Drexl, Andreas; Pesch, Erwin; Salewski, Frank

Zur Bedeutung der Modellbildung für die Entwicklung wissensbasierter Systeme

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 342

Nr. 342

Zur Bedeutung der Modellbildung für die Entwicklung wissensbasierter Systeme

Drexl / Pesch / Salewski

März 1994

Professor Dr. **Andreas Drexl**, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel.

PD Dr. **Erwin Pesch**, Assistant Professor, Rijksuniversiteit Limburg, Maastricht.

Dipl.-Informatiker **Frank Salewski**, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel.

A. Einführung

Kaum ein Gebiet der künstlichen Intelligenz[1] hat sich in jüngster Zeit so rasch entwickelt wie das der wissensbasierten bzw. beispielhaft Expertensysteme. Die Beschreibung und Darstellung von Wissen durch Verwendung von Regeln, Frames und semantischen Netzen wird zunehmend durch neuere Wissensrepräsentationen mittels Constraints ergänzt und verdrängt. Die auf diesen Wissensbasen arbeitenden Operatoren werden in entsprechenden constraint-basierten logikorientierten Programmiersprachen, Techniken zum temporalen und nichtmonotonen Schließen, sogenannten höheren Problemlösungsstrategien für Diagnosesysteme usw. bereitgestellt.[2]

Der vorliegende Beitrag diskutiert unterschiedliche Möglichkeiten der Modellierung als Voraussetzung für das Aufstellen von Wissensbasen im Hinblick auf die Entwicklung regelbasierter Expertensysteme. Wir beschäftigen uns hier exemplarisch mit einer Problemstellung aus der Investitionstheorie sowie mit einer aus dem Bereich des Projektmanagements, genauer der ressourcenorientierten Projektsteuerung. Für ein klassisches Problem aus der Investitionstheorie zeigen wir zunächst beispielhaft die Äquivalenz von binärer und regelbasierter Modellierung. Diese Feststellung läßt den Schluß zu, daß die Unterschiede zwischen herkömmlichen Modellen des Operations Research und Wissensbasen teilweise eher phänomenologischer Natur sind. Anschließend präsentieren wir für die Problemstellung aus dem Bereich des Projektmanagements eine äquivalente konzeptionelle Modellformulierung und zeigen exemplarisch Ergebnisse der Implementierung dieses Modells in CHARME, einer PROLOG-basierten Programmiersprache.

Der Beitrag ist wie folgt aufgebaut: In Abschnitt B charakterisieren wir die Komponenten wissensbasierter Systeme. Abschnitt C verdeutlicht mit Hilfe eines einfachen Beispiels aus der Investitionstheorie die grundsätzlichen Beziehungen zwischen mathematischer Optimierung und logischer Inferenz. In Abschnitt D geben wir einen Überblick über neuere, constraint-basierte KI-Sprachen. In Abschnitt E schildern wir zunächst die Annahmen eines Problems der Projektsteuerung, formulieren dann ein konventionelles Modell, erläutern die Beziehungen zu Regeln und leiten schließlich das (äquivalente) konzeptionelle Modell her. Abschnitt F stellt beispielhaft die Ergebnisse der Implementierung des konzeptionellen Modells in CHARME vor. Wir beschließen den vorliegenden Beitrag in Abschnitt G mit zusammenfassenden Hinweisen auf Ergebnisse und Perspektiven.

---

1 Vgl. z.B. **Boncsek** et al. (1981), **Winston** (1984), **Harmon / King** (1986), **Jackson** (1986) sowie **Luger / Stubblefield** (1989).

2 Siehe z.B. die Übersicht über die genannten und weitere Gebiete in **Puppe** (1991).

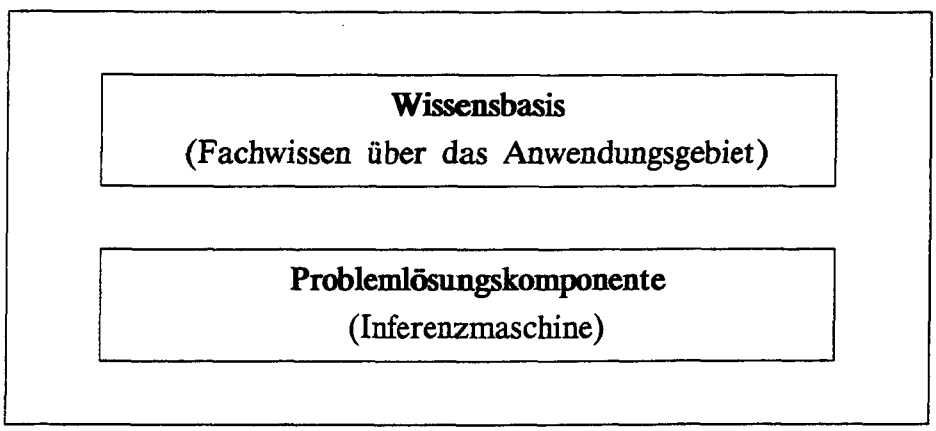
B. Komponenten wissensbasierter Systeme

Zwei Arten von "Wissen" werden üblicherweise bei wissensbasierten Systemen unterschieden:[3]

- Wissen über das Anwendungsgebiet (= **Wissensbasis**): In einem Programmpaket zur Produktionsplanung und -steuerung (PPS-System) ist Wissen u.a. über materialwirtschaftliche (z.B. über Bedarfsbeziehungen zwischen vor-/nachgelagerten Produkten) und zeitwirtschaftliche Zusammenhänge (z.B. über die Kopplung von Terminvorgaben und Reihenfolgebeziehungen) enthalten.[4]
- Allgemeines Problemlösungswissen (= **Problemlösungskomponente**): Darunter versteht man das Wissen (in Form von Planungsmethoden) darüber, wie mit dem Wissen über das Anwendungsgebiet umzugehen ist. In PPS-Systemen z.B. stellen Algorithmen zur Losgrößenplanung oder Durchlaufzeitterminierung sicher, daß die mengenmäßigen und/oder zeitlichen Abhängigkeiten berücksichtigt werden.

Während im Bereich der prozeduralen Programmierung diese beiden Arten des Wissens im Code untrennbar miteinander verbunden sind,[5] liegen beide in wissensbasierten Systemen streng voneinander getrennt vor (vgl. auch *Abbildung 1)*. Es liegt auf der Hand, daß die Trennung beider Arten von Wissen konzeptionell überlegen ist.

Abbildung 1: Komponenten wissensbasierter Systeme[6]



In wissensbasierten Systemen dient die Problemlösungskomponente der "Verarbeitung" des Fachwissens; diese Verarbeitung hat vor allem das Ziehen von Schlußfolgerungen zum

---

3 Vgl. z.B. **Kurbel** (1989, S. 17 ff.). Eine weitergehende Systematik liegt der KADS-Methodologie (Knowledge Acquisition and Documentation Structuring) zu Grunde; vgl. hierzu z.B. **Schreiber** (1990).

4 Die Wissensbasis stellt eine (mehr oder weniger stark) vereinfachte Abbildung des relevanten Ausschnitts der Realität dar. Unabhängig von der gewählten Codierung (prozedural oder deklarativ) sind damit Wissensbasis und Modell ceteris paribus identisch (vgl. hierzu auch Abschnitt E).

5 Eine Simplex-Prozedur zur Lösung linearer Optimierungsprobleme z.B. enthält sowohl Wissen über das Anwendungsgebiet "lineare Modelle" als auch Wissen über den Austauschmechanismus von Ecken des konvexen Polyeders — beides untrennbar im Code miteinander verbunden.

6 Quelle: **Kurbel** (1989, S. 18).

Gegenstand – man spricht daher auch von der "Inferenzmaschine" oder dem "Inferenzmechanismus".[7]

Wir gehen im folgenden nicht näher ein auf die Abgrenzung von klassischen Informationssystemen sowie wissensbasierten und Expertensystemen.[8] Auch die zusätzlichen Komponenten "(strukturierte) Wissensakquisition", "Dialog" sowie "Erklärung" in Expertensystemen sollen hier nicht weiter vertieft werden. Vielmehr wenden wir uns zunächst der regelbasierten Repräsentation von Wissen[9] zu, bevor wir dann zeigen, wie (klassisches und konzeptionelles) Modellwissen regelbasiert dargestellt werden kann.

Regeln sind die am häufigsten verwendete Form der Repräsentation von Wissen. Die allgemeine Form einer Regel lautet:

Wenn P *dann* K

P steht dabei für Prämisse, K für Konklusion, und beides sind logische Ausdrücke. Das bedeutet: Wenn die Prämisse erfüllt, das heißt P wahr ist, dann ist auch K wahr.[10]

Zur Verknüpfung von Ausdrücken dienen Operatoren, wobei über den Operator "$\rightarrow$" (Implikation) hinaus vor allem die Operatoren "$\wedge$" (und), "$\vee$" (oder) sowie "$\neg$" (Negation) von Bedeutung sind. Unter Verwendung des Operators "Implikation" läßt sich obige Regel auch kurz durch "$P \rightarrow K$" zum Ausdruck bringen.

C. Beispiel aus der Investitionstheorie

Operations Research und künstliche Intelligenz haben fundamentale Berührungspunkte im Bereich der Anwendung mathematischer Optimierungsmethoden auf logische Inferenzprobleme.[11] Zur Verdeutlichung diene folgendes einfache Beispiel aus dem Bereich der *Investitionstheorie:*[12] Bei der Planung von Investitionsprojekten sind häufig logische Bedingungen für verbundene und/oder sich gegenseitig ausschließende Projekte zu beachten. Nehmen wir beispielsweise an, daß insgesamt vier Investitionsprojekte A, B, C und D zur Wahl stehen. Die Projekte A und B sowie C und D schließen sich (aus technischen Gründen) gegenseitig aus. Projekt D kann (wiederum aus technischen Gründen) nur realisiert werden, wenn auch Projekt A realisiert wird. Projekt B kann analog nur realisiert werden, wenn auch Projekt C realisiert wird. Es stellt sich nun die Frage, welche Investitionsprogramme technisch zulässig sind. Zur Beantwortung dieser Frage können wir – unter Verwendung von Binärvariablen $x_j \in \{0,1\}$ (mit der Bedeutung $x_j = 1$ (0), wenn das Projekt j (nicht) realisiert werden soll) das folgende Restriktionensystem formulieren.

---

7 Siehe hierzu insbesondere **Waterman** (1986, S. 18 f.).

8 Vgl. hierzu z.B. **Kurbel** (1989, S. 25 ff.).

9 Darüber hinaus wird Wissen in der KI vor allem in Form von Frames oder semantischen Netzen dargestellt.

10 Ist die Prämisse P falsch, dann ist der Wahrheitsgehalt von K nicht bestimmbar.

11 Zu den Beziehungen zwischen mathematischer und logikorientierter Programmierung vgl. insbesondere **Williams** (1977), **Blair** et al. (1986), **Hooker** (1988), **Lodwick** (1989) sowie **Dincbas** et al. (1990).

12 Vgl. hierzu z.B. **Albach** (1975), **Blohm / Lüder** (1991) sowie **Kruschwitz** (1993).

$$
\begin{array}{lllllll}
x_A & + & x_B & & & \leq & 1 \\
x_A & & & & -x_D & \geq & 0 \\
 & -x_B & + & x_C & & \geq & 0 \\
 & & & x_C & + \; x_D & \leq & 1
\end{array}
$$

Man kann sich leicht überlegen, daß – in Vektorschreibweise – die Invesitionsprogramme $\mathbf{x} = (0,0,0,0)$, $\mathbf{x} = (0,1,1,0)$ und $\mathbf{x} = (1,0,0,1)$ technisch zulässig sind. Zu diesen "Programmentscheidungen" gelangt man auch, wenn man – unter Verwendung Boole'scher Variablen – die folgende äquivalente Menge von Regeln aufstellt.

$$
\begin{array}{llll}
x_A \rightarrow \neg x_B & & x_B \rightarrow \neg x_A \\
x_D \rightarrow x_A & & \\
x_B \rightarrow x_C & & \\
x_C \rightarrow \neg x_D & & x_D \rightarrow \neg x_C
\end{array}
$$

Die Ausführungen in diesem Abschnitt zeigen in einfacher Form den grundsätzlichen Zusammenhang zwischen mathematischer Optimierung und logischer Inferenz. In Abschnitt E und F werden wir darstellen, wie sich Optimierungsprobleme auf unterschiedliche Art und Weise in logische Inferenzprobleme umformen lassen und welche Konsequenzen das für die Laufzeiteffizienz des Inferenzmechanismus hat. Wir rekurrieren dabei auch auf constraint-basierte KI-Sprachen, denen wir uns daher im folgenden Abschnitt zunächst zuwenden.

D. Neuere, constraint-basierte KI-Sprachen

Vergleichsweise kleine Wissensbasen (mit einer geringen Anzahl von Regeln oder Restriktionen) definieren eine i.d.R. exponentiell in der Problemgröße anwachsende Zahl potentieller Lösungen, deren Zulässigkeit oder Optimalität nur durch (implizite) Enumeration (Backtracking) des gesamten Suchraumes bestimmt werden kann. Konsistenzprüfungen, als Teil der Inferenz, ergänzt durch Strategien wie heuristische Suche, Relaxationen, retrospektive und prospektive Techniken, etc.[13] können zwar den Suchbaum bei Backtrackverfahren (auch Branch & Bound) zur Lösung schwieriger (NP-vollständiger) Probleme drastisch beschränken, sind jedoch in prozeduralen Programmiersprachen nur mit großem Aufwand und häufig nur sehr problembezogen zu implementieren sind. Die Trennung von Wissensbasis und Problemlösungskomponente, ein wesentliches Merkmal logikorientierter Programmierung, liefert hier die erforderliche Flexibilität.

---

13 Ergänzt um weitere Methoden wie intelligentes Backtracking (intelligent backtracking), selektives Backtracking (selective backtracking), abhängigkeitsgerichtetes Backtracking (dependency-directed backtracking), Backjumping oder Backmarking; vgl. hierzu z.B. **Tate** (1985), **Bitner / Reingold** (1975), **Freuder / Wallace** (1992) sowie **Bruynooghe / Pereira** (1984).

Logikorientierte bzw. deklarative Programmiersprachen bauen implizit einen Suchbaum auf, wobei der Programmierer lediglich den Aufbau zu steuern, jedoch nicht selbst zu programmieren hat. Traditionelle logikorientierte Programmiersprachen (z.B. PROLOG[14]) erlauben zwar eine vergleichsweise einfache Problemimplementierung (basierend auf einer Wissensbasis in Form von Regeln) und stellen eine generische Suchstrategie (basierend auf einfachen, teilweise benutzergesteuerten Inferenzregeln) bereit, sie können jedoch i.d.R. als "general problem solver" nur zur Lösung einfacher Probleme Verwendung finden.

Die Suche mittels Backtracking gewann zunehmend an Bedeutung aufgrund der Anwendung in PROLOG und beim Lösen von *Constraint Satisfaction Problemen.*[15] PROLOG II[16] eröffnete dem Benutzer die Möglichkeit, Konzepte einfacher und klarer auszudrücken und ebnete den Weg zu dem, was heute allgemein als "logic programming with constraints" bezeichnet wird.[17] Damit wurde eine wesentliche Erweiterung gegenüber Programmiersprachen, die auf Wissensbasen in Form von Gleichungssystemen operieren, erreicht.[18] In PROLOG III[19] besteht die Möglichkeit, Constraints auf verschiedenen Datentypen zu definieren, wodurch die Ausdruckskraft und Performance der Sprache erheblich erweitert sowie Anwendungen ermöglicht werden, die in PROLOG nur schwer zu realisieren sind.[20]

Eines der ersten generischen Problemlösungssysteme, welches automatische Konsistenzprüfungen bzgl. eines Nebenbedingungssystems zur Suchraumreduzierung durchführte, war das System ALICE von Lauriere (1978). Es stellte sich für den Benutzer als Black Box dar, d.h. ohne Möglichkeiten zum Eingreifen bei der Steuerung der Suche. Obwohl Lauriere beansprucht, daß sein System ALICE praktische Probleme effizient löst, stellt erst die Entwicklung von CHIP[21] eine logische Programmierumgebung bereit, die reale Problemstellungen lösbar macht.[22]

PROLOG und CHIP sind die einzigen bekannt gewordenen Programmiersprachen, in denen Backtrackstrukturen und Konsistenzprüfungen implizit realisiert und vom Programmierer gesteuert werden können. Vor wenigen Jahren stellte die französische Firma Bull mit CHARME eine Weiterentwicklung von CHIP vor, in der der Benutzer parametergesteuert den Suchbaum beim Backtracking reduzieren kann.[23]

---

14 Vgl. z.B. **Clocksin / Mellish** (1984), **Kleine Büning / Schmitgen** (1988) sowie **Bothner / Kähler** (1991).

15 Auch als Entscheidungsproblem bezeichnet; vgl. insbesondere **Cox** (1984), **Clocksin / Mellish** (1984) sowie **De Kleer** (1986, 1989). Das Constraint Satisfaction Problem ist eng verwandt mit dem aus der Entscheidungstheorie bekannten "Satisfizierungsproblem"; vgl. hierzu z.B. **Dinkelbach** (1982).

16 Siehe insbesondere **Pique** (1988) sowie **Sterling / Shapiro** (1986).

17 Vgl. z.B. **Cohen** (1990).

18 Siehe vor allem **Leler** (1988).

19 Vgl. insbesondere **Colmerauer** (1987, 1990).

20 Siehe insbesondere **Krautter / Steinert** (1988) sowie **Jost / Skuppin** (1989).

21 Vgl. **Van Hentenryck** (1987) sowie **Dincbas** et al. (1990).

22 Umfangreiche Vergleiche verschiedener logikorientierter Sprachen findet man in **Mitterreiter / Radermacher** (1991).

23 Vgl. **BULL** (1991); siehe hierzu auch **Pesch** et al. (1993).

Wir beschreiben im folgenden eine Problemstellung, deren konzeptionelle Modellformulierung in CHARME sehr einfach und vergleichsweise effizient implementiert werden kann.

E. Konventionelle und konzeptionelle Modellierungsmöglichkeiten

Im folgenden wählen wir exemplarisch[24] eine Problemstellung aus dem Bereich der Netzplantechnik bzw. des Projektmanagements.[25] Wir beschreiben zunächst die Problemstellung, formulieren dann ein konventionelles (binäres) Modell und präsentieren schließlich ein (äquivalentes) konzeptionelles Modell. Wir diskutieren die Gemeinsamkeiten und Unterschiede und zeigen insbesondere, wie sich zwei Restriktionstypen des binären Modells in der Form einer "Regel" aufschreiben lassen.

1. Modellannahmen

Im einzelnen ist die betrachtete Problemstellung durch folgende Annahmen gekennzeichnet:[26]

- Gegeben sei ein Projekt in Form eines MPM Vorgangsknoten-Netzplans (Vorgänge als Knoten, Pfeile für Reihenfolgebeziehungen) mit j = 1,2,...,J Jobs (Vorgängen, Aktivitäten).
- $\mathcal{V}_j$ ($\mathcal{N}_j$) sei die Menge der Job j unmittelbar vorausgehenden (nachfolgenden) Jobs. Ohne Beschränkung der Allgemeinheit sei j = 1 (j = J) der einzige Startjob (Endjob). (Damit gilt $\mathcal{V}_1 = \phi$ und $\mathcal{V}_j \neq \phi$ für $j \neq 1$ sowie $\mathcal{N}_J = \phi$ und $\mathcal{N}_j \neq \phi$ für $j \neq J$.)
- Zur Bearbeitung der Jobs stehen r = 1,2,...,R erneuerbare Ressourcen (Personal, Betriebsmittel) zur Verfügung, und zwar mit K_r Einheiten pro Periode.
- Die Bearbeitung von Job j dauert d_j Zeiteinheiten und beansprucht von Ressource r genau k_{jr} Kapazitätseinheiten pro Periode.

Gehen wir nun davon aus, daß eine obere Schranke T für die Projektdauer (vom Anwender vorgegeben bzw. durch Addition der Bearbeitungsdauern oder Anwendung einer Heuristik ermittelt) gegeben ist, so können wir früheste Anfangs- und Endzeitpunkte FA_j und FE_j mit Hilfe der bei der Zeitplanung üblichen Vorwärtsrekursion sowie nach Setzen von $SE_J = T$ auch späteste Anfangs- und Endzeitpunkte SA_j und SE_j durch Rückwärtsrekursion berechnen.[27] Alle so ermittelten Zeitpunkte beziehen sich, sofern T "groß genug" gewählt wurde, auf einen hinsichtlich Reihenfolge- und Kapazitätsrestriktionen zulässigen Ablaufplan.

---

24 Hinweise zu weiteren betriebswirtschaftlichen Anwendungen von Expertensystemen findet man z.B. in **Mertens** et al. (1988), **Gabriel / Frick** (1991) sowie **Lehner** (1991).

25 Ansätze der KI zur Unterstützung der Netzplantechnik sind beispielsweise in **Zelewski** (1988), **Bartusch** et al. (1989) sowie **Klein** et al. (1990) zu finden.

26 Ein Überblick über weitere verwandte Problemstellungen ist beispielsweise in **Domschke / Drexl** (1991 b) zu finden.

27 Vgl. hierzu **Domschke / Drexl** (1991 a, S. 89 ff.).

Es stellt sich nun die Frage, welcher (bzgl. der Restriktionen zulässige) Ablaufplan die gesamte Projektdauer (Durchlaufzeit, makespan) minimiert. Dieses Entscheidungsproblem läßt sich auf unterschiedliche Art und Weise formulieren. Wir beginnen mit der Abbildung als "konventionelles" binäres Optimierungsproblem.

2. Konventionelle Modellformulierung

Verwenden wir Binärvariablen x_{jt} mit der Bedeutung $x_{jt} = 1$ (0), falls Job j in Periode t beendet (nicht beendet) wird, so können wir das Problem folgendermaßen formulieren:

$$\min \quad \sum_{t=FE_J}^{SE_J} t \cdot x_{Jt} \tag{1}$$

unter den Nebenbedingungen

$$\sum_{t=FE_j}^{SE_j} x_{jt} = 1 \qquad \text{(für alle j)} \tag{2}$$

$$\sum_{t=FE_h}^{SE_h} t \cdot x_{ht} \leq \sum_{t=FE_j}^{SE_j} (t-d_j)\, x_{jt} \qquad \text{(für alle j und } h \in \mathcal{V}_j) \tag{3}$$

$$\sum_{j=1}^{J} k_{jr} \sum_{q=t}^{t+d_j-1} x_{jq} \leq K_r \qquad \text{(für alle r und t)} \tag{4}$$

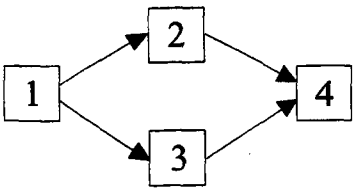
$$x_{jt} \in \{0, 1\} \qquad \text{(für alle j und } t = FE_j,\ldots,SE_j) \tag{5}$$

Zur Erläuterung: (1) führt zur Minimierung der Durchlaufzeit (makespan) des Projektes. (2) erzwingt die genau einmalige Bearbeitung eines Jobs. (3) sichert die Einhaltung der Reihenfolgebeziehungen zwischen Vorgängen. (4) verhindert Kapazitätsüberschreitungen. Binärvariablen brauchen in (5) nur für Perioden t im Intervall FE_j bis SE_j (Zeitfenster für die Bearbeitung von Job j) vorgesehen zu werden.

3. Darstellung in Form von Regeln

Wir betrachten das folgende bewußt sehr einfach gehaltene Beispiel (vgl. auch *Abbildung 2)*: Ein Projekt mit vier Jobs sei bis zum Zeitpunkt T = 3 zu bearbeiten. Dabei gilt: $d_1 = 0$, $d_2 = d_3 = d_4 = 1$, $\mathcal{V}_1 = \phi$, $\mathcal{V}_2 = \mathcal{V}_3 = \{1\}$, $\mathcal{V}_4 = \{2, 3\}$. Von der (einzigen) Ressource r = R = 1 sei eine Einheit pro Periode vorhanden ($K_1 = 1$); die Nachfragen seien $k_{11} = 0$, $k_{21} = k_{31} = k_{41} = 1$.

Abbildung 2: Netzplan für das Beispiel aus E.3



Mit T = 3 erhalten wir $FE_1 = 0$, $FE_2 = FE_3 = 1$, $FE_4 = 2$ sowie $SE_1 = 1$, $SE_2 = SE_3 = 2$, $SE_4 = 3$.

Die Zielfunktion (1) kann sehr einfach mit Hilfe des "minimize-Prädikats" modelliert werden. Die Restriktion (2) lautet für Job j = 2:

$$x_{21} + x_{22} = 1$$

Gehen wir nun davon aus, daß $x_{jt} = 1$ (0) dem Wert der Boole'schen Variablen "wahr" ("falsch") entspricht, so läßt sich diese Gleichung äquivalent durch die folgenden Regeln ausdrücken:

(a) $x_{21} \rightarrow \neg\, x_{22}$ (b) $x_{22} \rightarrow \neg\, x_{21}$

(c) $\neg\, x_{21} \rightarrow x_{22}$ (d) $\neg\, x_{22} \rightarrow x_{21}$

Das bedeutet: Wird Job 2 im Zeitpunkt 1 beendet, so impliziert das, daß er nicht gleichzeitig im Zeitpunkt 2 beendet werden darf (a). Wird Job 2 im Zeitpunkt 2 beendet, so impliziert das umgekehrt, daß er nicht gleichzeitig im Zeitpunkt 1 beendet werden darf (b). Wird Job 2 nicht im Zeitpunkt 1 beendet, so folgt daraus, daß er im Zeitpunkt 2 beendet werden muß (c). Wird Job 2 nicht im Zeitpunkt 2 beendet, so folgt daraus, daß er im Zeitpunkt 1 beendet werden muß (d).

Für die Reihenfolgebeziehung zwischen Job 2 und 4 lautet die Restriktion (3):

$$1\, x_{21} + 2\, x_{22} \leq (2\text{-}1)\, x_{42} + (3\text{-}1)\, x_{43}$$

Eine äquivalente Darstellung in Form von Regeln sieht folgendermaßen aus:

$$x_{21} \rightarrow x_{42} \vee x_{43}$$

$$x_{22} \rightarrow \neg\, x_{42} \wedge x_{43}$$

Das bedeutet: Wird Job 2 im Zeitpunkt 1 beendet, so kann Job 4 entweder im Zeitpunkt 2 oder im Zeitpunkt 3 beendet werden. Wird Job 2 jedoch im Zeitpunkt 2 beendet, so kann Job 4 nicht im Zeitpunkt 2 beendet werden, er muß vielmehr im Zeitpunkt 3 beendet werden.

Für die Restriktion (4) ergibt sich u.a.:

$$x_{21} + x_{31} \leq 1$$

$$x_{22} + x_{32} + x_{42} \leq 1$$

Die Formulierung von Regeln liefert unter der Annahme, daß (a)-(d) gelten:

$$x_{21} \rightarrow \neg\, x_{31}$$

$$x_{22} \rightarrow \neg\, x_{32} \wedge \neg\, x_{42}$$

Das bedeutet: Wird Job 2 im Zeitpunkt 1 beendet, so kann nicht auch noch gleichzeitig Job 3 im Zeitpunkt 1 beendet werden. Wird Job 2 im Zeitpunkt 2 beendet, so können weder Job 3 noch Job 4 im Zeitpunkt 2 beendet werden.

Aus der damit offensichtlichen inhaltlichen Äquivalenz der verschiedenen Darstellungsformen folgt, daß die Unterschiede zwischen herkömmlichen Modellen des Operations Research und regelbasierten Modellen (Systemen) teilweise eher phänomenologischer Natur sind – mit der Folge, daß (salopp formuliert) "das Modell an sich" und nicht die Art und Weise des "Aufschriebs" von eigentlichem Interesse sein sollte.

4. Konzeptionelle Modellformulierung

Wir formulieren nun ein zu (1)-(5) äquivalentes Modell. Dieses Modell ist der Denkweise bzw. dem Problemverständnis des Menschen "angemessener" und damit leichter zu verstehen, und zwar aus folgenden Überlegungen: Ein Mensch zerlegt ein Ablaufplanungsproblem nicht in binäre Einheiten (Variablen), wie es die obige konventionelle Modellbildung erfordert; vielmehr fordert er lediglich, daß alle in einem Zeitpunkt zu bearbeitenden Jobs nicht mehr als die verfügbaren Ressourcen nachfragen, wobei jeder Job a priori nur innerhalb eines vorgegebenen Zeitfensters eingeplant werden darf. Dieser Prozeß der (konzeptuellen) Modellbildung führt unmittelbar zu folgendem Modell:

$$\min \quad y_J \tag{6}$$

unter den Nebenbedingungen

$$y_h \leq y_j - d_j \qquad \text{(für alle j und alle } h \in \mathcal{V}_j\text{)} \tag{7}$$

$$\sum_{j \in \mathcal{A}_t} k_{jr} \leq K_r \qquad \text{(für alle r und t)} \tag{8}$$

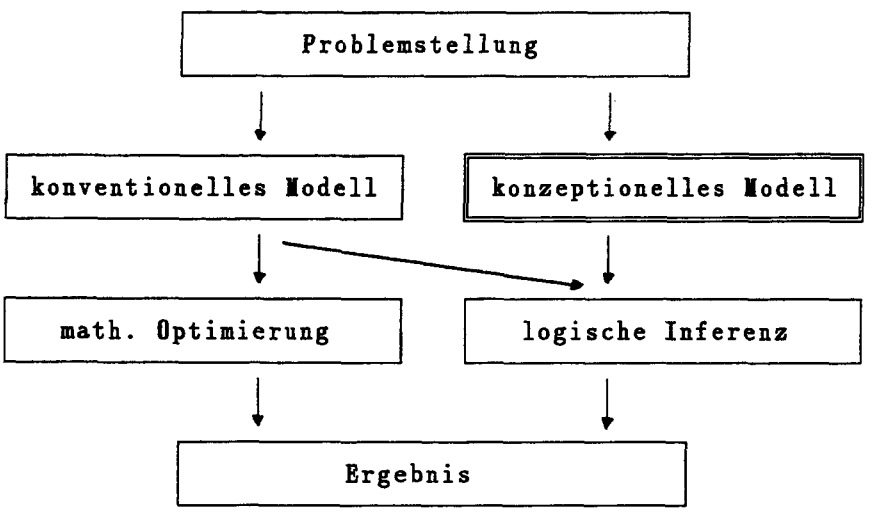
$$y_j \in \{FE_j, \ldots, SE_j\} \qquad \text{(für alle j)} \tag{9}$$

Diese Modellierung des Problems wurde unseres Wissens zuerst von **Talbot / Patterson** (1978) veröffentlicht. y_j ist dabei die ganzzahlig aus (dem Intervall) der Menge zulässiger Beendigungszeitpunkte zu wählende Beendigungsvariable für Job j. $\mathcal{A}_t$ bezeichnet die Menge der in der Periode t aktiven Jobs, d.h. derjenigen Jobs, die in der Periode t Ressourcen nachfragen.

Im Modell (6)-(9) bedarf das Symbol A_t einer näheren Erläuterung: Die Menge der in Periode t aktiven Jobs ist ex ante nicht bestimmbar, vielmehr ist diese Größe das Ergebnis der Festlegung von Bearbeitungsreihenfolgen. Weil die Größe A_t damit eigentlich nicht bekannt ist, durch (6)-(9) jedoch die Annahmen korrekt wiedergegeben werden, bezeichnen wir (6)-(9) als **konzeptionelles Modell.** Im Gegensatz dazu sind in (1)-(5) alle Größen ex ante ermittelbar. Die Folge ist, daß (1)-(5) mit Hilfe von Software zur binären (bzw. gemischt-ganzzahligen) Optimierung gelöst werden kann, während (6)-(9) mit Software zur ganzzahligen Optimierung nicht zu lösen ist. (6)-(9) ist jedoch **unmittelbar** in (PROLOG-basierten) Modellierungssprachen zu codieren und vom (Standard-) Inferenzmechanismus zu bearbeiten.

Das Modell (1)-(5) wurde "konventionell" (d.h. prozedural) mit Software zur gemischt-ganzzahligen Programmierung (LINDO) gelöst. Ferner wurden die beiden Modelle (1)-(5) und (6)-(9) in CHARME implementiert. *Abbildung 3* zeigt im Überblick, wie man auf unterschiedlichen Wegen von der Problemstellung zum Ergebnis gelangt, d.h. von demselben Ausgangspunkt (Problemstellung) ausgehend an dasselbe Ziel (Ergebnis).

Abbildung 3: Von der Problemstellung zum Ergebnis



Von besonderem Interesse ist in *Abbildung 3* das konzeptionelle Modell (doppelt umrandet). Zur Verdeutlichung der oben bereist genannten Vorzüge dieser Modellierung (entspricht Denkweise des Menschen etc.) geben wir im folgenden *ausgewählte Implementierungsdetails* wieder.

Der CHARME-Code der Kapazitätsrestriktion (4) lautet:[28]

---

28 Wie man sieht, lehnt sich die Syntax von CHARME an die Programmiersprache C an. Darüber hinaus ist die Umsetzung der jeweiligen Restriktion als "Constraint" unmittelbar und selbsterklärend nachvollziehbar und bedarf daher keiner weiteren Erläuterung.

```
{
for Rr in 1..R do
    for Tt in 1..T do
        {
        array SummeJj :: [1..J];
        for Jj in 1..J do
            {
            array SummeTt :: [Tt..Tt+D[Jj]-1];
            for Qq in Tt..Tt+D[Jj]-1 do
                if Qq >= FE[Jj]; Qq <= SE[Jj] then
                    SummeTt[Qq] = X[Jj,Qq]*Kk[Jj,Rr]
                else
                    SummeTt[Qq] = 0;
            sum (SummeTt, SummeJj[Jj]);
            }
        sum (SummeJj, Summe);
        Summe <= K[Rr];
        };
}
```

Im Vergleich dazu der CHARME-Code der Kapazitätsrestriktion (8):

```
{
for Rr in 1..R do
    for Tt in 1..T do
        {
        local RrSum;
        RrSum <= K[Rr];
        scal_prod (Active[Tt, 1..J], Kk[1..J, Rr], RrSum);
        };
}
```

Die Verwendung des (dynamischen) Parameters "Active" in der CHARME-Modellierung (8) entspricht – wie bereits erwähnt – der Art und Weise, wie der Mensch die Einhaltung von Kapazitätsrestriktionen überprüft, in dem er für alle im Zeitpunkt t "aktiven" Jobs prüft, ob ausreichend Ressourcen vorhanden sind.

Wie im folgenden gezeigt wird, hat die konzeptionelle Modellierung (über den Vorzug der Kompaktheit und der unmittelbaren Nachvollziehbarkeit hinaus) den weiteren Vorteil einer (im Mittel) besseren Laufzeiteffizienz. Um das zu zeigen, gehen wir im folgenden zunächst ausführlicher auf ein Beispiel ein. Anschließend berichten wir über numerische Ergebnisse, die mit beiden Implementierungen erzielt wurden.

F. Beispiel und numerische Ergebnisse

Wir betrachten folgendes Beispiel: Gegeben sei der Netzplan von *Abbildung 4*; er besteht aus den J = 10 Vorgängen 1 bis 10 mit dem (fiktiven[29]) Startjob 1 sowie dem (fiktiven) Endjob 10. Das Projekt muß spätestens im Zeitpunkt T = 11 fertiggestellt sein.

Abbildung 4: Netzplan für das Beispiel aus F

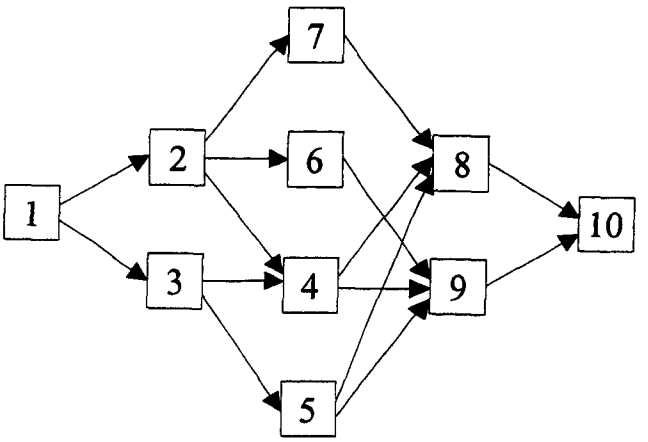


Tabelle 1 gibt die Dauern d_j der Vorgänge wieder. *Tabelle 2* enthält die Kapazitätsgebräuche (dabei bedeutet "–", daß keine Einheit der betreffenden Ressource nachgefragt wird) sowie *Tabelle 3* die Verfügbarkeit der R = 3 zu beachtenden Ressourcen für drei – ansonsten identische – Datensätze (a), (b) bzw. (c).

Tabelle 1 : Vorgangsdauern

j	1	2	3	4	5	6	7	8	9	10
d_j	0	1	2	2	2	2	3	3	4	0

Tabelle 2 : Kapazitätsgebräuche

j	1	2	3	4	5	6	7	8	9	10
k_{j1}	–	3	–	–	–	7	6	–	4	–
k_{j2}	–	6	–	4	4	–	–	7	–	–
k_{j3}	–	–	8	–	10	–	2	3	6	–

29 Ein Vorgang j mit $d_j = 0$ und $k_{jr} = 0$ (für alle r) wird als fiktiv bezeichnet.

Tabelle 3 : Ressourcenverfügbarkeit

Problem	(a)	(b)	(c)
K_1	12	10	8
K_2	8	8	7
K_3	12	11	10

Abbildung 5 enthält das Gantt-Diagramm einer optimalen Lösung für den Datensatz (c);[30] *Abbildung 6* gibt die zugehörige konsistente (zulässige) Belegung der drei Ressourcen wieder, wobei die Notation "j(k)" bedeutet, daß Job j genau k Einheiten der betreffenden Ressource r pro Zeiteinheit erfordert.[31]

Tabelle 4 zeigt den Einfluß einer Variation des Zeitpunktes T, in dem das Projekt spätestens fertiggestellt sein muß, auf die Anzahl zulässiger Lösungen des jeweiligen Problems. Man sieht, daß mit knapper werdenden Kapazitäten – erwartungsgemäß – deutlich weniger zulässige Ablaufpläne existieren, aus denen der Entscheidungsträger ggf. eine Auswahl treffen kann.

Tabelle 4: Anzahl zulässiger Lösungen

Problem	(a)	(b)	(c)
T = 8	–	–	–
T = 9	6	0	0
T = 10	134	30	2
T = 11	1.082	341	45
T = 12	5.389	2.031	391
T = 13	20.063	8.564	2.120
T = 14	61.338	28.814	8.549
T = 15	162.491	82.437	28.047

Die konzeptionelle Modellierung ist der konventionellen nicht nur hinsichtlich ihrer Kompaktheit vorzuziehen, sie führt auch zu einem günstigeren Antwortverhalten der Inferenzmaschine.[32] Im folgenden berichten wir über Erfahrungen bei der Lösung von 24 Testdatensätzen mit den Datensatzcharakteristika von Tabelle 5.[33]

---

30 In den Abbildungen 5 und 6 werden die (fiktiven) Vorgänge 1 und 10 der Einfachheit halber weggelassen.

31 Die für den Datensatz (c) optimale Lösung ist für die Datensätze (a) und (b) lediglich zulässig.

32 Die Umsetzung des Modells (1)-(5) in das MPS-Datenformat und die anschließende Lösung mit Hilfe von LINDO, einem Softwarepaket zur gemischt-ganzzahligen Optimierung, führt im Mittel zu (noch) höheren Rechenzeiten als die Umsetzung von (1)-(5) in CHARME.

33 Mit dem parametrisch spezifizierbaren Projektdaten-Generator ProGen erzeugt; vgl. **Kolisch** et al. (1992).

Abbildung 5: Job-Gantt für Datensatz (c)

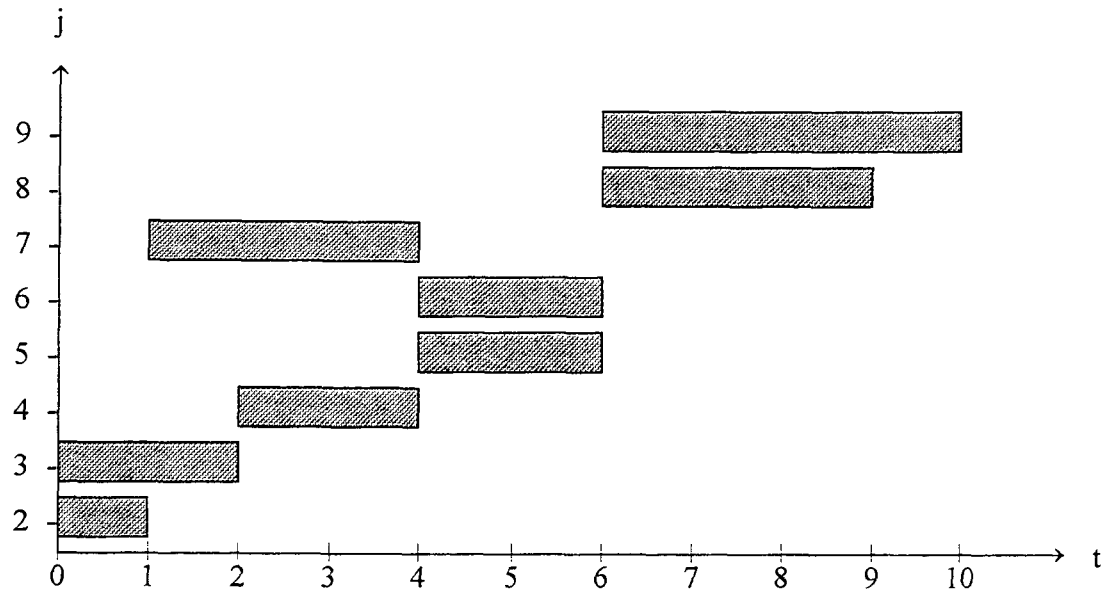


Abbildung 6: Ressourcen-Gantt für Datensatz (c)

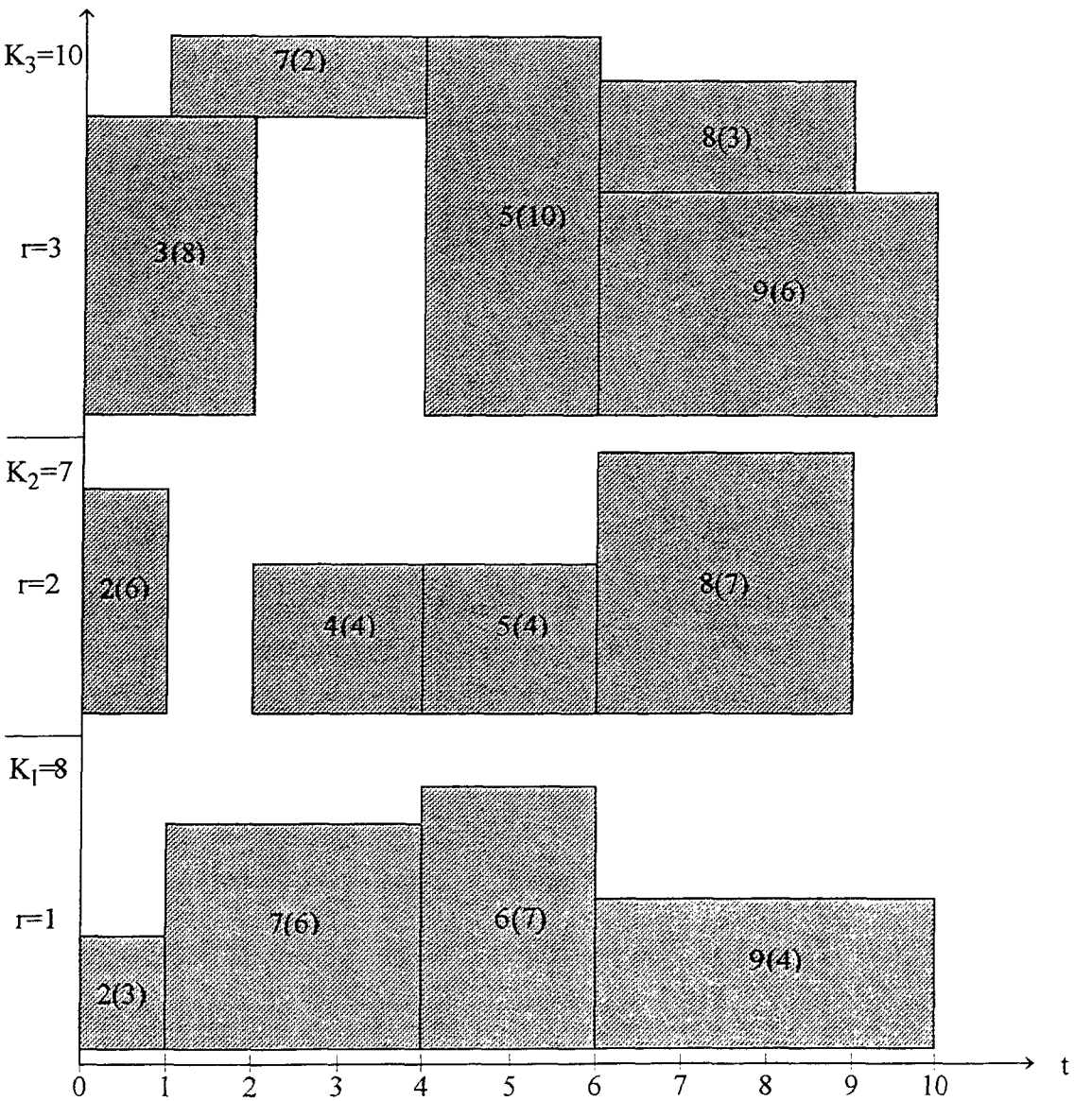


Tabelle 5: Datensatzcharakteristika

# Jobs	5	6	7	8	8	8	10	10
# Perioden	19	32	31	10	15	26	11	15

Von jedem der durch die Anzahl (#) der Jobs (keine fiktiven enthalten) und die Anzahl der Perioden gemäß *Tabelle 5* gekennzeichneten Projektplanungsprobleme wurden drei Datensätze erzeugt, die sich jeweils nur durch das Ressourcenangebot unterscheiden.

Die Anzahl existierender zulässiger Lösungen schwankt zwischen 0 und 383.668, im Mittel sind 34.285 Lösungen zulässig. Die durchschnittlich[34] für die Bestimmung aller Lösungen erforderliche Rechenzeit beträgt

- 4.054 CPU-sec für die konventionelle und
- 337 CPU-sec für die konzeptionelle

Modellformulierung. Damit ist letztere hinsichtlich des Antwortverhaltens um mehr als den Faktor 10 überlegen. Die Verteilung der CPU-sec für die einzelnen Modellierungsarten ist *Tabelle 6* zu entnehmen.

Tabelle 6: Verteilung der Rechenzeiten

CPU-sec	<10	11-100	101-1.000	1.001-1.0000	>10.000
konventionell	7	4	5	5	3
konzeptionell	10	3	9	2	0

G. Ergebnisse und Perspektiven

In diesem Beitrag wird für ein Problem aus dem Bereich des Projektmanagements eine konventionelle und eine konzeptionelle Modellformulierung vorgestellt und beispielhaft gezeigt, daß der PROLOG-Inferenzmechanismus auf dieser Basis die zulässigen (konsistenten) Lösungen findet. Die Effizienz beruht auf dem in CHARME implementierten Inferenzmechanismus und der Stärke der Restriktionen der Wissensbasis (6)-(9). Der derzeitige Stand der Forschung rechtfertigt wohl die Aussage, daß "... constraint logic programming is a useful computational logic for modeling, simulating, and verifying real-time discrete event processes" [35].

Die hier vorgelegten Ergebnisse sind über ihre "lokale" Bedeutung hinaus von Interesse: Bei der "gewöhnlichen" gemischt-ganzzahligen Modellierung von Problemen müssen alle Koeffizientenwertebereiche ex ante bekannt sein, damit ein "General Problem Solver" (z.B. Branch & Bound - Verfahren) eine zulässige oder auch optimale Lösung bestimmen kann. Eine konzep-

---

34 Auf einem 486 dx (33 MHz, 8 MB) unter SCO-UNIX.

35 **Ostroff** (1991).

tionelle Modellformulierung kann demgegenüber eine Problemstellung u.U. wesentlich kompakter repräsentieren und gleichzeitig, obwohl nicht alle ggf. erforderlichen Koeffizientenwertebereiche ex ante bekannt sind, von einem KI-basierten "General Problem Solver" bearbeitet werden.[36] Der Backtrackmechanismus wird dabei durch die Formulierung des Modells und die Vorgabe einiger Steuergrößen implizit definiert.

Für künftige Untersuchungen expertensystembasierter bzw. wissensbasierter Ansätze ergeben sich daraus eine Reihe von Implikationen; als Beispiele seien genannt:

- Bei der betriebswirtschaftlichen Modellbildung[37] steigt damit die Bedeutung des Herausarbeitens der Modellprämissen; sie bilden den aus "Anwendersicht" eigentlichen Kern der Arbeit.
- Von der Darstellung und kritischen Analyse der Prämissen sollte der gewählte Formalismus (konventionelle Modellierung, konzeptionelle Modellierung, regelbasiertes System) getrennt werden.
- Für die experimentelle Evaluation der Performance unterschiedlicher Systeme (Antwortverhalten, Lösungsgüte usw.) ergeben sich damit neue Möglichkeiten, die künftig verstärkt aufgegriffen und genutzt werden sollten.[38]

Generell werfen die hier präsentierten Ergebnisse die Frage auf, ob und unter welchen Bedingungen die in der Wissenschaft und in der Praxis teilweise bewußt forcierte Abgrenzung der Gegenstandsbereiche der speziellen Betriebswirtschaftslehren Operations Research und Wirtschaftsinformatik gerechtfertigt ist.

Literaturhinweise

Albach, H. (Hrsg.): Investitionstheorie. Köln 1975.

Bartusch, M. / Möhring, R.H. / Radermacher, F.J.: Design aspects of an advanced model-oriented DSS for scheduling problems in civil engineering. Decision Support Systems, Vol. 5 (1989), S. 321-344.

Bitner, J.R. / Reingold, E.M.: Backtrack programming techniques. Communications of the ACM, Vol. 18 (1975), S. 651-656.

Blair, C.E. / Jeroslow, R.G. / Lowe, J.K.: Some results and experiments in programming techniques for propositional logic. Computers & Operations Research, Vol. 13 (1986), S. 633-645.

Blohm, H. / Lüder, K.: Investition. 7. Aufl., München 1991.

Bonczek, R.H. / Holsapple, C.W. / Whinston, A.B.: Foundations of decision support systems. New York 1981.

---

36 Zu Vergleichen der Performance von "General Solvern" der künstlichen Intelligenz sowie der gemischt-ganzzahligen Programmierung vgl. z.B. **Dhar / Ranganathan** (1990) sowie **Jordan / Drexl** (1993).

37 Vgl. zur grundsätzlichen Bedeutung und zu den Möglichkeiten der betriebswirtschaftlichen Modellgestaltung z.B. **Schneeweiß** (1992).

38 Als (im Hinblick auf die Ausführungen von Abschnitt F naheliegendes) Beispiel sei folgendes genannt: Im Rahmen einer systematischen Untersuchung unter Verwendung der mit Hilfe von ProGen (vgl. **Kolisch** et al. (1992)) erzeugten Testdatensätze sollten verschiedene Implementierungen der in Abschnitt E vorgestellten Modelle und das daraus resultierenden Antwortverhalten logikorientierter und mathematischer Programmierungssysteme verglichen werden.

Bothner, P. / Kähler, W.-M.: Programmieren in PROLOG. Braunschweig 1991.

Bruynooghe, M. / Pereira, L.M.: Deduction revision by intelligent backtracking, in: **Campbell, J.A.** (Hrsg.): Implementation of Prolog. Chichester 1984, S. 194-215.

BULL: Artificial intelligence: CHARME - Eine Einführung. Ref.-Nr. 13.D2-5031, Köln 1991.

Clocksin, W.F. / Mellish, C.S.: Programming in Prolog. Berlin 1984.

Cohen, J.: Constraint logic programming languages. Communications of the ACM, Vol. 33 (1990), S. 52-68.

Colmerauer, A.: Opening the PROLOG III universe. Byte, Vol. 12 (1987), S. 177-182.

Colmerauer, A.: An introduction to PROLOG III. Communications of the ACM, Vol. 33 (1990), S. 69-90.

Cox, P.T.: Finding backtrack points for intelligent backtracking, in: **Campbell, J.A.** (Hrsg.): Implementation of Prolog. Chichester 1984, S. 216-233.

Dhar, V. / Ranganathan, N.: Integer programming vs. expert systems: An experimental evaluation. Communications of the ACM, Vol. 33 (1990), S. 323-336.

Dincbas, M. / Simonis, H. / van Hentenryck, P.: Solving large combinatorial problems in logic programming. Journal of Logic Programming, Vol. 8 (1990), S. 75-93.

Dinkelbach, W.: Entscheidungsmodelle. Berlin 1982.

Domschke, W. / Drexl, A.: Einführung in Operations Research. 2. Aufl., Berlin 1991 a.

Domschke, W. / Drexl, A.: Kapazitätsplanung in Netzwerken - Ein Überblick über neuere Modelle und Methoden. OR Spektrum, Bd. 13 (1991 b), S. 63-76.

Freuder, E.C. / Wallace, R.J.: Partial constraint satisfaction. Artificial Intelligence, Vol. 58 (1992), S. 21-70.

Gabriel, R. / Frick, D.: Expertensysteme zur Lösung betriebswirtschaftlicher Problemstellungen. Zeitschrift für betriebswirtschaftliche Forschung, Jg. 43 (1991), S. 544-565.

Harmon, P. / King, D.: Expertensysteme in der Praxis - Perspektiven, Werkzeuge, Erfahrungen. München 1986.

van Hentenryck, P.: Consistency techniques in logic programming. Ph.D. Thesis, University Namur, Belgium 1987.

Hooker, J.N.: A quantitative approach to logical inference. Decision Support Systems, Vol. 4 (1988), S. 45-69.

Jackson, P.: Introduction to expert systems. Reading (Mass.) 1986.

Jordan, C. / Drexl, A.: A comparison of logic and mixed-integer programming solvers for batch sequencing with sequence-dependent setups. Arbeitspapier, Universität Kiel 1993.

Jost, T. / Skuppin, R.: Technical diagnosis based on numerical models using PROLOG III. Conf. Proc. ESPRIT 1989, Kluwer, S. 513-527.

de Kleer, J.: An assumption based truth maintenance system. Artificial Intelligence, Vol. 28 (1986), S. 127-162.

de Kleer, J.: A comparison of ATMS and CSP techniques. Proc. IJCAI 1989, S. 290-296.

Kleine Büning, H. / Schmitgen, S.: PROLOG - Grundlagen und Anwendungen. 2. Aufl., Stuttgart 1988.

Klein, J. / König, W. / Stiasni, Chr.: Wissensbasierte Generierung von Projektplänen bei Dienstleistungsproduktion mit Kleinseriencharakter - Ein Ansatz mit den wissensbasierten Systemen NETEX und BEREX, in: **Ehrenberg, D.** et al. (Hrsg.): Wissensbasierte Systeme in der Betriebswirtschaft. Berlin 1990, S. 453-469.

Kolisch, R. / Sprecher, A. / Drexl, A.: Characterization and generation of a general class of resource-constrained project scheduling problems: Easy and hard instances. Arbeitspapier, Universität Kiel Kiel 1992.

Krautter, W. / Steinert, M.: A knowledge representation for model-based reasoning using PROLOG III. Conf. Proc. ESPRIT 1988, North-Holland, S. 814-825.

Kruschwitz, L.: Investitionsrechnung. 5. Aufl., Berlin - New York 1993.

Kurbel, K.: Entwicklung und Einsatz von Expertensystemen - Eine anwendungsorientierte Einführung in wissensbasierte Systeme. Berlin 1989.

Lauriere, J.: A language and a program for stating and solving combinatorial problems. Artificial Intelligence, Vol. 10 (1978), S. 29-127.

Leler, W.: Constraint Programming Languages: Their Specification and Generation. Reading (Mass.) 1988.

Lehner, F.: Expertensysteme für Organisationsaufgaben. Zeitschrift für Betriebswirtschaft, Jg. 61 (1991), S. 737-754.

Lodwick, W.A.: Constraint propagation, relational arithmetic in AI systems and mathema tical programs. Annals of Operations Research, Vol. 21 (1989), S. 143-148.

Luger, G.F. / Stubblefield, W.A.: Artificial intelligence and the design of expert systems. Redwood City (California) 1989.

Mertens, P. / Borowski, V. / Geis, W.: Betriebliche Expertensystem-Anwendungen - Eine Materialsammlung. Berlin 1988.

Mitterreiter, I. / Radermacher, F.J.: Some notes on experiments on the running time behaviour of some algorithms solving propositional logical problems. Arbeitspapier, FAW / Ulm 1991.

Ostroff, J.S.: Constraint logic programming for reasoning about discrete event processes. Journal of Logic Programming, Vol. 11 (1991), S. 243-270.

Pesch, E. / Drexl, A. / Kolen, A. : Modellbasierte Inferenz in CHARME. Arbeitspapier, Rijksuniversiteit Limburg, Maastricht 1993.

Pique, J.-F.: Prolog II, a step on the Prolog road. AICOM, Vol. 1 (1988), S. 4-16.

Puppe, F.: Einführung in Expertensysteme. 2. Aufl., Berlin 1991.

Schneeweiß, Chr.: Planung 2 - Konzepte der Prozeß- und Modellgestaltung. Berlin 1992.

Schreiber, G.: A KADS design description language. Esprit-KADS, University of Amsterdam 1990.

Sterling, L. / Shapiro, E.: The Art of Prolog. Cambridge (Mass.) 1986.

Talbot, F.B. / Patterson, J.H.: An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. Management Science, Vol. 24 (1978), S. 1163-1174.

Tate, A.: A review of knowledge-based-planning techniques. Expert Systems, Vol. 85 (1985), S. 89-111.

Waterman, D.A.: A guide to expert systems. Reading (Mass.) 1986.

Williams, H.P.: Logical problems and integer programming. Bull. Inst. Math. Appl., Vol. 13 (1977), S. 18-20.

Winston, P.H.: Artificial intelligence. 2. Aufl., Reading (Mass.) 1984.

Zelewski, S.: Ansätze der Künstlichen Intelligenz-Forschung zur Unterstützung der Netzplan-technik. Zeitschrift für betriebswirtschaftliche Forschung, Jg. 40 (1988), S. 1112-1129.

Zusammenfassung

Der vorliegende Beitrag beschäftigt sich mit der Entwicklung regelbasierter Expertensysteme. Das Hauptaugenmerk gilt unterschiedlichen Möglichkeiten der Modellierung als Voraussetzung für das Aufstellen von Regeln. Wir behandeln (außer einem sehr einfachen Beispiel aus der Invesitionstheorie) exemplarisch eine Problemstellung aus dem Bereich des Projektmanagements und zeigen die Äquivalenz von binärer und regelbasierter Modellierung. Diese Feststellung läßt den Schluß zu, daß die Unterschiede zwischen herkömmlichen Modellen des Operations Research und Wissensbasen teilweise eher phänomenologischer Natur sind. Ferner präsentieren wir eine kompakte konzeptionelle Modellierung, die direkt in deklarativen Sprachen wie CHARME codiert und vom Inferenzmechanismus ausgewertet werden kann.

Summary

The paper deals with the development of rule-based expert systems. In addition to a simple example from investment theory we focus on different models for resource-constrained project scheduling problems as a basis for formulating rules. We show that the binary model formulation may be equivalently stated via rules (boolean expressions), which means that a linkage between Artificial Intelligence and Operations Research can be stated. Furthermore, we present a compact conceptual model. Whereas it is impossible to use the conceptual model by general purpose mixed-integer programming packages, it is, however, directly accessible to the inference engine of declarative logic programming languages like CHARME.