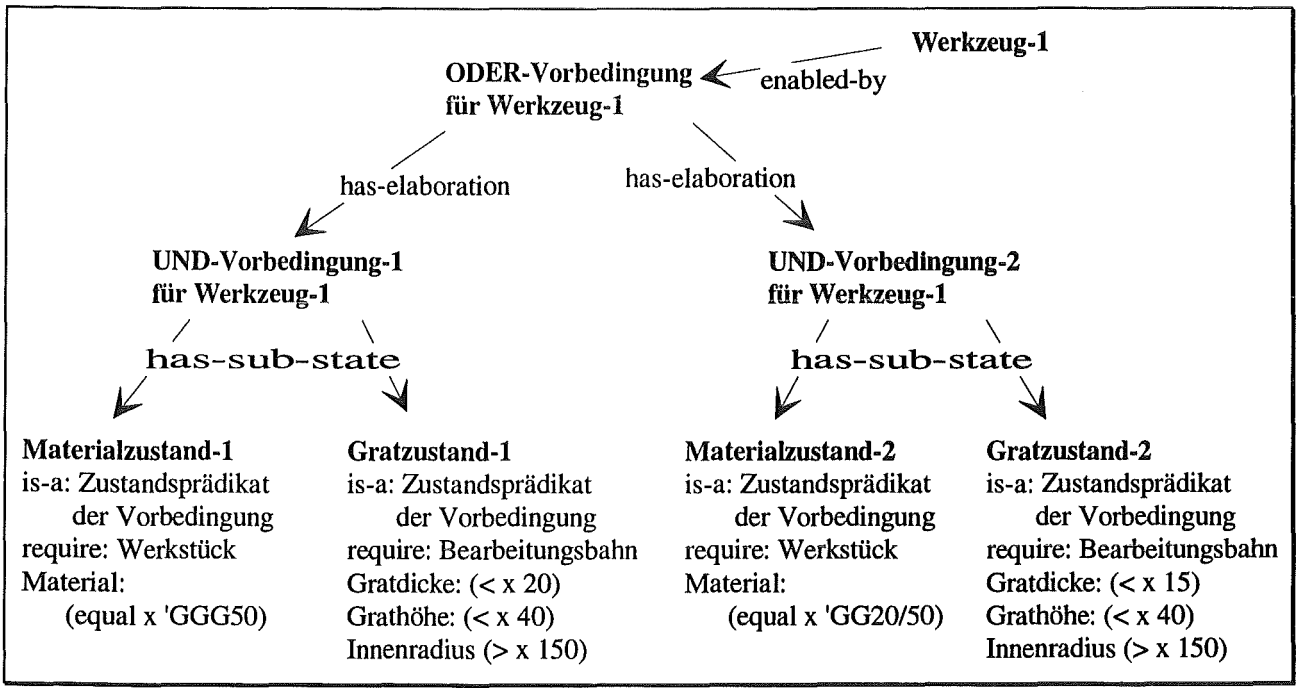


Abb. 8.9: Die drei Charakteristika heißen hier Grathöhe, Gratdicke engster Bogen und Material. Ein Zustandsprädikat überprüft die Charakteristika des Werkstücks, ein weiteres die der Bearbeitungsbahn. Das Werkzeug ist anwendbar, falls entweder das Material des Werkstücks globularer Grauguß GGG 50, die Gratdicke < 20 mm, die Grathöhe < 40 mm und der minimale Innenradius > 150 mm ist oder es sich beim Material um Grauguß GG 20/25 handelt und der Grat nicht mehr als 15 mm Dicke, 40 mm Höhe und mindestens 150 mm Innenradius besitzt.

Um ein Werkzeug auszuwählen, ordnet man ihm eine Nützlichkeit zu, dargestellt durch einen numerischen Wert im Attribut *utility*. Der Wert kann errechnet werden aus Bearbeitungsgeschwindigkeit und dem derzeitigen Verschleiß. Alternativ kann die Bearbeitungsgeschwindigkeit direkt eine Funktion des Verschleißes sein.

Ein Planungsverfahren kann alle denkbaren Werkzeugkombinationen und -zuordnungen zu Bahnelementen durchprobieren. Das wäre sehr aufwendig und besitzt exponentielle Komplexität. Die Suche nach den geeigneten Werkzeugen läßt sich vereinfachen, wenn man von einer Initiallösung ausgeht und fordert, daß jedes zusätzliche Werkzeug einschließlich der zusätzlichen Werkzeugwechselzeit die Gesamtbearbeitungsdauer verkürzen muß. Als Initiallösung wählt man ein Werkzeug, das alle Bahnelemente bearbeiten kann, z.B. den Fingerfräser. Gibt es kein solches Werkzeug, beginnt man mit einer möglichst einfachen Kombination.

Man wählt ein beliebiges Werkzeug und ordnet ihm alle Bearbeitungsbahnen zu, die es bearbeiten kann. Für diese Lösung ermittelt man die Bearbeitungsdauer. Ist sie kürzer, gilt sie als neue Initiallösung. Nun wählt man ein weiteres Werkzeug bis alle verfügbaren Werkzeuge überprüft wurden. Die Lösung am Ende des Verfahrens ist lokal, aber nicht unbedingt global optimal, weil nicht jede denkbare Kombination von Werkzeugen untersucht wurde.

8.3. Beispiel aus der Projektplanung

Im Beispiel für die COMETOS-Fertigungszelle treten keine wirklich komplizierten Zeitverhältnisse auf und der Vorteil der Zeitpropagierung für die Reihenfolgeplanung wird nicht unmittelbar klar. Deswegen wird an einem simplifizierten Projektplanungsbeispiel gezeigt, wo die Stärken der Zeitplanung liegen.

Ein Projekt bestehe aus den Vorgängen 1 bis 7. Folgende Informationen seien bekannt:

- Vorgang 1 unmittelbar vor Vorgang 5, mittelbar vor[3] Vorgängen 2 sowie vor, unmittelbar vor, während, überlappt oder zeitgleich zu Vorgang 4; Dauer: 1 h. Start: 8.00 h.
- Vorgang 2 vor Vorgang 3
- Vorgang 3: Dauer: 1 h, Ende: 14:00 h
- Vorgang 4 während Vorgang 3 und geht von 10.00 h bis 12.00 h. Vorgang 4 nicht nach Vorgang 3
- Vorgang 5 unmittelbar vor Vorgang 3
- Vorgang 6 geht von 10.30 h bis 11.00 h
- Vorgang 7 beginnt frühestens um 10.45 h, dauert 50 Minuten und endet spätestens um 11.45 h

[3]unmittelbar vorher = "meets", vgl. Kapitel 6.5.1

- Die Planung soll für spätere Entscheidungen aufgrund von Störfaktoren möglichst wenig festlegen.

Mit Allens Propagation entsteht das Netzwerk von Abbildung 8.10.

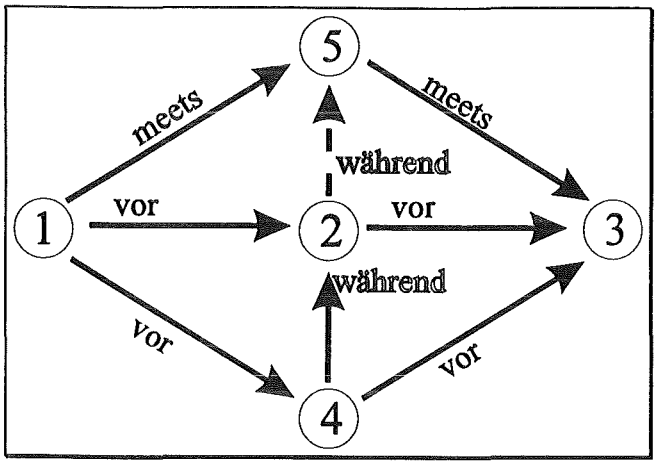


Abb. 8.10: Allens Propagation fügt u.a. den gestrichtelten während-Pfeil ein. Die Relationen des Wegs 1 - 4 - 1 wurden errechnet. Weil sie in keiner bekannten zeitlichen Beziehung zu diesem Netzwerk stehen, fehlen hier die Knoten 6 und 7.

Die Darstellung würde unübersichtlich, wollte man alle sich ergebenden Beziehungen darstellen. Deswegen werden hier nur die wichtigsten dargestellt. Die in Kapitel 6 entwickelte Netzwerkreduktion erlaubt nun eine wesentliche Vereinfachung dieses Netzwerks, indem Relationen gestrichen werden (vgl. Abbildung 8.11).

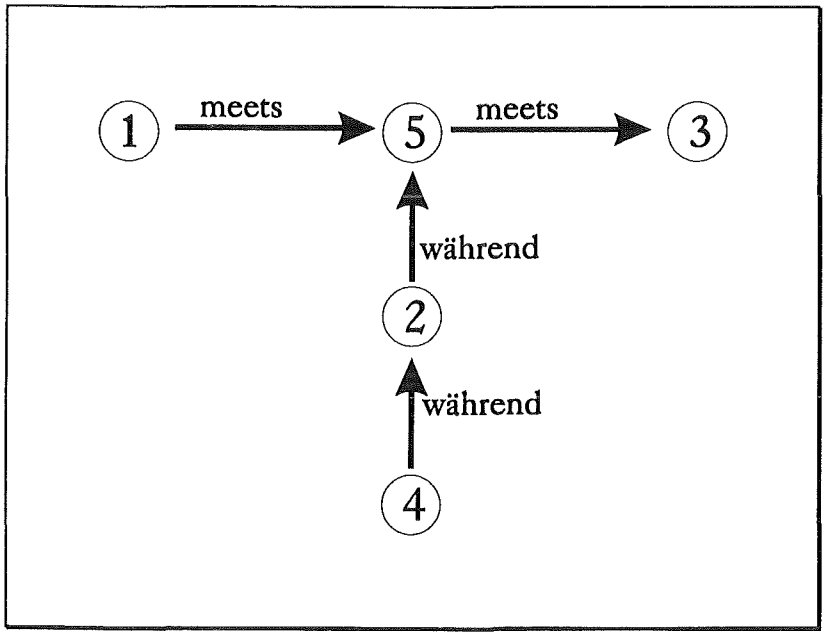


Abb. 8.11: Diese Vereinfachung entstand durch das Kreuzen zweier transitiver Ketten. Die Anzahl der Beziehungen wurde verringert und das Netzwerk wesentlich einfacher.

Weil der hier vorgestellte Ansatz die Propagation fester Zeiten integriert, haben die Start- und Endpunkte der Vorgänge 6 und 7 Konsequenzen für ihre Zeitbeziehungen, vgl. Abbildung 8.12.

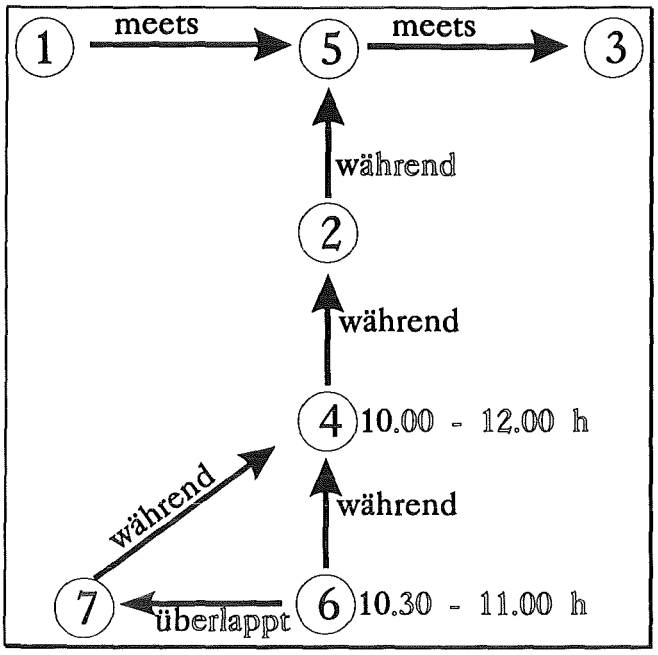


Abb. 8.12: Aus Zeitrelationen folgen Grenzen für feste Zeiten und umgekehrt aus festen Zeiten Zeitbeziehungen. Vorgang 7 startet zwischen 10.00 h und 14.00 h. Er überlappt eindeutig mit Vorgang 6 und ist während[4] Vorgang 4.

Die Propagierung der festen Zeiten und die Rückkopplung zur Propagation von Relationen ist mit keinem anderen Planer möglich. Einige weitere Ergebnisse der Planung sind:

- Vorgang 5 dauert von 9.00 - 13.00 h.
- Vorgang 2 dauert mehr als zwei und weniger als vier Stunden. Er startet nach 9.00 und vor 10.00 h und endet nach 12.00 und vor 13.00 h.

Mit bisherigen Planern wären derartige Ergebnisse nur über große Umwege möglich gewesen, weil sie entweder nicht alle Zeitbeziehungen darstellen können und unscharfe Angaben wie Intervalle für den Start- oder Endzeitpunkt nicht verarbeiten können. Typische CP/M-Planer z.B. wären weder in der Lage, das Netz entsprechend zu vereinfachen, noch können sie z.B. eine *overlaps*-Beziehung handhaben. Bisherigen nicht-linearen Zeitplanern wäre der Übergang zu festen Zeitbeziehungen nicht gelungen. Keiner der Planer hätte aus einem relativ unübersichtlichen Netzwerk der Abbildung 8.38 durch Reduktion ein übersichtliches Netzwerk wie in Abbildung 8.39 herleiten können.

8.4. Vergleich der Ergebnisse mit den bisherigen Möglichkeiten

Ein übergreifender Ansatz formalisiert und löst drei sehr unterschiedliche Reihenfolgeprobleme. Das Beispiel der Maschinenbelegung zeigt, daß die Darstellung vollständig, präzise und klar ist (vgl. S.128). Die Aussagen sind auf die weniger komplizierten Beispiele aus der Bahnelementeoptimierung und Projektplanung sofort übertragbar. Die Darstellung ist leicht erweiterbar und besitzt transparente Strukturen, die eine Problemmodellierung in der Denkweise des menschlichen Planers erlauben. Um das zu verdeutlichen, kann man überlegen, wie eine konventionelle Implementierung z.B. in Turbo-Pascal aussähe. Tabelle 8.29 vergleicht den Ansatz mit einer solchen Realisierung.

[4] Auf die Unterscheidung zwischen during, starts, finishes wurde hier aus Gründen der Anschaulichkeit verzichtet. Im Ergebnis hat sie hier keinerlei Konsequenzen.

Problem	allgemeines Reihenfolgeplanungs-werkzeug	konventionelle Implementierung
Maschinenbelegungsplanung	1. Modellierung: Der Anwender generiert 50 A/S-Objekte für den Arbeitsplan und ordnet sie an. Von Planungsobjekten schafft er Unterklassen, denen er Attributwerte zuweist. Zur Generierung reicht der Aufruf der entsprechenden Methode. 2. Verfahren: Die beiden Verfahren JOB-SUCHE und TAGESPRODUKTIONEN-SORTIERUNG sind so speziell meist nicht übertragbar. Der Anwender muß hier die Implementierung anpassen. Alle weiteren Verfahren stellt das System bereit. 3. Erweiterung: Das System kann ohne zusätzliche Programmierung erweitert werden. Im Beispiel des Werkzeugverschleißes muß man den Aktivitäten nur zusätzliche Bedingungen zuweisen. Zusatzaufwand: 0,5 Manntage. Gesamtaufwand: je nach Programmiererfahrung 2 - 5 Manntage.	Das Beispielproblem ist bereits sehr komplex. Das bedeutet, daß die Ergebnisse auch in einer konventionellen Umgebung nur durch Simulation berechnet werden können. Jede Bedingung, die das Planungswerkzeug als Objekt darstellt, wird bei konventioneller Programmierung im besten Fall eine IF-Abfrage, meistens aber eine eigene Prozedur. Sauberen Programmierstil vorausgesetzt ergibt das mindestens 35 Prozeduren, die die 35 Zustände ersetzen. Dabei sind die Such- und Sortierverfahren noch nicht mitgerechnet, die durch die vielfältigen Alternativen innerhalb des Arbeitsplans entstehen. Ob ein derartiges Programm noch transparent genug ist, um im Nachhinein auch den Werkzeugverschleiß zu integrieren, ist fragwürdig. Programmstücke aus den anderen beiden Problemen sind nicht verwertbar. Geschätzter Gesamtaufwand: mindestens zwei Monate.
Bahnelementeoptimierung	1. Modellierung: Zur Modellierung ist eine dreistufige Hierarchie erforderlich, die in etwa einer halben Stunde realisierbar ist. 2. Verfahren: Neben dem sukzessiven Einfügen, der 2- und 3-optimalen Verbesserung braucht man ein Programmstück, das Bahnelemente zu Bahngruppen zusammenfaßt. Gesamtaufwand: 2 - 4 Tage	Neben dem Eröffnungs- und Verbesserungsverfahren sind Tests erforderlich, um Kurzzyklen zu vermeiden, kein Element doppelt aufzunehmen usw. Programmstücke aus den anderen beiden Problemen sind nicht verwertbar. Geschätzter Aufwand: 1 - 2 Wochen
Projektplanung	Lediglich die Aktivitäten mit Zeitfenstern und Zeitbeziehungen müssen erklärt werden. Das Verfahren hat eine Laufzeit von O(n4). Gesamtaufwand: ca. 1 Stunde	Keine Vergleichsmöglichkeit, weil das Problem mit herkömmlichen Methoden nicht lösbar ist.

Tab. 8.29: Vergleich des Realisierungsaufwands beim Reihenfolgeplanungsansatz und bei herkömmlicher Programmierung

Die Übersicht verdeutlicht, daß gleiche Probleme mit konventionellen Methoden wesentlich aufwendigerer Lösungen bedürfen. Dabei gehen oft Transparenz und Übersichtlichkeit des Programms verloren. Dagegen entwickelt der Reihenfolgeplaner die Verfahren auf Basis der Problemstruktur. Deswegen ist er offen gegenüber Modifikationen und Erweiterungen.

Der vorliegende Ansatz ist nicht komplexer als andere Verfahren. Der zugrundeliegende Digraph ist stets ein gerichteter Baum, auf den alle gängigen Such- und Sortierverfahren anwendbar sind. Das Beispiel der Bahnelementeoptimierung zeigt, daß sich daraus Möglichkeiten zur Komplexitätsreduktion ergeben, die das System sofort nutzt. Gerade die Strukturierung deckt solche Chancen auf und unterstützt den Anwender, der die Komplexität optimieren möchte.

9. Zusammenfassung

Bis heute gibt es kein Werkzeug, das speziell die Reihenfolgeplanung unterstützt. Allgemeinere Hilfsmittel wie Computersprachen bieten wenig Hilfe, spezielle Werkzeuge wie Modellierungsumgebungen für CAM-Probleme decken nicht die Bandbreite der Probleme ab.

Andererseits steigen ständig die Aufwendungen für Investitionen im Bereich der Werkzeugmaschinen. Maschinenstillstandszeiten werden immer teurer. Eine Chance zur Vermeidung von Stillständen liegt in der Arbeitsorganisation. Allein durch Umordnen von Reihenfolgen kann eine Produktion rentabler werden. Ein gutes Beispiel dafür ist die COMETOS-Fertigungszelle, die für konventionelle Planungssoftware wenig Optimierungspotential besitzt. Ihre Reihenfolgeprobleme sind sehr unterschiedlich. Ein System zur Reihenfolgeplanung, das sich ohne großen Aufwand auf die Problematik maßschneidern läßt, ist eine Chance, die Planung zu automatisieren.

9.1. Stand der Forschung

Seit Jahrzehnten werden Reihenfolgeprobleme im Operations Research und in der Informatik erforscht. Die wichtigsten Ergebnisse sind:

- hierarchische und nichtlineare Planer sowie Metaplaner,
- Netze zur Repräsentation von Nebenbedingungen wie Petri-Netze und A/S-Netze,
- Netze zur Repräsentation von Hierarchien,
- Kalküle zum temporalen Schließen,
- eine Reihe von Heuristiken zur anwendungsspezifischen Erzeugung von Reihenfolgen,
- intelligente Suchverfahren.

Nachteil der hauptsächlich aus dem OR kommenden Methoden ist ihr hoher Grad an Spezialisation. So entstand die Idee zu einem allgemeineren Werkzeug für die Reihenfolgeplanung. Kernpunkte der Bearbeitung sind:

- eine allgemeine Wissensrepräsentation zu erarbeiten und zu formalisieren. Sie soll helfen, Probleme zu formulieren und Lösungsverfahren zu entwickeln.
- ein Schema zu finden, in das gängige Verfahren zur Reihenfolgeplanung integriert werden können.
- einen mächtigen Zeitplaner zu entwickeln, der auch komplexe Zeitverhältnisse handhabt, wie sie in Reihenfolgeproblemen auftreten können.

9.2. Wissensrepräsentation

Objekte sind Mitglied zweier Hierarchien:

- Eine begriffliche Hierarchie ordnet sie in einen Zusammenhang mit anderen Objekten ein, z.B. *Roboter* als Unterbegriff von *Maschine*.
- Eine weitere Hierarchie weist einem Objekt eine Rolle bei der Problemlösung zu, die sich im Lauf der Planung ändern kann.

Beide Hierarchien sind unabhängig voneinander und leicht zu erweitern. Das gleiche gilt für die Beziehungen zwischen Objekten. Die Wissensrepräsentation wird damit sehr

transparent und ist einfach adaptierbar. Sie stellt alle Arten von Objekten einschließlich der Methoden einheitlich dar.

Die erarbeitete Wissensdarstellung formalisiert und erweitert die an der Carnegie-Mellon-Universität erarbeitete Repräsentation von Aktivitätswissen als Digraphen. Sie integriert zwei Grundformen der Problemdarstellung von Reihenfolgeproblemen. Im Gegensatz zur üblichen Darstellung als lineares Optimierungsproblem kann sie Reihenfolgeprobleme mit vielfältigen Beziehungen in wesentlich weniger und anschaulicheren Nebenbedingungen formulieren. Das wurde bewiesen. Für das Travelling-Salesman-problem sind z.B. 4 * (n-1)(n-2)/2 Nebenbedingungen erforderlich, bei der üblichen Darstellung als lineares Optimierungsproblem steigt die Anzahl der Nebenbedingungen exponentiell. Darüberhinaus kann man auch Lösungsverfahren in die Darstellung problemindividuell integrieren, indem man Knoten geeignete Verfahren zuordnet.

Mischformen von Reihenfolgeproblemen stellen nicht nur an die effektive Suche im Suchraum hohe Anforderungen, sondern auch an den Aufbau des Suchraums selbst. Die Frage, ob eine Erweiterung der Teillösung zulässig ist, wird zu einem nichttrivialen Problem. Eine Stärke der entwickelten Darstellung ist, daß sie beide Probleme, das Generieren des Suchraums und die Suche darin unterstützt. Man kann Mischformen von Problemen oder Probleme aus der Praxis, die sich selten auf klassische Reihenfolgeprobleme direkt abbilden lassen, mit wesentlich geringerem Aufwand vollständig, präzise und klar darstellen.

Die Darstellung von Zeiten erweitert das gängige Konzept der Normen für CP/M-Netze um die Begriffe minimale und maximale Dauer.

9.3. Inferenzen

Die ausgewählten und weiterentwickelten Inferenzen sind zum größten Teil domänenunabhängig und lassen sich leicht spezialisieren. Sie planen Reihenfolgen, indem sie Knoten in einem Digraphen verifizieren.

Ein problemindividuelles Lösungsverfahren bettet man in ein allgemeines Schema zur Propagation ein. Der Anwender kann spezielle Lösungsverfahren einfach umsetzen und testen. Zwischen Aufwand und Güte der Lösung kann er für jedes Teilproblem neu abwägen. Dabei verbessert er die Leistung des Gesamtsystems. Besonders bei Problemen aus der Praxis oder Mischformen von Reihenfolgeproblemen ist das attraktiv, weil sich dort Teilprobleme oft mit geringerem Aufwand lösen lassen als das globale Problem. Auch das wurde gezeigt.

Der neuartige Zeitplaner arbeitet nichtlinear und verarbeitet durchgängig quantitative wie qualitative Zeitinformationen. Um die Laufzeit zu verbessern und die Transparenz zu erhöhen, vereinfacht er qualitative Zeitinformationen stärker als bisherige Verfahren. Er kann über Zeitpunkte und Dauern inferieren, dabei Vorgaben durch Intervalle berücksichtigen und durch einen Digraphen mit beliebigen zeitlichen Relationen propagieren. Besonders betont sei die logische Durchgängigkeit der Zeitpropagation sowie seine Komplexität von $O(n^4)$.

Die Propagationsverfahren ergänzen sich und sind auf die einheitliche Wissensrepräsentation abgestimmt. Zur Wahrheitserhaltung verwenden sie Kontexte, die ein Backtrackingalgorithmus an jedem Entscheidungsknoten generiert bzw. löscht. Die wichtigsten Punkte sind:

1. Ein Konzept vermindert Redundanzen in temporalen Digraphen, wenn es sie nicht völlig vermeidet.
2. Feste Zeiteinheiten können zum ersten Mal durch einen temporalen Digraphen über alle Zeitbeziehungen hinweg propagiert werden.

3. Ein nichtlinearer, durchgängiger Propagierungsansatz behandelt qualitative und quantitative Zeitinformationen einheitlich.
4. Lösungsverfahren kann man in ein allgemeines Schema einbetten.
5. Der Ansatz erweitert A/S-Netze um Prüfprädikate und dynamische Revisionen.

Drei Beispiele zeigen, daß der Reihenfolgeplaner mit sehr unterschiedlichen Problemen fertig wird. Maschinenbelegungsplanung und die Optimierung von Bahnelementen, beides Beispiele aus der Anwendung von COMETOS, sowie ein einfaches Beispiel aus der Projektplanung hat der Planer einheitlich dargestellt und gelöst. Hinsichtlich der Lösungsgüte und Komplexität steht er konventionellen Planern nicht nach, weil er Lösungsstrategien konventionaler Planer ohne Laufzeitverlust verwenden kann. Der Unterschied ist, daß ein Anwender mit dem vorliegenden Darstellungsansatz Lösungen schneller entwickelt. Er modelliert die Problemwelt und weist ihren Objekten geeignete Verfahren zu.

	vorher	jetzt
Darstellung von Reihenfolgeproblemen	unterschiedliche Ansätze, je nach Problemtyp	eine einheitliche Darstellung für Reihenfolgeprobleme
	Trennung von Darstellung des Problems und der Lösung	Integration von Problem und Lösungsstrategie durch Zuweisung von Verfahren zu Knoten in einem gerichteten Baum
Komplexität der Darstellung	Die Formulierung als lineares Optimierungsproblem benötigt für Reihenfolgeprobleme z.B. beim Travelling Salesman Problem eine exponentielle Anzahl von Nebenbedingungen	Übersichtlichere Darstellung: beim Travelling Salesman Problem z.B. müssen nur alle existierenden Pfeile dargestellt werden.
Lösungsverfahren	nach Problemtyp klassifiziert	Aufgespalten in Funktionalitäten, um die Entwicklung neuer Verfahren zu unterstützen
Wissen	unterschiedlich dargestellt oder propagiert	einheitliche Darstellung und Propagation
Darstellung fester Zeiten	Zeitfenster für Start- und Endpunkte oder Dauern, nicht aber beides	Start-, Endzeitpunkte, Dauern jeweils in Form von Fenstern, d.h. mit Constraints
Komplexität	abhängig vom gewählten Verfahren	Ein beliebiges Verfahren zur Reihenfolgeplanung läßt sich integrieren. Die Komplexität hängt von diesem Verfahren ab.

Tab. 9.1: Fazit für die Reihenfolgeplanung:

	vorher	jetzt
Redundanzen in Zeitnetzen	transitive vorher- und nachher-Ketten löschten redundante Verbindungen	ganze transitive Ketten werden als Redundanzen erkannt und entfernt
Unvollständige Zeitinformation	nur qualitativ zulässig	qualitativ und/oder quantitativ zulässig
Zeitpunkte und Dauern	über vorher/nachher-Beziehungen	über jede denkbare Zeitbeziehung, nicht-lineare Planung Planung auch über deren Constraints möglich
Integration qualitative/quantitative Propagierung	keine	in beide Richtungen

Tab. 9.2: Fazit für die Zeitpropagation:

9.4. Erweiterungsmöglichkeiten

Mit dem hier entwickelten Reihenfolgeplaner wird es einfacher, Reihenfolgeprobleme zu modellieren und Verfahren zu optimieren. Mit einer zusätzlichen Bibliothek von Lösungsverfahren wird sich die Arbeit des Anwenders darauf beschränken, Verfahren zusammenzustellen und Schnittstellen zu schreiben. Das hier vorgestellte Modell ist trotz seiner Vielfalt einfach und für einen OR-Experten oder Informatiker verständlich. Der Verfasser arbeitet derzeit an drei Erweiterungen, zu denen er bald Publikationen plant:

- Ein Hopfield-Netzwerk kann die Suche nach geeigneten Lösungsverfahren automatisieren. Der Anwender muß dann lediglich das Problem und die Ziele formulieren.
- Wenn Zeitwerte nur vage bekannt sind, stellt man sie besser unscharf dar. Dafür ist eine einfache Darstellung gesucht, um die Laufzeit des Zeitplaners nicht unnötig zu erhöhen.
- Alternativen zwischen Vorgängen sollte der Planer ebenfalls berücksichtigen können. Dazu muß man die Logik der Zeitplanung erweitern.

Die vorliegende Arbeit ist nicht als Lehrbuch gedacht. Die in Kapitel 8 gezeigten Beispiele sind nur ein Ausschnitt der Anwendungsmöglichkeiten. Ordnet man sie in einen größeren Zusammenhang ein, kann man sie hinsichtlich der Realisierung wesentlich erweitern.

Der nicht-lineare Zeitplaner findet dort Einsatzgebiete, wo bisher mit konventionellen Zeitplanungsverfahren gearbeitet wird. Im Gegensatz zu herkömmlichen Planern kann er unvollständiges Wissen verarbeiten. Im Anlagenbau und in der Montage sind selten alle Zeitbeziehungen, geschweige denn alle quantitativen Zeitinformationen von vorneherein bekannt. Ein konventioneller Planer hat damit Probleme. Der nicht-lineare Planer nutzt die vorhandenen Informationen und zeigt Freiräume an. "Kritische Pfade" sind damit überflüssig. Kommt später zeitliches Wissen hinzu, kann er es verarbeiten und den Plan konkretisieren. Weitere Einsatzgebiete sind die Logistik sowie Aufgaben im kommerziellen Bereich, z.B. bei Mergers und Akquisitions oder beim Gang zur Börse. Beide Bereiche besitzen komplexe Pläne mit Vorgängen, die über vielfältige Zeitbeziehungen miteinander verwoben sind.

Der Reihenfolgeplaner kann als Komponente für die Steuerung von Fertigungszentren und -straßen eingesetzt werden. Innerhalb des CIM-Konzepts plant man auf verschiedenen Ebenen. Der vorliegende Planer kann unterschiedliche Planungsprobleme

zusammenfassen. Will man etwa innerhalb einer Fertigungsstraße nicht nur die Belegung der Maschinen planen, sondern auch einen günstigen Arbeitsplan für neue Werkstücke finden, dann ist die Trennung der Probleme in Maschinenbelegungsplanung und Kürzeste-Wege-Problem für den Anwender nicht immer einsichtig. Mit diesem Planer kann er seine Aufgabe als ein zusammenhängendes Problem formulieren und lösen. Neben den verschiedenen Planungsebenen der PPS bieten sich auch im Verkehrsbereich interessante Anwendungsmöglichkeiten. Im Flug- und Bahnverkehr muß man oft bei der Wahl einer Route ihre momentane Belastung mitberücksichtigen. Ein kurzer Weg ist deswegen nicht immer der schnellste. Die Belastung einer Route ist jedoch das Ergebnis der Wegebelegungsplanung, die Probleme beeinflussen sich gegenseitig. Deswegen wünscht man sich einen integrativen Ansatz. Der Reihenfolgeplaner kann beide Probleme gemeinsam darstellen und erlaubt, Verfahren zu wählen, die die Dynamik von Belegungsplanung und Routenplanung berücksichtigen.

Für derart große Probleme muß der Planer auf eine andere Softwareplattform portiert werden. Eine schnelle Sprache wie C++ oder Smalltalk sollte KnowledgeCraft ersetzen. Für den Reihenfolgeplaner und separat für den Zeitplaner wird man sich eine Oberfläche wünschen, damit ein Anwender ausschließlich mit Maus- und Menüoperationen sein Problem formulieren kann. Parallelrechner bieten sich zur weiteren Verbesserung an, weil der Reihenfolgeplaner seine Probleme hierarchisch zerlegt.

Der hier entwickelte Formalismus schließlich erlaubt, einen neuen Standard zur Formulierung von Reihenfolgeproblemen zu entwickeln, der der Lösung eher entgegenkommt als die bisherige Formulierung. Daraus kann man die Klassifizierung von Problemen formal standardisieren und entsprechende Lösungsverfahren entwickeln. Der Mensch wird damit Planungsprobleme leichter lösen können. Dazu einen Beitrag geleistet zu haben ist der Wunsch des Verfassers.

Literaturverzeichnis

[Allen 83] . Allen, J. F.: Maintaining Knowledge about Temporal Intervalls. Communications of the ACM, Vol. 26, No. 11, S. 832 - 843. 1983.

[Arff 89] Arff, St.; Hasle, G.: Alplan: A knowledge-based system for production planning in aluminium foundries. Proceedings of the 3rd international conference on expert systems and the leading edge in production and operations management, Hilton Head Island, S. 603 - 618. 1989.

[Arff 91] Arff, St.; Hasle, G.; Stokke, G.; Gjerlow, J. C.: AI Approaches to Production Management. Expert systems with applications. New York, NY. 3.1991.2, S. 229 - 239. 1991.

[Arora 80] Arora, R. K.; Rana, S. P.: Scheduling in a Semi-Ordered Flow-Shop Without Intermediate Queues. AIIE Transactions Vol. 12, S. 263 - 271. 1980.

[Baker 74] Baker, K. R.: Introduction to sequencing and scheduling. John Wiley & Sons, Inc.. 1974.

[Baker 84] Baker, K. R.: Sequencing Rules and Due-Date Assignments in a Job Shop. Management Science Vol. 30 No 9 S. 1093 - 1104. 1984.

[Baudinet 89] Baudinet, M.: Temporal logic programming is complete and expressive. Proceedings of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas. 1989.

[Baumgarten 90] Baumgarten, B.: Petri-Netze. BI Wissenschaftsverlag, Mannheim, Wien, Zürich. 1990.

[Bellmore 68] Bellmore, M.; Nemhauser, G. L.: The Traveling Salesman Problem: A Survey. Operations Research 16, H. 3, S. 538 - 558. 1968.

[Britt 87] Britt, D. L.; Geoffroy, A. L.; Schaefer, P. R.; Gobring, J. R.: Scheduling spacecraft operations. Third Conference on AI for Space Applications, Huntsville, AL. 1987.

[Brucker 81] Brucker, P.: Scheduling. in: Jürgen Perl (Hrsg.): Studientexte Informatik, Akademische Verlagsgesellschaft Wiesbaden. 1981.

[Brzoska 89] Brzoska, C.: Temporal Logic Programming. A Survey. Universität Karlsruhe, Fakultät für Informatik. 1989.

[Calvo 89] Calvo, P.; Carminati, T.; Fraternali, P.; Miriello, S.: CIBELE, a Knowledge Based System for Short-Term Production Scheduling. The Second International Conference on Industrial and Engineering Applications of Artificial Intelligence & Expert Systems. Tullahoma, Tenness., S. 339 - 343. 1989.

[Camurri 90] Camurri, A.; Frixione, M.: A Hybrid System for the Hierarchical Control of FMS. in: Rzevski, G. (ed.): Applications of Artificial Intelligence in Engineering V. Springer-Verlag, S. 47 - 66. 1990.

[Canzi 90] Canzi, U.; Guida, G.; Maifredi, G.; Paolucci, E. et al.: CRONOS-III: Requirements for a knowledge-based scheduling tool covering a broad class of production environments. in: Gottlob, G., Nejdl, W. (eds.): Expert Systems in Engineering, S. 168 - 175. 1990.

[Carlier 84] Carlier, J.; Chretienne, Ph.; Girault, C.: Modelling scheduling problems with timed petri nets. in: Rozenberg, G. (Ed.): Advances in Petri Nets. Springer Verlag, S. 62 - 82. 1984.

[Chapman 87] Chapman, D.: Planning for Conjunctive Goals. Artificial Intelligence 32, S. 333 - 337. 1987.

[Choong 86] Choong, Y. F.; Maimon, O. Z.: On dynamic routing in FMS. IEEE International Conference on Robotics and Automation, S. 1476 - 1481. 1986.

[Christofides 69] Christofides, N.; Eilon, S.: An Algorithm for the Vehicle-dispatching Problem. Operational Research Quarterly 20, S. 309 - 319. 1969.

[Christofides 72] Christofides, N.; Eilon, S.: Algorithms for Large-scale Travelling Salesman Problems. Operational Research Quarterly 23, S. 511 - 518. 1972.

[Christofides 79] Christofides, N.; Mingozzi, A.; Toth, P.; Sandi, C. (eds.): Combinatorial Optimization. John Wiley & Sons, Chichester, New York, Brisbane, Toronto. 1979.

[Coad 91] Coad, P.; Yourdon, E.: Object-Oriented Analysis. 2nd Edition. Yourdon Press, Prentice Hall Building, Englewood Cliffs, New Jersey 07632, 1991.

[Collinot 88] Collinot, A.; Le Pape, C.; Pinoteau, G.: SONIA: a knowledge-based scheduling system. Artificial Intelligence in Engineering, Vol. 3, No. 2, S. 86 - 94. 1988.

[Conway 67] Conway, R. W.; Maxwell, W. L.; Miller, L. W.: Theory of Scheduling. Addison-Wesley. 1967.

[Copas 90] Copas, C.; Browne, J.: A rule-based scheduling system for flow type assembly. International Journal on Production Research, Vol. 28, No. 5, S. 981 - 1005. 1990.

[Croes 58] Croes, G. A.: A method for solving Traveling-Salesman Problems. Operations Research 6, S. 791 - 812. 1958.

[Dagli 89] Dagli, C. H.; Hoffman, G. E.: Application of Distributed Knowledge Bases in Intelligent Manufacturing. Proceedings of the IEEE International Conference on Systems Engineering, Dayton, Ohio, S. 239 - 242. 1989.

[Davis 85] Davis, L.: Job Shop Scheduling with Genetic Algorithms. Proceedings of an international conference on genetic algorithms and their applications. Pittsburgh, PA. 1985.

[Davis 87] Davis, L.; Ritter, F.: Schedule Optimization with Probabilistic Search. Conference on artificial intelligence applications. 3, Kissimmee, Fla.. 1987.

[Day 71] Day, J. E.; Hottenstein, M. P.: Review of Sequencing Rules. S. 11 - S. 39. Springer 1971.

[Dempster 82] Dempster, M. A. H.; Lenstra, J. K.; Rinnooy Kan, A. H. G.: Deterministic and Stochastic Scheduling. Proceedings of the NATO advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems. Held in Durham, England, July 6-17, 1981. D. Reidel Publishing Company, Dordrecht, Boston, London. 1982.

[Dincmen 84] Dincmen, M.; Cebi, T.: Prioritätsregeln in der Werkstattfertigung. wt - Zeitschrift für industrielle Fertigung 74(1984) 85-88. 1984.

[Domschke 85] Domschke, W.: Logistik: Rundreisen und Touren. R. Oldenbourg Verlag München Wien. 1985.

[Dorn 89] Dorn, J.: Wissensbasierte Echtzeitplanung. Friedr. Vieweg & Sohn, Braunschweig/Wiesbaden. 1989.

[Doulgeri 87] Doulgeri, Z.; Hibberd, R. D.; Husband, T. M.: The Scheduling of Flexible Manufacturing Systems. 1987.

[Engelke 88] Engelke, H.; Spur, G.; Hirn W.; Viehweger, B.: Untersuchung von Maßnahmen zur Produktivitätssteigerung einer Fertigungslinie durch rechnergestützte Simulation. Zeitschrift für wirtschaftliche Fertigung 78, 8, S. 368 - 374. 1988.

[Fisher 82] Fisher, M. L.: Worst-Case Analysis of Heuristic Algorithms for Scheduling and Packing. in: Dempster et al.: Deterministic and Stochastic Scheduling. D. Reidel Publishing Company. 1982.

[Fox 84] Fox, M. S.; Smith, S. F.: ISIS - a knowledge-based system for factory scheduling. Expert Systems, Oxford 1(1), S. 25 - 49. 1984.

[Fox 85] Fox, B. R.; Kempf, K. G.: Opportunistic Scheduling for Robotic Assembly. in: International conference on robotics and automation 2. St. Louis, MO, S. 880 - 889. 1985.

[Fox 90] Fox, M. S.; Sycara, K. P.: Overview of CORTES: a constraint based approach to production planning, scheduling and control. in: Proceedings of the Fourth International conference on Expert Systems in Productionand Operations Management, Hilton Head, South Carolina. 1990.

[Freedman 88] Freedman, P.; Malowany, A.: The Analysis and Optimization of Repetition within Robot Workcell Sequencing Problems. International Conference on Robotics and Automation, S. 1278 - 1281. 1988.

[Freedman 90] Freedman, P.; Alami, R.: Repetitive Sequencing: from workcell tasks to workcell cycles. IEEE Conference Proceedings on Robotics and Automation, S. 1962 - 1967. 1990.

[Freedman 91] Freedman, P.: Time, Petri Nets, and Robotics. IEEE Transactions on Robotics and Automation, Vol. 7, No. 4, S. 417 - 433. 1991.

[Friedland 79] Friedland, P.: Knowledge-Based Experiment Design in Molecular Genetics. International joint conference on artificial intelligence. 6, S. 285 - 287, Tokyo, 20. - 23.8.1979.

[Garfinkel 79] Garfinkel, R. S.: Branch and Bound Methods for Integer Programming. in: Christofides, N.: Combinatorial Optimization, S. 1 - 20. 1979.

[Georgeff 89] Georgeff, M. P.; Ingrand, F. F.: Decision-Making in an Embedded Reasoning System. International joint conference on artificial intelligence. 11, Detroit, Mich., S. 972 - 978. 1989.

[Gilhooly 82] Gilhooly, K. J.: Thinking. Academic Press Inc., London. 1982.

[Gilmore 89] Gilmore, J. F.; Williams, D. L.; Thornton, S.: A Knowledge-Based Approach to Planning and Scheduling. Proceedings of the Society of Photo-Optical Instrumentation Engineers. Redondo Beach, Calif 1095, S. 906 - 919. 1989.

[Glover 89] Glover, F.; Greenberg, H. J.: New approaches for heuristic search: A bilateral linkage with artificial intelligence. European Jornal of Operational Research 39, 119-130. North Holland. 1989.

[Grant 86] Grant, T. J.: Lessons for O.R. from A.I.: A scheduling Case Study. Journal of the Operational Research Society, Vol. 37, No. 1, S. 41 - 57. 1986.

[Grefenstette 85] Grefenstette, J.; Gopal, R.; Rosmaita, B.; Van Gucht,D.: Genetic Algorithms for the Traveling Salesman Problem. Proceedings of an international conference on genetic algorithms and their applications. Pittsburgh, PA. 1985.

[Groh 68] Groh, H.: Lokale Enumeration - Ein Verfahren zur Maschinenbelegung für die Werkstattfertigung. in: AWF (ed.): 50 Jahre AWF, 1918 - 1968, S. 117 - 139, Freiburg i. Br. 1968

[Günther 84] Günther, S.: Zur Repräsentation temporaler Beziehungen in SRL. KIT-Report 21, TU Berlin. 1984.

[Guerrieri 88] Guerrieri, E.: Expert Systems and Prolog in Multiple Project Management. in: Sriram, D.; Adey, R. A. (eds.): Knowledge Based Expert Systems in Engineering: Planning and Design. Computational Mechanics Publication, S. 185 - 209. 1988.

[Halevi 80] Halevi, G.; Weill, R.: Development of Flexible Optimum Process Planning Procedures. Annals of the CIRP 29(1), S. 313 - 317. 1980.

[Heinrich 80] Heinrich, C.; Schneeweiß, C.; Vaterrodt, H. J.: Reihenfolgeplanung mit Hilfe heuristischer Verfahren in einer Unternehmung der Automobilbranche. in: Operations Research Proceedings, Springer Verlag, S. 258 - 264. 1980.

[Hertzberg. 86] Hertzberg, J.: Planerstellungs-Methoden der Künstlichen Intelligenz. Informatik-Spektrum 9. 1986.

[Hilliard 88] Hilliard, M. R.; Liepins, G. E.; Palmer, M.: Machine Learning applications to job shop scheduling. International conference on industrial and engineering applications of artificial intelligence and expert systems. Proceedings 1, Tullahoma, Tenn., S. 728 - 737. 1988.

[Hindi 90] Hindi, K. S.; Singh, M. G.: Hybrid knowledge-based systems for industrial scheduling. in: Decisional structures in automated manufacturing: Proceedings of the IFAC/CIRP/IFIP/IFORS workshop, Oxford. Eds.: Villa, A., Murari, G., S. 63 - 68 1990.

[Hrycej 88a] Hrycej, T.: Temporal Prolog. European Conference on Artificial Intelligence. Conference Proceedings, S. 296 - 301. 1988.

[Hrycej 88b] Hrycej, T.: Transitivity in relations between time intervals. In: Heyer, Krems, and Görz (eds.): Wissensarten und ihre Darstellung. Springer-Verlag. 1988.

[Huber 91] Huber, P.; Jensen, K.; Shapiro, R. M.: Hierarchies in Coloured Petri Nets. in: Jensen, K., Rozenberg, G. (Eds.): High-level Petri Nets. Springer Verlag, S. 215 - 243. 1991.

[Isele 87] Isele, J.: unveröffentlichter Bericht. Kernforschungzentrum Karlsruhe. 1987.

[Johnson 54] Johnson, S. M.: Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly. New York, NY, S. 61 - 68. 1954.

[Kadaba 91] Kadaba, N.; Nygard, K. E.; Juell, P. L.: Integration of Adaptive Machine Learning and Knowledge-Based Systems for Routing and Scheduling Applications. Expert systems with applications. 1991.

[Kahn 77] Kahn, K.; Gorry, G. A.: Mechanizing Temporal Knowledge. Artificial Intelligence, Vol. 9. 1977.

[Kanet 86] Kanet, J. J.: Tactically delayed versus non-delay scheduling: An experimental investigation. European Journal of Operational Research 24, North Holland, S. 99 - 105. 1986.

[Keck 68] Keck, H.: Vergleich von Prioritätsregeln mit Hilfe der Simulation. Eine Literaturübersicht.. in: Bussmann, K. F.. 1968.

[Kloth 88] Kloth, M.: Wissensbasierte Planungssysteme in der Layoutplanung. KOMMTECH 88, 7-10. Juli in Essen, S. 12.3.01 - 12.3.21. 1988.

[Kramer 89] Kramer, R.; Morlock, M.: Wissensbasierte Systeme im Operations Research I (Heuristiken). Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe. 1989.

[Kreimeier 86a] Kreimeier, D.: Planungshilfsmittel für die Disposition in teilautonomen Arbeitsgruppen. Technische Rundschau 43, S. 29 - 34. 1986.

[Kreimeier 86b] Kreimeier, D.: Rechnereinsatz bei der Disposition in teilautonomen Arbeitsgruppen. Kommtech 1986, Essen, S. 60-2 - 60-16. 1986.

[Lageweg 77] Lageweg, B. J.; Lenstra, K. J.; Rinnooy Kan, A. H. G.: Job-Shop Scheduling by implicit enumeration. Management Science 4, S. 441 - 449. 1977.

[Lageweg 79] Lageweg, B. J.; Lenstra, J. K.; Rinnooy Kan, A. H. G.: Minimizing Maximum Lateness on one Machine: Algorithms and Applications. in: Christofides, N.: Combinatorial Optimization, S. 371 - 388. 1979.

[Lamatsch 86] Lamatsch, A.; Morlock, M.; Neumann, K.; Rubach, T.: Schedule - An Expert System for Machine Scheduling. Annals of Operatiojns Research 16, S. 425 - 438. 1988.

[Lawler 82] Lawler, E. L.; Lenstra, J. K.; Rinnoy Kan, A. H. G.: Recent Developments in deterministic Sequencing and Scheduling: A Survey. in: Dempster, M. A. H., Lenstra, J. K., Rinnoy Kan, A. H. G.: Deterministic and Stochastic Scheduling. D. Reidel, Boston MA. 1982.

[Lawo 89] Lawo, M.: Hochflexible Handhabungssysteme zum Verputzen grosser Gussstücke (BMFT VP2.450.FM3). Kernforschungszentrum Karlsruhe GmbH. 1989.

[Lawo 90a] Lawo, M.; Droll, H.; Göbell, A.; Häfele, K.-H. et al.: COMETOS ein hochflexibles Handhabungssystem zur Bearbeitung mittelgroßer Gußstücke. KfK-Nachrichten 2. 1990.

[Lawo 90b] Lawo, M.; Schlegelmilch, S.: Modellbasierte Systeme zur Unterstützung der Diagnose und Modellbildung industrieller Handhabungssysteme. KfK-Nachrichten 2. 1990.

[Lawo 91] Lawo, M.; Droll, H.; Göbell, A.; Häfele K.-H. et al.: Automatisierungs- und Steuerungskonzept für ein hochflexibles Handhabungssystem zum Gußputzen. G. Hommel (Hrsg.): Prozeßrechensysteme '91. Springer-Verlag. 1991.

[Lawrence 86] Lawrence, S. R.; Morton, T. E.: PATRIARCH: Hierarchical production scheduling. Symposium on Real Time Optimization in Automated Manufacturing Facilities, NBS. 1986.

[Lee 86] Lee, E. J.; Mirchandani, P. B.: Scheduling with setups on a two-machine FMS. IEEE International Conference on Robotics and Automation, S. 1483 - 1489. 1986.

[Leinemann 91] Leinemann, K.: Advanced tele-operator support for fusion plant maintenance. Proceedings'91 International Symposium on Advanced Robot Technology, Tokyo, Japan, S. 465 - 472. 1991.

[Le Pape 85] Le Pape, C.: SOJA: A daily workshop scheduling system. Proceedings of the Fifth Technical Conference of the British Computer Society. Warwick, UK 1985.

[Lin 65] Lin, S.: Computer Solutions of the Traveling Salesman Problem. The Bell System Technical Journal 44(7), S. 2245 - 2269. 1965.

[Liu 90] Liu, P. S.; Fu, L. C.: An Efficient Method of Solving Problems of Classification and Selection Using Minimum Spanning Tree in a Flexible Manufacturing System. IEEE Proceedings on Robotics and Automation, S. 2148 - 2154. 1990.

[Lotter 92] Lotter, J.: unveröffentlichter Bericht. Kernforschungszentrum Karlsruhe, Dezember 1992.

[Martel 82] Martel, C.: Scheduling Uniform Machines with Release Times, Deadlines and Due Time. in Dempster et al.: Deterministic and Stochastic Scheduling. D. Reidel Publishing Company. 1982.

[McDermott 82] McDermott, D.: A Temporal Logic for Reasoning About Processes and Plans. Cognitive Science 6. 1982.

[Mehlhorn 84] Mehlhorn, Kurt: Graph Algorithms and NP-Completeness. Springer Verlag Berlin Heidelberg New York Tokyo. 1984.

[Morlock 88] Morlock, M., Kramer R.: Wissensbasierte Systeme im Operations Research I (Heuristiken). Vorläufige Vorlesungsausarbeitung WS 87/88, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe, 1988.

[Moser 91] Moser, P.: Arbeitsgruppe Flexible Fertigungssteuerung. Fraunhofer-Institut für Informations- und Datenverarbeitung, Karlsruhe. 1991.

[Müller-Merbach 70] Müller-Merbach, H.: Optimale Reihenfolgen. Springer. 1970.

[Neumann 75a] Neumann, K.: Operations Research Verfahren, Band I. Carl Hanser Verlag. 1975.

[Neumann 75b] Neumann, K.: Operations Research Verfahren Band III. Carl Hanser Verlag. 1975.

[Nilsson 82] Nilsson, N. J.: Principles of Artificial Intelligence. Springer-Verlag, Berlin, Heidelberg, New York. 1982.

[Noronha 91] Noronha, S. J.; Sarma, V. V. S.: Knowledge-Based Approaches for Scheduling Problems: A Survey. IEEE transactions on knowledge and data engineering 3 (2). 1991.

[Oliver 87] Oliver, I. M.; Smith, D. J.; Holland, J. R. C.: A study of Permutaion crossover operators on the travelling salesman problem. Proceedings of the Second International Conference on Genetic Algorithms. Cambridge, MA. 1987.

[Ow 87] Ow, P. S.; Smith, S. F.: Two Design Principles for knowledge-based systems. Decision Sciences 18, S. 430 - 447. 1987.

[Panwalkar 77] Panwalkar, S. S.; Iskander, W.: A Survey of Scheduling Rules. Operations Research, Vol. 25, No. 1, S. 45 - 61. 1977.

[Parunak 86] Parunak, H. V. D.; White, J. F.; Lozo, P. W.; Judd, R. et al.: An architecture for heuristic factory control. American Control Conference, Seatle, WA. 1986.

[Pearl 84] Pearl, J.: Heuristics - Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley Publishing Company, Inc., 1984.

[Proth 86] Proth, J. M.: Group technology in production management: a tool to simplify some scheduling problems. IEEE International Conference on Robotics and Automation S. 1641 - 1644. 1986.

[Puppe 89] Puppe, F.: Expertensysteme. Springer-Verlag. 1989.

[Reece 90] Reece, G. A.; Findler, N. V.: Toward a Blackboard Framework for Opportunistic Planning and Dynamic Scheduling in a Modular Job Shop Scheduling System. International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. Charleston, SC, Vol. 1, S. 151 - 155. 1990.

[Rembold 91] Rembold, U.: The Role of Manufacturing Models for the Information Technology of the Factory of the Nineties. Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe. 1991.

[Ricken 89] Ricken, A.: Konzeption und Implementierung eines Verfahrens zur Maschinenbelegung unter Berücksichtigung

verschiedener Problemklassen. WBK, Universität Karlsruhe. 1989.

[Rossier 86] Rossier, Y.; Troyon, M.; Liebling, Th. M.: Probabilistic Exchange Algorithms and Euclidean Traveling Salesman-Problem. OR Spektrum 8. 1986.

[Sacerdoti 73] Sacerdoti, E. D.: Planning in a hierarchy of abstraction spaces. International joint conference on artificial intelligence. 3, Stanford, 20.-23.8.1973, S. 115 - 135. 1973.

[Sacerdoti 77] Sacerdoti, E. D.: A Structure for Plans and Behavior. Elsevier North-Holland, Inc.. 1977.

[Sanoff 91] Sanoff, S. P.; Poilevey, D.: Integrated information processing for production scheduling and control. Computer-integrated manufacturing systems. London. 4.1991.3, S. 164 - 175. 1991.

[Sathi 85] Sathi, A.; Fox, M. S.; Greenberg, M.: Representation of Activity Knowledge for Project Management. IEEE Transactions on pattern analysis and machine intelligence, Vol. PAMI-7, No. 5, S. 531 - 552. 1985.

[Siegel 74] Siegel, Th.: Optimale Maschinenbelegung. Erich Schmidt Verlag. 1974.

[Simons 82] Simons, B.: On Scheduling with Release Times and Deadlines. in Dempster et al.: Deterministic and Stochastic Scheduling. D. Reidel Publishing Company. 1982.

[Smith 83] Smith, S. F.: Exploiting Temporal Knowledge to Organize Constraints. Intelligent Systems Laboratory, The Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA. 1983.

[Solberg 80] Solberg, J. J.: Capacity Planning with a Stochastic Workflow Model. AIIE Transactions Vol. 12, S. 116 - 122. 1980.

[Stefik 80] Stefik, M. J.: Planning with constraints. Stanford University, Report No. STAN-CS-80-784. 1980.

[Stefik 81] Stefik, M. J.: Planning and metaplanning (MOLGEN: Part 2). Artificial Intelligence 16, S. 141 - 169, 1981.

[Sussman 73] Sussman, G. J.: A Computer Model of Skill Acquisition. American Elsevier Publishing Company, Inc. New York London Amsterdam. 1973.

[Tang 86] Tang, Ch. S.: A Job Scheduling Model for a Flexible Manufacturing Machine. in: 1986 IEEE international Conference on Robotics and Automation, S. 152 - 155. 1986.

[Tate 74] Tate, A.: INTERPLAN: A plan generation system which can deal with interaction between machine goals. Machine Intelligence Research Unit Memorandum MIP-R-109, University of Edinburgh, Edinburgh, 1974.

[Tate 76] Tate, A.: Project planning using a hierarchic nonlinear planner. Department of Artificial Intelligence Research Rep. No. 25, University of Edinburgh, Edinburgh, 1976.

[Vere 83] Vere, S. A.: Planning in Time: Windows and Durations for Activities and Goals. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-5, No 3, S. 246 - 267. 1983.

[Vilain 82] Vilain, M. B.: A system for reasoning about time. AAAI-82, Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, Pennsylvania, S. 197 - 201. 1982.

[Vilain 86] Vilain, M. B.; Kautz, H.: Constraint Propagation Algorithms for Temporal Reasoning. Proceedings AAAI 86, Philadelphia, PA. 1986.

[Waldinger 75] Waldinger, R.: Achieving several goals simultaneously. SRI Artificial Intelligence Center Tech. Note 107, Menlo Park, CA, S. 94 - 136. 1975.

[Warnecke 83] Warnecke, H. J.; Geitner, U. W.: Gesichtspunkte bei Verfahren der Fertigungssteuerung. wt Zeitschrift für industrielle Fertigung 73, S. 95 - 98. 1983.

[Warren 74] Warren, D. H. D.: WARPLAN: A system for generating plans. Department of Computational Logic, Memo No. 76, University of Edinburgh, Edinburgh, 1974.

[Wiendahl 84] Wiendahl, H. P.; Buchmann, W.: Realisierung alternativer Fertigungssteuerungsstrategien. VDI-Z Bd. 126 Nr. 10 - Mai (II). 1984.

[Wilkins 83] Wilkins, D. E.: Representation in a Domain-Independent Planner. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, 8-12 August Karlsruhe, Vol. 2, S. 733 - 740. 1983.

[Winter 91] Winter, R.: Mehrstufige Planung. Dissertation. Springer 1991.

[Wu, S 87] Wu, S.-Y.; Wysk, R. A.: MPECS - An Intelligent Flexible Machining Cell Controller. Proceedings of the European simulation Multiconference, Wien, S. 71 - 76. 1987.

Anhang

Abkürzungsverzeichnis

<	Zeitbeziehung before, vorher
>	Zeitbeziehung after, nachher
<'	Vorher-Zeitbeziehungen: <, m
>'	Nachher-Zeitbeziehungen: >, mi
A	1. Menge der Aktivitäten, 2. Reihenfolgeproblemtyp: alle vorkommenden Elemente sind bekannt, alle Elemente müssen in die Reihenfolge aufgenommen werden [Müller-Merbach 70, S. 2f].
AK	Reihenfolgeproblemtyp: Einschränkung von Typ A: die Kosten zwischen den Elementen sind a priori bekannt und konstant [Müller-Merbach 70, S. 2f].
AV	Reihenfolgeproblemtyp: Einschränkung von Typ A: die Kosten zwischen den Elementen sind nicht bekannt und variabel [Müller-Merbach 70, S. 2f].
A/S	Activity/State: Aktivität/Zustand
B	Reihenfolgeproblemtyp: alle vorkommenden Elemente sind bekannt, aber nicht alle Elemente müssen in die Reihenfolge aufgenommen werden [Müller-Merbach 70, S. 2f].
BP	Menge aller Besitzprädikate
c_B	Bewertung eines Pfeils im Digraph
COMETOS	Coordinate Measuring and Tooling System
CPM	Critical Path Method (vgl. [Neumann 75, S. 191])
D	Digraph
D_B	beschriebener Digraph
d	Zeitbeziehung during, während
d'	Während-Relationen, Zusammenfassung der Zeitbeziehungen s, d, f
di	Zeitbeziehung during-inverse, contains, enthält
di'	Enthaltenseins-Relationen, Zusammenfassung der Zeitbeziehungen si, di, fi
E	Menge aller Pfeile im Digraphen
f	Zeitbeziehung finishes, beendet
FA	Frühester Anfangszeitpunkt
FE	Frühester Endzeitpunkt
fi	Zeitbeziehung finishes-inverse, finished-by, beendet-von
INST	Menge aller Instanzen
KC	KnowledgeCraft™
KfK	Kernforschungszentrum Karlsruhe

KOZ	Kürzeste-Operationszeit-Regel: Heuristik für die Maschinenbelegungsplanung
m	Zeitbeziehung meets, unmittelbar vorher
MAX_Dauer	Maximale Zeitdauer
mi	Zeitbeziehung meets inverse, met-by, unmittelbar nachher
MIN_Dauer	Minimale Zeitdauer
$NACH_1$	Im Zeitgraph Z = [I, E, c], v ∈ E: {w ∈ I: c[v, w] = <'}
$NACH_2$	Im Zeitgraph Z = [I, E, c], v ∈ E: {w ∈ I: c[w, v] = di'}
NB	Menge aller Objekte der Nebenbedingungen
O	Menge aller Objekte eines Reihenfolgeproblems
o	Zeitbeziehung overlaps, überlappt
OOA	Objektorientierte Analyse
oi	Zeitbeziehung overlapped-by, überlappt-von
ODER	Menge aller ODER-Knoten
OR	Operations Research
R	Menge aller Relationen
IR	Menge der reellen Zahlen
PERT	Program Evaluation and Review Technique (vgl. [Neumann 75, S. 191])
PROT	Menge aller Prototypen
s	1. Startknoten, 2. Zeitbeziehung starts, beginnt
SA	Spätester Anfangszeitpunkt
SE	Spätester Endzeitpunkt
si	Zeitbeziehung starts-inverse, started-by, begonnen von
seq	gewählte Sequenz
TL	Teillösung
TSP	Travelling Salesman Problem
UND	Menge aller UND-Knoten
V	Menge aller Knoten in einem Digraph
VOR_1	Im Zeitgraph Z = [I, E, c], v ∈ E: {w ∈ I: c[w, v] = <'}
VOR_2	Im Zeitgraph Z = [I, E, c], v ∈ E: {w ∈ I: c[w, v] = di'}
Z	Menge aller Zustände
Zf	Zielfunktion
ZP	Menge aller Zustandsprädikate
ZR	Zustandsraum
ZWISCHEN	Im Zeitgraph Z = [I, E, c], v ∈ E: {u ∈ I: c[v, u] = <' ∧ c[u, w] = <'} = $NACH_1(w) \setminus NACH_2(v)$

Die COMETOS-Zelle im Überblick

Die Hardware der COMETOS-Zelle besteht aus

- 2 KUKA Portalrobotern mit der AEG-Steuerung r500
- einem Oldelft-Laserscanner
- einer Sensorkugel
- einer Anzahl von Werkzeugen (Schruppscheibe, Fräser etc.), deren Auswahl von der Bearbeitungsaufgabe abhängt,
- 4 IBM-kompatible 386er PCs, von denen einer als Server dient. Die übrigen drei werden einer der Bearbeitungsaufgaben - Teachen, Messen, Bearbeiten - zugeordnet,
- mindestens 4 Paletten, auf die die Werkstücke gespannt werden,
- bei Bedarf einem fahrerlosen Transportsystem.

Ein derartiges System hat einen Platzbedarf von 16,5 mal 15 Metern. Falls auf das fahrerlose Transportsystem verzichtet werden kann, verringert er sich auf 10 mal 8 Meter.

Dazu kommen folgende Softwarekomponenten:

- MS-DOS,
- ein AEG-eigenes Betriebssystem und DOROB,
- die Kommunikationssoftware 3+ Open, die die PCs untereinander verbindet, und natürlich
- die COMETOS-Computerprogramme.

Die Roboterzelle sollte von einem Techniker betreut werden. Für die tägliche Arbeit wie das Programmieren oder die Überwachung genügt auch eine angelernte Kraft. Abgesehen vom Auf- und Abspannen der Werkstücke kann die Arbeit in der Zelle von einer einzigen Person bewältigt werden. Das gilt besonders dann, wenn der Palettentransport automatisiert ist. In diesem Fall wird das System auch mannlose Schichten fahren können.

Die Grenzen des Systems sind dort, wo Roboter nicht mehr eingesetzt werden können. So können Innenputzarbeiten nur in Ausnahmefällen ausgeführt werden, weil Stellen im Inneren des Werkstücks für den Roboter nicht zugänglich sind. Dabei ist zu beachten, daß mit der Roboterzelle nie sämtliche Putztätigkeiten automatisiert werden sollten sondern nur der geplante Putzaufwand an Außengraten. Innenputzarbeiten und ungeplanter Putzaufwand sollte vom Konzept her weiterhin den Gußputzern vorbehalten bleiben.

Grundsätzlich kann man die Roboterzelle auch für andere Einsatzzwecke nutzen. Die Arbeitsgeräte sind so ausgelegt, daß Werkstücke bis zu einer Größe von 2800 mm x 1200 mm x 700 mm auf fünf Seiten bearbeitet werden können. Wie so etwas aussehen kann, hängt aber sehr vom Einzelfall ab.

Arbeiten mit der COMETOS-Roboterzelle[1]

Ein Werkstück wird im Automatikbetrieb mit den folgenden Schritten bearbeitet:

1. Das Werkstück wird auf eine Palette gespannt.

[1] Eine detaillierte Beschreibung der Zelle findet sich bei [Lawo 89], [Lawo 90a-c], [Lawo 91]. Grundbegriffe der Robotik erklärt Naval [Naval 89].

2. Der Bearbeitungsroboter trennt Steiger ab.
3. Der Meßroboter fährt entlang der Gratbahn und scant mit einem Laserscanner den Grat ab.
4. Ein Computer berechnet aus den Meßdaten das Putzprogramm.
5. Der Bearbeitungsroboter putzt das Werkstück.
6. Das Werkstück wird abgespannt.

Mit diesem Schema können sehr unterschiedliche Werkstücke bearbeitet werden. Wie man am Layout der Zelle erkennt, sind die verschiedenen Stationen über das fahrerlose Transportsystem verbunden.

COMETOS verwendet eine neue Art der Programmierung. Der Anwender greift den Roboterarm an einer Sensorkugel und führt ihn so über den Grat des Werkstückes. Dazu genügt es, wenn der Abstand zum Gratfuß ungefähr gleich ist. Weil keine allzugroße Genauigkeit beim Führen verlangt wird, geht das Teachen einfacher und schneller. Die Anforderungen an den Anwender sind weitaus geringer als beim konventionellen Teachen.

Daß diese flexible Art des Teachens möglich ist, verdankt das System einem Laserscanner. Er befindet sich wie die Sensorkugel am Arm des Roboters und mißt Lage und Höhe des Grates, der abgetragen werden soll. Im Gegensatz zu anderen Sensoren arbeitet er auch dann noch genau, wenn sich sein Abstand zur Oberfläche etwas ändert. Der Scanner verfügt über ein Sensorfenster. Der Grat muß innerhalb dieses Fensters liegen; der Sensor muß deswegen nicht auf den Zehntelmillimeter genau eingestellt werden.

Sensorkugel und Laserscanner erlauben es, die Teachzeit von mehreren hundert Stunden bei konventioneller off-line Programmierung auf 4 bis 5 Stunden zu senken. Erst dann ist an den wirtschaftlichen Einsatz der Roboterzelle zu denken. Der Computer berechnet automatisch aus den gemessenen Daten die Bearbeitungsprogramme. Der Bediener gibt lediglich einzelne aus der Messung nicht ableitbare Daten, wie Werkzeugwahl und -orientierung, ein. Der Computer hilft dem Bediener, indem er eine Optimierung vorschlägt[2].

Der Anwender führt den Roboter nicht in einem Zug um das Werkstück; vielmehr unterteilt er den Weg des Roboters in mehrere Abschnitte, die Bahnelemente heißen. Bahnelemente sind sinnvoll zusammengefaßte Sequenzen von Bahnpunkten. Bahnpunkte sind Punkte, die der Roboter mit Hilfe der Robotersteuerung ansteuern kann. Bei den Bahnelementen unterscheidet man weiterhin zwischen Luft- und Gratbahnen. Nur auf Gratbahnen wird der Laserscanner messen. Gratbahnen heißen beim Messen Meßbahnen und beim Putzen Bearbeitungsbahnen. Luftbahnen generiert man beispielsweise für Wege zur Werkzeugwechseleinrichtung oder zum Werkstück hin.

Sobald ein Werkstück einmal geteacht wurde, weiß das System, wie alle anderen Werkstücke des gleichen Typs gemessen und bearbeitet werden müssen.

Darstellung des TSP

Das TSP wird meist mit einem vollständigen, schlichten und bewerteten Graphen G = [V, E, C] und symmetrischer Kostenmatrix $C = (c_i)_{nn} \geq 0$ dargestellt. Gesucht ist ein kostenminimaler Weg, bei dem alle Knoten mindestens einmal passiert werden. Eine Variable x_{ij} indiziert, ob der Weg von i nach j benutzt wird oder nicht. c_{ij} sind die Kosten für die Benutzung des Wegs von i nach j.

Prämissen des Problems sind:

[2]Die Bedienung ist ausführlich von Isele [Isele 91] beschrieben worden.

1. $c_{ij} \leq c_{ik} + c_{kj}$ mit i, j, k $\in \{1, .. , n\}$ (Dreiecksungleichung)
2. $c_{ii} := \infty$ (keine Zyklen an einem Knoten)
3. $c_{ij} > 0$. Die Bedingung verhindert Zyklen über mehrere Knoten. Sie wird gern relaxiert, aber diese Spezialfälle sollen bis auf weiteres nicht interessieren.
4. $\sum_{i=1}^{n} x_{ij} = 1;\ j \in \{1 \cdots n\}$ (Jeder Knoten des Lösungswegs hat nur einen Vorgänger)
5. $\sum_{j=1}^{n} x_{ij} = 1;\ i \in \{1 \cdots n\}$ (Jeder Knoten des Lösungswegs hat nur einen Nachfolger)
6. $x_{ij} \in \{0, 1\}$
7. Bedingungen zur Verhinderung von Kurzzyklen.

Eine Zielfunktion lautet dann:

Minimiere $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$

Eine umfassendere Beschreibung des TSP findet man z.B. bei Müller-Merbach [Müller-Merbach 70, S. 69f].

Begriffe und Prämissen der Maschinenbelegungsplanung

Begriffe

j Auftrag, Job, Los. " (...) zeiterforderliche Tätigkeiten, die wir in der Regel mit j_1, ... ,j_n oder auch kurz mit 1, ... , n bezeichnen." [Brucker 81, S. 2].

m Maschine: Einrichtung, "die in der Lage ist, eine bestimmte Bearbeitung auszuführen" [Seelbach 75, S. 14], also alle Betriebsmittel mit beschränkter Kapazität.

Bearbeitung, Operation, Teiljob, Arbeitsgang: "bestimmte Tätigkeit, die von einer bestimmten Maschine an einem bestimmten Auftrag ausgeführt wird.

p_{ji} Bearbeitungs- oder Operationszeit. Dauer einer Operation von Auftrag j auf Maschine i.

Auftragsgrößen:

w_{jm} Wartezeit. Zeit zwischen dem Ende einer Operation und dem ablaufplanbedingten Beginn der darauffolgenden Operation des gleichen Auftrags j, vor Maschine m.

D_j Durchlaufzeit. Summe aus Bearbeitungszeiten und Wartezeiten des Auftrags j.

c_j Abschlußzeitpunkt. Zeitpunkt, zu dem die letzte Operation des Auftrags j beendet wird.

d_j Fälligkeitstermin. Zeitpunkt, zu dem Auftrag j beendet sein soll.

t_j Terminüberschreitung. Die positive Zeitdifferenz zwischen Fertigstellungs- und Fälligkeitstermin: $c_j - d_j$.

Maschinengrößen:

l_{jm} Leerzeit: Zeit, während der die Maschine m vor der Bearbeitung des Auftrags j ablaufplanbedingt nicht arbeitet.

B_m Belegungszeit einer Maschine, die Summe aus ihren Bearbeitungs- und Leerzeiten.

Rüstzeit: Zeit, die aus technischen Gründen nötig ist, um eine Maschine bereitzustellen.

Maschinenfolge des Auftrags j: zeitliche Reihenfolge, in der j die Maschinen durchläuft.

Auftragsfolge der Maschine m: zeitliche Reihenfolge, in der die Maschine m ihre Aufträge bearbeitet.

Maschinenbelegungsplan, Schedule: Auftragsfolge aller Maschinen.

Prämissen

Von den folgenden Prämissen wird bei der Maschinenbelegungsplanung üblicherweise ausgegangen. Einige der Prämissen gelten nicht für die Problemstellung der vorliegenden Arbeit. Sie wurden mit einem Stern gekennzeichnet.

1.* Unterbrechungen durch Wartung oder Störung werden nicht berücksichtigt.

2. "Auf jeder Maschine kann zu jeder Zeit nur ein Job bearbeitet werden." [Morlock 89, S. 6]

3.* Jeder Auftrag bildet eine Einheit. Kein Auftrag kann gleichzeitig mehrere Maschinen belegen.

4.* Zwischen zwei Operationen kann ein Auftrag zwischengelagert werden. Dafür gibt es stets ausreichend Platz.

5.* Keine Operation darf unterbrochen oder vorzeitig beendet werden (wohl aber ein Auftrag siehe 4.).

Darstellung des Scheduling Problems

Scheduling ist "the art of assigning resources to tasks in order to insure the termination of these tasks in a reasonable amount of time" [Dempster 82, S. 3]. Die Symbolik bei der Maschinenbelegung ist nicht eindeutig. Die hier verwendete wurde von Seelbach [Seelbach 75, S. 14ff] und Brucker [Brucker 81, S. 1-20] übernommen. Die englischen Entsprechungen findet man z.B. bei Lawler [Lawler 82, S. 36ff]. Dort werden die Begriffe auch ausführlicher erklärt.

Einen bekannten Klassifizierungsansatz verwenden z.B. von Conway, Maxwell und Miller [Conway 67]. Die Symbole a/b/c/d/e bedeuten:

a Anzahl der Jobs (Aufträge). Falls sie nicht festgelegt ist, schreibt man "o".

b Kapazitätsrestriktionen, meist die Anzahl der Maschinen und "o", falls sie nicht feststeht.

c Nachfolgestruktur oder logische Präzendenzrelation, d.h. in wie weit ein Auftrag Maschinen in einer bestimmten Reihenfolge passieren muß. Die Aufträge sind entweder voneinander unabhängig, oder es handelt sich um eine der folgenden Nachfolgestrukturen: *Open Shop*: Aufträge können die Maschinen wahlfrei durchlaufen. *Flow Shop*: Alle Aufträge durchlaufen die Maschinen in der gleichen, vorgegebenen Reihenfolge. *Job Shop*: Aufträge durchlaufen die Maschinen in vorgebenen, aber unterschiedlichen Reihenfolgen. Bei Montage- und Sortierproblemen ist die Nachfolgestruktur ein gerichteter Baum, dessen Wurzel die Quelle (Montageproblem, MB) bzw. die Senke (Sortierproblem SB) des Netzwerkes ist. Allgemein spricht man von Baumproblemen (B). Bei bestimmten

Problemen kann eine Tätigkeit unabhängig von den Vorgängertätigkeiten nicht zu beliebigen Zeiten begonnen werden. Vielmehr ist ein Bereitstellungstermin zu berücksichtigen. Das Symbol dafür ist b(i) ≠ 0. "o" bedeutet beliebige Nachfolgestruktur.

d Wenn die Durchführungsdauer auf bestimmte Werte beschränkt ist, schreibt man hier z.B. d(i) = 1, d(i) = {2;3;4}. "o" steht für keinerlei Beschränkungen der Durchführungsdauer.

e schließlich ist die Zielfunktion, das oder die Kriterien, nach denen optimiert werden soll. Die Zielkriterien aus betriebswirtschaftlicher Sicht sind oft widersprüchlich. Gutenberg sprach vom "Dilemma der Ablaufplanung" [Gutenberg 70]. Für eine Diskussion optimaler Pläne und unterschiedlicher Zielsetzungen vergleiche etwa [Brucker 81, S. 8f], [Seelbach 75], [Hoss 65].

Andere Klassifizierungsansätze bestehen aus nur drei Parametern (vgl. z.B. Lawler et al. in [Lawler 82, S. 36ff]. Sie unterscheiden sich aber nicht grundsätzlich von dem hier skizzierten Ansatz.

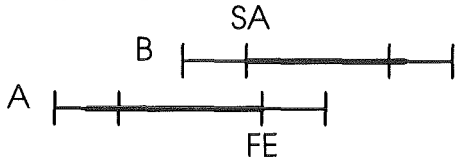
Schlußfolgerungstabelle 2 für zeitliche Inferenz

Werden feste Zeiten vorgegeben, können Relationen weiter eingeschränkt werden. Dafür gilt die folgende Schlußfolgerungstabelle

A before B

einhaltbar, wenn

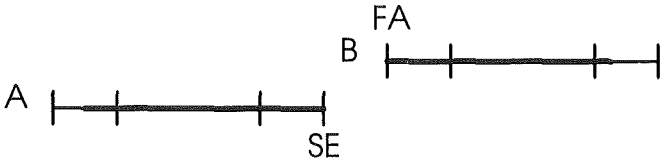
- ☐ $FE_A < SA_B$



A meets B

einhaltbar, wenn

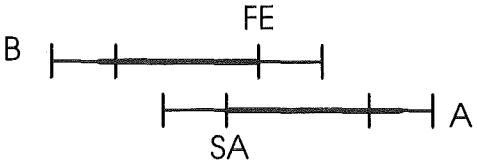
- ☐ $FE_A \leq SA_B$ (vgl. before) und
- ☐ $SE_A \geq FA_B$



A after B

einhaltbar, wenn

- ☐ $SA_A > FE_B$

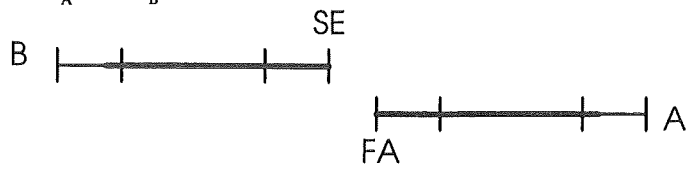


A met-by B

einhaltbar, wenn

- ☐ $SA_A \geq FE_B$ (vgl. after) und

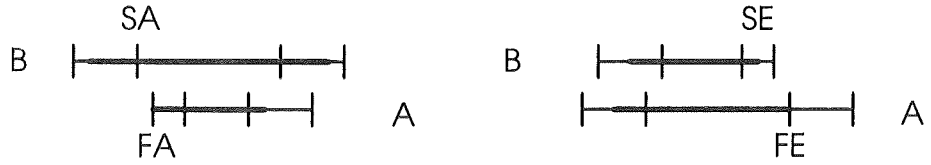
- $FA_A \leq SE_B$



A overlaps B

einhaltbar, wenn

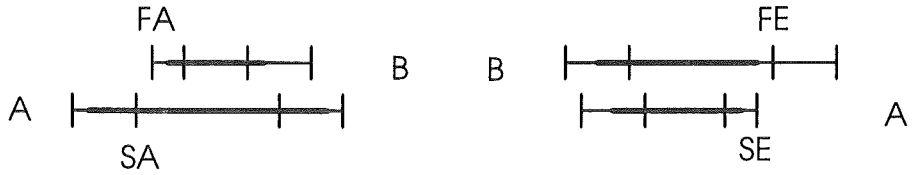
- $FA_A \leq SA_B$ und
- $FE_A \leq SE_B$ und
- $SE_A \geq FA_B$



A overlapped by B

einhaltbar, wenn

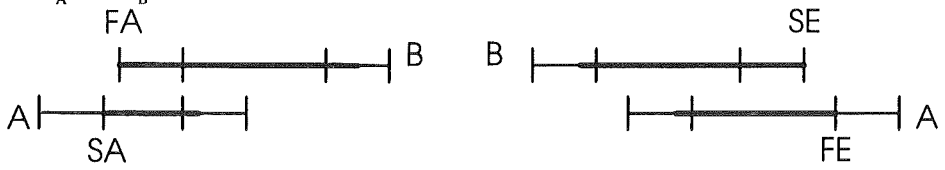
- $SA_A \geq FA_B$ und
- $SE_A \geq FE_B$ und
- $FA_A \leq SE_B$



A during B

einhaltbar, wenn

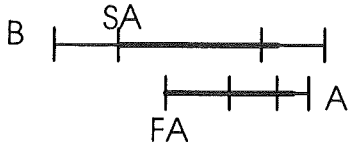
- $SA_A > FA_B$ und
- $FE_A < SE_B$



A starts B

einhaltbar, wenn

- $SA_A \geq FA_B$ (vgl. during) und
- $FE_A \leq SE_B$ (vgl. during) und
- $FA_A \leq SA_B$

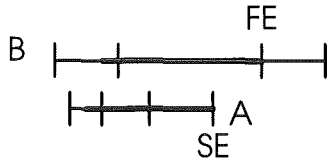


A finishes B

einhaltbar, wenn

- $SA_A \geq FA_B$ (vgl. during) und

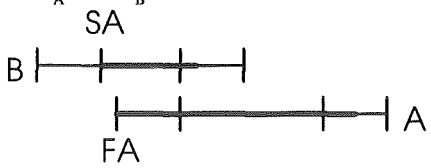
- $FE_A \leq SE_B$ (vgl. during) und
- $SE_A \geq FE_B$

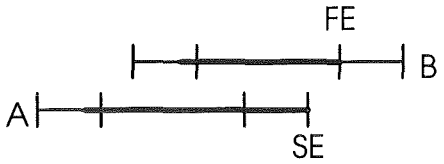


A contains B

einhaltbar, wenn

- $FA_A < SA_B$ und
- $SE_A > FE_B$

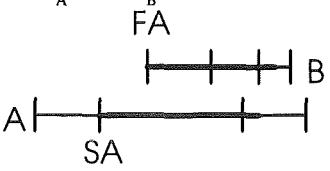




A started-by B

einhaltbar, wenn

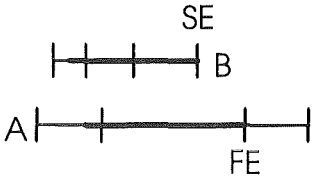
- $FA_A \leq SA_B$ (vgl. contains) und
- $SE_A \geq FE_B$ (vgl. contains) und
- $SA_A \geq FA_B$



A finished-by B

einhaltbar, wenn

- $FA_A \leq SA_B$ (vgl. contains) und
- $SE_A \geq FE_B$ (vgl. contains) und
- $FE_A \leq SE_B$



A time-equal B

einhaltbar, wenn

- $FA_A \leq SA_B$ und
- $SA_A \geq FA_B$
- $FE_A \leq SE_B$
- $SE_A \geq FE_B$