Hybride Optimierung als Werkzeug zur Lösung von KMU-relevanten Planungs- und Steuerungsproblemen

(Forschungsbericht des Projekts V, 2000)

Dipl.-Inf. Sven Hader
Fakultät für Informatik
Technische Universität Chemnitz

Innovationskolleg INK 17/A1 - 1

Bildung eines vernetzten Logistik- und Simulationszentrums

Zusammenfassung

Eine Vielzahl von Unternehmen muß heute in einem Marktumfeld agieren, das geprägt ist durch zunehmenden Konkurrenzdruck einerseits und wachsende Kundenforderungen, vor allem nach hoher Produktindividualität und -qualität sowie kurzer Lieferzeit und niedrigem Preis, andererseits. Dadurch wird es zunehmend erforderlich, jegliche betriebliche Entscheidungsfindung unter dem Gesichtspunkt der Optimalität zu betrachten.

In diesem Forschungsbericht wird das neuartige Konzept der adaptiv-hybriden Optimierung vorgestellt und eine konkrete Realisierung des Konzepts in Form des Optimierungssystems DynamO beschrieben. DynamO ist in der Lage, praxisrelevante Planungs- und Steuerungsprobleme rechnergestützt und automatisch zu lösen und kann dadurch die Qualität und Effizienz der Entscheidungsfindung in Unternehmen (speziell KMU) entscheidend verbessern.

Inhaltsverzeichnis

1 Einleitung 1

2 Die Lösung von Planungs- und Steuerungsproblemen mittels Optimierung 3

2.1 Grundlagen 3
2.2 Anforderungen an die zu lösenden Probleme 4
2.3 Beispiele für lösbare Planungs- und Steuerungsprobleme 5

3 Die hybride Optimierung 7

3.1 Grundlagen 7
3.2 Taxonomie hybrider Optimierungsverfahren 8
3.3 Parametrierung hybrider Optimierungsverfahren 10

4 DynamO - ein hybrides Optimierungssystem auf Multiagentenbasis 11

4.1 Das Konzept eines adaptiv-hybriden Optimierungsverfahrens 11
4.2 Multiagentensysteme 13
4.3 Das Optimierungssystem DynamO 14

5 Ein Anwendungsbeispiel 19

5.1 Problemstellung 19
5.2 Wesentliche Merkmale des Problems 20
5.3 Ergebnisse 21

Anhang A: Die Aufgabenbeschreibungssprache von DynamO 23

A.1 Datenobjekte 23
A.2 Arithmetische Ausdrücke 24
A.3 Beschreibungsblöcke 26

Anhang B: Schnittstellen von DynamO 33

B.1 Die Benutzerschnittstelle 33
B.1.1 Verwaltung von Optimierungsaufgabenbeschreibungen 34
B.1.2 Konfigurierung des hybriden Optimierungsverfahrens 35
B.1.3 Durchführung der Optimierung 37
B.1.4 Verwaltung von Optimierungsergebnissen 38
B.2 Die Dateischnittstelle 39
B.3 Die Optimierungsaufgaben-Programmschnittstelle 39
B.4 Die Generierungs-Programmschnittstelle 40
B.5 Die Verarbeitungs-Programmschnittstelle 41

Anhang C: Installation von DynamO 43
C.1 Systemvoraussetzungen 43
C.2 Installation 43
C.3 Bekannte Probleme 44

Literaturverzeichnis 45

1 Einleitung

Die aktuelle Marktsituation für Unternehmen ist gekennzeichnet durch zunehmenden Konkurrenzdruck einerseits und wachsende Kundenforderungen, vor allem nach hoher Produktindividualität und -qualität sowie kurzer Lieferzeit und niedrigem Preis, andererseits. Kleine und mittlere Unternehmen (im folgenden als *KMU* bezeichnet) mit ihren geringen personellen und finanziellen Ressourcen sind dadurch in besonderem Maße gezwungen, geeignete Maßnahmen zur Steigerung ihrer Flexibilität und Effizienz bei gleichzeitiger Reduzierung der Kosten zu ergreifen, um am Markt bestehen zu können. Somit ist es erforderlich, jegliche betriebliche Entscheidungsfindung, sei es die Projektierung von Produktions- und Logistiksystemen, der Entwurf von Produkten oder die Produktionsplanung und Fertigungssteuerung, unter dem Gesichtspunkt der *Optimalität* zu betrachten. Die qualitativ hochwertige und termingerechte Lösung derartiger Problemstellungen kann vom Menschen allein oft nicht mehr bewältigt werden, vor allem wegen der hohen Komplexität der Aufgabe. Aus diesem Grund kommen zur Unterstützung der Entscheidungsfindung zunehmend rechnergestützte, automatisch arbeitende Verfahren zum Einsatz, die nicht nur deren Qualität und Effizienz erhöhen, sondern auch den Menschen zeitlich entlasten.

Betriebliche Problemstellungen, für die quantitativ erfaßbare Zielkriterien, wie z.B. Kosten, Auslastung und Termintreue, formulierbar sind, können mit den Mitteln der *Optimierung* direkt gelöst werden. Existiert zusätzlich ein adäquates rechnerinternes Modell des Problembereichs, z.B. in Form eines Analysetools oder Simulators, kann die Problemlösung durch Anwendung rechnergestützter Optimierungsverfahren automatisiert werden. Da kein „universelles“ Optimierungsverfahren existiert, hat die Festlegung, mit welchem Verfahren und mit welchen Verfahrensparametern ein Problem zu lösen ist, entscheidenden Einfluß auf die erreichbare Lösungsqualität und Effizienz. In den allermeisten KMU kann eine fundierte Entscheidung darüber jedoch wegen des unzureichenden Wissensstandes nicht getroffen werden, so daß die Verfahrensauswahl und -parametrierung eher von „zufälligen“ Faktoren („nehmen wir schon immer“, „darüber habe ich gerade was gelesen/gehört“, etc.) beeinflußt wird. Dadurch bleibt wertvolles Optimierungspotential im Unternehmen ungenutzt und seine Wettbewerbsfähigkeit wird geschwächt.

In dieser Arbeit wird das neuartige Konzept eines *adaptiv-hybriden Optimierungsverfahrens* vorgestellt, das es auch ohne Spezialkenntnisse ermöglicht, eine Vielzahl von Planungs- und Steuerungsproblemen (und darüber hinaus auch andere Optimierungsprobleme) rechnergestützt und automatisch zu lösen. Wesentlicher Vorteil dieses Konzepts ist die „intelligente“ Ausnutzung der für die Optimierung zur Verfügung stehenden Zeit durch selbsttätige Auswahl und Parametrierung geeigneter Optimierungsverfahren, ohne daß ein Eingriff durch den Nutzer notwendig ist. Dadurch ist das Konzept insbesondere für den Einsatz in Softwaretools für KMU geeignet. Ein derartiges Softwaretool, das Optimierungssystem **DynamO**, wird in der vorliegenden Arbeit eingehend beschrieben.

2 Die Lösung von Planungs- und Steuerungsproblemen mittels Optimierung

2.1 Grundlagen

Unter *Optimierung* wird in der allgemeinsten Bedeutung die Suche nach dem bezüglich eines Gütekriteriums besten Element einer Menge verstanden. Diese Suche kann als Lösung eines Optimierungsproblems mittels eines Optimierungsverfahrens beschrieben werden. Das Optimierungsproblem spezifiziert die zu betrachtende Menge sowie das zu verwendende Gütekriterium. Das Optimierungsverfahren spezifiziert die Art der Auswahl von zu bewertenden Elementen aus der Menge.

In der Praxis orientiert sich die Beschreibung von Optimierungsproblemen meist an der gebräuchlichen mathematischen Notation:

$$\begin{aligned} &\text{minimiere}^{1} \quad f(\mathbf{x}) \quad , \quad \mathbf{x} \in S \\ &\text{unter} \qquad g_1(\mathbf{x}) \geq 0 \,, \ldots \\ &\qquad\qquad h_1(\mathbf{x}) = 0 \,, \ldots \end{aligned}$$

$\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ ist ein Parametervektor der Dimension *n* (≡ Anzahl Parameter). *S* ist eine Menge, die mindestens alle zu betrachtenden Elemente enthält (der *Suchraum*, meist R^n). $f(\mathbf{x})$ ist die Zielfunktion und repräsentiert das Gütekriterium. $g_i(\mathbf{x})$ ist die i-te Ungleichungs-Nebenbedingung, $h_j(\mathbf{x})$ die j-te Gleichungs-Nebenbedingung. Der Suchraum und die Gesamtheit der Nebenbedingungen definieren die Menge *L* (den *Lösungsraum*), die die im einführenden Absatz genannte Menge repräsentiert:

$$L = \left\{ \mathbf{x} \in S \;\middle|\; g_i(\mathbf{x}) \geq 0 \;\forall g_i \;\wedge\; h_j(\mathbf{x}) = 0 \;\forall h_j \right\}$$

Ein Parametervektor $\mathbf{x} \in L$ wird als „gültig“ bezeichnet, da er alle Nebenbedingungen erfüllt und somit (potentiell) als Lösung in Frage kommt.

Ziel einer Optimierung ist das Finden des bzgl. des Gütekriteriums „besten“ Parametervektors $\mathbf{x}^*$ des Lösungsraums (*globaler Minimierer*), d.h.

$$f^* = f(\mathbf{x}^*) \leq f(\mathbf{x}) \;\forall \mathbf{x} \in L$$

f^* wird dabei als *globales Minimum* bezeichnet.

Da das Finden eines globalen Minimierers für bestimmte Problemklassen durchaus NP-vollständig sein kann (z.B. Travelling-Salesman-Problem), ist man in der Praxis meist schon mit einer „guten“ bis „sehr guten“ Lösung zufrieden.

---

[1] Wenn nicht anders angegeben, wird in dieser Arbeit davon ausgegangen, daß es sich bei der Optimierung um eine Minimierung handelt. Diese Annahme stellt keine Einschränkung dar, denn ein Maximierungsproblem mit Zielfunktion $f(\mathbf{x})$ ist äquivalent zu einem Minimierungsproblem mit Zielfunktion $-f(\mathbf{x})$.

2.2 Anforderungen an die zu lösenden Probleme

Alle Probleme, die durch Optimierung gelöst werden sollen, müssen in die im vorigen Kapitel beschriebene Form (Menge + Gütekriterium) überführbar sein. Die Überführung kann z.B. folgendermaßen erfolgen:

(1) Alle Parameter des zu lösenden Problems, die als „Stellgrößen" anzusehen sind, d.h. deren Wert bei der Optimierung verändert werden kann, werden identifiziert, ggf. geeignet geordnet und von 1 beginnend durchnumeriert. Die Anzahl derartiger Parameter sei *n*.

(2) Für jeden Parameter $x_i, i = 1 \ldots n$ wird der zugehörige Definitionsbereich D_i festgelegt, d.h. die Menge von Werten, die der Parameter prinzipiell annehmen kann.
Der Suchraum des Problems ergibt sich somit als $S = D_1 \times D_2 \times \ldots \times D_n$.

(3) Es wird festgelegt, welche Anforderungen ein Element des Suchraums (d.h. eine konkrete Parameterbelegung) erfüllen muß, um gültig zu sein und somit prinzipiell als Lösung in Frage zu kommen. Die Anforderungen sind so zu formulieren, daß sie einerseits „unmögliche" Elemente[1] und andererseits möglichst viele Elemente, von denen bekannt ist, daß sie als Lösung nicht in Frage kommen, ausschließen.
Der Lösungsraum *L* ergibt sich somit als Menge aller Elemente des Suchraums, die alle gestellten Anforderungen erfüllen.

(4) Es werden Bewertungskriterien festgelegt, deren Anwendung auf ein Element des Lösungsraums es erlaubt, möglichst genau zwischen „guten" und „schlechten" Elementen zu unterscheiden.

Wenn für ein zu lösendes Problem die vier Schritte dieser Methode erfolgreich ausgeführt werden können[2], handelt es sich um ein Optimierungsproblem. Um es mit Hilfe der in dieser Arbeit betrachteten Optimierungsverfahren lösen zu können, muß es jedoch eine Reihe zusätzlicher Eigenschaften aufweisen:

- Die Definitionsbereiche der Parameter müssen in Teilmengen von *R* (Bereich reeller Zahlen) überführbar sein, so daß die Parameter nur noch Zahlenwerte annehmen[3].
- Die Anforderungen an gültige Elemente müssen, falls vorhanden, als Funktionen formulierbar sein, die bei Angabe eines Elements einen Zahlenwert aus *R* liefern, der die Erfüllung der jeweiligen Anforderung widerspiegelt.
- Die Bewertungskriterien für gültige Elemente müssen in einer Zielfunktion zusammenfaßbar sein, die bei Angabe eines Elements einen seiner Eignung entsprechenden Zahlenwert aus *R* liefert[4].

---

[1] Die „Unmöglichkeit" bestimmter Elemente ergibt sich dadurch, daß bestimmte Parameterkonstellationen Naturgesetzen, technischen Normen o.ä. widersprechen.

[2] Der dritte Schritt (Anforderungen an gültige Elemente) kann entfallen, wenn alle Elemente des Suchraums gültig sind.

[3] Selbst bei Parametern, deren Werte nicht-numerisch sind (z.B. Materialbezeichnungen), stellt diese Forderung i.allg. keine allzu große Einschränkung dar, da sich solche Parameter meist in den Bereich der ganzen Zahlen „umkodieren" lassen.

[4] In dieser Arbeit wird der Bereich der *mehrkriteriellen Optimierung* ausgeklammert.

Es ist zu beachten, daß die Anforderungsfunktionen und die Zielfunktion nicht zwingend als explizite mathematische Formeln formuliert sein müssen, sondern auch durch Algorithmen spezifiziert sein können. Mit anderen Worten, die Funktionen dürfen auch durch externe Programme, z.B. Datenbanken, Analysetools oder Simulatoren, repräsentiert sein.

2.3 Beispiele für lösbare Planungs- und Steuerungsprobleme

Prinzipiell stellen sämtliche Planungs- und Steuerungsprobleme, die in Unternehmen zu lösen sind, Optimierungsprobleme dar. Die Menge der zu betrachtenden Elemente wird durch die bei der jeweiligen Planung bzw. Steuerung vorhandenen Freiheitsgrade bzw. Varianten festgelegt. Das Gütekriterium ergibt sich durch das Ziel, das durch die Planung bzw. Steuerung erreicht werden soll (möglichst geringe Investitionskosten, möglichst hohe Auslastung, etc.).

Obwohl somit alle Planungs- und Steuerungsprobleme prinzipiell durch Optimierung gelöst werden können, treten in der Praxis häufig Schwierigkeiten auf, da die Probleme z.T. nicht die in Kapitel 2.2 genannten Eigenschaften aufweisen. Besonders die Formulierung einer Zielfunktion, die alle Bewertungskriterien quantitativ und im richtigen Verhältnis zusammenfaßt, ist oft sehr kompliziert[1]. Wegen der Vielzahl und Verschiedenartigkeit möglicher Bewertungskriterien kann für diese Problematik leider keine einfache Lösung angeboten werden. Ein möglicher Ansatz ist die sukzessive Optimierung des jeweiligen Problems mit verschiedener Gewichtung der einzelnen Kriterien und anschließender manueller Bewertung der gefundenen Lösungen durch den Fachmann. Ein anderer Ansatz ist die Verwendung eines Verfahrens der *mehrkriteriellen* Optimierung, auf die im Rahmen dieser Arbeit jedoch nicht eingegangen werden kann.

Da, wie bereits begründet, alle Planungs- und Steuerungsprobleme prinzipiell durch Optimierung gelöst werden können, sollen im folgenden nur einige konkrete Anwendungsbeispiele aufgezählt werden, um den großen Einsatzbereich der Optimierung im Unternehmen zu verdeutlichen. Problemstellungen, die von Projekt V im Rahmen des Innovationskollegs exemplarisch mittels Optimierung gelöst wurden, werden dabei mit (*) gekennzeichnet.

Fabrikplanung:

- Planung der hierarchischen und räumlichen Struktur des Produktionssystems (*)
- Bestimmung der notwendigen Anzahl Fertigungsplatzgruppen/Fertigungsplätze (*)
- Bestimmung der notwendigen Anzahl Werker (*)
- Dimensionierung von Puffern und Lagern (*)

Produktionsplanung:

- Bestimmung geeigneter Fertigungs- und Beschaffungslosgrößen

Fertigungssteuerung:

- Bestimmung der Einschleusreihenfolge und -termine der Fertigungsaufträge (*)
- Bestimmung geeigneter Reaktionen auf Störungen der Fertigung

[1] Man denke nur an eine geeignete Kombination der Kriterien „hohe Auslastung“ und „geringe Durchlaufzeit“.

3 Die hybride Optimierung

3.1 Grundlagen

Leider existiert trotz des in den letzten Jahren zunehmenden Einsatzes hybrider Optimierungsverfahren bis heute weder eine abgeschlossene formale Theorie noch eine allgemein anerkannte Terminologie der hybriden Optimierung. Die im folgenden eingeführten Begriffe und beschriebenen Konzepte stellen deshalb nur zum Teil bekanntes „Lehrbuchwissen" dar, entstammen ansonsten jedoch der Arbeit des Autors auf diesem Gebiet. Den Ausgangspunkt der Betrachtungen bildet die (nicht zuletzt wegen ihrer Trivialität) allgemein anerkannte Definition des Begriffs „hybride Optimierung".

Definition 3.1 (hybride Optimierung)

Die *hybride Optimierung* beschäftigt sich mit der Lösung von Optimierungsproblemen durch Anwendung hybrider Optimierungsverfahren.

Ungleich schwerer fällt eine exakte Definition des Begriffs „hybrides Optimierungsverfahren", die erst eine scharfe Abgrenzung von hybriden und nicht-hybriden Verfahren ermöglichen würde. Grundlage einer geeigneten Definition bildet das Verständnis des Begriffs „hybrid". Dieser Begriff (abgeleitet vom lateinischen *hybrida*) wird in den verschiedensten Fachgebieten verwendet, u.a. in der Biologie, Chemie, Systemtheorie, Linguistik und Informatik, wobei sich die bezeichneten Objekte bzw. Prozesse inhaltlich zumeist stark unterscheiden. Die fundamentale Gemeinsamkeit besteht jedoch darin, daß stets eine Kombination *verschiedenartiger* Elemente bzw. Konzepte vorliegt.

Definition 3.2 (hybrides Optimierungsverfahren)

Ein Optimierungsverfahren wird als *hybrid* bezeichnet, wenn es aus unterschiedlichen Optimierungsverfahren zusammengesetzt ist oder charakteristische Konzepte unterschiedlicher Optimierungsverfahren beinhaltet[1].

Worin liegen die Vorteile hybrider gegenüber nicht-hybriden Optimierungsverfahren ?

Ein Optimierungsverfahren kann im wesentlichen durch die drei Eigenschaften

- *Lösungsqualität*
  (Maß für die Güte der ermittelten Optimierungsergebnisse)
- *Effizienz*
  (Maß für den Aufwand zur Ermittlung von Lösungen der geforderten Qualität)
- *Anwendungsbereich*
  (Maß für die Menge der in geforderter Qualität lösbaren Optimierungsprobleme)

charakterisiert werden. Aus Sicht des Praktikers im Unternehmen sollte ein Optimierungsverfahren sowohl eine hohe Lösungsqualität und Effizienz als auch einen großen Anwendungsbereich besitzen, der zumindest alle im Unternehmen anfallenden Optimierungsprobleme

[1] Im Rahmen dieser Arbeit kann leider nicht näher darauf eingegangen werden, wann zwei Optimierungsverfahren als „unterschiedlich" anzusehen sind. Kurz gesagt sind zwei Verfahren unterschiedlich, wenn sie verschiedenen Kategorien eines Klassifikationsschemas zugeordnet werden können.

umfaßt. Nicht-hybride Optimierungsverfahren zeigen jedoch i.allg. Schwächen bei mindestens einer der genannten Eigenschaften. So besitzen Genetische Algorithmen eine hohe Lösungsqualität und einen großen Anwendungsbereich, aber nur eine geringe Effizienz. Gradientenverfahren hingegen besitzen eine hohe Lösungsqualität und Effizienz, sind aber nur für konvexe Optimierungsprobleme erfolgversprechend einsetzbar (→ kleiner Anwendungsbereich).

Die Grundidee der hybriden Optimierung besteht nun in der Kombination unterschiedlicher Optimierungsverfahren, die sowohl Stärken als auch Schwächen besitzen. Ziel ist es, diese Schwächen zu mildern oder gar zu eliminieren, ohne die Stärken zu beeinträchtigen. Je nachdem, welche Schwäche es primär zu beheben gilt, wird vorrangig eine

- Verbesserung der Lösungsqualität
- Steigerung der Effizienz
- Erweiterung des Anwendungsbereichs

angestrebt, wobei die Verbesserung einer Eigenschaft möglichst nicht durch eine (merkliche) Verschlechterung der anderen beiden Eigenschaften erkauft werden soll. Exemplarisch sei an dieser Stelle nur auf die Steigerung der Effizienz kurz eingegangen:

> Die besondere Stärke Genetischer Algorithmen besteht in ihrer Fähigkeit, komplexe Suchräume effizient nach erfolgversprechenden Teilbereichen zu durchmustern, während die abschließende Konvergenz zum Optimum eher langsam erfolgt. Hier liegt die Kombinierung mit einem effizienten lokalen Suchverfahren nahe, das für jeden der identifizierten „guten“ Teilbereiche in kurzer Zeit dessen lokales Optimum ermittelt.

3.2 Taxonomie hybrider Optimierungsverfahren

Da in der Fachliteratur nach meinem Kenntnisstand bisher kein umfassendes Klassifikationsschema für hybride Optimierungsverfahren beschrieben wurde, wird in diesem Kapitel ein eigener Ansatz vorgestellt. Unterschieden wird zwischen den drei Grundtypen „assimiliert-hybrid“, „eingebettet-hybrid“ und „separat-hybrid“, die in der Praxis sowohl in Reinform als auch gemischt auftreten können (siehe Bild 3.1).

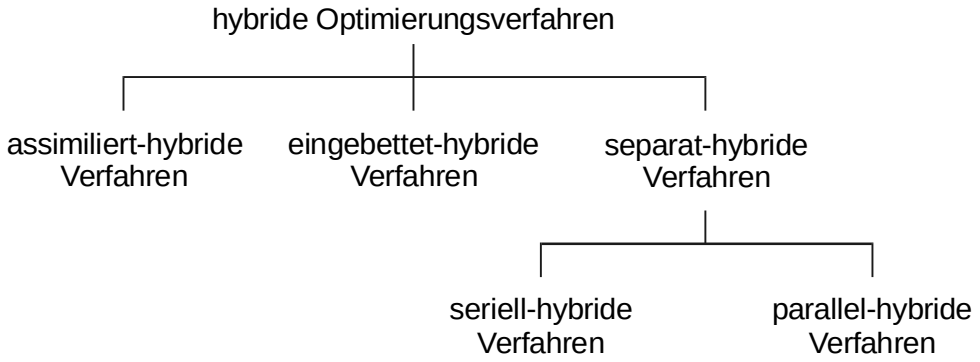

Bild 3.1 Klassifizierung hybrider Optimierungsverfahren

Assimiliert-hybride Verfahren

Assimiliert-hybride Verfahren bestehen aus einem Basis-Optimierungsverfahren, in das charakteristische Konzepte anderer Optimierungsverfahren integriert sind. Anders ausgedrückt: das Basisverfahren „assimiliert" Ideen anderer Optimierungsverfahren, indem es diese in den eigenen Algorithmus einbaut (als echte Erweiterung oder als Ersatz existierender Bestandteile).

Ein Beispiel für ein solches Verfahren ist das *Parallel Recombinative Simulated Annealing* (siehe /MAHF-93/). Bei diesem Verfahren wird ein Genetischer Algorithmus um das Konzept des „Boltzmann-Versuchs" des Simulated Annealing erweitert.

Eingebettet-hybride Verfahren

Eingebettet-hybride Verfahren bestehen aus einem Basis-Optimierungsverfahren, in das andere Optimierungsverfahren („Hilfsverfahren") direkt oder über entsprechende Schnittstellen „eingebettet" sind. Die Hilfsverfahren werden vom Basisverfahren aufgerufen, um während der Optimierung bestimmte, meist wiederkehrende Teilaufgaben zu lösen.

Ein Beispiel für ein solches Verfahren ist das *Simulated Annealing with Local Optimization* (siehe /DESA-96/). Bei diesem Verfahren wird als Basisverfahren Simulated Annealing und als Hilfsverfahren ein effizienter lokaler Optimierer verwendet. Jeweils nach Erzeugung eines neuen Parametervektors durch das Basisverfahren wird der lokale Optimierer gestartet, der von diesem Vektor ausgehend das nächste lokale Optimum ermittelt. Der Zielfunktionswert des gefundenen lokalen Optimums ersetzt dann den Zielfunktionswert des Ursprungsvektors.

Separat-hybride Verfahren

Separat-hybride Verfahren bestehen aus einer Reihe gleichberechtigter, „separater" Optimierungsverfahren, die ausschließlich über ihre Standardschnittstellen miteinander gekoppelt sind. Je nach Datenfluß zwischen den beteiligten Verfahren kann eine weitere Unterteilung in seriell-hybride und parallel-hybride Verfahren vorgenommen werden.

Seriell-hybrid bedeutet, daß die Ergebnisse eines Verfahrens als Eingangsinformationen eines anderen Verfahrens verwendet werden. Die Reihenfolge der Verfahren ist entweder fest vorgegeben oder wird durch ein explizites Steuermodul dynamisch festgelegt.

Ein Beispiel für ein solches Verfahren ist die *Combined 2-Phase Optimization* (siehe /SYRJ-95/). Bei diesem Verfahren durchläuft die Optimierung zwei Stufen: in der ersten Stufe erfolgt mittels eines globalen Verfahrens eine überblicksmäßige Erforschung des gesamten Suchraums, in der anschließenden zweiten Stufe mittels eines lokalen Verfahrens die Lokalisierung des Optimums im erfolgversprechendsten Bereich des Suchraums.

Parallel-hybrid bedeutet, daß die beteiligten Verfahren unabhängig voneinander ausgeführt werden und erst abschließend eine Auswahl bzw. Kombinierung der Ergebnisse erfolgt. Die einzelnen Verfahren können deshalb sowohl seriell in beliebiger Reihenfolge als auch parallel ausgeführt werden.

Ein Konzept für ein solches Verfahren wird in /COTT-95/ beschrieben. Dabei werden ein Genetischer Algorithmus und ein Branch-and-Bound-Verfahren parallel ausgeführt, um kombinatorische Optimierungsprobleme zu lösen. Der Genetische Algorithmus liefert als

Ergebnis eine obere Schranke und das Branch-and-Bound-Verfahren eine untere Schranke des globalen Optimums, die sich mit zunehmender Optimierungsdauer annähern.

3.3 Parametrierung hybrider Optimierungsverfahren

Der Anwender im Unternehmen stellt meist folgende Anforderungen an ein rechnergestütztes Optimierungswerkzeug:

- Das Werkzeug muß möglichst alle praktisch anfallenden Optimierungsprobleme in der geforderten Qualität lösen können.
- Das Werkzeug muß unter möglichst guter Ausnutzung der zur Verfügung stehenden Zeit und Ressourcen effizient arbeiten.
- Das Werkzeug muß einfach zu bedienen sein.

Während die ersten beiden Anforderungen durch hybride Optimierungsverfahren i.allg. mindestens so gut erfüllt werden wie durch nicht-hybride Verfahren (siehe Kapitel 3.1), stellt die dritte Anforderung für hybride Verfahren ein nichttriviales Problem dar. Die Ursache dafür erschließt sich bei näherer Betrachtung des Begriffs der „einfachen Bedienbarkeit“. Neben allgemeinen Aspekten (vor allem: Gestaltung der Mensch-Maschine-Schnittstelle) ist darunter zu verstehen, daß der Nutzer keine Spezialkenntnisse zur Optimierung benötigt, um mit Hilfe des Werkzeugs Optimierungsprobleme effizient und in annehmbarer Qualität zu lösen. Das heißt insbesondere, daß beim Nutzer keine oder nur rudimentäre Kenntnisse zu den im hybriden Verfahren kombinierten Optimierungsverfahren und i.allg. keine Kenntnisse zur geeigneten Parametrierung dieser Verfahren (Festlegung von Schrittweiten, Abbruchbedingungen, etc.) vorausgesetzt werden dürfen.

Demgegenüber steht die Erkenntnis, daß ein hybrides Optimierungsverfahren über eine Vielzahl einzustellender Verfahrensparameter verfügt, die sich aus den Verfahrensparametern der darin enthaltenen Verfahren sowie zusätzlichen Verfahrensparametern, die u.a. die Art der Verfahrenskopplung (siehe Taxonomie) und ggf. die Reihenfolge der Verfahrensanwendungen spezifizieren, zusammensetzt. Kurz gesagt, die Parametrierung eines hybriden Verfahrens ist *per se* aufwendiger als die eines nicht-hybriden Verfahrens. Somit stellt sich die Frage, wer die Parametrierung des hybriden Optimierungsverfahrens übernimmt, wenn der Nutzer damit nicht betraut werden soll (s.o.). Einerseits kann mit dieser Aufgabe der Entwickler des Optimierungsverfahrens betraut werden. Das führt dazu, daß bereits beim Entwurf die beteiligten Verfahren, deren Kopplungsart und sämtliche Verfahrensparameter festgelegt werden müssen. Da zu diesem Zeitpunkt i.allg. sehr wenig über die beim späteren Einsatz zu lösenden Optimierungsprobleme bekannt ist, muß mit mehr oder weniger starken Einbußen bei Lösungsqualität und Effizienz gerechnet werden. Andererseits kann mit dieser Aufgabe das Optimierungsverfahren selbst betraut werden, indem ihm „intelligente“ Methoden zur Verfügung gestellt werden, seine Parametrierung selbsttätig vorzunehmen. Der Entwicklungsaufwand ist bei dieser Variante zwar erheblich größer, dafür kann sich das Verfahren sehr gut an das konkret zu lösenden Problem anpassen und somit eine erheblich größere Lösungsqualität und Effizienz erreichen als bei der ersten Variante.

Im folgenden Kapitel wird das Konzept eines *adaptiv-hybriden Optimierungsverfahrens* auf Multiagentenbasis vorgestellt, das sich selbsttätig parametriert und somit die o.g. Forderung nach „einfacher Bedienbarkeit“ bei hoher Lösungsqualität und Effizienz erfüllt.

4 DynamO - ein hybrides Optimierungssystem auf Multiagentenbasis

4.1 Das Konzept eines adaptiv-hybriden Optimierungsverfahrens

Wie bereits im Kapitel 3.3 erläutert, besteht die wesentliche Schwierigkeit beim Einsatz hybrider Optimierungsverfahren in deren Parametrierung. Da die geeignete Parametrierung nicht auf den Anwender im Unternehmen „abgewälzt" werden kann, der meist nur über sehr begrenzte Kenntnisse zur Optimierung verfügt, wurde im Rahmen des Innovationskollegs nach Ansätzen gesucht, bei denen das hybride Verfahren selbst (mehr oder weniger vollständig) die Parametrierung übernimmt. Im Ergebnis dieser Untersuchungen wurde ein Konzept für ein *adaptiv-hybrides Optimierungsverfahren* („AHO-Verfahren") entwickelt, das im folgenden vorgestellt wird.

Wesentliches Merkmal des AHO-Verfahrens ist die explizite *Planung* des Optimierungsprozesses. Planung bedeutet im allgemeinen Sinn die Generierung eines *Plans*, d.h. einer Folge von *Aktionen*, die unter Einhaltung bestimmter *Nebenbedingungen* einen gegebenen *Startzustand* in einen gewünschten *Endzustand* überführen. Bei Anwendung des Planungsbegriffs auf das Gebiet der Optimierung ergeben sich folgende Zuordnungen:

Zustand	↔	bisher beste Lösung des Optimierungsproblems
Startzustand	↔	vom Anwender vorgegebene Startlösung (bzw. „keine Startlösung")
Endzustand	↔	optimale[1] Lösung des Optimierungsproblems
Aktion	↔	Anwendung eines konkreten Optimierungsverfahrens mit einer konkreten Belegung der Verfahrensparameter auf das Optimierungsproblem
Nebenbed.	↔	begrenzte verfügbare Ressourcen (Rechnerkapazität, Zeit u.a.)
Plan	↔	Folge sukzessiver Anwendungen von Optimierungsverfahren auf das Optimierungsproblem, die zur Ermittlung der optimalen Lösung führen

Somit kann das Problem der Parametrierung eines hybriden Optimierungsverfahren als Planungsproblem angesehen werden. Da jedoch die Wirkung einer Aktion, d.h. das Ergebnis einer Verfahrensanwendung, in den allermeisten Fällen nicht exakt vorhersehbar ist (z.B. wegen Stochastik im Verfahrensalgorithmus und/oder im zu lösenden Problem), kann die Planung nicht vollständig und endgültig *vor* Beginn der eigentlichen Optimierung durchgeführt werden. Hier bietet sich ein Ansatz an, der in der Fachliteratur als „Interleaving Planning and Execution" (etwa: „Überlappen von Planung und Ausführung", siehe u.a. /AMBR-88/) bezeichnet wird. Dabei wird ein Plan vor Beginn der Ausführung nur soweit spezifiziert, wie es mit den vorhandenen Informationen möglich ist und dann während der Ausführung auf Basis der dabei gewonnenen Informationen verfeinert bzw. „repariert". Dieser Ansatz wurde im AHO-Verfahren realisiert.

[1] Wie bereits in Kapitel 2.1 angedeutet, wird in der Praxis der Begriff der „optimalen Lösung" meist durch „gute Lösung" ersetzt, wobei die Entscheidung, ob eine Lösung „gut" ist, meist subjektiv getroffen wird.

Bild 4.1 zeigt den allgemeinen Aufbau des AHO-Verfahrens. Vor Beginn der Optimierung stellt der Anwender eine Beschreibung des zu lösenden Optimierungsproblems sowie ggf. eine Menge von Startlösungen bereit und spezifiziert die für die Optimierung maximal aufzuwendende Zeit.

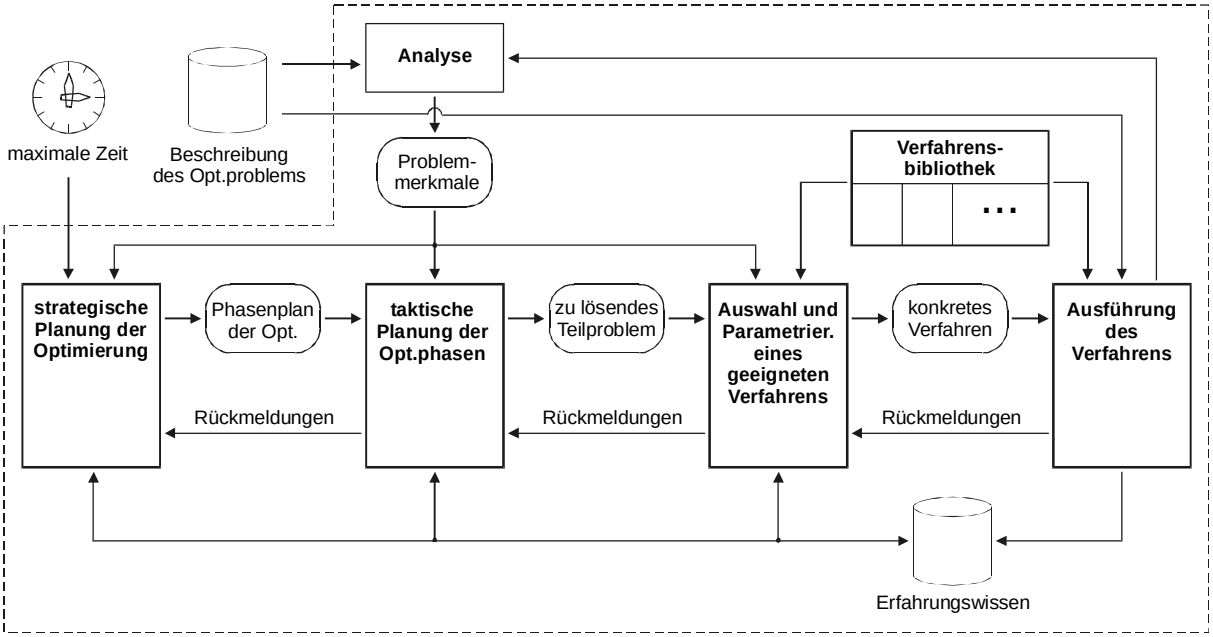

Bild 4.1 Aufbau des AHO-Verfahrens

Das AHO-Verfahren leitet durch entsprechende Analysen die wesentlichen explizit bestimmbaren Problemmerkmale ab. Dazu gehören u.a. Anzahl und Definitionsbereiche der Parameter sowie Aufbau von Zielfunktion und Nebenbedingungen (linear, quadratisch, usw.). Diese Merkmale werden zusammen mit einer Schätzung der Maximalzahl untersuchbarer Parametervektoren bei der *strategischen Planung* der Optimierung verwendet, um einen groben Plan des Optimierungsprozesses aufzustellen. Der Plan unterteilt den Optimierungsprozeß in einzelne Phasen (z.B. Erforschungsphase, Konvergenzphase) und legt die dafür maximal aufzuwendenden Zeiten fest. Anschließend werden die einzelnen Phasen abgearbeitet. Die in der jeweils aktuellen Phase zu lösenden Teilprobleme und die zur Lösung maximal verfügbaren Zeiten werden durch die *taktische Planung* sukzessiv festgelegt. Für jedes zu lösende Teilproblem erfolgt die *Auswahl* eines geeigneten Verfahrens der Verfahrensbibliothek (meist aus mehreren potentiellen Kandidaten) sowie deren *Parametrierung*[1]. Beide Vorgänge werden wissensbasiert gesteuert, wobei insbesondere vom Entwickler vorgegebene Heuristiken sowie Erfahrungswissen aus früheren Optimierungen ähnlicher Probleme bzw. dem bisherigen Verlauf der aktuellen Optimierung genutzt werden. Anschließend erfolgt die Ausführung des gewählten Verfahrens, die entweder zur Lösung des Teilproblems führt oder mit einem Mißerfolg endet. In letzterem Fall wird zur Auswahl zurückgekehrt, die versucht, ein alternatives Verfahren zu finden. Gelingt dies nicht, muß zur taktischen oder sogar zur strategischen Planung zurückgekehrt werden, um auf den Mißerfolg durch Umplanung zu reagieren.

[1] Die Aufteilung der Planung auf Hierarchieebenen verschiedenen Abstraktionsniveaus, die durch
strategische Planung → taktische Planung → Auswahl
realisiert wird, entspricht dem Ansatz des „Hierarchical Task Network Planning“ (siehe u.a. /EROL-95/).

Das Konzept des AHO-Verfahrens spezifiziert die Struktur des verwendeten Planungsalgorithmus, nicht jedoch die bei der Planung zu verwendenden Regeln (z.B. zur Aufteilung in Phasen) und die in der Verfahrensbibliothek enthaltenen Optimierungsverfahren. Dadurch bleiben dem Entwickler bei der Überführung in konkrete Softwaretools für bestimmte Problemdomänen ausreichend Freiheitsgrade, um eine möglichst gute Anpassung an diese Domänen zu erreichen.

Nicht näher spezifiziert wurde bisher auch die Art der Auswahl geeigneter Verfahren aus der Verfahrensbibliothek sowie die Art der Konfliktlösung, falls mehrere Verfahren potentiell in der Lage sind, ein gegebenes Teilproblem zu lösen. Im nächsten Kapitel wird deshalb ein Ansatz vorgestellt, der diese Problematik auf der Grundlage eines Multiagentensystems behandelt.

4.2 Multiagentensysteme

In den letzten beiden Jahrzehnten haben sich Multiagentensysteme, die dem Forschungsgebiet der Verteilten Künstlichen Intelligenz entstammen, von einem theoretischen Forschungsgegenstand zu einer praktisch einsetzbaren Technologie entwickelt (für eine Einführung siehe z.B. /MÜLL-93/). Leider existiert eine Vielzahl von Definitionen der Begriffe „Agent“ und „Multiagentensystem“, die für den Nicht-Fachmann meist nur schwer zu verstehen und unterscheiden sind. Die in dieser Arbeit verwendete Sichtweise auf diese beiden Begriffe ist absichtlich vereinfachend, beinhaltet aber m.E. die wesentlichen Aspekte, die zum Verständnis des Multiagenten-Einsatzes im Rahmen des AHO-Verfahrens notwendig sind.

Ein *Multiagentensystem* besteht aus einer Ansammlung interagierender autonomer Hardware- und/oder Software-Systeme, die *Agenten* genannt werden. Jeder Agent besitzt spezielle Fähigkeiten, verfügt über eigenes Wissen und verfolgt ggf. eigene Ziele. Die Besonderheit von Multiagentensystemen besteht darin, daß in ihnen die Problemlösung nicht zentral gesteuert wird, sondern sich dezentral durch die Prinzipien von Kooperation und Konkurrenz ergibt.

Eine häufig auftretende Aufgabe in Multiagentensystemen ist das Finden desjenigen Agenten, der am besten für die Lösung eines gegebenen Problems geeignet ist. Diese Aufgabe wird in der Fachliteratur als „Task Allocation Problem“ bezeichnet. Ihre Lösung ist trivial, wenn genau ein Agent in der Lage ist, das Problem zu lösen, kann jedoch äußerst kompliziert sein, wenn mehrere Agenten dazu in der Lage sind. Ein weitverbreitetes Verfahren zur Lösung dieser Aufgabe ist das sogenannte „Contract Net Protocol“ (siehe /SMIT-80/). Es verwendet einen wechselseitigen Verhandlungsprozeß zwischen einem *Manager* (dem Agenten, der die Lösung eines Problems benötigt) und einer Menge potentielle *Vertragsnehmer* (die Agenten, die das Problem lösen können). Der Prozeß unterteilt sich im wesentlichen in folgende vier Abschnitte:

(1) Der Manager sendet eine Ausschreibung des Problems an alle Agenten, von denen er annimmt, daß sie das Problem lösen können (im Zweifelsfall an alle Agenten).

(2) Jeder dieser Agenten prüft, basierend auf seinem lokalen Wissen, ob und, wenn ja, wie gut er das Problem lösen könnte. Wenn er sich für geeignet hält, sendet er dem Manager eine Bewerbung.

(3) Der Manager wertet die eingegangenen Bewerbungen aus, wählt die beste davon aus[1] und schließt einen Vertrag über die Lösung des Problems mit dem entsprechenden Agenten ab (der dadurch zum Vertragsnehmer wird).

(4) Der Vertragsnehmer löst das Problem entweder selbst oder zerlegt es in Teilprobleme, die er seinerseits zur Lösung ausschreibt und von anderen Agenten lösen läßt, und sendet zum Schluß dem Manager die Lösung des Problems.

Der beschriebene Ansatz kann verwendet werden, um die Verfahrensauswahl im AHO-Verfahren zu realisieren. Dazu wird jedes Optimierungsverfahren der Verfahrensbibliothek als Agent modelliert. Ein solcher Agent enthält neben dem eigentlichen Verfahrensalgorithmus (seiner „Fähigkeit“) auch Informationen über seine Eignung für die Lösung bestimmter Probleme und die dann jeweils günstigste Belegung seiner Verfahrensparameter (sein „Wissen“). Diese Informationen sind initial vom Entwickler vorzugeben, können danach jedoch entsprechend der gesammelten Erfahrung automatisch modifiziert werden. Die Gesamtheit der Verfahrensagenten bildet ein Multiagentensystem, das in der Lage ist, für ein von der taktischen Planung festgelegtes Teilproblem selbständig das am besten geeignete Verfahren zu ermitteln und auszuführen.

4.3 Das Optimierungssystem DynamO

DynamO ist ein automatisches Optimierungssystem auf der Basis des in den vergangenen Kapiteln vorgestellten adaptiv-hybriden Optimierungsverfahrens (siehe auch /HADE-00/). Das Anwendungsgebiet dieses Systems sind Aufgaben der statischen Parameteroptimierung mit einer Zielfunktion[2], wobei insbesondere Aufgaben gelöst werden können, deren Zielfunktion und/oder Nebenbedingungen durch zeitintensive externe Programme, wie z.B. Simulatoren, gegeben sind. Das Optimierungssystem wurde so gestaltet, daß es weitgehend ohne Spezialkenntnisse zur Optimierung bedient werden kann und verfügt über eine Reihe von Schnittstellen, die eine Kopplung mit anderen Softwarewerkzeugen ermöglichen (siehe Anhang B). Aus diesen Gründen sowie wegen des geringen Ressourcenbedarfs ist es insbesondere für den praxisorientierten Einsatz im Unternehmen, speziell in KMU, geeignet.

Die von DynamO zu lösenden Optimierungsprobleme müssen vom Anwender unter Nutzung einer formalen Beschreibungssprache spezifiziert werden (zur Syntax und Semantik dieser Sprache siehe Anhang A). Existieren externe Programme, die die benötigte Problembeschreibung automatisch generieren, können diese über eine spezielle Schnittstelle von DynamO genutzt werden (näheres hierzu in Anhang B.4). So wurde z.B. für ein Fabrikplanungsproblem, das im Rahmen des Innovationskollegs zu bearbeiten war, ein Programm entwickelt, das die entsprechende Problembeschreibung auf der Basis von Informationen generiert, die es aus einer SQL-Datenbank liest.

---

[1] Als Auswahlkriterien können dem Manager u.a. Informationen der Bewerber zum zu erwartenden Lösungsaufwand oder zur erreichbaren Lösungsqualität dienen.

[2] Aufgaben mit mehreren Zielfunktionen sind somit nicht direkt lösbar. Ein möglicher Ausweg ist die Formulierung einer zusammengesetzten Zielfunktion als gewichtete Summe der einzelnen Zielfunktionen. Insbesondere bei stark gegenläufigen Zielfunktionen sollten zur Lösung jedoch besser Verfahren der mehrkriteriellen Optimierung verwendet werden.

Die Benutzerschnittstelle von DynamO ist fensterorientiert und entspricht dem von Windows 95/98/NT bekannten Look-and-Feel-Standard (siehe Bild 4.2). Sie beinhaltet u.a. einen Editor für die Problembeschreibung sowie Tools zur Visualisierung des Optimierungsprozesses.

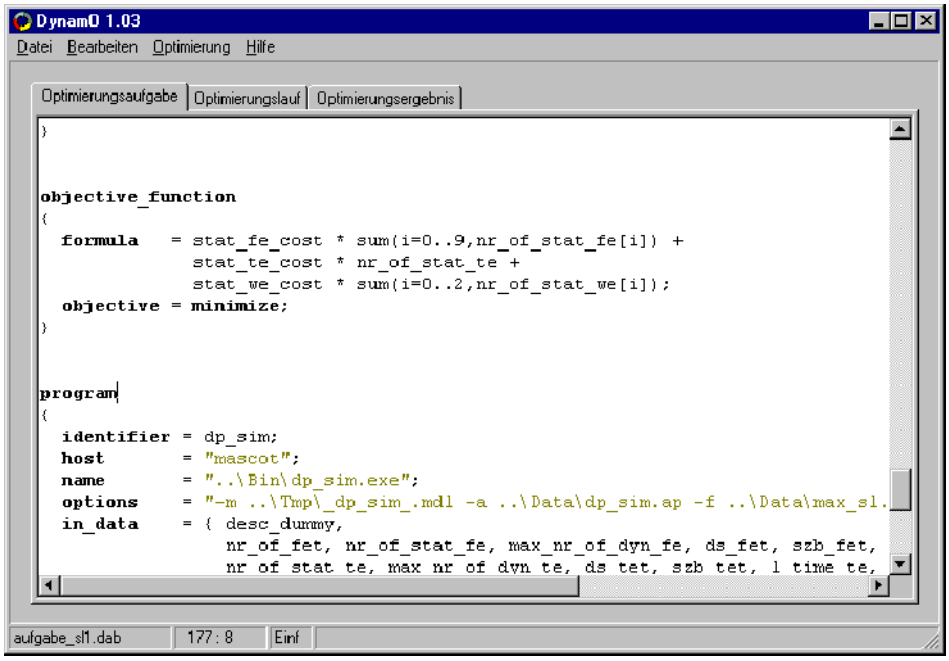

Bild 4.2 Hauptfenster des Optimierungssystems DynamO

Nachdem umfangreiche Untersuchungen darüber durchgeführt wurden, welche heute gebräuchlichen Optimierungsverfahren für eine hybride Verwendung in DynamO besonders geeignet sind, wurden die folgenden Optimierungsverfahren ausgewählt:

- Vollständige Enumeration (EN)
- Globale Zufallssuche (GZS)
- Lokale Zufallssuche (LZS)
- Methode des steilsten Abstiegs (MSA)
- Methode von Hooke & Jeeves (HJM)
- Simplexmethode von Nelder & Mead (NMM)
- Genetischer Algorithmus (GA)
- Simulated Annealing (SA)

Auf die genaue Funktionsweise der einzelnen Verfahren kann an dieser Stelle nicht näher eingegangen werden. Hier sollte der interessierte Leser auf die gebräuchlichen Lehrbücher zur Nichtlinearen Optimierung zurückgreifen oder das große Angebot von Fachartikeln im Internet nutzen. Um zumindest einen groben Überblick zu geben, enthält die folgende Tabelle in aller Kürze wesentliche Eigenschaften der aufgezählten Verfahren:

	Globales Verfahren ?	Benötigt Startlösung ?	Populations-basiert ?	Besondere Vorteile ?	Besondere Nachteile ?
EN	ja	nein	nein	Lösungsgarantie, wenn max. erlaubte Versuchs-zahl ≥ Mächtigkeit des Suchraums	nur für kombinatorische Probleme; geringe Effizienz
GZS	ja	nein	nein	gleichmäßige Unter-suchung des gesamten Suchraums	geringe Effizienz
LZS	nein	ja	nein	großer Anwendungs-bereich	langsame Konvergenz zum lokalen Optimum
MSA	nein	ja	nein	rasche Konvergenz zum lokalen Optimum bei konvexen Problemen	wenn Gradiente unbe-kannt, Gradienten-approximation notwendig
HJM	nein	ja	nein	rasche Konvergenz zum lokalen Optimum	Probleme bei Abhängig-keiten zwischen Optimie-rungsparametern
NMM	nein	nein	ja	automatische Anpassung an zu lösendes Problem	Probleme, wenn Mächtig-keit des Lösungsraums << Mächtigkeit des Such-raums
GA	ja	nein	ja	universelle Einsetzbarkeit bei großer Lösungsqualität	geringe Effizienz
SA	ja	ja	nein	großer Anwendungs-bereich	geringe Effizienz

Jedes der genannten Optimierungsverfahren bildet zusammen mit Heuristiken zur Bestimmung seiner Eignung für konkrete Probleme und zur geeigneten Festlegung seiner Verfahrens-parameter einen Softwareagenten, der in der Verfahrensbibliothek gespeichert ist. Der Austausch bzw. die Ergänzung von Verfahren ist dem Anwender nicht möglich, kann vom Entwickler jedoch problemlos vorgenommen werden.

Vor dem Start einer Optimierung kann der Anwender festlegen („markieren“), welche der Optimierungsverfahren in den internen Planungsprozeß einbezogen werden sollen und somit prinzipiell zum Einsatz kommen können. Besitzt der Anwender kein ausreichendes Wissen zur Optimierung im allgemeinen bzw. zum aktuell zu lösenden Problem, sollten alle vorhandenen Optimierungsverfahren markiert werden. Ist hingegen entsprechendes Wissen vorhanden, können durchaus auch nur bestimmte oder gar nur ein einziges Verfahren markiert werden, wodurch sich ggf. die Planung vereinfacht und die Effizienz der Optimierung erhöht[1].

Die zweite Möglichkeit der Einflußnahme des Anwenders auf die Optimierung ist die Angabe der maximalen Anzahl von Parametervektoren, die im Verlauf der Optimierung untersucht werden dürfen (d.h. der maximalen Anzahl „Versuche“). Der konkrete Wert richtet sich nach der Komplexität des zu lösenden Problems, der zur Verfügung stehenden Zeit und der ge-

[1] Wird nur ein Verfahren markiert, erübrigt sich die Planung vollständig; in diesem Fall wird statt einer Kombination von Optimierungsverfahren stets nur das markierte zur Problemlösung verwendet.

wünschten Lösungsqualität. Eine einfache Vorgehensweise zur Ermittlung dieses Wertes ist z.B. diese:

(1) Festlegung der für die gesamte Optimierungsstudie maximal aufzuwendenden Zeit

(2) Messung oder Schätzung der Zeit, die zur Bewertung eines Parametervektors notwendig ist (d.h. der Laufzeit aller externen Programme, die bei der Berechnung der Zielfunktion und dem Test aller Nebenbedingungen ausgeführt werden[1])

(3) Berechnung der Anzahl Parametervektoren, die in der in (1) festgelegten Zeit im Mittel bewertet werden können

(4) Durchführung einer ersten Optimierung mit einem Zehntel der in (3) berechneten Anzahl, ggf. unter Angabe bereits bekannter Startlösungen

(5) Test des Optimierungsergebnisses auf Plausibilität und Qualität; falls die Qualität bereits als ausreichend angesehen wird, Ende der Optimierungsstudie

(6) Durchführung einer zweiten Optimierung mit der verbleibenden Anzahl Versuche unter Angabe des Ergebnisses aus (4) als Startlösung[2]

(7) Ende der Optimierungsstudie

DynamO wurde vollständig in C/C++ entwickelt (Borland C++ Builder 4.0), wobei u.a. die Softwarewerkzeuge YACC und LEX zum Einsatz kamen. Das Optimierungssystem umfaßt etwa 40.000 Zeilen selbstentwickelten Quelltext. Nähere Informationen zu DynamO (Bedienung, Schnittstellen, Installation, etc.) können dem Anhang dieser Arbeit entnommen werden.

[1] Wenn bei der Bewertung eines Parametervektors keine externen Programme zum Einsatz, kann man je nach Rechenleistung etwa 10...100 msec pro Bewertung annehmen.

[2] Die Optimierung kann jederzeit manuell abgebrochen werden, z.B. wenn man feststellt, daß über einen längeren Zeitraum (z.B. 20% der Gesamtlaufzeit) keine neue bessere Lösung gefunden wurde.

5 Ein Anwendungsbeispiel

5.1 Problemstellung

Zur Illustration der Leistungsfähigkeit des Optimierungssystems DynamO soll eine Problemstellung aus dem Bereich der Fabrikplanung verwendet werden. Sie wurde der Dissertation „Simulationsbasierte Dimensionierung von Produktionssystemen mit definiertem Potential an Leistungsflexibilität“ von A. Kobylka entnommen, wo sie (als Teilproblem) durch die Planungsmethode DYNAMIS-P bearbeitet wurde (siehe /KOBY-00/). Wegen des Umfangs dieses Problems sei für eine vollständige Beschreibung auf o.g. Arbeit verwiesen.

Betrachtet wird ein Fertigungssystem mit zehn Fertigungsplatzgruppen, einem Transportsystem und drei Werkertypen (siehe Bild 5.1). Jede Fertigungsplatzgruppe besteht aus einer nicht-leeren Menge identischer Fertigungsplätze („Maschinen“) und besitzt einen eigenen Eingangs- und Ausgangspuffer unbegrenzter Kapazität. Das Transportsystem besteht aus einer nicht-leeren Menge identischer, unabhängig agierender Transporter. Jeder Werkertyp besteht aus einer nicht-leeren Menge identisch-qualifizierter Werker[1]. Innerhalb des Fertigungssystems werden keine stochastischen Einflüsse berücksichtigt.

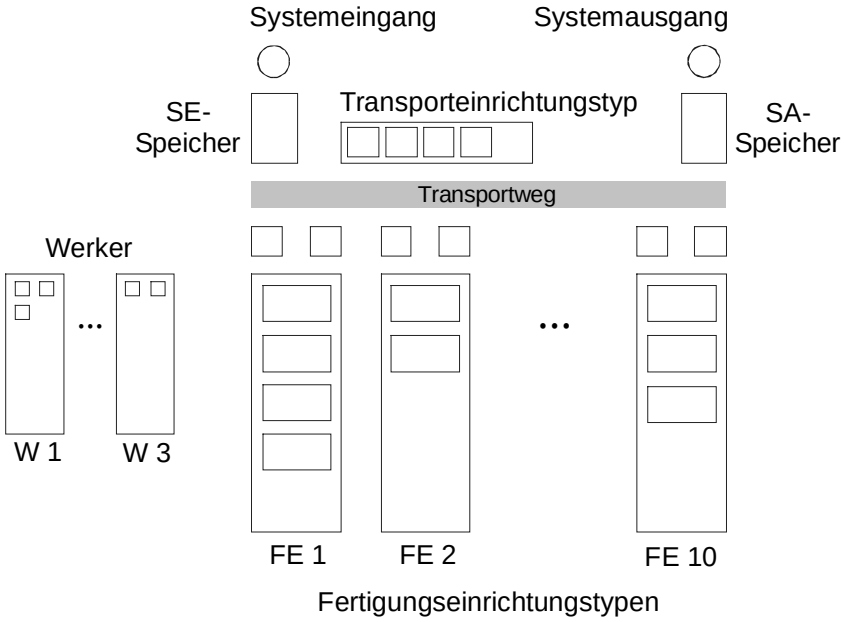

Bild 5.1 Allgemeine Struktur des Fertigungssystems

Das Produktionsprogramm des Fertigungssystems für ein Jahr besteht aus 1900 Fertigungsaufträgen, deren Einschleuszeitpunkte und spätesten Ausschleuszeitpunkte vorgegeben sind. Per Definition sei ein Fertigungsauftrag gleich einem Fertigungslos gleich einem Transportlos („Unteilbarkeit“ eines Auftrags).

[1] „Identisch-qualifiziert“ bedeutet, daß alle Werker einer Gruppe dieselben Maschinen bedienen können.

Optimierungsparameter

Die Parameter des Optimierungsproblems sind die Zahl von Fertigungsplätzen der einzelnen Fertigungsplatzgruppen, die Zahl der Transporter und die Zahl von Werkern der einzelnen Werkertypen ($\rightarrow$ 14 ganzzahlige Parameter). Die Zahl von Fertigungsplätzen kann dabei zwischen 1 und 30, die Zahl der Transporter zwischen 1 und 20 sowie die Zahl von Werkern zwischen 1 und 50 variiert werden.

Zielfunktion

Sei $N_{F,i}$ die Zahl von Fertigungsplätzen der i-ten Fertigungsplatzgruppe, N_T die Zahl der Transporter und $N_{W,i}$ die Zahl von Werkern des i-ten Werkertyps. Die zu minimierende Zielfunktion ist dann

$$f(.) = 1.0 * \sum_{i=1}^{10} N_{F,i} + 0.5 * N_T + 0.1 * \sum_{i=1}^{3} N_{W,i} .$$

Nebenbedingungen

Sei $N_{FA,late}$ die Zahl verspätet fertiggestellter Fertigungsaufträge[1]. Sei $T_{FA,late}$ die Summe der Verspätungszeiten der Fertigungsaufträge. Folgende Nebenbedingungen sind einzuhalten:

$N_{FA,late} \leq 95$

(*„höchstens 5% der Fertigungsaufträge dürfen verspätet fertiggestellt werden“*)

$T_{FA,late} \leq 2000\,\text{h}$

(*„die Gesamt-Verspätungszeit der Fertigungsaufträge darf 2000 h nicht übersteigen“*)

Ein Modell des zu optimierenden Fertigungssystems wurde in Form eines Simulators implementiert. Dabei wurde besonderes Augenmerk auf eine hohe Effizienz, d.h. geringe Laufzeit des Simulators gelegt. Ziel war die Durchführung der gesamten Simulation für eine Systemvariante in nicht mehr als 1..2 Sekunden auf „normaler“ Rechnerhardware. Da dieses Ziel mit kommerzieller Simulationssoftware nicht zu erreichen war, wurde der Simulator vom Autor vollständig in C++ implementiert und in Bezug auf die Laufzeit optimiert. Durch dieses Vorgehen konnte die gewünschte Laufzeit erreicht werden, ohne die Qualität der Simulationsergebnisse negativ zu beeinflussen.

5.2 Wesentliche Merkmale des Problems

Bei der beschriebenen Aufgabe handelt es sich um ein *kombinatorisches*, *nichtlineares*[2], *deterministisches*, *restringiertes*, *algorithmisch formuliertes* Optimierungsproblem.

Die Suchraumgröße, d.h. die Anzahl möglicher Systemvarianten, beträgt ca. $1.5*10^{21}$.

---

[1] Ein Fertigungsauftrag sei „verspätet fertiggestellt“, wenn sein realer Ausschleuszeitpunkt nach dem spätest-erlaubten Ausschleuszeitpunkt liegt.

[2] Das Problem ist nichtlinear, da zwar die Zielfunktion linear ist, die Nebenbedingungs-Größen $N_{FA,late}$ und $T_{FA,late}$ jedoch nichtlinear von den Optimierungsparametern abhängen.

Die geschätzte Lösungsraumgröße beträgt ca. $2.1*10^{20}$, d.h. etwa 14 % der Elemente des Suchraums sind gültig[1]. Anders ausgedrückt: etwa 14 % der möglichen Systemvarianten sind gültig (erfüllen alle Nebenbedingungen) und kommen damit als Lösung potentiell in Frage.

Die folgende Tabelle zeigt die geschätzte Zielfunktionswert-Verteilung gültiger Systemvarianten[2]:

Wertebereich	Anteil in %
(-∞ ... 80)	0.000
[80 ... 100)	0.021
[100 ... 120)	0.317
[120 ... 140)	2.903
[140 ... 160)	12.047
[160 ... 180)	25.842
[180 ... 200)	30.762
[200 ... 220)	19.847
[220 ... 240)	6.918
[240 ... 260)	1.237
[260 ... 280)	0.100
[280 ... ∞)	0.005

5.3 Ergebnisse

Unter Verwendung des hybriden Optimierungssystems DynamO wurde eine Optimierungsstudie des beschriebenen Problems durchgeführt, wobei alle enthaltenen Optimierungsverfahren markiert waren und eine maximale Anzahl Versuche von 20.000 festgelegt wurde. Um die Robustheit von DynamO zu testen, wurden Optimierungsläufe mit fünf verschiedenen Startlösungen und drei verschiedenen Zufallszahlengenerator-Startwerten durchgeführt. Es zeigte sich, daß DynamO stets Lösungen mit einem Zielfunktionswert im Bereich 46.9 ... 47.1 fand. In den meisten Fällen wurde folgende Lösung gefunden:

Anzahl Fertigungsplätze N_F: (4, 4, 4, 4, 5, 5, 4, 4, 4, 4)

Anzahl Transporter N_T: 2

Anzahl Werker N_W: (10, 13, 16)

Zielfunktionswert $f(.)$: 46.9

Untersuchungen dieser Lösung ergaben, daß es sich dabei mit sehr hoher Wahrscheinlichkeit um den globalen Minimierer des beschriebenen Problems handelt.

[1] Dieser Wert wurde auf der Basis von 300.000 zufällig generierten Elementen des Suchraums bestimmt.

[2] Diese Verteilung wurde auf der Basis von 300.000 zufällig generierten Elementen des Suchraums bestimmt, unter denen sich 41.951 gültige Systemvarianten befanden.

Anhang A: Die Aufgabenbeschreibungssprache von DynamO

Um DynamO einsetzen zu können, muß der Nutzer eine formale Beschreibung der zu lösenden Optimierungsaufgabe bereitstellen. Die Elemente einer solchen Aufgabenbeschreibung orientieren sich an der gebräuchlichen mathematischen Darstellung von Optimierungsaufgaben (siehe hierzu auch Kapitel 2.1), wie z.B.

$$\underbrace{\text{minimiere}}_{\text{(Opt.art)}}: \quad \underbrace{f(x,y)}_{\text{(Parameter)}} = \underbrace{2*x^2 + 3*y}_{\text{(Zielfunktion)}}$$

$$\text{mit:} \quad \underbrace{g_1: x < y \ , \ g_2: x+y = 10}_{\text{(Nebenbedingungen)}}$$

Neben diesen Elementen können zusätzlich *benannte Konstanten*, *benannte Formeln* und *Schnittstellen zu externen Berechnungsprogrammen* (Simulatoren o.ä.) definiert werden.

A.1 Datenobjekte

Das wesentliche Konstrukt einer Aufgabenbeschreibung ist das *Datenobjekt*. Datenobjekte repräsentieren die für die Optimierung wesentlichen (Zahlen-)Objekte des behandelnden Problems. Ein Datenobjekt wird durch die Merkmale

- eindeutiger Name[1]
- Zahlentyp (ganze Zahlen, reelle Zahlen)
- Struktur (skalares Objekt, Vektor, Matrix)

beschrieben und besteht, entsprechend seiner Struktur, aus einem oder mehreren skalaren *Datenelementen*. Ein Datenelement speichert genau einen (Zahlen-)Wert, der je nach Art des Datenelementes folgendermaßen zugewiesen wird:

- numerische Konstante
  → Wert wird vor Beginn der Optimierung einmalig zugewiesen
- numerische Formel („Formelabhängige“)
  → Wert ist Ergebnis der Berechnung einer zugeordneten Formel
- algorithmische Formel („Programmabhängige“)
  → Wert ist Ergebnis der Ausführung eines zugeordneten externen Programmes
- Optimierungsparameter
  → Wert wird während der Optimierung vom Optimierungsverfahren modifiziert

Der Zugriff auf die Elemente eines Datenobjekts erfolgt durch Indizierung mittels **`[ .. ]`**[2], z.B. **`dobj1`**, **`dobj2[0]`**, **`dobj3[3][1]`**. Es ist zu beachten, daß die Indizierung mit 0 beginnt und nicht mit 1.

---

[1] Datenobjektnamen sind nichtleere Zeichenketten, die nur aus Buchstaben, Ziffern und dem Unterstrich ‘_’ bestehen dürfen und deren erstes Zeichen ein Buchstabe ist. Es existiert keine Längenbegrenzung.

[2] Bei skalaren Datenobjekten entfällt die Indizierung, da Datenobjekt und -element identisch sind.

A.2 Arithmetische Ausdrücke

Wesentliche Teile eines Optimierungsproblems werden mittels *arithmetischer Ausdrücke* beschrieben (z.B. Zielfunktion, Nebenbedingungen). Arithmetische Ausdrücke können folgende Konstrukte beinhalten:

(1) Zahlen:

ganze Zahlen, z.B. **`123`**, **`-1`**, **`1.2e+3`** (= 1200)
reelle Zahlen, z.B. **`123.7`**, **`-0.7`**, **`1.2e-3`** (= 0.0012)

(2) Datenelemente:

DObjName od. *DObjName***`[`***Index1***`]`** od. *DObjName***`[`***Index1***`][`***Index2***`]`**

Indizes müssen angegeben werden, falls es sich bei *DObjName* um ein Vektor- oder Matrix-Datenobjekt handelt.

Bsp.:	**`dobj1`**	(skalares Datenobjekt)
	`dobj2[0]`	(Element 0 eines Vektor-Datenobjekts)
	`dobj3[5][1*1+2]`	(Element 5,3 eines Matrix-Datenobjekts)

(3) arithmetische Konstanten:

`e`	2.718...	(Eulersche Konstante)
`pi`	3.141...	(π)

(4) arithmetische Operatoren:

`+` , **`-`** , **`*`** , **`/`** , **`div`**, **`mod`**, **`^`** (Potenz)

(5) arithmetische Funktionen:

`floor(`*X***`)`**	größte ganze Zahl $\le X$, d.h. $\lfloor X \rfloor$ („entier“)
`ceil(`*X***`)`**	kleinste ganze Zahl $\ge X$, d.h. $\lceil X \rceil$
`abs(`*X***`)`**	Betrag von X, d.h. $\lvert X \rvert$
`sgn(`*X***`)`**	Vorzeichen-Funktion ($X < 0$: -1 , $X \ge 0$: 1)
`exp(`*X***`)`**	Exponentialfunktion e^X
`ln(`*X***`)`**	natürlicher Logarithmus ln X , $X > 0$
`lg(`*X***`)`**	dekadischer Logarithmus lg X , $X > 0$
`sqr(`*X***`)`**	Quadrat von X
`sqrt(`*X***`)`**	Quadratwurzel von X , $X \ge 0$
`sin(`*X***`)`**	Sinus von X
`cos(`*X***`)`**	Cosinus von X

`dim1(`*DObjName***`)`**	1. Dimension von *DObjName* (liefert 0, falls Skalar)
`dim2(`*DObjName***`)`**	2. Dimension von *DObjName* (liefert 0, falls Skalar bzw. Vektor)

`min(`*X,Y***`)`**	kleinerer der Werte X und Y
`max(`*X,Y***`)`**	größerer der Werte X und Y

(6) arithmetische Folgen:

sum(*HVar***=***UG***..***OG*$_A$**,***X***)**

Summe von *X* für alle ganzzahligen *HVar*-Werte von *UG* bis *OG*

entspricht $\sum_{HVar=UG}^{OG} X$

prod(*HVar***=***UG***..***OG***,***X***)**

Produkt von *X* für alle ganzzahligen *HVar*-Werte von *UG* bis *OG*

entspricht $\prod_{HVar=UG}^{OG} X$

mit *HVar* - lokal eindeutige Hilfsvariable,
Namenskonvention siehe „Datenobjekt“ (i.allg. **i**, **j**, ..)
UG, *OG* - untere bzw. obere Grenze des Zählbereichs
X - arithmetischer Ausdruck (der i.allg. *HVar* enthält)

(7) Hilfsvariablen:

werden durch ihre Namen repräsentiert
(sind nur innerhalb von sum/prod-Ausdrücken erlaubt, s.o.)

(8) bedingte Ausführung:

if(*B*$_1$ *vop B*$_2$**,***X***,***Y***)**

liefert *X*, wenn die Bedingung *B*$_1$ *vop B*$_2$ erfüllt ist, sonst *Y*

B$_1$, *B*$_2$ – arithmetische Ausdrücke
vop – Vergleichsoperator aus { **<** , **<=** , **==** , **!=** , **>=** , **>** }

(9) Klammerung:

(.. **)** Zusammenfassung arithmetischer Ausdrücke

Bsp.:
```
1 + 2 * 3
(1+2) * 3
dobj1 + 2 * sqrt(dobj3[1][2])
3.7e-1 - sum(i=0..dim1(dobj2)-1,dobj2[i]^2)
if(dobj1 < sqrt(dobj2[0]),0.391,ln(11.8))
```

A.3 Beschreibungsblöcke

Aufgabenbeschreibungen bestehen aus einer Reihe von Blöcken der allgemeinen Form

```
<blockname>
{
  <anweisung> ;
  ...
}
```

Dabei spezifiziert <blockname> die Art (≡ Funktion) des jeweiligen Blockes. Die Reihenfolge der Blöcke ist prinzipiell beliebig, solange Datenobjekte stets erst definiert und anschließend verwendet werden. Der Einheitlichkeit und Übersichtlichkeit halber sollte jedoch die Reihenfolge eingehalten werden, in der die Blöcke in den folgenden Abschnitten beschrieben sind.

Kommentare können an beliebiger Stelle innerhalb und außerhalb von Blöcken eingefügt werden. Einzeilenkommentare werden mittels **//** ... realisiert, Mehrzeilenkommentare mittels **/*** ... ***/**.

Der Definitionsblock `data_objects`

In diesem Block werden alle Datenobjekte definiert, die im Rahmen der Aufgabenbeschreibung benötigt werden[1]. Dabei wird für jedes Datenobjekt der Zahlentyp[2] und die Struktur angegeben. Eine Definition besitzt die allgemeine Form

(**int** | **float**) *DObjName* [**[***Dim1***]**[**[***Dim2***]**]]**;**

Bsp.:
```
data_objects
{
  int dobj1;              // ganze Zahl
  float dobj2[5];         // Vektor reeller Zahlen der Länge 5
  int dobj3[4][2];        // Matrix ganzer Zahlen der Dimension 4x2
}
```

In jeder Aufgabenbeschreibung muß mindestens ein Datenobjekt definiert sein.

Der Konstantenblock `constants`

In diesem Block können Datenelemente als numerische Konstanten definiert werden. Eine Definition besitzt eine der allgemeinen Formen

DObjName [**[***Index1***]**[**[***Index2***]**]] **=** *ArithAus***;**

DObjName **= {** *ArithAus1***,** *ArithAus2***,** ... **};**

DObjName **= { {** *ArithAus11***,** *ArithAus12***,** ... **}, {** *ArithAus21***,** *ArithAus22***,** ... **},** ... **};**

---

[1] Die Anzahl definierbarer Datenobjekte wird nur durch den verfügbaren Hauptspeicher begrenzt, jedoch darf jedes Datenobjekt nur aus maximal 1000 Datenelementen bestehen.

[2] ganze Zahlen - **int** - $-2^{31}..2^{31}-1$
reelle Zahlen - **float** - $\pm 1.79*10^{308}$

*ArithAus** ist dabei ein konstanter arithmetischer Ausdruck, der den Wert der numerischen Konstante repräsentiert. Die letzten beiden Formen können verwendet werden, um alle Datenelemente von Vektor- bzw. Matrix-Datenobjekten gleichzeitig als numerische Konstanten zu definieren.

Bsp.:
```
constants
{
  dobj1 = -7;
  dobj3 = { {11,12}, {21,22}, {31,32}, {41,42} };
}
```

Der Parameterblock `parameters`

In diesem Block können Datenelemente als Optimierungsparameter definiert werden[1]. Eine Definition besitzt die allgemeine Form

DObjName [**[***Index1***]**[**[***Index2***]**]] **=** *ArithAus1* **..** *ArithAus2***;**

wobei *ArithAus1* und *ArithAus2* konstante arithmetische Ausdrücke sind, die die untere bzw. obere Grenze des Parameter-Definitionsbereichs angeben.

Bsp.:
```
parameters
{
  dobj1 = 1 .. 100;
  dobj2[0] = 0.3 .. 5.8;
}
```

In jeder Aufgabenbeschreibung muß mindestens ein Optimierungsparameter definiert sein.

Der Startpunktblock `starting_points`

In diesem Block können „Startpunkte“ für die Optimierung definiert werden, d.h. Belegungen der Optimierungsparameter, die (wahrscheinlich) in der Nähe des gesuchten Optimums liegen[2]. Die Definition eines Startpunkts besitzt die allgemeine Form

{ *ArithAus1***,** *ArithAus2***,** ... **};**

wobei *ArithAus** konstante arithmetische Ausdrücke sind, die den jeweiligen Wert des ersten, zweiten, ... Optimierungsparameters[3] repräsentieren. Die Liste muß exakt so viele Ausdrücke enthalten, wie Optimierungsparameter existieren.

Bsp.:
```
starting_points
{
  { 0.32, 15, -100.9 };
  { 0.58, 12, 0.0 };
}
```

[1] Eine Aufgabenbeschreibung darf nicht mehr als 1000 Optimierungsparameter enthalten.

[2] Eine Aufgabenbeschreibung darf nicht mehr als 100 Startpunkte enthalten.

[3] Die Numerierung entspricht der Reihenfolge, in der die Optimierungsparameter definiert wurden.

Der Abhängigenblock `dependents`

In diesem Block können Datenelemente als Formelabhängige definiert werden[1]. Eine Definition besitzt die allgemeine Form

DObjName [**[***Index1***]**[**[***Index2***]**]] **=** *ArithAus***;**

wobei *ArithAus* ein arithmetischer Ausdruck ist, der die numerische Formel repräsentiert, die mit dem Datenelement zu verknüpfen ist. Wenn im Laufe der Optimierung der Wert dieses Datenelementes benötigt wird, dann wird dieser anhand der zugeordneten Formel berechnet[2].

Bsp.:
```
dependents
{
  dobj1 = sqrt(3*dobj3[1][0]) + 2;
  dobj2[3] = dobj1 / 4.3;
}
```

Der Nebenbedingungsblock `constraints`

In diesem Block können Nebenbedingungen definiert werden, die eine gültige Lösung der Optimierungsaufgabe erfüllen muß[3]. Eine Definition besitzt die allgemeine Form

*ArithAus1 VOp ArithAus2***;**

wobei *ArithAus1* und *ArithAus2* arithmetische Ausdrücke sind und *VOp* ein Vergleichsoperator aus { **<**, **<=**, **==**, **!=**, **>=**, **>** }.

Bsp.:
```
constraints
{
  sum(i=1..5,dobj2[i]) < 100;
  sqrt(dobj1) >= 2*dobj3[1][2];
}
```

Es ist darauf zu achten, daß keine sich widersprechenden Nebenbedingungen definiert werden (z.B. **`dobj1 > 10`** , **`dobj1 < 8`**), da die Optimierung sonst keine gültige Lösung finden kann.

Der Zielfunktionsblock `objective_function`

In diesem Block wird das Ziel der Optimierung durch Angabe einer Zielfunktion (Funktion, die die Güte einer Lösung bewertet) und der Optimierungsart (Minimierung oder Maximierung) definiert. Der Block muß jeweils genau einen Eintrag der Form

`formula =` *ArithAus***`;`** und
`objective =` (**`minimize`** | **`maximize`**) **`;`**

---

[1] Eine Aufgabenbeschreibung darf nicht mehr als 1000 Formelabhängige enthalten.

[2] Genaugenommen wird der Wert einer Formelabhängigen nur dann neu berechnet, wenn sich die Werte der in der Formel enthaltenen Datenelemente geändert haben.

[3] Eine Aufgabenbeschreibung darf nicht mehr als 1000 Nebenbedingungen enthalten.

enthalten, wobei *ArithAus* ein arithmetischer Ausdruck ist, der die Zielfunktion repräsentiert, und **`minimize`** für Minimierung und **`maximize`** für Maximierung steht.

Bsp.:
```
objective_function
{
  formula   = dobj1 - sum(i=0..dim1(dobj2)-1,dobj2[i]);
  objective = minimize;
}
```

In jeder Aufgabenbeschreibung muß genau ein Zielfunktionsblock enthalten sein.

Der Programmblock `program`

In diesem Block wird die Schnittstelle zu einem externen Programm definiert, das zur Bestimmung der Werte von Programmabhängigen verwendet wird[1]. Der Block kann folgende Einträge enthalten:

```
identifier = Ident;
host       = HostName;
name       = ProgName;
options    = ProgOpts;
in_data    = { DObjNameI1, DObjNameI2, ... };
out_data   = { DObjNameO1, DObjNameO2, ... };
in_format  = InFormat;
out_format = OutFormat;
in_file    = InFileName;
out_file   = OutFileName;
```

Der **`identifier`**-Eintrag weist der Schnittstelle einen eindeutigen Namen zu (Namensbildung analog zu Datenobjekten, siehe Kapitel A.1). Dieser Eintrag ist obligatorisch.

Der **`host`**-Eintrag gibt den DNS-Namen des Host-Rechners an, auf dem das Programm läuft[2], z.B. `mascot.informatik.tu-chemnitz.de`. Dieser Eintrag ist optional; standardmäßig wird davon ausgegangen, daß das Programm auf demselben Rechner läuft wie das Optimierungssystem.

Der **`name`**-Eintrag gibt den Aufrufnamen (inkl. Pfad) des Programmes an. Der Name muß in Anführungszeichen ″...″ eingeschlossen sein. Dieser Eintrag ist obligatorisch.

Der **`options`**-Eintrag gibt die Kommandozeilenargumente an, die beim Programmaufruf zu verwenden sind. Die Argumente müssen in Anführungszeichen ″...″ eingeschlossen sein. Dieser Eintrag ist optional.

Der **`in_data`**-Eintrag gibt die Namen der Datenobjekte an, deren Werte als Eingangsdaten an das Programm zu übergeben sind. Dieser Eintrag ist optional, i.allg. wird einem Programm jedoch mindestens der Wert eines Datenobjekts (meist Optimierungsparameter) übergeben.

---

[1] Eine Aufgabenbeschreibung darf nicht mehr als 100 Programmblöcke enthalten.

[2] Die Funktionalität, Programme auf anderen Rechnern auszuführen, steht in der aktuellen Version des Optimierungssystems leider noch nicht zur Verfügung.

Der **out_data**-Eintrag gibt die Namen der Datenobjekte an, deren Werte als Ergebnisdaten vom Programm geliefert werden[1]. Dieser Eintrag ist optional, i.allg. wird von einem Programm jedoch mindestens der Wert eines Datenobjekts (Programmabhängige) geliefert.

Der **in_format**-Eintrag spezifiziert die Art der Datenübergabe an das Programm. Dieser Eintrag ist optional; standardmäßig wird **i_face**[2] angenommen.

Der **out_format**-Eintrag spezifiziert die Art der Datenübernahme vom Programm. Dieser Eintrag ist optional; standardmäßig wird **i_face** angenommen.

Der **in_file**-Eintrag gibt den Namen (inkl. Pfad) der Datei an, in der die an das Programm zu übergebenden Eingangsdaten zu speichern sind. Der Name muß in Anführungszeichen ″...″ eingeschlossen sein. Dieser Eintrag ist optional und hat nur dann eine Wirkung, wenn mittels **in_format** eine dateiorientierte Übergabeart (z.B. **i_face**) festgelegt wurde.

Der **out_file**-Eintrag gibt den Namen (inkl. Pfad) der Datei an, aus der die vom Programm zu übernehmenden Ergebnisdaten gespeichert sind. Der Name muß in Anführungszeichen ″...″ eingeschlossen sein. Dieser Eintrag ist optional und hat nur dann eine Wirkung, wenn mittels **out_format** eine dateiorientierte Übernahmeart (z.B. **i_face**) festgelegt wurde.

Der Eigenschaftsblock `properties`

In diesem Block können bestimmte Eigenschaften des Optimierungsproblems, deren Kenntnis bei der Optimierung von Vorteil sein könnte, angegeben werden[3]. Dadurch kann die interne Eigenschaftserkennung des Optimierungssystems unterstützt werden, was ggf. die Qualität und Effizienz der Optimierung erhöht[4]. Eine solche Angabe besitzt die allgemeine Form

EigTyp EigName **=** *EigWert* **;**

wobei *EigTyp* einen Datentyp aus { **int**, **float**, **bool**, **text** }, *EigName* den eindeutigen Namen der Eigenschaft und *EigWert* deren Wert repräsentiert. Folgende Beziehungen existieren zwischen *EigTyp* und *EigWert*:

EigTyp:		*EigWert*:	
	int		ganze Zahl
	float		Zahl
	bool		Zahl ≠ 0 oder **true** → wahr
			Zahl = 0 oder **false** → falsch
	text		Zeichenkette, ggf. in ″...″ eingeschlossen

---

[1] Achtung: Ein Datenobjekt darf nur dann in einer **out_data**-Liste auftreten, wenn keines seiner Elemente eine Konstante, Parameter oder Formelabhängige ist (→ alle Elemente werden als Programmabhängige definiert). Außerdem darf ein Datenobjekt höchstens in der **out_data**-Liste eines Programmblocks auftreten.

[2] **i_face** ist eine einfache Datenschnittstelle auf der Basis von Textdateien, die im Kapitel B.3 näher beschrieben ist. Andere Übergabe-/Übernahmearten stehen in der aktuellen Version des Optimierungssystems leider nicht zur Verfügung.

[3] Eine Aufgabenbeschreibung darf nicht mehr als 1000 Eigenschaften spezifizieren.

[4] Bestehen Unklarheiten darüber, welche Eigenschaftsausprägungen ein Optimierungsproblem aufweist, sollte die Eigenschaftserkennung ganz- oder teilweise dem Optimierungssystem überlassen werden (Grundsatz: „besser keine Information als eine falsche“).

Folgende Eigenschaften können angegeben werden:

bool obj_func_stochastic	Zielfunktion stochastisch ?
bool cons_stochastic	Mindestens eine Nebenbeding. stochastisch ?
bool problem_stochastic	Optimierungsproblem stochastisch ?
bool obj_func_linear	Lineare Zielfunktion ?
bool obj_func_quadratic	Quadratische Zielfunktion ?
bool obj_func_nonlinear	Sonstige Zielfunktion ?
bool cons_linear	Alle Nebenbeding. linear ?
bool cons_quadratic	Nur lineare und (mind. eine) quadratische Nebenbeding. ?
bool cons_nonlinear	Mindestens eine nicht-lineare/quadratische Nebenbeding. ?
bool problem_linear	Lineares Optimierungsproblem ?
bool problem_quadratic	Quadratisches Optimierungsproblem ?
bool problem_nonlinear	Sonstiges Optimierungsproblem ?
bool problem_convex	Ist das Optimierungsproblem konvex ?
float invalid_ratio	Anteil ungültiger Punkte am Suchraum
float runtime_succ_trial	mittlere Dauer der Bewertung eines gültigen Punktes (in sec, > 0)
float runtime_fail_trial	mittlere Dauer der Bewertung eines ungültigen Punktes (in sec, > 0)

Anhang B: Schnittstellen von DynamO

Das Optimierungssystem DynamO verfügt über eine Reihe von Benutzer-, Datei- und Programmschnittstellen, die in diesem Kapitel näher erläutert werden (siehe Bild B.1).

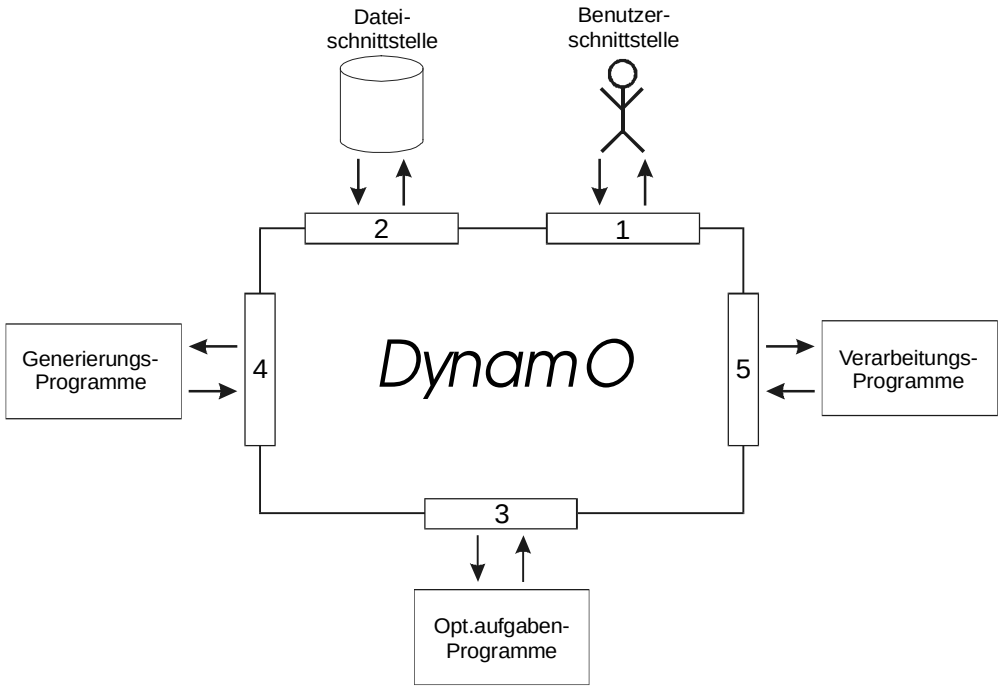

Bild B.1 Schnittstellen des Optimierungssystems DynamO

B.1 Die Benutzerschnittstelle

Die Benutzerschnittstelle besitzt folgende wesentliche Funktionen zur Kommunikation zwischen dem Nutzer und dem Optimierungssystem:

- Verwaltung von Optimierungsaufgabenbeschreibungen
  (Generieren, Laden, Anzeigen, Editieren, Speichern, Drucken)
- Konfigurierung des hybriden Optimierungsverfahrens
- Durchführung der Optimierung
- Verwaltung von Optimierungsergebnissen
  (Visualisieren, Verarbeiten, Speichern)

Die Benutzerschnittstelle ist fensterorientiert und entspricht dem von Windows 95/98/NT bekannten Look-and-Feel-Standard. Das Hauptfenster, das nach dem Start von DynamO erscheint, besteht aus den folgenden drei Bereichen (siehe Bild B.2):

- *Menüleiste*
  menügesteuerte Anwahl von Programmfunktionen per Tastatur oder Maus
- *Arbeitsbereich*
  Darstellung von Editoren, Tabellen und Diagrammen
  der Arbeitsbereich ist unterteilt in die Arbeitsflächen *Optimierungsaufgabe*, *Optimierungslauf* und *Optimierungsergebnis*

- *Statuszeile*
  Bereitstellung zusätzlicher Informationen, wie Dateinamen, Cursorpositionen, Editiermodi und Status-/Fehlermeldungen

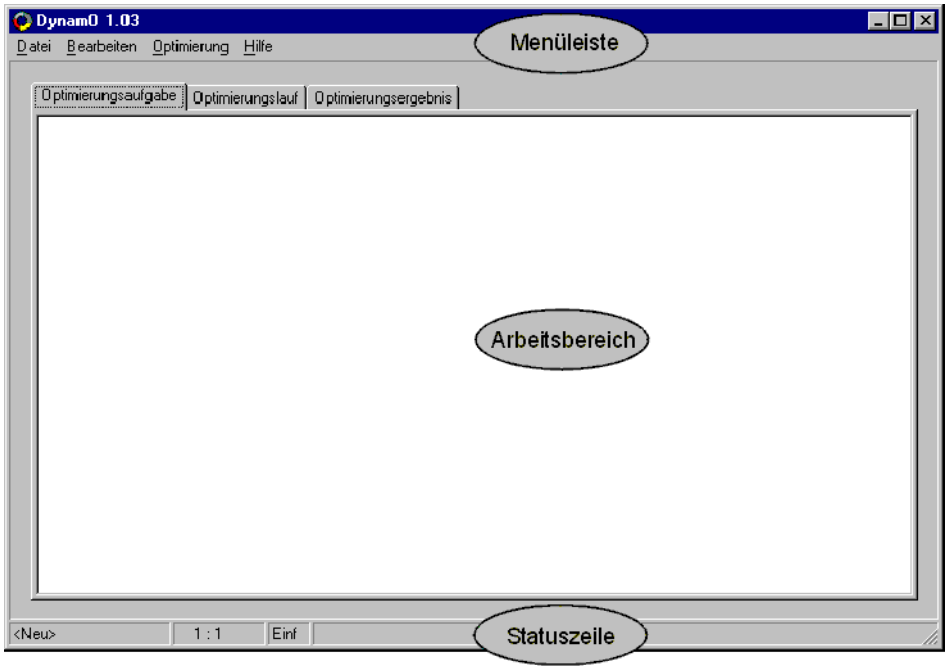

Bild B.2 Hauptfenster von DynamO

B.1.1 Verwaltung von Optimierungsaufgabenbeschreibungen

Eine Optimierungsaufgabenbeschreibung spezifiziert die Informationen zu einer Optimierungsaufgabe, die zur Durchführung einer automatisierten Optimierung notwendig sind (zum Inhalt und zur Syntax siehe Anhang A). Die Beschreibung, die gerade im Aufgabeneditor (Arbeitsfläche „Optimierungsaufgabe“) enthalten ist, wird als *aktive* Beschreibung bezeichnet und spezifiziert die Aufgabe, die aktuell durch Optimierung zu lösen ist. Folgende Funktionen zur Verwaltung von Optimierungsaufgabenbeschreibungen stehen zur Verfügung:

- Leeren des Aufgabeneditors:

  Diese Funktion wird über den Menüpunkt Datei→Neu→Leere Aufgabenbeschreibung (Shortcut Ctrl-N) oder Datei→Schließen aktiviert. Wenn die aktive Beschreibung modifiziert und noch nicht gespeichert wurde, wird vor dem Leeren gefragt, ob dies geschehen soll.

- Erzeugen einer Beschreibungsschablone im Aufgabeneditor:

  Diese Funktion wird über den Menüpunkt Datei→Neu→Aufgabenbeschreibungsschablone (o. Prog.) bzw. Datei→Neu→Aufgabenbeschreibungsschablone (m. Prog.) aktiviert und erzeugt im Aufgabeneditor das „Gerüst“ einer Beschreibung ohne bzw. mit Programmblock, das dann vom Nutzer erweitert werden kann. Wenn die aktive Beschreibung modifiziert und noch nicht gespeichert wurde, wird vor dem Erzeugen gefragt, ob dies geschehen soll.

- Laden einer Beschreibung aus einer Textdatei in den Aufgabeneditor:

  Diese Funktion wird über den Menüpunkt Datei→Öffnen (Shortcut Ctrl-O) aktiviert. Wenn die aktive Beschreibung modifiziert und noch nicht gespeichert wurde, wird vor

dem Laden gefragt, ob dies geschehen soll. Beim Laden der neuen Beschreibung erfolgt kein Test auf syntaktische und semantische Korrektheit.

- Generieren einer Beschreibung durch ein externes Programm und Laden in den Aufgabeneditor:

  Diese Funktion wird über den Menüpunkt Datei→Generieren aktiviert. Wenn die aktive Beschreibung modifiziert und noch nicht gespeichert wurde, wird vor dem Generieren gefragt, ob dies geschehen soll. Das Prinzip der Beschreibungsgenerierung wird in Kapitel B.4 näher erläutert. Beim Laden der generierten Beschreibung erfolgt kein Test auf syntaktische und semantische Korrektheit.

- Speichern der Beschreibung im Aufgabeneditor in einer Textdatei:

  Diese Funktion wird über den Menüpunkt Datei→Speichern (Shortcut Ctrl-S) bzw. Datei→Speichern Unter aktiviert (nur möglich, wenn Arbeitsfläche „Optimierungsaufgabe“ aktiv). Im ersten Fall wird die mit der Beschreibung assoziierte Textdatei verwendet[1], im zweiten Fall muß explizit der Name der Textdatei angegeben werden. Wenn die Textdatei bereits existiert, wird vor dem Speichern gefragt, ob sie überschrieben werden soll.

- Drucken der Beschreibung im Aufgabeneditor:

  Diese Funktion ist in der aktuellen Version der Software leider noch nicht verfügbar.

Der Aufgabeneditor dient zur Darstellung sowie manuellen Erzeugung bzw. Modifizierung von Optimierungsaufgabenbeschreibungen. Er besitzt die Funktionalität klassischer Windows-Editoren (z.B. NotePad), wie Cut&Paste und Suchen&Ersetzen (siehe Menüpunkt Bearbeiten), darüber hinaus aber folgende Erweiterungen:

- Syntax-Highlighting für Optimierungsaufgabenbeschreibungen
- Anspringen der i-ten Textzeile über den Menüpunkt Bearbeiten→Gehe zu
- Umschalten zwischen Einfüge- und Überschreib-Modus über die Taste Ins
- Löschen von Textzeilen über den Shortcut Ctrl-Y
- Anzeige von Cursorposition und Editiermodus in der Statuszeile

Die Undo-Funktion (Menüpunkt Bearbeiten→Rückgängig*) ist in der aktuellen Version der Software leider noch nicht verfügbar.*

B.1.2 Konfigurierung des hybriden Optimierungsverfahrens

DynamO verwendet ein adaptives hybrides Optimierungsverfahren, bestehend aus folgenden Einzelverfahren (zur Funktionsweise siehe Kapitel 4 sowie /HADE-97/, /HADE-98a/, /HADE-98b/, /HADE-99a/ und /HADE-00/):

- Vollständige Enumeration
- Globale Zufallssuche
- Lokale Zufallssuche

[1] Eine Datei ist dann mit einer Beschreibung „assoziiert“, wenn die Beschreibung aus der Datei geladen oder zuletzt in dieser Datei gespeichert wurde.

- Gradientenverfahren
- Simplexmethode von Nelder & Mead
- Methode von Hooke & Jeeves
- Simulated Annealing
- Genetischer Algorithmus

Der nutzer-zugängliche Teil der Konfigurierung des hybriden Verfahrens erfolgt über den Menüpunkt Optimierung→Optionen (siehe Bild B.3). Da sich der hybride Mechanismus, basierend auf den Merkmalen der zu lösenden Aufgabe, weitgehend selbst konfiguriert, sind nur wenige Einstellmöglichkeiten vorhanden.

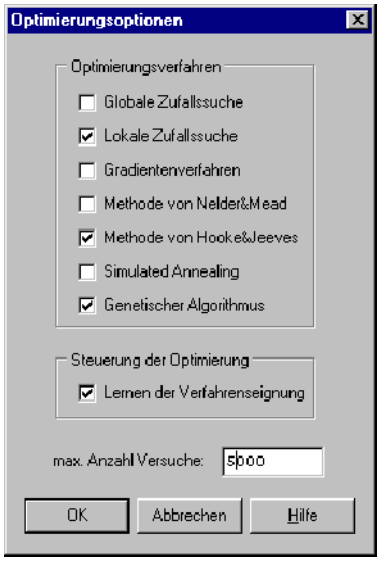

Bild B.3 Einstellung der Optimierungsoptionen in DynamO

Der Nutzer kann festlegen, welche Einzelverfahren während der nächsten Optimierung prinzipiell zum Einsatz kommen dürfen[1], ob bei der Steuerung der Optimierung Erkenntnisse zur Verfahrenseignung, die im bisherigen Verlauf der Optimierung gewonnen wurden, berücksichtigt werden sollen und wieviele Parameterkonstellationen („Punkte“) insgesamt maximal erzeugt und untersucht werden dürfen[2]. Sollte die Beschreibung der zu lösenden Aufgabe keine gültige Startlösung enthalten (Block `starting_points`, siehe Kapitel A.3), dann muß mindestens ein Optimierungsverfahren aktiviert werden, daß keine Startlösung benötigt, um eine Optimierung zu ermöglichen (siehe hierzu die Tabelle im Kapitel 4.3).

Es ist zu beachten, daß Optionsänderungen erst beim nächsten Start einer Optimierung wirken, d.h. keinen Einfluß auf eine evtl. gerade laufende Optimierung haben.

[1] Das heißt nicht, daß die vom Nutzer ausgewählten Verfahren auch zum Einsatz kommen, sondern schränkt lediglich die Freiheitsgrade der automatischen Konfigurierung entsprechend ein (ein nicht ausgewähltes Verfahren kommt garantiert nicht zum Einsatz).

[2] Die maximale Anzahl zu untersuchender Punkte stellt nicht nur ein Abbruchkriterium dar, sondern wird auch in die Steuerung der Optimierung (Auswahl und Parametrierung der Einzelverfahren) einbezogen.

B.1.3 Durchführung der Optimierung

Grundvoraussetzung für die Durchführung einer Optimierung ist die Existenz einer Optimierungsaufgabenbeschreibung im Aufgabeneditor. Der Start der Optimierung erfolgt über den Menüpunkt Optimierung→Start (Shortcut F9). Im Arbeitsbereich wird dabei automatisch die Arbeitsfläche „Optimierungslauf“ angezeigt (siehe Bild B.4).

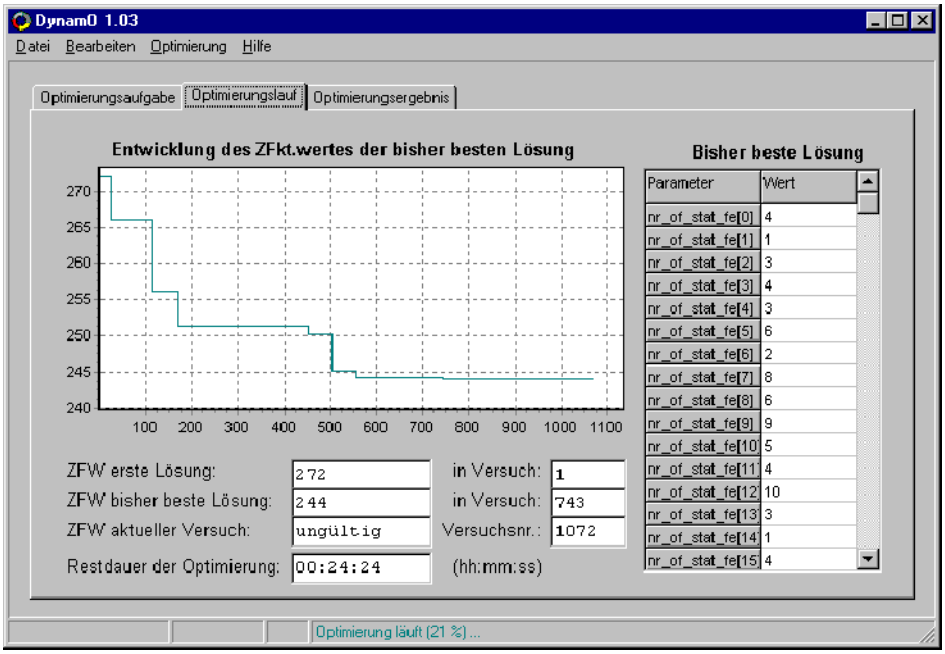

Bild B.4 Darstellung des Optimierungsverlaufs in DynamO

Auf dieser Arbeitsfläche werden während der Optimierung folgende aktuellen Informationen bereitgestellt:

- Zielfunktionswert und Versuchsnummer der ersten gültigen Lösung, die gefunden wurde
- Parameterbelegung[1], Zielfunktionswert und Versuchsnummer der bisher besten gefundenen Lösung
- Zielfunktionswert/Versuchsnummer-Diagramm aller Verbesserungen der bisher besten Lösung
- Zielfunktionswert und Versuchsnummer des zuletzt bewerteten Punktes
- geschätzte Restdauer der Optimierung

Zusätzlich wird in der Statuszeile angezeigt, wieviel Prozent der maximal erlaubten Versuchszahl bisher „verbraucht“ wurden.

Die Optimierung endet spätestens nach Erreichen der maximal erlaubten Versuchszahl, kann jedoch über den Menüpunkt Optimierung→Stop jederzeit abgebrochen werden.

[1] Tabelle im rechten Teil der Arbeitsfläche „Optimierungslauf“

B.1.4 Verwaltung von Optimierungsergebnissen

Nach dem Ende einer Optimierung wird automatisch ein Ergebnisbericht generiert und in den Ergebniseditor geladen. Inhalt und Format dieses Berichts entsprechen dem der ERG_1-Datei, die im Kapitel B.5 beschrieben ist. Folgende Funktionen zur Verwaltung von Optimierungsergebnissen stehen zur Verfügung:

- Leeren des Ergebniseditors:

  Der Ergebniseditor wird automatisch geleert, wenn der Aufgabeneditor geleert, eine neue Optimierungsaufgabenbeschreibung geladen/generiert oder eine Optimierung gestartet wird. Wenn der aktuelle Ergebnisbericht noch nicht gespeichert wurde, wird vor dem Leeren gefragt, ob dies geschehen soll.

- Generieren eines Ergebnisberichts nach Durchführung einer Optimierung und Laden in den Ergebniseditor:

  Siehe voriges Kapitel.

- Speichern des Ergebnisberichts im Ergebniseditor in einer Textdatei:

  Diese Funktion wird über den Menüpunkt Datei→Speichern (Shortcut Ctrl-S) bzw. Datei→Speichern Unter aktiviert (nur möglich, wenn Arbeitsfläche „Optimierungsergebnis“ aktiv). Im ersten Fall wird die mit dem Bericht assoziierte Textdatei verwendet[1], im zweiten Fall muß explizit der Name der Textdatei angegeben werden. Wenn die Textdatei bereits existiert, wird vor dem Speichern gefragt, ob sie überschrieben werden soll.

- Verarbeiten des Ergebnisberichts durch ein externes Programm

  Diese Funktion wird über den Menüpunkt Datei→Verarbeiten aktiviert. Das Prinzip der Ergebnisverarbeitung wird im Kapitel B.5 näher erläutert.

- Drucken des Ergebnisberichts im Ergebniseditor:

  Diese Funktion ist in der aktuellen Version der Software leider noch nicht verfügbar.

Der Ergebniseditor dient zur Darstellung sowie manuellen Modifizierung[2] von Ergebnisberichten. Er besitzt die Funktionalität klassischer Windows-Editoren (z.B. NotePad), wie Cut&Paste und Suchen&Ersetzen (siehe Menüpunkt Bearbeiten), darüber hinaus aber folgende Erweiterungen:

- Syntax-Highlighting für Ergebnisberichte
- Anspringen der i-ten Textzeile über den Menüpunkt Bearbeiten→Gehe zu
- Umschalten zwischen Einfüge- und Überschreib-Modus über die Taste Ins
- Löschen von Textzeilen über den Shortcut Ctrl-Y
- Anzeige von Cursorposition und Editiermodus in der Statuszeile

Die Undo-Funktion (Menüpunkt Bearbeiten→Rückgängig*) ist in der aktuellen Version der Software leider noch nicht verfügbar.*

---

[1] Eine Datei ist dann mit einem Ergebnisbericht „assoziiert“, wenn der Bericht zuletzt in dieser Datei gespeichert wurde.

[2] Achtung: Bei der Modifizierung, insbesondere von Nicht-Kommentarzeilen ist zu beachten, daß Probleme bei der Verarbeitung durch externe Programme entstehen können, die bestimmte Formate erwarten !

B.2 Die Dateischnittstelle

Die Dateischnittstelle besitzt folgende Funktionen:

- Lesen von Textdateien, die Optimierungsaufgabenbeschreibungen enthalten
- Sichern von Optimierungsaufgabenbeschreibungen in Textdateien
- Sichern von Optimierungsergebnissen in Textdateien

Das Lesen einer Optimierungsaufgabenbeschreibung führt dazu, daß die Textdatei, deren Name angegeben wurde, in den Aufgabeneditor von DynamO geladen wird. Eine ggf. bereits im Editor befindliche Beschreibung wird dabei (nach Rückfrage) gelöscht. Beim Laden erfolgt jedoch noch kein Test, ob es sich um eine korrekte Beschreibung handelt; dieser Test wird erst beim Start einer Optimierung durchgeführt. Diese Schnittstellenfunktion kann vom Nutzer über den Menüpunkt Datei→Öffnen aktiviert werden, wird zusätzlich jedoch auch von der Generierungs-Programmschnittstelle verwendet (siehe Kapitel B.4).

Das Sichern einer Optimierungsaufgabenbeschreibung führt dazu, daß die aktuell im Aufgabeneditor befindliche Beschreibung in der Textdatei, deren Name angegeben wurde, gespeichert wird (Rückfrage, falls Datei bereits existiert). Beim Speichern erfolgt jedoch kein Test, ob es sich um eine korrekte Beschreibung handelt. Diese Schnittstellenfunktion kann vom Nutzer über die Menüpunkte Datei→Speichern und Datei→Speichern unter aktiviert werden (nur wenn Aufgabeneditor aktiv), wird zusätzlich jedoch auch DynamO-intern verwendet.

Das Sichern von Optimierungsergebnissen führt dazu, daß die aktuell im Ergebniseditor befindlichen Ergebnisse in der Textdatei, deren Name angegeben wurde, gespeichert werden (Rückfrage, falls Datei bereits existiert). Diese Schnittstellenfunktion kann vom Nutzer über die Menüpunkte Datei→Speichern und Datei→Speichern unter aktiviert werden (nur wenn Ergebniseditor aktiv), wird zusätzlich jedoch auch DynamO-intern verwendet.

B.3 Die Optimierungsaufgaben-Programmschnittstelle

Die Optimierungsaufgaben-Programmschnittstelle besitzt folgende Funktionen zur Arbeit mit externen Programmen, die Teile der zu lösenden Optimierungsaufgabe repräsentieren (Analyseprogramme, Simulatoren, etc.):

- Übergabe aller für die Programmausführung notwendigen Eingangsdaten
- Starten des Programmes und Warten auf dessen Beendigung
- Übernahme der vom Programm gelieferten Ergebnisdaten

Immer dann, wenn während einer Optimierung der Wert einer Programmabhängigen benötigt wird, erfolgt die automatischen Ausführung des zugeordneten externen Programms[1]. „Zugeordnet“ ist dabei das Programm, dessen Programmblock im **`out_data`**-Eintrag den Namen des Datenobjekts enthält. In der folgenden Erläuterung wird davon ausgegangen, daß für

[1] Genauer gesagt erfolgt die Programmausführung nicht bei jedem Zugriff auf den Wert der Programmabhängigen, sondern nur dann, wenn der Wert für die aktuelle Belegung der Programm-Eingangsgrößen noch nicht ermittelt wurde.

dieses Programm sowohl als Datenübergabe- als auch Datenübernahmeart **`i_face`** festgelegt wurde (eine Kurzbeschreibung zu **i_face** finden Sie am Ende dieses Kapitels).

Als erstes werden von DynamO die aktuellen Werte aller Datenobjekte, die im **`in_data`**-Eintrag angegeben sind, ermittelt und entsprechend ihrer Reihenfolge in diesem Eintrag im **i_face**-Format in eine Textdatei geschrieben, deren Name im **`in_file`**-Eintrag festgelegt ist. Anschließend wird das Programm unter Angabe der im **`options`**-Eintrag enthaltenen Kommandozeilenargumente gestartet. Das Programm liest die Eingangsdaten, führt entsprechende Aktionen durch (meist Analyse/Bewertung einer Systemvariante), schreibt die Ergebnisdaten und beendet sich. DynamO liest die Ergebnisdatei, deren Name dem im **`out_file`**-Eintrag angegebenen entsprechen und das **i_face**-Format aufweisen muß und weist allen Datenobjekten, die im **`out_data`**-Eintrag aufgezählt sind, entsprechend ihrer Reihenfolge in diesem Eintrag die gelesenen Werte zu. Anschließend wird die Optimierung fortgesetzt.

Kurzbeschreibung der Datenschnittstelle i_face

i_face ist eine auf Textdateien basierende Datenschnittstelle, bei der numerische Daten als Skalare behandelt oder zu Vektoren oder zweidimensionalen Matrizen zusammengefaßt werden können (vgl. Konzept Datenelement→Datenobjekt der Optimierungsaufgabenbeschreibungssprache, Kapitel A.1). Die Speicherung in Textdateien erfolgt zeilenweise, wobei

- der Wert eines Skalars in einer Zeile steht
- die Werte eines Vektors nacheinander in einer Zeile stehen (Trennung durch Leerzeichen oder Tab‘s)
- die Werte einer Matrixzeile nacheinander in einer Zeile (Trennung durch Leerzeichen oder Tab‘s) und die Zeilen einer Matrix in aufeinanderfolgenden Zeilen stehen

Ausführlichere Informationen zu i_face entnehmen Sie bitte /HADE-98c/.

B.4 Die Generierungs-Programmschnittstelle

Die Generierungs-Programmschnittstelle besitzt folgende Funktionen zur Arbeit mit externen Programmen, die automatisch Optimierungsaufgabenbeschreibungen generieren:

- Starten des Programms, Kommunikation zwischen DynamO und dem Programm und Warten auf Beendigung des Programms

Nachdem der Nutzer das Programm spezifiziert hat, das die Beschreibung generieren soll (Menüpunkt Datei→Generieren), startet DynamO das Generierungsprogramm, wobei als Kommandozeilenargument

`--stdin_handle` <IHandleNr> `--stdout_handle` <OHandleNr>

angegeben wird. <IHandleNr> und <OHandleNr> sind dabei die Nummern zweier anonymer Pipes, die als Kommunikationsverbindung zwischen den beiden Prozessen dienen. <IHandleNr> referenziert dabei aus Sicht des Generierungsprogramms eine Lese-Pipe (Eingabe, „Tastatur“), <OutHandleNr> eine Schreib-Pipe (Ausgabe, „Bildschirm“). Folgendes Kommunikationsprotokoll zwischen den beiden Prozessen muß eingehalten werden:

- Die Kommunikation erfolgt durch Austausch von Textzeilen, die mit **$** beginnen. Alle anderen Zeilen werden ignoriert.
- Wenn das Generierungsprogramm die gewünschte Beschreibung erzeugt und als Textdatei abgespeichert hat, teilt es DynamO den Namen dieser Datei durch Senden der Zeile **$a** <Dateiname> mit.
- Wenn das Generierungsprogramm nicht in der Lage ist, die gewünschte Beschreibung zu erzeugen, teilt es DynamO dies durch Senden der Zeile **$e** mit.

Nachdem Dynamo eine **$a**- oder **$e**-Zeile empfangen hat, wird das Generierungsprogramm automatisch beendet[1]. War die Generierung der Optimierungsaufgabenbeschreibung erfolgreich, wird diese anschließend über die Dateischnittstelle automatisch geladen (siehe Kapitel B.2).

B.5 Die Verarbeitungs-Programmschnittstelle

Die Verarbeitungs-Programmschnittstelle besitzt folgende Funktionen zur Arbeit mit externen Programmen, die die Ergebnisse durchgeführter Optimierungen weiterverarbeiten:

- Speichern der Optimierungsergebnisse in temporären Dateien
- Starten des Programms und Warten auf dessen Beendigung

Diese Schnittstelle ist nur nach einer „erfolgreichen“ Optimierung nutzbar, d.h. wenn während der Optimierung mindestens eine gültige Lösung gefunden wurde.

Nachdem der Nutzer das Programm spezifiziert hat, das die Optimierungsergebnisse weiterverarbeiten soll (Menüpunkt Datei→Verarbeiten), erzeugt DynamO zwei temporäre Textdateien mit Informationen zur durchgeführten Optimierung.

Die erste Datei (im folgenden kurz ERG_1-Datei genannt) enthält Kurzinformationen zur durchgeführten Optimierung in folgender Reihenfolge[2]:

- Name der Datei mit der Beschreibung der gelösten Optimierungsaufgabe
- Anzahl Optimierungsparameter
- Wurde mindestens eine gültige Lösung gefunden ? (`1` - ja , `0` - nein)
- Belegung der Optimierungsparameter der besten gefundenen Lösung (alles `0`, falls keine gültige Lösung gefunden)
- Zielfunktionswert der besten gefundenen Lösung (`0`, falls keine gültige Lösung gefunden)

Der Name dieser Datei kann in der Konfigurationsdatei von DynamO festgelegt werden (siehe Anhang C), standardmäßig wird <akt. Verzeichnis>`\_dp_wopt_.erg1` verwendet.

[1] Wird DynamO während der Ausführung des Generierungsprogramms beendet, wird automatisch auch letzteres beendet.

[2] Jede Information steht in einer eigenen Zeile; Kommentarzeilen beginnen mit #.

Die zweite Datei (im folgenden kurz ERG_2-Datei genannt) enthält ausführlichere Informationen zur besten gefundenen Lösung in folgender Reihenfolge[1]:

- Anzahl Optimierungsparameter
- Belegung der Optimierungsparameter der besten gefundenen Lösung
- Zielfunktionswert der besten gefundenen Lösung
- Belegung aller Programmabhängigen („Kenngrößen“) der besten gefundenen Lösung[2]

Der Name dieser Datei entspricht dem Namen der ERG_1-Datei, wobei deren Extension durch `.erg2` ersetzt wird (existiert keine Extension, wird `.erg2` an den Namen angehängt).

Nachdem die beiden Dateien erzeugt wurden, startet DynamO das Verarbeitungsprogramm, wobei als Kommandozeilenargument

`--result_file` <ERG_1-Datei>

angegeben wird. Das Verarbeitungsprogramm führt basierend auf den Optimierungsergebnissen entsprechende Aktionen durch, z.B. Visualisierung, Generierung von Reports oder Speicherung in einer Datenbank. Nach Beendigung des Verarbeitungsprogramms kann die Arbeit mit DynamO fortgesetzt werden[3].

[1] Jede Information steht in einer eigenen Zeile; Kommentarzeilen beginnen mit **#**.

[2] Die Belegung jeder Programmabhängigen wird in einer Zeile (Skalar, Vektor) bzw. einer Folge von Zeilen (Matrix) angegeben. Unmittelbar davor steht eine Kommentarzeile der Form **# `Kenngroesse`** *DON*, wobei *DON* den in Hochkommas eingeschlossenen Namen des jeweiligen Datenobjekts angibt.

[3] Wird DynamO während der Ausführung des Verarbeitungsprogramms beendet, wird automatisch auch letzteres beendet.

Anhang C: Installation von DynamO

C.1 Systemvoraussetzungen

DynamO kann auf Rechnern ausgeführt werden, die mindestens über einen Pentium 100, 32 MByte Hautspeicher und 2 MByte freie Festplattenkapazität verfügen. Die Bildschirmauflösung sollte mindestens 800x600 betragen, die Farbtiefe mindestens 8 Bit.

DynamO wurde bisher unter Windows NT 4.0 und Windows 2000 erfolgreich getestet. Die Arbeit unter Windows 95/98/Me sollte ebenfalls möglich sein, wurde jedoch noch nicht verifiziert.

C.2 Installation

Wegen des geringen Schwierigkeitsgrades der Installation bzw. Deinstallation von DynamO existiert zur Zeit keine automatische Installationssoftware. Mit den folgenden Hinweisen sollte es jedoch problemlos möglich sein, Dynamo „per Hand“ zu installieren.

- Anlegen des DynamO-Hauptverzeichnisses
  (o.B.d.A. sei der Verzeichnisname `D:\Programme\DynamO`)
- Anlegen eines Programm- und Dokument-Verzeichnisses in diesem Verzeichnis
  (o.B.d.A. seien die Verzeichnisnamen `Bin` und `Doc`)
- Kopieren der Dateien `dp_opt.exe`, `dp_wopt.exe` und `dp_wopt.ini` in das DynamO-Programmverzeichnis
- Anpassen der Datei `dp_wopt.ini` im DynamO-Programmverzeichnis:

  Eintrag 1: <Name des DynamO-Programmverzeichnisses>`\dp_opt.exe`
  Eintrag 2: <Name des Verzeichnisses für temporäre Dateien[1]>`\_dp_opt_.dab`
  Eintrag 3: <Name des Verzeichnisses für temporäre Dateien>`\_dp_opt_.erg`
  Eintrag 4: <Name des Verzeichnisses für temporäre Dateien>`\_dp_wopt_.erg1`
  Eintrag 5: <Name des DynamO-Programmverzeichnisses>

  (alle Einträge sind in Hochkommas ’ .. ’ einzuschließen)
- Kopieren der Datei `DynamO.doc` in das DynamO-Dokumentverzeichnis
- Wenn DynamO von der Kommandozeile auch ohne vollständige Pfadangabe gestartet werden soll, muß der Name des DynamO-Programmverzeichnisses in den Suchpfad eingefügt werden (bei Windows NT über Systemverwaltung→System→Umgebung).

Damit ist DynamO vollständig installiert und kann durch Angabe von

<Name des DynamO-Programmverzeichnisses>`\dp_wopt.exe`

auf der Kommandozeile oder durch Doppelklick auf `dp_wopt.exe` im Windows Explorer gestartet werden. Komfortabler ist jedoch das Ablegen einer Verknüpfung zu `dp_wopt.exe` auf dem Desktop (Name sinnvollerweise `DynamO`).

[1] meistens `C:\Temp` oder `C:\Tmp`

C.3 Bekannte Probleme

In der Statuszeile von DynamO werden nicht die vorgesehenen Informationen (Cursorposition, etc.) sondern Teile der Hauptmenü-Einträge angezeigt !

Ursache dieses Verhaltens ist eine fehlerhafte Version der Librarydatei `comctl32.dll` (unter Windows NT 4.0 im Verzeichnis `C:\WINNT\system32` enthalten). Diese Datei wird u.a. bei der Installation bestimmter Service Packs und neuerer Versionen des Internet Explorers aktualisiert. Korrekt funktioniert die Version 4.72, als fehlerhaft bekannt ist z.Z. die Version 5.00. Verschiedene Versionen der Librarydatei können von

`http://www.microsoft.com/msdownload/ieplatform/ie/comctrlx86.asp`

heruntergeladen werden[1].

Das Syntax-Highlighting im Aufgabeneditor funktioniert nicht bei `/*` ... `*/`-Kommentaren !

Das Syntax-Highlighting für diese Kommentarart wurde nicht realisiert, da dadurch der Parse-Aufwand erheblich steigen würde, was zu starkem Flackern der Anzeige bei Eingabe von Text führen könnte.

Die Optimierung „bleibt stehen", d.h. die Arbeitsfläche „Optimierungslauf" wird nicht mehr aktualisiert und es werden keine neuen Versuche mehr durchgeführt, obwohl die Optimierung noch nicht beendet ist !

Dieser Fehler kann auftreten, wenn die Generierung und Bewertung von Versuchspunkten so schnell erfolgt (wegen geringem Rechenaufwand oder sehr schnellem Rechner), daß sich die Puffer der zur internen Kommunikation verwendeten Pipes vollständig füllen. Dadurch „verklemmen" die beteiligten Lese- und Schreibprozesse, wodurch die Optimierung zum Erliegen kommt.

Die „stehengebliebene" Optimierung kann über den Menüpunkt Optimierung→Stop abgebrochen werden; der Fehler wird jedoch wahrscheinlich bei der nächsten Optimierung derselben Aufgabe wieder auftreten.

Da das beschriebene Verhalten der Pipes nicht dem in den offiziellen Dokumenten zum Betriebssystem (WinSDK u.a.) beschriebenen entspricht, konnte bis jetzt noch keine Lösung für dieses Problem gefunden werden.

[1] Leider erlaubt die Lizenzpolitik von Microsoft nicht, die (korrekte) Librarydatei zusammen mit dem Optimierungssystem weiterzugeben.

Literaturverzeichnis

/AMBR-88/ Ambros-Ingerson, J. A.; Steel, S.: *Integrating Planning, Execution and Monitoring*. In: Proceedings of the Seventh National Conference on Artificial Intelligence, AAAI Press, 1988, S. 83-88.

/COTT-95/ Cotta, C.; Aldana, J.F.; Nebro, A.J.; Troya, J.M.: *Hybridizing Genetic Algorithms with Branch and Bound Techniques for the Resolution of the TSP*. In: Pearson, D.W.; Steele, N.C.; Albrecht, R.F. (Hrsg.): Artificial Neural Nets and Genetic Algorithms. Springer-Verlag, Wien, 1995, S. 277-280.

/DESA-96/ Desai, R.; Patil, R.: *SALO: Combining Simulated Annealing and Local Optimization for Efficient Global Optimization*. Proceedings of the 9th Florida AI Research Symposium, 1996, S. 233-237.

/EROL-95/ Erol, K.: *Hierarchical Task Network Planning: Formalization, Analysis and Implementation*. University of Maryland at College Park, 1995.

/HADE-94/ Hader, S.: *Regelbasierte Optimierung von Produktionssystemen*. In: Kampe, G.; Zeitz, M. (Hrsg.): 9. Symposium Simulationstechnik. Vieweg-Verlag, 1994, S. 603-606.

/HADE-95/ Hader, S.: *A Knowledge-Based Approach to the Optimization of Technical Systems and Its Application to Inventory Systems*. In: Bogataj, L. (Hrsg.): Inventory Modelling - Lecture Notes of the International Postgraduate Summer School. vol. 2, ISIR, Budapest,1995, S. 87-92.

/HADE-97/ Hader, S.: *MASCOT: A Multiagent System for Hybrid Optimization*. In: Papachristos, S.; Ganas, I. (Hrsg.): Proceedings of the Third ISIR Summer School on Inventory Modelling in Production and Supply Chains. ISIR, 1997, S. 155-165.

/HADE-98a/ Hader, S.: *Ein Multiagentensystem zur simulationsbasierten Optimierung*. In: Kleine Büning, H. (Hrsg.): Workshop SiWiS '98 - Simulation in Wissensbasierten Systemen. Universität-GH-Paderborn, Bericht tr-ri-98-194, Reihe Informatik, 1998.

/HADE-98b/ Hader, S.: *A Multiagent Simulation Optimization System*. In: Bargiela, A.; Kerckhoffs, E. (Hrsg.): Simulation Technology: Science and Art - 10th European Simulation Symposium 1998. Society for Computer Simulation International, 1998, S. 124-128.

/HADE-98c/ Hader, S.: *Das C++ Schnittstellen-Modul i_face*. Dokumentation, TU Chemnitz, 1998, http://www.tu-chemnitz.de/~hader/i_face.

/HADE-99a/ Hader, S.: *Ein Multiagentenansatz zur Realisierung selbstadaptierender hybrider Optimierungsverfahren*. In: Köchel, P. (Hrsg.): KI-Methoden in der simulationsbasierten Optimierung - 13. Workshop der ASIM-Arbeitsgruppe "Simulation und Künstliche Intelligenz". Chemnitzer Informatik-Bericht CSR-99-03, TU Chemnitz, 1999, S. 79-88.

/HADE-99b/ Hader, S.: *Hybride Optimierung als Werkzeug zur Lösung von KMU-relevanten Planungs- und Steuerungsproblemen*. In: Enderlein, H. (Hrsg.): Zukunftsweisende Unternehmens- und Fabrikkonzepte - 10. Tage des System- und Betriebsingenieurs. 1999, S. 202.

/HADE-00/ Hader, S.; Kobylka, A.; Kreißig, U.: *Praxisrelevante Optimierungsstrategien für die simulationsbasierte dynamische Dimensionierung von Produktionssystemen (DYNAMIS-P)*. In: Enderlein, H. (Hrsg.): IBF-Fachtagung "Vernetzt planen und produzieren". 2000, S. 89-93.

/KOBY-00/ Kobylka, A.: *Simulationsbasierte Dimensionierung von Produktionssystemen mit definiertem Potential an Leistungsflexibilität*. Dissertation, Fakultät für Maschinenbau und Verfahrenstechnik, TU Chemnitz, 2000.

/KÖCH-98/ Köchel, P.: *Optimierung ereignisdiskreter Systeme via Simulation. Teil 1*. Forschungsbericht des Innovationskollegs „Bildung eines vernetzten Logistik- und Simulationszentrums“, TU Chemnitz, 1998.

/KÖCH-00/ Köchel, P.: *Simulationsbasierte Optimierung – ein Zugang zur Behandlung von Planungs- und Steuerungsproblemen*. In: Enderlein, H. (Hrsg.): IBF-Fachtagung "Vernetzt planen und produzieren". 2000, S. 83-88.

/MAHF-93/ Mahfoud, S.W.; Goldberg, D.E.: *Parallel Recombinative Simulated Annealing: A Genetic Algorithm*. IlliGAL Report No. 93006, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois at Urbana-Champaign, 1993.

/MÜLL-93/ Müller, J. (Hrsg.): *Verteilte Künstliche Intelligenz - Methoden und Anwendungen*. BI Wissenschaftsverlag, Mannheim, 1993.

/SMIT-80/ Smith, R.G.: *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*. IEEE Transactions on Computers, 29(19980)12, S. 1104-1113.

/SYRJ-95/ Syrjakow, M.; Szczerbicka, H.: *Simulation and Optimization of Complex Technical Systems*. In: Ören, T.I.; Birta, L.G. (Hrsg.): Proceedings of the 1995 Summer Computer Simulation Conference. Society for Computer Simulation International, 1995, S. 86-95.