Das wissensbasierte Optimierungssystem DIM_EXPERTE

(Forschungsbericht des Projekts V, 1998)

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

Innovationskolleg INK 17/A1 - 1

Bildung eines vernetzten Logistik- und Simulationszentrums

Zusammenfassung

DIM_EXPERTE ist ein wissensbasiertes Optimierungssystem zur Lösung von Problemstellungen der *statischen Parameteroptimierung*. Es verwendet ein hybrides Optimierungsverfahren aus regelbasierter Optimierung und Gradientensuche. Besonderes Augenmerk wird in DIM_EXPERTE auf die Verwendung externer Analyseprogramme (speziell Simulationsprogramme) bei der Bewertung von Lösungsalternativen gelegt. Dadurch ist es insbesondere für technische Domänen geeignet.

Dieser Forschungsbericht gibt eine Übersicht über den Aufbau und die Verwendung von DIM_EXPERTE.

Inhaltsverzeichnis

1 Einleitung

Die *statische Parameteroptimierung* beschäftigt sich mit der Suche nach dem Optimalwert statischer Zielfunktionen bezüglich eines Satzes von Parametern. Statisch bedeutet hierbei, daß die Zielfunktion zeitinvariant ist, d.h. daß gleiche Parameterwerte unabhängig von der Zeit stets den gleichen Zielfunktionswert ergeben.

Aufgabenstellungen dieser Art treten beim Entwurf und der Umgestaltung technischer Systeme recht häufig auf. Bei der Komplexität praxisrelevanter Systeme liegen die zu optimierenden Zielfunktionen meist nicht als explizite Formeln vor, sondern algorithmisch in Form von Analyseprogrammen (z.B. FEM, Simulation). Diese Programme besitzen i.allg. eine längere Laufzeit (Minuten bis Stunden), so daß in der für die Gesamtoptimierung vorgesehenen begrenzten Zeit nur eine relativ kleine[1] Anzahl möglicher Systemvarianten („Szenarios“) untersucht werden kann.

Um trotz der begrenzten Zeit eine „akzeptable“ Lösung zu finden, muß das verwendete Optimierungsverfahren möglichst zielgerichtet arbeiten. Das bedeutet, daß möglichst viel Information über die zu lösende Aufgabe bei der Generierung neuer Systemvarianten verwendet wird. Um die notwendige Information bereitzustellen, existieren zwei Ansätze:

- Informationen werden vorgegeben
  Vorteil: können von Beginn der Optimierung an genutzt werden
  liegen in aggregierter Form vor
  Nachteil: sind meist unvollständig und unsicher („Heuristiken“)

  Bsp.: Informationen über die Gestalt des Problemraums wie Konvexität, Vorhandensein lokaler Minima usw. oder über die Beziehung Parameter → Zielfunktion

- Informationen werden während der Optimierung gesammelt
  Vorteil: sind (zumindest lokal) exakter als Heuristiken
  Nachteil: Informationssammlung kostet Zeit
  müssen vor der Verwendung geeignet aggregiert werden

  Bsp.: Gradientenverfahren
  „Erforschung“ eines lokalen Gebietes des Problemraums durch Schritte in verschiedene Richtungen und anschließende Schätzung der Gradiente

Um die Effizienz und Robustheit von Optimierungsverfahren zu erhöhen, sollte eine Kombination beider Ansätze der Informationsbereitstellung verwendet werden. Traditionelle Optimierungsverfahren verwenden überwiegend die Informationssammlung, während vorgegebene Informationen kaum genutzt werden können.

Die wissensbasierte Optimierung ist in der Lage, sowohl vorgegebene Informationen zu verwenden („statisches Wissen“) als auch Informationen zu sammeln („dynamisches Wissen“). Dadurch bietet sie die Möglichkeit, zielgerichtet und damit in kürzerer Zeit akzeptable Lösungen für Optimierungsaufgaben zu finden.

Im folgenden wird das wissensbasierte Optimierungssystem DIM_EXPERTE vorgestellt. Dabei wird besonderes Augenmerk auf die spezielle Form der Aufgaben- und Wissensbeschreibung und die simulationsbasierte Optimierung in DIM_EXPERTE gelegt.

[1] bezogen auf die Größe des Problemraums

Zum besseren Verständnis dieses Forschungsberichts sind grundlegende Kenntnisse auf den Gebieten *Optimierung, regelbasierte Wissensverarbeitung* und *PROLOG* (Variablenkonzept) von Vorteil.

Folgende stilistischen Konventionen gelten:

- so erscheint **`Quelltext`**
- so erscheinen *typisierte formale Ausdrücke*$_{Typ1,Typ2,..}$
  der Typ eines Ausdrucks folgt dem Ausdrucksnamen tiefgestellt, bei mehreren möglichen Typen als Kommaliste

  Typen:

Z	-	Zahl	ZA	-	arithm. Ausdruck
GZ	-	ganze Zahl	GZA	-	ganzzahliger arithm. Ausdruck
ZK	-	Zeichenkette	ZKA	-	Zeichenkettenausdruck
L	-	allgemeine Liste	LA	-	Listenausdruck
ZL	-	Zahlenliste	ZLA	-	Zahlenlistenausdruck
ZKL	-	Zeichenkettenliste	ZKLA	-	Zeichenkettenlistenausdruck
RL	-	Referenzliste			
BL	-	Bedingungsliste			
R	-	Referenz	RA	-	Referenzausdruck
UV	-	ungebundene Variable			
			A	-	allgemeiner Ausdruck

  Bsp.: *FaktName*$_{ZK}$ - repräsentiert eine Zeichenkette
  Wert$_{ZA,L}$ - repräsentiert einen arithmetischen Ausdruck oder eine Liste

- optionale Konstrukte werden in [..] eingeschlossen

2 Die Aufgabenbeschreibung

DIM_EXPERTE benötigt zur Lösung von Optimierungsaufgaben eine Reihe von grundsätzlichen Informationen, die in einer *Aufgabenbeschreibung* zusammengefaßt werden. Aufgabenbeschreibungen werden vom Nutzer als Textdateien angelegt und vor dem Beginn der Optimierung von DIM_EXPERTE geladen.

Da DIM_EXPERTE den Namen der Aufgabenbeschreibungsdatei verwendet, um Informationen zur jeweiligen Aufgabe abzulegen, muß jede neue bzw. geänderte Aufgabenbeschreibung unter einem neuen, eindeutigen Namen abgespeichert werden.

Die Elemente einer Aufgabenbeschreibung orientieren sich an der gebräuchlichen mathematischen Darstellung von Optimierungsaufgaben, z.B.

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

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

Neben diesen Elementen können in DIM_EXPERTE zusätzlich *benannte Konstanten* und *Formeln* definiert werden. Da die Zielfunktion (bzw. ihr zugrundeliegende Werte) durch ein externes Analyseprogramm repräsentiert wird, muß zusätzlich die Schnittstelle zu diesem Programm definiert werden.

2.1 Datenobjekte

Das wesentliche Konstrukt einer Aufgabenbeschreibung ist das *Datenobjekt*. Datenobjekte repräsentieren die für die Optimierung wesentlichen (Zahlen-)Objekte des betrachteten Systems. Sie bestehen aus einem oder mehreren skalaren *Datenelementen* (Zahlen). Datenobjekte werden beschrieben durch

- einen eindeutigen Namen[1]
- einen Zahlentyp (ganze Zahlen, reelle Zahlen)
- eine Struktur (skalares Objekt, Vektor, Matrix) und
- ihre Funktion:
  - + Parameter: die eigentlichen Optimierungsparameter
  - + Konstanten: wichtige symbolische Konstanten
  - + Abhängige: wichtige Formeln
  - + Kenngrößen: vom Analyseprogramm gelieferte „Leistungskenngrößen“ eines Systems

[1] als Datenobjektnamen können beliebige nichtleere Zeichenketten verwendet werden; Zeichenketten, die nur aus Buchstaben, Ziffern und dem Unterstrich ‘_’ bestehen und deren erstes Zeichen ein Buchstabe ist, brauchen nicht quotiert werden, alle anderen Zeichenketten sind zu quotieren z.B. **`dobj1, DObj1, 'hallo welt'`**, **`'123'`**, **`'#$1'`**

2.2 Arithmetische Ausdrücke

Wesentliche Teile eines Optimierungsproblems werden mittels *arithmetischer Ausdrücke* beschrieben (z.B. Zielfunktion, Nebenbedingungen). Arithmetische Ausdrücke in der Aufgabenbeschreibung können aus den folgenden Elementen bestehen:

(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) Datenobjekte: *DObjName*$_{ZK}$[**`[`***Index1*$_{GZA}$**`]`**[**`[`***Index2*$_{GZA}$**`]`**]]

Indizes[1] müssen angegeben werden, falls es sich bei dem Datenobjekt *DObjName* um einen Vektor oder eine Matrix handelt

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

(3) arithmetische Operatoren: **`+`** , **`-`** , **`*`** , **`/`** , **`div`**, **`mod`**, **`^`** (Potenz)

(4) arithmetische Funktionen:

`round(`*X*$_{ZA}$**`)`**	Runden von *X* auf die nächste ganze Zahl
`trunc(`*X*$_{ZA}$**`)`**	Abrunden von *X* auf die nächst kleinere ganze Zahl
`abs(`*X*$_{ZA}$**`)`**	Betrag von *X*
`exp(`*X*$_{ZA}$**`)`**	Exponentialfunktion e^X
`log(`*X*$_{ZA}$**`)`**	natürlicher Logarithmus ln *X* , $X > 0$
`sqrt(`*X*$_{ZA}$**`)`**	Quadratwurzel von *X* , $X \geq 0$
`sin(`*X*$_{ZA}$**`)`**	Sinus von *X*
`cos(`*X*$_{ZA}$**`)`**	Cosinus von *X*

`dim1(`*DObjName*$_{ZK}$**`)`**	1. Dimension von *DObjName* (liefert 0, falls Skalar)
`dim2(`*DObjName*$_{ZK}$**`)`**	2. Dimension von *DObjName* (liefert 0, falls Skalar bzw. Vektor)

`min(`*X*$_{ZA}$**`,`***Y*$_{ZA}$**`)`**	kleinere der Zahlen *X* und *Y*
`max(`*X*$_{ZA}$**`,`***Y*$_{ZA}$**`)`**	größere der Zahlen *X* und *Y*

(5) arithmetische Folgen:

`sum(`*HVar*$_{ZK}$**`=`***UG*$_{GZA}$**`..`***OG*$_{GZA}$**`,`***X*$_{ZA}$**`)`**

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

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

[1] die Indizierung beginnt mit 1

prod(*HVar*$_{ZK}$**=***UG*$_{GZA}$**..***OG*$_{GZA}$**,***X*$_{ZA}$**)**

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, in dem *HVar* auftritt

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

(7) Klammerung: **(** .. **)**

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

2.3 Aufgabenbeschreibungsdateien

Aufgabenbeschreibungsdateien[1] bestehen aus einer Reihe von Textblöcken der allgemeinen Form

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

Dabei spezifiziert <blockname> die Funktion des jeweiligen Blockes. Sowohl die möglichen Werte von <blockname> als auch die Reihenfolge der entsprechenden Blöcke in der Beschreibungsdatei sind fest vorgegeben.

Kommentare können an beliebiger Stelle innerhalb der Beschreibungsdatei stehen. Einzeilenkommentare werden mittels **//** ... realisiert, Mehrzeilenkommentare mittels **/*** ... ***/**.

[1] Namenskonvention: Extension **.dab**, z.B. **ffs_1.dab**

Die Block-Reihenfolge ([..] - optionale Blöcke)

`datenobjekte`-Block
[`konstanten`-Block]
`parameter`-Block
[`abhaengige`-Block]
`zielfunktion`-Block
[`nebenbedingungen`-Block]
`programm`-Block

Der Definitionsblock `datenobjekte`

In diesem Block müssen alle Datenobjekte definiert werden[1], die im Rahmen der Optimierung verwendet werden. Dabei wird für jedes Datenobjekt der Zahlentyp[2] und die Struktur angegeben (Notation wie in C):

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

Dieser Block muß der erste Block der Beschreibungsdatei sein. Er ist obligatorisch.

Der Konstantenblock `konstanten`

In diesem Block können ganze Datenobjekte oder Datenelemente daraus als Konstanten definiert werden. Die Konstantenwerte werden als einzelne Zahlen oder als Zahlenlisten angegeben.

Bsp.:
```
konstanten
{
  dobj1 = -7;
  dobj2[1..3] = { 100.7, -3.1, 1.234e-3 };
  dobj3 = { {11,12}, {21,22}, {31,32}, {41,42} };
}
```

Dieser Block kann weggelassen werden. Ist er vorhanden, muß er unmittelbar dem Datenobjektblock folgen.

---

[1] Maximalzahl definierbarer Datenobjekte z.Z. 500
jedes Datenobjekt darf aus maximal 1000 Datenelementen bestehen

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

Der Parameterblock `parameter`

In diesem Block werden ganze Datenobjekte oder Datenelemente daraus als Optimierungsparameter definiert[1]. Parameter werden beschrieben durch ihren Wertebereich und ggf. einen Startwert.

Bsp.:
```
parameter
{
  dobj1 = wb(1..100), sw(1);
  dobj2[4..5] = wb(0..1), sw(0.5);
  dobj3[1][2] = wb(-1000..10);
}
```

Dieser Block muß dem Konstantenblock folgen. Er ist obligatorisch.

Der Abhängigenblock `abhaengige`

In diesem Block können für Datenobjekte oder Datenelemente daraus explizite Berechnungsformeln in Form von arithmetischen Ausdrücken angegeben werden[2]. Wenn im Laufe der Optimierung der Wert von Abhängigen benötigt wird, dann wird dieser anhand der entsprechenden Formel berechnet.

Bsp.:
```
abhaengige
{
  dobj1 = 3*dobj3[1][1] + 2;
  dobj2[3] = dobj1 / 4.3;
}
```

Dieser Block kann weggelassen werden. Ist er vorhanden, muß er unmittelbar dem Parameterblock folgen.

Der Zielfunktionsblock `zielfunktion`

In diesem Block wird die zu optimierende Zielfunktion (arithmetischer Ausdruck) und die Optimierungsart { **`maximum`**, **`minimum`** } spezifiziert.

Bsp.:
```
zielfunktion
{
  ( dobj1 - sum(i=1..dim1(dobj2),dobj2[i]) ) -> minimum;
}
```

Dieser Block muß dem Abhängigenblock folgen. Er ist obligatorisch.

[1] Maximalzahl definierbarer (skalarer) Parameter z.Z. 200

[2] Maximalzahl definierbarer (skalarer) Abhängiger z.Z. 200

Der Nebenbedingungsblock `nebenbedingungen`

In diesem Block können Nebenbedingungen angegeben werden, die eine gültige Lösung der Optimierungsaufgabe erfüllen muß[1]. Nebenbedingungen bestehen aus zwei arithmetischen Ausdrücken, die durch einen Vergleichsoperator { **<**, **<=**, **==**, **!=**, **>=**, **>** }miteinander verknüpft sind.

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

Dieser Block kann weggelassen werden. Ist er vorhanden, muß er unmittelbar dem Zielfunktionsblock folgen.

Der Programmblock `programm`

In diesem Block wird die Schnittstelle zum externen Analyseprogramm definiert (siehe dazu auch Abschnitt 3). Die Definition beinhaltet Informationen zum Namen des Programms, zu Programmoptionen, der Art des dem Analyseprogramm zugrundeliegenden Modells {**deterministisch** oder **stochastisch**}, dem Typ der Dateischnittstelle (z.Z. nur **i_face**[2]) sowie den Eingabegrößen und Ausgabegrößen („Kenngrößen“) des Programms.

Bsp.:
```
programm
{
  name = '/usr/bin/analyse1';
  options = '-s -d test1.in';
  art = stochastisch;
  format = i_face;
  eingabe = { SIM_ANZ3, dobj1, dobj2 };
  ausgabe = { dobj3 };
}
```

Dieser Block muß dem Nebenbedingungsblock folgen. Er ist obligatorisch.

---

[1] Maximalzahl Nebenbedingungen z.Z. 100

[2] **i_face** kennzeichnet eine einfache Datenschnittstelle, bei der der Datenaustausch mittels einfacher ASCII-Textdateien erfolgt, die zeilenorientiert aufgebaut sind; Programmbibliotheken und Beschreibungen für diese Schnittstelle sind im Internet unter `http://www.tu-chemnitz.de/~hader/i_face.html` zugänglich.

[3] zur Bedeutung des speziellen Datenobjekts **SIM_ANZ** siehe Abschnitt 4.3.4

3 Die Ankopplung externer Analyseprogramme

Die Bewertung von Lösungsalternativen während der Optimierung erfolgt in DIM_EXPERTE mittels externer Analyseprogramme. Dabei handelt es sich um Programme, denen eine (parametrische) Beschreibung der zu bewertenden Systemvariante übergeben wird und die eine oder mehrere zugehörige Kenngrößen liefern[1]. Die für den Datenaustausch notwendige Schnittstelle muß vom Nutzer in der verwendeten Aufgabenbeschreibungsdatei definiert werden. Im folgenden wird der Datenaustausch zwischen DIM_EXPERTE und den Analyseprogrammen beschrieben und auf notwendige Eigenschaften dieser Programme eingegangen.

3.1 Datenaustausch DIM_EXPERTE ↔ Analyseprogramm

Der Datenaustausch zwischen DIM_EXPERTE und dem der aktuellen Aufgabe zugeordneten Analyseprogramm erfolgt jeweils dann, wenn während der Optimierung eine vollständig spezifizierte Systemvariante bewertet werden soll. Der Datenaustausch erfolgt in fünf Schritten:

(1) DIM_EXPERTE erzeugt im Programmverzeichnis des Analyseprogramms eine Textdatei (Eingabedatei), deren Name sich aus dem Namen des Analyseprogramms und der Extension **`.in`** zusammensetzt. In diese Datei werden entsprechend dem gewählten Schnittstellenformat (z.Z. nur **`i_face`**) die aktuellen Werte der als Eingabegrößen definierten Datenobjekte eingetragen.

(2) DIM_EXPERTE startet das Analyseprogramm als eigenständigen Systemprozeß und wartet auf die Beendigung dieses Prozesses.

(3) Das Analyseprogramm liest den Inhalt der Eingabedatei, ermittelt für die entsprechende Systemvariante eine Reihe von Kenngrößen (z.B. Durchsatz, Durchlaufzeit, Kosten) und erzeugt im eigenen Programmverzeichnis eine Textdatei (Ausgabedatei), deren Name sich aus seinem eigenen Namen und der Extension **`.out`** zusammensetzt. In diese Datei werden die ermittelten Kenngrößen eingetragen.

(4) Das Analyseprogramm beendet seine Arbeit, wobei im Exitstatus Informationen über den Erfolg der Programmausführung verschlüsselt werden.

(5) DIM_EXPERTE wertet den Exitstatus des Analyseprogramms aus. Folgende Werte werden unterschieden:

0 - erfolgreiche Ausführung
1 - erfolgreiche Ausführung + Ctrl-C-Signal[2] empfangen
sonst - Fehler bei der Ausführung

Liegt der Exitstatus in {0,1}, dann liest DIM_EXPERTE den Inhalt der Ausgabedatei entsprechend dem gewählten Schnittstellenformat und weist die gelesenen Werte den als Ausgabegrößen definierten Datenobjekten zu.

---

[1] Die Bestimmung der Kenngrößen erfolgt je nach Art des im Analyseprogramm verwendeten Modells durch Berechnung, Simulation, Datenbankabfrage, etc.

[2] Mittels Ctrl-C kann der Nutzer eine laufende Optimierung vorzeitig beenden. Als Ergebnis wird dabei die beste bisher gefundene Systemvariante geliefert.

Nachdem die Kenngrößen der Systemvariante gelesen wurden, kann DIM_EXPERTE deren Zielfunktionswert bestimmen und die Einhaltung von Nebenbedingungen testen.

3.2 Notwendige Eigenschaften von Analyseprogrammen

Analyseprogramme müssen eine Reihe von Eigenschaften besitzen, damit sie mit DIM_EXPERTE gekoppelt werden können. Dazu gehören:

- Das Programm muß im Batch-Modus ausführbar sein, d.h. es darf weder manuelle Nutzereingaben erwarten noch Ausgaben auf dem Bildschirm durchführen. Der Datenaustausch erfolgt ausschließlich über die Eingabe- und die Ausgabedatei.

- Der Nutzer muß Ausführungsrechte für das Programm sowie Lese- und Schreibrechte für das Programmverzeichnis besitzen.

- Das Programm muß Ctrl-C-Signale, die im Normalfall zum Programmabbruch führen, abfangen können. Das Auftreten eines solchen Signals wird im vom Programm gelieferten Exitstatus vermerkt.

- Das Programm muß Dateien entsprechend dem verwendeten Schnittstellenformat[1] lesen und schreiben können.

[1] Programmbibliotheken und Beschreibungen für das einzige bisher erlaubte Schnittstellenformat `i_face` sind im Internet unter `http://www.tu-chemnitz.de/~hader/i_face.html` zugänglich.

4 Die Optimierung

Optimierungsverfahren basieren meist auf der sukzessiven Synthese bzw. Modifikation von *Szenarios*. Szenarios repräsentieren Varianten des zu optimierenden Systems (Lösungsalternativen) und werden durch folgende Informationen beschrieben:

- konkrete Werte für alle Optimierungsparameter des Systems (→ Systemvariante)
- ermittelte Leistungskenngrößen der Systemvariante
- ermittelter Zielfunktionswert der Systemvariante („Güte“).

Ein Szenario ist *gültig*, wenn es sämtliche vorgegebenen Nebenbedingungen erfüllt.

4.1 Simulation und Optimierung

Wenn die zu optimierende Zielfunktion f($\mathbf{X}$) stochastischen Einflüssen unterworfen ist[1], dann handelt es sich bei jeder Bestimmung des Zielfunktionswertes für einen konkreten Parametervektor $\mathbf{X}_i$ um die *Realisierung* einer (unbekannten) *Zufallsvariable* f($\mathbf{X}_i$). Um statistisch aussagekräftige Ergebnisse zu erhalten, wird die Wertbestimmung N mal wiederholt und eine Schätzung $\overline{x}_i$ für den Erwartungswert E(f($\mathbf{X}_i$)) sowie eine Schätzung s_i^2 für die Varianz $D^2X(f(\mathbf{X}_i))$ des Zielfunktionswertes erstellt.

Im folgenden gehen wir davon aus, daß die Realisierungen der Zufallsvariable f($\mathbf{X}_i$) unabhängig und normalverteilt sind, d.h. es gilt

$$f(\mathbf{X}_i) \sim N(\mu_i,\sigma_i^2) \text{ mit } \mu_i = E(f(\mathbf{X}_i)) \text{ und } \sigma_i^2 = D^2X(f(\mathbf{X}_i)) .$$

Das zweiseitige *Konfidenzintervall*[2] zum Niveau 1 - α für den Erwartungswert μ_i ergibt sich dann als

$$[\mu_{i,u},\mu_{i,o}] \equiv \left[\overline{x}_i - \frac{s_i}{\sqrt{N}} t_{N-1;1-\alpha/2} \, , \, \overline{x}_i + \frac{s_i}{\sqrt{N}} t_{N-1;1-\alpha/2} \right]$$

mit $t_{N-1;1-\alpha/2}$ - t-Verteilung mit N-1 Freiheitsgraden

Bsp.: N = 6, $\overline{x}_i = 12.7$, $s_i^2 = 9.3$, $\alpha = 0.05$

$$\rightarrow [\mu_{i,u},\mu_{i,o}] \equiv [9.194, 16.206] \rightarrow P(9.194 \le \mu_i \le 16.206) = 0.95$$

Bei der Optimierung interessiert weniger die Frage nach dem genauen Zielfunktionswert μ_i eines Parametervektors $\mathbf{X}_i$, sondern die, ob ein Parametervektor $\mathbf{X}_i$ „besser“ oder „schlechter“ ist als ein anderer Parametervektor $\mathbf{X}_j$ (Variantenvergleich). Mit Hilfe eines zweiseitigen Konfidenzintervalls der Differenz $\delta_{ij} = \mu_i - \mu_j$ der Erwartungswerte kann diese Frage (evtl.) beantwortet werden (siehe hierzu auch /BANK-95/). Wir gehen im folgenden davon aus, daß die N_i Realisierungen von f($\mathbf{X}_i$) unabhängig von den N_j Realisierungen von f($\mathbf{X}_j$) sind.

---

[1] z.B. bei Realisierung des Systemmodells durch ein stochastisches Simulationsprogramm

[2] ein Konfidenzintervall zum Niveau 1 - α gibt ein Intervall $[\mu_u,\mu_o]$ an, das den wahren, unbekannten Parameter μ einer Verteilung mit Wahrscheinlichkeit 1 - α überdeckt (α meist 0.01, 0.05 oder 0.1)

Fall 1: $N_i = N_j = N$

$$[\delta_{ij,u},\delta_{ij,o}] \equiv \left[\overline{x}_i - \overline{x}_j - \sqrt{\frac{s_i^2 + s_j^2}{N}}\, t_{\nu,1-\alpha/2} \;,\; \overline{x}_i - \overline{x}_j + \sqrt{\frac{s_i^2 + s_j^2}{N}}\, t_{\nu,1-\alpha/2} \right]$$

mit $\nu = 2N-2$

Fall 2: $N_i \neq N_j$ (Empfehlung: $N_i, N_j \geq 6$)

$$[\delta_{ij,u},\delta_{ij,o}] \equiv \left[\overline{x}_i - \overline{x}_j - \sqrt{\frac{s_i^2}{N_i} + \frac{s_j^2}{N_j}}\, t_{\nu,1-\alpha/2} \;,\; \overline{x}_i - \overline{x}_j + \sqrt{\frac{s_i^2}{N_i} + \frac{s_j^2}{N_j}}\, t_{\nu,1-\alpha/2} \right]$$

$$\text{mit } \nu = \frac{\left(\frac{s_i^2}{N_i} + \frac{s_j^2}{N_j} \right)^2}{\frac{\left(s_i^2/N_i\right)^2}{N_i - 1} + \frac{\left(s_j^2/N_j\right)^2}{N_j - 1}}$$

Anhand des ermittelten Konfidenzintervalls $[\delta_{ij,u},\delta_{ij,o}]$ können folgende Schlüsse gezogen werden[1]:

- wenn $\delta_{ij,o} < 0$ ist, dann kann Variante $\mathbf{X}_i$ als „schlechter“ („besser“) als Variante $\mathbf{X}_j$ angesehen werden
- wenn $\delta_{ij,u} > 0$ ist, dann kann Variante $\mathbf{X}_i$ als „besser“ („schlechter“) als Variante $\mathbf{X}_j$ angesehen werden
- ansonsten kann anhand der vorliegenden Daten keine der beiden Aussagen getroffen werden

Tritt der zuletzt genannte Fall auf, dann können die beiden Varianten $\mathbf{X}_i$ und $\mathbf{X}_j$ entweder als „ununterscheidbar“ angesehen werden oder man versucht, durch Erhöhung der Stichprobenumfänge N_i und N_j das Konfidenzintervall soweit zu verkleinern, bis eine besser/schlechter-Aussage möglich wird.

4.2 Wissensbasierte Optimierung

Die *wissensbasierte Optimierung* beschäftigt sich mit der Lösung von Optimierungsproblemen unter Verwendung explizit repräsentierten Fachwissens (aus einer *Wissensbasis*). Dabei wird meist versucht, das Verhalten menschlicher Experten bei der Optimierung nachzubilden.

Menschliche Experten verwenden bei der Optimierung zumeist einen „manuell-iterativen“ Ansatz, bei dem ein Startszenario in einer Folge von Trial-and-Error-Schritten sukzessive bis zum Erreichen eines „optimalen“ Szenarios verbessert wird. Neue Szenarios werden durch

[1] die Aussagen beziehen sich auf den Fall „Maximierung“ („Minimierung“ jeweils in Klammern)

Modifizierung bereits getesteter Szenarios[1] generiert (sequentielle Optimierung), wobei die Art der Modifizierung anhand von Heuristiken festgelegt wird.

Um diesen Ansatz nachbilden zu können, muß die Wissensbasis zwei Arten von fachspezifischem Wissen enthalten:

- **Analysewissen**
  Dieses Wissen wird verwendet, um Szenarios zu bewerten und vorkommende Schwachstellen, Engpässe bzw. Leistungsreserven zu identifizieren.
- **Transformationswissen**
  Dieses Wissen wird verwendet, um Szenarios so zu ändern, daß die erkannten Schwachstellen nicht mehr auftreten. Es handelt sich dabei vor allem um Wissen darüber, wie durch Änderung von Parameterwerten die Güte von Szenarios beeinflußt werden kann.

Wird das verwendete Fachwissen in Form von *WENN-DANN-Regeln* repräsentiert, dann spricht man von *regelbasierter Optimierung*. Regeln werden i.allg. als adäquates Mittel zur Darstellung (speziell technischen) Fachwissens angesehen. Sie bilden abgeschlossene Wissenseinheiten, die (relativ) unabhängig voneinander erzeugt und verarbeitet werden können.

WENN-DANN-Regeln besitzen die folgende allgemeine Struktur:

WENN <Bedingung> **DANN** <Aktion>

Der Bedingungsteil beschreibt die Menge der „Situationen“, in denen die Regel anwendbar ist. Er besteht aus einer Reihe konjunktiv verknüpfter logischer Ausdrücke.

Der Aktionsteil beschreibt die Aktionen/Modifikationen, die bei Anwendung der Regel auszuführen sind und zu einer neuen Situation führen. Er besteht aus einer Reihe von Aktionsanweisungen.

Die Verarbeitung der Regeln (*Inferenz*) erfolgt durch einen *Regelinterpreter*. Verschiedene Inferenzstrategien sind möglich, im folgenden soll stellvertretend die häufig verwendete sog. *Tiefe-Zuerst-Suche* illustriert werden.

[1] meist wird das bisher beste gefundene Szenario verwendet

Start: Initialisierung der Ausgangssituation S_0 , i := 0

Zyklus i: (1) Bestimmen aller in der aktuellen Situation S_i anwendbaren Regeln

wenn (mindestens eine anwendbare Regel existiert) dann

(2) Ordnen der anwendbaren Regeln bzgl. ihrer „Erfolgsaussichten“ (unter Verwendung interner Heuristiken)

(3) Ausführen der „erfolgversprechendsten“, noch nicht getesteten Regel j
→ neue Situation $S_{i+1,j}$

wenn ($S_{i+1,j}$ = Zielsituation S_Z) dann **Ende** (ERFOLG)
sonst $S_{i+1} := S_{i+1,j}$, i := i+1

sonst wenn (Backtracking möglich[1]) dann Backtracking[2]
sonst **Ende** (MISSERFOLG)

Schema der Tiefe-Zuerst-Suche

Bei der Verwendung eines Regelinterpreters zur Optimierung sind eine Reihe von Besonderheiten zu beachten (stark vereinfachte Darstellung):

- eine Situation S_i wird durch das beste bisher bekannte Szenario repräsentiert, der Ausgangssituation S_0 entspricht dabei meist das vom Nutzer vorgegebene Startszenario
- am Ende jedes Zyklus erfolgt ein zusätzlicher Test, ob die neue Situation S_{i+1} „besser“ als die vorige Situation S_i ist (d.h. ob sich ein verbessertes Szenario ergeben hat); ist das nicht der Fall, dann kommt es zum Backtracking
- da die Zielsituation S_Z i.allg. nicht apriori bekannt ist, müssen andere Abbruchbedingungen formuliert werden (z.B. Anzahl erfolgloser Szenariotests)

4.3 Realisierung in DIM_EXPERTE

DIM_EXPERTE basiert auf dem im Abschnitt 4.2 beschriebenen regelbasierten Optimierungsansatz, verwendet jedoch eine Reihe von Erweiterungen. Die wesentlichen Erweiterungen sollen in den folgenden Abschnitten erläutert werden.

4.3.1 Verwendung von Variablen in Regeln

Um die Ausdrucksmöglichkeiten von Regelbedingungen zu erweitern, können Variablen eingesetzt werden. Diese erlauben es, eine ganze Klasse von Situationen in einem Bedingungsteil zu beschreiben und so ggf. auch auf apriori unbekannte Situationen (bekannten Typs) zu reagieren.

---

[1] Backtracking ist möglich, wenn in mindestens einem der durchlaufenen Zyklen eine anwendbare, noch nicht getestete Regel existiert

[2] Zyklus k sei der zuletzt durchlaufende Zyklus, der eine anwendbare, noch nicht getestete Regel enthält
→ $S_k := S_i$, i := k , gehe zu Schritt 3 von Zyklus k

Bsp.: „wenn die Auslastung einer Maschine größer als 0.95 ist, dann ..."

ohne Variablen:

```
WENN  masch_auslastung[masch1] > 0.95  DANN ...
WENN  masch_auslastung[masch2] > 0.95  DANN ...
...
```

mit Variablen:

```
WENN  masch_auslastung[MNr] > 0.95  DANN ...
```

Variablen ermöglichen es auch, Werte temporär zwischenzuspeichern und vom Bedingungsteil an den Aktionsteil einer Regel zu übergeben.

Variablen sind nicht typisiert und brauchen nicht explizit definiert zu werden. Ihr Gültigkeitsbereich beginnt mit dem ersten Auftreten in der Regel und reicht bis zum Ende der Regel. Variablen werden bei ihrem ersten Auftreten an einen Wert *gebunden*, danach verhalten sie sich wie Konstanten (eine zweite Wertbindung ist nicht möglich).

Variablennamen sind Zeichenketten, die aus Buchstaben, Ziffern und dem Unterstrich '_' bestehen können, wobei das erste Zeichen ein Großbuchstabe oder Unterstrich sein muß[1].

Die spezielle Variable _ (*anonyme Variable*) kann an den Stellen verwendet werden, an denen eine ungebundene Variable erwartet wird, der zu bindende Wert jedoch nicht von Interesse ist. Zwei Vorkommen der anonymen Variable repräsentieren stets zwei unterschiedliche Variablen.

4.3.2 Repräsentation und Verarbeitung von qualitativem Fachwissen

Fachleute können meist Auskunft darüber geben, wie sich Größen in einem System beeinflussen und wie groß diese Einflußnahme ungefähr ist („gering", „groß", ..), das Nennen von allgemeingültigen Zahlenwerten ist jedoch kaum möglich. Um den Prozeß der Wissensakquisition zu erleichtern, sollte diese Art qualitativen Fachwissens repräsentiert und verarbeitet werden können. Qualitatives Fachwissen erlaubt auch den inkrementellen Aufbau von Wissensbasen, indem zunächst nur allgemeine qualitative Aussagen repräsentiert und diese dann bei wachsender Einsicht in das System sukzessive verfeinert werden.

Die Nutzung qualitativen Fachwissens beschränkt sich in DIM_EXPERTE z.Z. auf die Verwendung *qualitativer Parametermodifikatoren*. Diese ermöglichen es, Änderungen von Optimierungsparametern qualitativ, also ohne Angabe konkreter Zahlenwerte, zu beschreiben. Es existieren folgende Arten qualitativer Parametermodifikatoren:

- Vergrößerung/Verkleinerung eines Parameters um einen qualitativen Wert (z.B. **`wenig`**, **`mittel`**, **`viel`**, **`klein`**, **`gross`**)
- Vergrößerung/Verkleinerung eines Parameters ohne Angabe eines qualitativen Wertes (Schrittweite nicht bekannt)

[1] Variablennamen der Form **`VAR_*`** dürfen nicht verwendet werden, da diese für interne Zwecke reserviert sind.

- Änderung eines Parameter um einen qualitativen Wert
  (Richtung der Änderung nicht bekannt)
- Änderung eines Parameters ohne Angabe eines Wertes
  (Richtung der Änderung und Schrittweite nicht bekannt)

Durch Anwendung einer oder mehrerer dieser Modifikatoren auf ein konkretes Szenario entsteht ein „qualitatives" Szenario, das eine ganze Klasse von Szenarios beschreibt (z.B. „alle Szenarios mit Parameter x > 13"). Um aus dieser Klasse ein möglichst gutes Szenario auszuwählen, wird in DIM_EXPERTE ein numerisches Suchverfahren verwendet. Dieses bestimmt zuerst anhand der qualitativen Richtungs- und Schrittweitenangaben und ggf. zusätzlicher Experimente mit dem System eine konkrete Suchrichtung sowie Anfangs- und Endschrittweiten. Danach erfolgt eine Suche entlang der ermittelten Richtung. Das beste gefundene Szenario dieser Suche wird als Repräsentant seiner Klasse ausgewählt und die regelbasierte Optimierung fortgesetzt.

Ausgehend vom angegebenen qualitativen Änderungswert (**`wenig`**, **`viel`**, etc.) werden die für das Suchverfahren benötigten quantitativen Schrittweiten bestimmt. Dabei handelt es sich jeweils um Zahlentripel der Form

[<AnfSW>, <MaxSW>, <MinSW>]

wobei <AnfSW> die Anfangsschrittweite, <MaxSW> die maximale Schrittweite und <MinSW> die minimale Schrittweite bezeichnet[1]. Alle drei Zahlen müssen im Bereich (0,1] liegen; zusätzlich muß die Bedingung <MaxSW> ≥ <AnfSW> ≥ <MinSW> erfüllt sein. Alle Angaben sind als Bruchteile vom Gesamtwertebereich des jeweiligen Parameters zu verstehen.

Bsp.: Parameter **`x`** mit Wertebereich [-20,30]
Änderungswert **`mittel`** → [0.1,0.2,0.05]

d.h. Anfangsschrittweite 10% ≡ 5
maximale Schrittweite 20% ≡ 10
minimale Schrittweite 5% ≡ 2.5

Ausgehend vom Änderungswert und dem Parameternamen wird nach dem zu verwendenden Zahlentripel folgendermaßen gesucht (Erläuterung erfolgt anhand von **`mittel`** und **`x`**):

- Wert von Attribut **`mittel`** des Fakts **`x`**
- Wert von Attribut **`standard`** des Fakts **`x`**
- Wert von Attribut **`standard`** des Fakts **`default_s`** (systemdefiniert)
- programminterne Defaultwerte (systemdefiniert)

Nähere Angaben zur Definition und Modifikation von Fakten durch den Nutzer erfolgen in Abschnitt 4.4.1.

[1] das numerische Optimierungsverfahren kann die Schrittweiten in den vorgegebenen Grenzen dynamisch ändern („erfolgsorientiert")

4.3.3 Szenarioverwaltung

In DIM_EXPERTE wird zwischen *bewerteten* und *unbewerteten* Szenarios unterschieden:

Bewertete Szenarios repräsentieren vollständige Systemvarianten, bei denen alle Parameter feste Zahlenwerte besitzen und die Güte der Variante bereits bestimmt wurde.
Sämtliche während einer Optimierung bewerteten Szenarios werden gespeichert (in einer internen Liste namens **`all_szenario`**) und sind über Referenzen adressierbar. Somit werden Mehrfachbewertungen verhindert, die einen erheblichen zusätzlichen Aufwand bedeuten würden.

Unbewertete Szenarios repräsentieren Zwischenstufen zu vollständigen Systemvarianten, bei denen noch nicht alle Parameter feste Zahlenwerte besitzen und deshalb auch noch keine Güte bestimmt werden konnte. Sie entstehen durch Kopieren von bewerteten Szenarios und anschließender Modifikation von Parametern.
Zu jedem Zeitpunkt der Optimierung existiert genau ein unbewertetes Szenario (auch „aktuelles Szenario“ genannt). Alle durchgeführten Parametermodifikationen wirken ausschließlich auf dieses Szenario.

Während einer Optimierung kann der Fall eintreten, daß mehrere „beste“ Szenarios gefunden werden, die „gleichwertig“ sind. Gleichwertig sind gültige Szenarios, die bei deterministischer Zielfunktion denselben Zielfunktionswert besitzen bzw. bei stochastischer Zielfunktion „ununterscheidbar“ sind (siehe Abschnitt 4.1).

DIM_EXPERTE verwaltet alle gleichwertig besten Szenarios der aktuellen Optimierung in der internen Liste **`opt_szenario`**. Manipulieren die verwendeten Optimierungsregeln stets das bisher beste Szenario (*lokale* Strategie), dann werden sukzessive alle Elemente dieser Liste probiert, bis ein neues bestes Szenario gefunden wird. Die Kapazität der Liste ist beschränkt (vom Nutzer einstellbar, Standardwert 3).

Alle gleichwertig besten Szenarios des letzten Optimierungsschrittes werden in der internen Liste **`last_szenario`** gespeichert.

4.3.4 Simulationsbasierte Optimierung

Damit die in Abschnitt 4.1 beschriebenen Konfidenzintervalle berechnet werden können, benötigt jedes gültige bewertete Szenario Informationen über die Varianzen der zugehörigen Kenngrößen und über die zur Ermittlung verwendete Stichprobenanzahl N. Um diese Informationen verwalten zu können, wurde folgendes Konzept realisiert:

DIM_EXPERTE definiert ein skalares Datenobjekt **`SIM_ANZ`** vom Typ **`int`**, daß bei Optimierung stochastischer Zielfunktionen die Stichprobenanzahl N enthält. Der Nutzer kann dieses Datenobjekt als Konstante mit einem Wert größer 0 definieren und somit die zu verwendende Stichprobenanzahl festlegen. Standardmäßig wird der Wert 1 angenommen.

DIM_EXPERTE geht davon aus, daß zu jedem definierten Kenngrößen-Datenobjekt <kenngroesse> ein nutzerdefiniertes Datenobjekt <kenngroesse>**`_d2x`** gleicher Struktur existiert, daß die Varianz dieser Kenngröße enthält. Wird dieses Datenobjekt vom Nutzer nicht definiert, wird für die jeweilige Kenngröße eine Varianz von 0 angenommen.

Für die Aktualisierung der Liste der gleichwertig besten Szenarios (siehe vorigen Abschnitt) wurde ein spezielles zweistufiges Verfahren implementiert.

(1) Aus der Menge der (bisher) gleichwertig besten Szenarios und einer Menge neu bewerteter Szenarios wird durch Bildung paarweiser Konfidenzintervalle eine neue Menge gleichwertig bester Szenarios erzeugt.

(2) Da die Liste der gleichwertig besten Szenarios nur eine beschränkte Anzahl Einträge erlaubt (Standardwert: 3), muß die neu erzeugte Menge evtl. verkleinert werden. In diesem Fall werden die Szenarios mit den „schlechtesten“ Erwartungswertschätzungen $\overline{x}_i$ aus der Menge entfernt, bis die Menge in die Liste paßt.

4.4 Wissensrepräsentation in DIM_EXPERTE

Die für die regelbasierte Optimierung notwendigen Fakten und Regeln werden in nutzerlesbarer Form in Textdateien[1] (*Regeldateien*) gespeichert. Je nach zu lösendem Problem werden eine oder mehrere Regeldateien vom Nutzer ausgewählt und in DIM_EXPERTE eingelesen.

Kommentare können an beliebiger Stelle innerhalb von Regeldateien auftreten. Einzeilenkommentare werden mittels **//** ... realisiert, Mehrzeilenkommentare mittels **/*** ... ***/**.

4.4.1 Fakten

Fakten sind Zusammenfassungen von Attribut-Wert-Paaren, vergleichbar mit den **struct**-Konstrukten in C. Während der Optimierung können Fakten sowohl modifiziert als auch erzeugt und gelöscht werden („dynamisches Wissen“).

Fakten sind folgendermaßen aufgebaut:

FAKT(*FaktName*$_{ZK}$ **):**
{ [*AttrName*$_{ZK}$ **:=** *AttrWert*$_{Z,ZK,L}$ [**,** ..]] **}**
[**CF** *CertaintyFactor*$_{Z}$] **;**

Faktenname:

Fakten besitzen einen eigenen Namensraum, in dem jeder Fakt einen eindeutigen Namen besitzt. Als Faktennamen können beliebige nichtleere Zeichenketten verwendet werden; Zeichenketten, die nur aus Buchstaben, Ziffern und dem Unterstrich ‘_’ bestehen und deren erstes Zeichen ein Kleinbuchstabe ist, brauchen nicht quotiert werden, alle anderen Zeichenketten sind zu quotieren. Eine (logische) Längenbegrenzung des Namens existiert nicht.

Attribut-Wert-Paar:

Der Attributname definiert eine bestimmte „Eigenschaft“, während der Wert die konkrete Ausprägung dieser Eigenschaft repräsentiert.

[1] Namenskonvention: Extension **.rd**, z.B. **ffs_1.rd**

Bsp.: **haarfarbe** (Attributname) - **blond** (Wert)

Attributnamen sind beliebige Zeichenketten, die im Rahmen des Faktes eindeutig sein müssen. Attributwerte können Zahlen bzw. Zahlenlisten oder Zeichenketten bzw. Zeichenkettenlisten sein.

Wertlisten werden folgendermaßen dargestellt: **{** [$Wert_{Z,ZK,L}$ [**,** ..]] **}**

Certainty Factor:

Zu jedem Fakt kann ein Certainty Factor (CF) im Bereich 0..1 (Default) angegeben werden. Dieser ist ein Maß für die „Sicherheit“ der im jeweiligen Fakt zusammengefaßten Attribut-Wert-Paare. Der CF wird z.Z. nicht ausgewertet.

Vordefinierte Fakten:

Name: **inferenz_S**
Attribute: **start**
Wert = **ja** zu Beginn der Optimierung
kann zur Ausführung von Initialisierungsregeln verwendet werden
stop
Wert = **nein** zu Beginn der Optimierung
bei Wert = **ja** wird Optimierung beendet

Name: **inferenz**
Attribute: **prioritaet**
Wert = **reihenfolge** zu Beginn der Optimierung
legt das Ordnungskriterium für die anwendbaren Regeln fest
Werte: **reihenfolge**
(zuerst definierte Regeln zuerst [S[1]])
max_spri
(Regeln mit größerem CF zuerst [D])
min_anwendungen
(Regeln mit wenigsten Anwendungen bisher zuerst [D])
max_erfolg
(Regeln mit meisten Erfolgen[2] zuerst [D])
max_erfolg_spri
min_misserfolg
(Regeln mit wenigsten Mißerfolgen zuerst [D])
min_misserfolg_spri
max_erfolgsrate
(Regeln mit bester Erfolgsrate zuerst [D])
max_erfolgsrate_spri
max_erfolgsrate_last
(Regeln mit bester Erfolgsrate zuerst, die zuletzt angewendete

[1] statische Prioritäten (S) bleiben während der Optimierung konstant, dynamische Prioritäten (D) können sich während der Optimierung ändern (je nach Verlauf der Optimierung)

[2] als „Erfolg“ zählt, wenn eine Regel mittel- oder unmittelbar zu einer Verbesserung des Zielfunktionswertes geführt hat

Regel jedoch zuletzt [D])
max_erfolgsrate_last_spri
max_verbesserung
(Regeln mit größter Zielfunktionswertverbesserung zuerst [D])
max_verbesserung_spri

Name: **aufgabe**
Attribute: **anz_param** - Anzahl Parameter der Optimierungsaufgabe

4.4.2 Regeln

Regeln sind autonome Wissenseinheiten, die angeben, welche Aktionen in konkreten Situationen ausgeführt werden sollten, um bestimmte Ziele zu erreichen („statisches Wissen“). Sie tragen meist heuristischen Charakter.

allgemeiner Aufbau:
REGEL(*RegelName*$_{ZK}$**):**
WENN { [*Bedingung* [**,** ..]] **}**
DANN { [*Aktion* [**,** ..]] **}**
[**CF** *CertaintyFactor*$_{Z}$] **;**

Regelname:

Regeln besitzen einen eigenen Namensraum, in dem jede Regel einen eindeutigen Namen besitzt. Als Regelnamen können beliebige nichtleere Zeichenketten verwendet werden; Zeichenketten, die nur aus Buchstaben, Ziffern und dem Unterstrich ‘_’ bestehen und deren erstes Zeichen ein Kleinbuchstabe ist, brauchen nicht quotiert werden, alle anderen Zeichenketten sind zu quotieren. Eine (logische) Längenbegrenzung des Namens existiert nicht.

Bedingungen:

Bedingungen sind Konstrukte, deren Abarbeitung einen der logischen Werte TRUE oder FALSE liefert. Die Abarbeitung von Bedingungen führt nicht zu Modifikationen der statischen oder dynamischen Wissensbasis, ggf. können jedoch Variablen gebunden werden. Der Bedingungsteil einer Regel ist erfüllt (d.h. die Regel ist anwendbar), wenn alle Bedingungen den Wert TRUE liefern.

Vergleiche:

Ausdruck1$_{ZA,ZKA}$	**<**	*Ausdruck2*$_{ZA,ZKA}$
Ausdruck1$_{ZA,ZKA}$	**<=**	*Ausdruck2*$_{ZA,ZKA}$
Ausdruck1$_{ZA,ZKA,LA,RA}$	**==**	*Ausdruck2*$_{ZA,ZKA,LA,RA}$
Ausdruck1$_{ZA,ZKA,LA,RA}$	**!=**	*Ausdruck2*$_{ZA,ZKA,LA,RA}$
Ausdruck1$_{ZA,ZKA}$	**>**	*Ausdruck2*$_{ZA,ZKA}$
Ausdruck1$_{ZA,ZKA}$	**>=**	*Ausdruck2*$_{ZA,ZKA}$

die zu vergleichenden Ausdrücke müssen Werte gleichen Typs[1] liefern
(Typen: Zahlen, Zeichenketten, Listen, Referenzen)

[1] Werte verschiedenen Typs sind immer ungleich !!

Wertbindung von Variablen:

UVar$_{UV}$ **:=** *Ausdruck*$_{A}$

der Wert des Ausdrucks wird an die (bisher ungebundene) Variable gebunden

UVar$_{UV}$ **:= zaehle(***UG*$_{GZA}$**,** *OG*$_{GZA}$**)**

der Wert von *UG* wird an die (bisher ungebundene) Variable gebunden; über Backtracking wird die Variable sukzessive an die folgenden ganzen Zahlen gebunden, bis der Wert von *OG* erreicht ist (kurz: „Generierung einer Folge ganzer Zahlen“)

Szenarioverbesserung:

szenario_verbesserung(*Referenz*$_{RA}$**)**

testet, ob das mittels *Referenz* referenzierte, bereits bewertete Szenario in die Liste der gleichwertig besten Szenarios aufgenommen werden würde(!)
(schlägt auch fehl, wenn es sich bereits in der Liste befindet)

Test der Nebenbedingungen:

teste_nebenbedingungen(*Referenz*$_{RA}$**)**

testet, ob das mittels *Referenz* referenzierte, bereits bewertete Szenario gültig ist, d.h. alle Nebenbedingungen erfüllt

Sonstiges:

cut

verhindert ein Backtracking der Bedingungen, die vor dem Cut im Bedingungsteil stehen

Aktionen:

Aktionen sind Konstrukte, deren Abarbeitung zu Modifikationen der statischen oder dynamischen Wissensbasis führt. Grundsätzlich wird davon ausgegangen, daß alle Aktionen einer Regel korrekt abgearbeitet werden.

Wertbindung von Variablen:

siehe Bedingungen

Modifikation von Fakten:

$***FaktName*$_{ZK}$**.***AttrName*$_{ZK}$ **:= *Ausdruck*$_{A}$

der Wert von Ausdruck wird dem Attribut *AttrName* von Fakt *FaktName* zugewiesen (ggf. wird das Attribut und der Fakt vorher erzeugt)

loesche_fakt($***FaktName*$_{ZK}$)**

löscht den Fakt *FaktName* aus der dynamischen Wissensbasis

Modifikation von Parametern:

<parameter> **:=** *AbsWert*$_{ZA}$

mit <parameter> ::= *DObjName*$_{ZK}$[**[***Index1*$_{GZA}$**]**[**[***Index2*$_{GZA}$**]**]]

weist dem Parameter den Wert des arithm. Ausdrucks *AbsWert* zu
der Wert muß innerhalb des Wertebereichs des Parameters liegen

erhoehe <parameter> **um** *Schritt*$_{ZA,ZKA}$
erhoehe <parameter>
verringere <parameter> **um** *Schritt*$_{ZA,ZKA}$
verringere <parameter>

vergrößert/verkleinert den Wert des Parameters um einen Zahlenwert bzw. einen qualitativen Wert (siehe Abschnitt 4.3.2)

aendere <parameter> **um** *Schritt*$_{ZA,ZKA}$
aendere <parameter>

ändert den Wert des Parameters um einen Zahlenwert bzw. einen qualitativen Wert (siehe Abschnitt 4.3.2)

Szenariobewertung:

UVar$_{UV}$ **:= bewerte_szenario**

bewertet das aktuelle (quantitative) Szenario und bindet die (bisher ungebundene) Variable *UVar* an die Referenz des bewerteten Szenarios
die Referenz steht danach als einziges Element in der **last_szenario**-Liste

UVar$_{UV}$ **:= optimiere_szenario**

führt eine Suche in der Klasse der durch das aktuelle (qualitative) Szenario beschriebenen Szenarios durch und bindet die (bisher ungebundene) Variable *UVar* an die Referenz des besten gefundenen Szenarios
die Referenzen der ununterscheidbar besten Szenarios, die in diesem Optimierungsschritt ermittelt wurden, stehen danach in der **last_szenario**-Liste

Szenarioverwaltung:

setze_akt_szenario(*Referenz*$_{RA}$**)**

eine Kopie des mittels *Referenz* referenzierten, bereits bewerteten Szenarios wird zum „aktuellen“ Szenario, in dem alle folgenden Parametermodifikationen ausgeführt werden

add_opt_szenario(*Referenz*$_{RA}$**)**

versucht, das mittels *Referenz* referenzierte, bereits bewertete Szenario in die Liste der gleichwertig besten Szenarios (**opt_szenario**) aufzunehmen

die Erfüllung der Nebenbedingungen wird intern getestet, bei Nichterfüllung bzw. Nichtaufnahme in die Liste erfolgt „stilles Scheitern“[1] der Aktion

`add_opt_szenarios`

versucht, möglichst alle Szenarios der bei der letzten Szenariobewertung gefundenen gleichwertig besten Szenarios (stehen in **`last_szenario`**) in die Liste der gleichwertig besten Szenarios (**`opt_szenario`**) aufzunehmen
die Erfüllung der Nebenbedingungen wird intern getestet, bei Nichterfüllung bzw. Nichtaufnahme in die Liste erfolgt „stilles Scheitern“ der Aktion

Sonstiges:

`cut`

verhindert ein Backtracking für den Regelinterpreter-Zyklus, in dem die Cut-Aktion abgearbeitet wurde und für alle vorhergehenden Zyklen

`print(` [*Ausdruck*$_A$ [**`,`** ..]] **`)`**

Ausgabe der Werte der Ausdrücke oder der Ausdrücke selbst (falls mittels **`'..'`** quotiert) auf der Standardausgabe
spezieller Ausdruck **`nl(`**Zeilenwechsel)

`print_opt_szenarios`

Ausgabe des aktuellen Inhalts der Liste der gleichzeitig besten Szenarios (**`opt_szenario`**)

Certainty Factor:

Zu jeder Regel kann ein Certainty Factor (CF) im Bereich 0..1 (Default) angegeben werden. Dieser ist ein Maß für die „Erfolgsaussichten“ der jeweiligen Regel, zu einer Verbesserung der Lösung beizutragen. Der CF kann zur Ordnung der anwendbaren Regeln herangezogen werden.

Arithmetische Ausdrücke:

Arithmetische Ausdrücke können aus den folgenden Elementen bestehen:

(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) numerischer Wert von Datenobjekten:

`#`[**`nr(`***RefNr*$_{GZ}$**`).`**]*DObjName*$_{ZK}$[**`[`***Index1*$_{GZA,UV}$**`]`**[**`[`***Index2*$_{GZA,UV}$**`]`**]]

Indizes müssen angegeben werden, falls es sich um einen Vektor oder eine Matrix handelt
werden ungebundene Variablen als Indizes angegeben, dann werden diese an den

[1] „stilles Scheitern“ bedeutet, daß die Aktion nicht (vollständig) ausgeführt wird, die Regelausführung jedoch bei der nächsten Aktion fortsetzt

Wert 1 gebunden und als Index verwendet; über Backtracking werden die Variablen sukzessive an die folgenden ganzen Zahlen gebunden, bis die jeweilige Dimensionsgrenze dieses Datenobjekts erreicht ist

mit Hilfe des **nr(.)**-Konstrukts kann explizit dasjenige Szenario angegeben werden, dessen Datenobjekt *DObjName* verwendet werden soll

(3) numerische Attribute von Datenobjekten:

#*DObjName*$_{ZK}$[**[***Index1*$_{GZA,UV}$**]**[**[***Index2*$_{GZA,UV}$**]**]]**.***AttrName*$_{ZK}$

Indizes müssen angegeben werden, falls es sich um einen Vektor oder eine Matrix handelt
werden ungebundene Variablen als Indizes angegeben, dann werden diese an den Wert 1 gebunden und als Index verwendet; über Backtracking werden die Variablen sukzessive an die folgenden ganzen Zahlen gebunden, bis die jeweilige Dimensionsgrenze dieses Datenobjekts erreicht ist

AttrName:
- **dim1** - erste Dimension (falls Vektor oder Matrix)
- **dim2** - zweite Dimension (falls Matrix)
- **ug** - untere Grenze des Wertebereichs (falls Parameter)
- **og** - obere Grenze des Wertebereichs (falls Parameter)
- **sw** - nutzerdefinierter Startwert (falls Parameter)

(4) numerische Attribute von Fakten:

**[int]$***FaktName*$_{ZK}$**.***AttrName*$_{ZK}$
**[float]$***FaktName*$_{ZK}$**.***AttrName*$_{ZK}$
**[number]$***FaktName*$_{ZK}$**.***AttrName*$_{ZK}$

(5) arithmetische Operatoren: **+** , **-** , ***** , **/** , **div** , **mod** , **^** (Potenz)

(6) arithmetische Funktionen:

round(*X*$_{ZA}$**)** Runden von X auf die nächste ganze Zahl
trunc(*X*$_{ZA}$**)** Abrunden von X auf die nächst kleinere ganze Zahl
abs(*X*$_{ZA}$**)** Betrag von X
exp(*X*$_{ZA}$**)** Exponentialfunktion e^X
log(*X*$_{ZA}$**)** natürlicher Logarithmus ln X , $X > 0$
sqrt(*X*$_{ZA}$**)** Quadratwurzel von X , $X \geq 0$
sin(*X*$_{ZA}$**)** Sinus von X
cos(*X*$_{ZA}$**)** Cosinus von X

min(*X*$_{ZA}$**,***Y*$_{ZA}$**)** kleinere der Zahlen X und Y
max(*X*$_{ZA}$**,***Y*$_{ZA}$**)** größere der Zahlen X und Y

summe(*UVar*$_{UV}$**,** *BedListe*$_{BL}$**)**
produkt(*UVar*$_{UV}$**,** *BedListe*$_{BL}$**)**
minimum(*UVar*$_{UV}$**,** *BedListe*$_{BL}$**)**
maximum(*UVar*$_{UV}$**,** *BedListe*$_{BL}$**)**

die Bedingungsliste *BedListe* wird mittels Backtracking so oft wie möglich abgearbeitet, wobei alle Wertbindungen von *UVar* (erlaubt sind nur Zahlen)

in einer internen Liste gespeichert werden
über dieser Liste wird abschließend die jeweilige Aktion ausgeführt (Summe/Produkt bilden, Minimum/Maximum suchen) und die resultierende Zahl als Ergebnis geliefert

finde(*UVar*$_{UV}$**,** *BedListe*$_{BL}$**)**

die Bedingungsliste *BedListe* wird abgearbeitet und die Wertbindung von *UVar* (erlaubt sind nur Zahlen) als Ergebnis geliefert
über Backtracking können sukzessive weitere Wertbindungen von *UVar* ermittelt werden

anz_elemente(*Liste*$_{LA}$**)**

Anzahl Elemente der Liste

get_element_i(*Liste*$_{ZLA}$**,***Index*$_{GZA}$**)**

Index-tes Element der Zahlenliste(!) *Liste*

(7) arithmetische Ausdrücke können mittels **(** .. **)** geklammert werden

Zeichenkettenausdrücke:

Zeichenkettenausdrücke können aus den folgenden Elementen bestehen:

(1) Zeichenketten (ggf. mittels **'**..**'** quotiert)

(2) Zeichenkettenattribute von Datenobjekten:

#*DObjName*$_{ZK}$[**[***Index1*$_{GZA,UV}$**]**[**[***Index2*$_{GZA,UV}$**]**]]**.***AttrName*$_{ZK}$

AttrName: **art** - Wert aus { **parameter** , **konstante** , **kenngroesse** , **abhaengige** }
typ - Wert aus { **int** , **float** }
struktur - Wert aus { **skalar** , **vektor** , **matrix** }

(3) Zeichenkettenattribute von Fakten:

**[atom]$**FaktName$_{ZK}$**.***AttrName*$_{ZK}$

(4) Zeichenkettenfunktionen:

finde(*UVar*$_{UV}$**,** *BedListe*$_{BL}$**)**

die Bedingungsliste *BedListe* wird abgearbeitet und die Wertbindung von *UVar* (erlaubt sind nur Zeichenketten) als Ergebnis geliefert
über Backtracking können sukzessive weitere Wertbindungen von *UVar* ermittelt werden

get_element_i(*Liste*$_{ZKLA}$**,***Index*$_{GZA}$**)**

Index-tes Element der Zeichenkettenliste *Liste*

Listenausdrücke:

Listenausdrücke können aus den folgenden Elementen bestehen:

(1) Listen der Form **{** [$Element_{Z,ZK,L}$ [**,** ..]] **}**

(2) Listen-Attribute von Fakten:

**[number_list]$**$FaktName_{ZK}$**.**$AttrName_{ZK}$
**[atom_list]$**$FaktName_{ZK}$**.**$AttrName_{ZK}$
**[list]$**$FaktName_{ZK}$**.**$AttrName_{ZK}$

(3) Listen-Funktionen:

finde($UVar_{UV}$**,** $BedListe_{BL}$**)**

die Bedingungsliste *BedListe* wird abgearbeitet und die Wertbindung von *UVar* (erlaubt sind nur Listen) als Ergebnis geliefert
über Backtracking können sukzessive weitere Wertbindungen von *UVar* ermittelt werden

set_element_i($Liste_{L}$**,**$Index_{GZA}$**,**$NeuWert_{A}$**)**

liefert eine Kopie von *Liste*, bei der das *Index*-te Element durch den Wert des Ausdrucks *NeuWert* ersetzt wurde

verknuepfe_listen($Liste1_{L}$**,**$Liste2_{L}$**)**

liefert eine Liste, die sich aus der Verknüpfung der Listen *Liste1* und *Liste2* ergibt

ordne_liste($Liste_{L}$**,**$OrdArt_{ZK}$**)**

liefert eine Kopie von *Liste*, die nach *OrdArt* geordnet wurde
OrdArt ∈ { **aufsteigend** , **absteigend** }
sinnvoll nur für Zahlen- und Zeichenkettenlisten

finde_alle($UVar_{UL}$**,** $BedListe_{BL}$**)**

die Bedingungsliste *BedListe* wird mittels Backtracking so oft wie möglich abgearbeitet, wobei die Wertbindungen von *UVar* in einer Liste gesammelt werden; diese Liste wird als Ergebnis geliefert

Referenzausdrücke:

Referenzausdrücke können aus den folgenden Elementen bestehen:

(1) Referenzen der Form **nr(**$Index_{GZ}$**)**

(2) Referenz-Attribute von Fakten:

**[ref]$**$FaktName_{ZK}$**.**$AttrName_{ZK}$

(3) **szenario_referenz(**$SzArt_{ZK}$**)**

liefert über Backtracking Referenzen auf alle durch *SzArt* beschriebenen Szenarios

SzArt: **`opt_szenario`** - die gleichwertig besten Szenarios
`last_szenario` - die gleichwertig besten Szenarios des letzten Optimierungsschritts
`all_szenario` - alle Szenarios

5 Aufruf und Bedienung von DIM_EXPERTE

DIM_EXPERTE kann je nach Erfordernissen sowohl interaktiv als auch automatisch ausgeführt werden. Der Aufruf erfolgt mittels

`dim_experte` [**`--`**[1] Optionen] .

Folgende Optionen werden unterstützt:

`-h` Anzeige eines Hilfstextes zu DIM_EXPERTE

`-a` <datei> Angabe einer existierenden und lesbaren Aufgabenbeschreibungsdatei, die das zu lösende Problem enthält
Dateiname ggf. mit Pfad, aber ohne Extension (**`.dab`** wird angehängt)
(nur automatischer Modus, siehe Abschnitt 5.2)

`-r` <datei> Angabe einer existierenden und lesbaren Regeldatei, die Regeln zum zu lösenden Problem enthält
Dateiname ggf. mit Pfad, aber ohne Extension (**`.rd`** wird angehängt)
Option darf mehrfach auftreten
(nur automatischer Modus, siehe Abschnitt 5.2)

`-e` <datei> Angabe einer (i.allg. noch nicht existierenden) Datei, in die DIM_EXPERTE die Ergebnisse der Optimierung einträgt
Dateiname ggf. mit Pfad, aber ohne Extension (**`.erg`** wird angehängt)
wenn nicht angegeben, wird **`dim_experte.erg`** verwendet
(nur automatischer Modus, siehe Abschnitt 5.2)

`-P` <datei> Angabe einer Datei, in der DIM_EXPERTE den Verlauf der Sitzung protokolliert

`-n` neben einer wissensbasierten Optimierung wird zu Vergleichszwecken auch eine numerische Optimierung (Zufallssuchverfahren) durchgeführt
(nur automatischer Modus, siehe Abschnitt 5.2)

Als Basis für alle vom Nutzer ausführbaren Dateioperationen verwendet DIM_EXPERTE das beim Start aktuelle Verzeichnis. Dieses kann im interaktiven Modus vom Nutzer verändert werden (siehe Abschnitt 5.1).

Als Basis für alle vom System ausführbaren Dateioperationen (Hilfstexte laden, ...) verwendet DIM_EXPERTE das in der Environmentvariable DIM_EXPERTE_HOME angegebene Verzeichnis. Existiert diese Variable nicht, dann wird das aktuelle Verzeichnis verwendet.

In den folgenden Abschnitten werden die beiden Bedienungsmodi von DIM_EXPERTE beschrieben.

[1] der Doppelstrich **`--`** vor den DIM_EXPERTE-Optionen muß unbedingt angegeben werden !

5.1 Der interaktive Modus

DIM_EXPERTE arbeitet im interaktiven Modus, wenn die **-a** - Option nicht angegeben wurde. In diesem Fall werden eventuelle **-r** -, **-e** - und **-n** - Optionen ignoriert.

Die Angabe durchzuführender Aktionen erfolgt im interaktiven Modus *kommandozeilen-orientiert*, d.h. der Nutzer gibt über die Tastatur Kommandos (Wortgruppen) ein, die von DIM_EXPERTE analysiert und die entsprechenden Aktionen ausgeführt werden. Zur Analyse wird ein einfaches Schlüsselwortverfahren verwendet, das Aktionen an (meist mehreren alternativen) intuitiven Schlüsselwörtern erkennt. Die Stellung der Schlüsselwörter in den Kommando-Wortgruppen ist dabei ohne Belang. Dadurch wird eine einfache Bedienbarkeit und hohe Flexibilität der Eingabe erreicht.

Bei der Beschreibung der Schlüsselwörter gelten folgende Konventionen:

- ein * in einem Schlüsselwort bezeichnet eine beliebige Zeichenkette
- Schlüsselwörter der Form [wort1, wort2, ..] werden nur erkannt, wenn alle Schlüsselwortteile der Liste in der Wortgruppe enthalten sind
  alternativ: wort1 + wort2 + ...
- Schlüsselwörter der Form {wort1, wort2, ..} werden erkannt, wenn mindestens einer der Schlüsselwortteile der Liste in der Wortgruppe enthalten ist

5.1.1 Wichtige Aktionsgruppen

Name: LADE
Beschreibung: Laden von Dateien
Schlüsselwörter: { lade*, lese*, konsultiere* }
Bsp.: ***lade** die regeldatei*

Name: SPEICHERE
Beschreibung: Speichern von Dateien
Schlüsselwörter: { speicher*, schreibe*, sicher* }
Bsp.: ***speicher**e alle internen regeln*

Name: LÖSCHE
Beschreibung: Löschen
Schlüsselwörter: { loesch*, entfern*, vernicht* }
Bsp.: ***loesch**e alle internen regeln*

Name: SETZE
Beschreibung: Setzen/Wertzuweisen
Schlüsselwörter: { setze*, beleg*, schalte*, wechs*, aender* }
Bsp.: ***wechs**le das verzeichnis*

Name: ZEIGE
Beschreibung: Zeigen von Informationen
Schlüsselwörter: { zeige*, [gib,aus], [gebe*,aus] }
Bsp.: ***gib** alle datenobjekte **aus***

5.1.2 Wichtige Aktionen

Name: LADE_REGELN
Beschreibung: lädt eine Regeldatei
(der Dateiname wird explizit abgefragt, bei Angabe von **?** werden die Namen der im aktuellen Verzeichnis existierenden Regeldateien angezeigt)
Schlüsselwörter: LADE + { regel*, heuristik* }
Bsp.: ***lade*** *die* ***regel****datei*

Name: LADE_AUFGABE
Beschreibung: lädt eine Aufgabenbeschreibungsdatei
(der Dateiname wird explizit abgefragt bei Angabe von **?** werden die Namen der im aktuellen Verzeichnis existierenden Aufgabenbeschreibungs-dateien angezeigt)
Schlüsselwörter: LADE + { aufgabe*, modell*, problem* }
Bsp.: ***lese*** *das optimierungs****problem***

Name: SPEICHERE_LÖSUNG
Beschreibung: speichert das Ergebnis einer Optimierung in einer Datei
(der Dateiname wird explizit abgefragt)
Schlüsselwörter: SPEICHERE + { ergebnis*, loesung* }
Bsp.: ***speicher****e die* ***loesung*** *der optimierung*

Name: SPEICHERE_REGELN
Beschreibung: speichert alle nutzerdefinierten Regeln der Wissensbasis in einer Datei
(der Dateiname wird explizit abgefragt)
Schlüsselwörter: SPEICHERE + { regel*, heuristik* }
Bsp.: ***sicher****e alle* ***regel****n*

Name: LÖSCHE_EINE_REGEL
Beschreibung: entfernt eine bestimmte nutzerdefinierte Regel aus der Wissensbasis
(der Regelname wird explizit abgefragt)
Schlüsselwörter: LÖSCHE + { regel, heuristik }
Bsp.: ***loesch****e die* ***regel***

Name: LÖSCHE_ALLE_REGELN
Beschreibung: entfernt alle nutzerdefinierten Regeln aus der Wissensbasis
Schlüsselwörter: LÖSCHE + { regeln, [alle, regel*], heuristiken, [alle, heuristik*] }
Bsp.: ***loesch****e die* ***regeln***

Name: LÖSCHE_EINEN_FAKT
Beschreibung: entfernt einen bestimmten nutzerdefinierten Fakt aus der Wissensbasis
(der Faktname wird explizit abgefragt)
Schlüsselwörter: LÖSCHE + fakt
Bsp.: ***entfern****e den* ***fakt***

Name: LÖSCHE_ALLE_FAKTEN
Beschreibung: entfernt alle nutzerdefinierten Fakten aus der Wissensbasis
Schlüsselwörter: LÖSCHE + { fakten, [alle, fakt*] }
Bsp.: ***entferne alle fakten***

Name: SETZE_DIRECTORY
Beschreibung: verwendet das angegebene Verzeichnis als Arbeitsverzeichnis (das Verzeichnis wird explizit abgefragt, wobei sowohl absolute als auch relative Pfadangaben möglich sind)
Schlüsselwörter: SETZE + { *verzeichnis, *directory }
Bsp.: ***wechs****le das arbeits****verzeichnis***

Name: SETZE_OPT_SZENARIO_SIZE
Beschreibung: setzt die Größe der **`opt_szenario`**-Liste (Default: 3, Werte in [1..10])
Schlüsselwörter: SETZE + opt_szenario_size
Bsp.: ***aender****e die* ***opt_szenario_size***

Name: ZEIGE_EINE_REGEL
Beschreibung: gibt eine bestimmte nutzerdefinierte Regel auf dem Bildschirm aus (der Regelname wird explizit abgefragt)
Schlüsselwörter: ZEIGE + { regel, heuristik }
Bsp.: ***zeige*** *die* ***regel***

Name: ZEIGE_ALLE_REGELN
Beschreibung: gibt alle nutzerdefinierten Regeln auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + { regeln, [alle, regel*], heuristiken, [alle, heuristik*] }
Bsp.: ***zeige*** *die* ***regeln***

Name: ZEIGE_EINEN_FAKT
Beschreibung: gibt einen bestimmten nutzerdefinierten Fakt auf dem Bildschirm aus (der Faktname wird explizit abgefragt)
Schlüsselwörter: ZEIGE + fakt
Bsp.: ***gib*** *den* ***fakt aus***

Name: ZEIGE_ALLE_FAKTEN
Beschreibung: gibt alle nutzerdefinierten Fakten auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + { fakten, [alle, fakt*] }
Bsp.: ***gib alle fakten aus***

Name: ZEIGE_AUFGABE
Beschreibung: gibt die geladene Aufgabenbeschreibung auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + { aufgabe*, modell*, problem* }
Bsp.: ***zeige*** *die* ***aufgabe***

Name: ZEIGE_DATENOBJEKTE
Beschreibung: gibt den Datenobjekt-Teil der geladenen Aufgabenbeschreibung auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + datenobjekt*
Bsp.: ***zeige*** *alle* ***datenobjekt****e*

Name: ZEIGE_KONSTANTEN
Beschreibung: gibt den Konstanten-Teil der geladenen Aufgabenbeschreibung auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + konstante*
Bsp.: ***zeige** alle **konstante**n*

Name: ZEIGE_PARAMETER
Beschreibung: gibt den Parameter-Teil der geladenen Aufgabenbeschreibung auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + parameter
Bsp.: ***zeige** alle **parameter***

Name: ZEIGE_ABHÄNGIGE
Beschreibung: gibt den Abhängigen-Teil der geladenen Aufgabenbeschreibung auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + abhaengige*
Bsp.: ***zeige** alle **abhaengige**n*

Name: ZEIGE_NEBENBEDINGUNGEN
Beschreibung: gibt den Nebenbedingungs-Teil der geladenen Aufgabenbeschreibung auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + { nebenbedingung*, restriktion*, constraint* }
Bsp.: ***zeige** alle **nebenbedingung**en*

Name: ZEIGE_ZIELFUNKTION
Beschreibung: gibt den Zielfunktions-Teil der geladenen Aufgabenbeschreibung auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + zielfunktion
Bsp.: ***zeige** die **zielfunktion***

Name: ZEIGE_SCHNITTSTELLE
Beschreibung: gibt den Schnittstellen-Teil der geladenen Aufgabenbeschreibung auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + { *schnittstelle, *interface }
Bsp.: ***zeige** die programm**schnittstelle***

Name: ZEIGE_LÖSUNG
Beschreibung: gibt die Lösung einer zuvor durchgeführten Optimierung auf dem Bildschirm aus
Schlüsselwörter: ZEIGE + { ergebnis*, loesung* }
Bsp.: ***zeige** die gefundene **loesung***

Name: ZEIGE_ZUSTAND
Beschreibung: zeigt den aktuellen Zustand von DIM_EXPERTE aus
(geladene Dateien, durchgeführte Optimierungen, usw.)
Schlüsselwörter: ZEIGE + { *zustand, *status }
Bsp.: ***zeige** den system**zustand***

Name: OPTIMIERE
Beschreibung: führt eine Optimierung der geladenenen Aufgabe durch
es kann zwischen wissensbasierter Optimierung und numerischer Optimierung (Zufallssuche) gewählt werden
Schlüsselwörter: { optimier*, loes*, dimensionier* }
Bsp.: ***optimier**e das system*

Name: HILFE
Beschreibung: menügeführte Hilfe zu DIM_EXPERTE
Schlüsselwörter: { helfe*, hilf* }
Bsp.: ***hilf** mir mal*

Name: ENDE
Beschreibung: beendet DIM_EXPERTE
Schlüsselwörter: { stop, ende, halt, exit, quit }
Bsp.: ***ende***

5.2 Der automatische Modus

DIM_EXPERTE arbeitet im automatischen Modus (Batch-Modus), wenn die **`-a`** - Option angegeben wurde. In diesem Fall werden auch eventuell angegebene **`-r`** -, **`-e`** - und **`-n`** - Optionen beachtet.

Die **`-a`** - Option dient zur Angabe der Aufgabenbeschreibungsdatei, die die zu lösende Aufgabe enthält. Dem Dateinamen kann ggf. ein Pfad vorangestellt werden, die Extension **`.dab`** ist jedoch nicht anzugeben.

Die **`-r`** - Option dient zur Angabe der Regeldatei, die Regeln zur zu lösenden Aufgabe enthält. Dem Dateinamen kann ggf. ein Pfad vorangestellt werden, die Extension **`.rd`** ist jedoch nicht anzugeben. Diese Option kann auch mehrmals angegeben werden, wobei dann alle angegebenen Regeldateien in die Wissensbasis geladen werden.

Die **`-e`** - Option dient zur Angabe der Ergebnisdatei, in die die Ergebnisse der Optimierung geschrieben werden sollen. Dem Dateinamen kann ggf. ein Pfad vorangestellt werden, die Extension **`.erg`** ist jedoch nicht anzugeben. Fehlt diese Option, dann werden die Ergebnisse in die Datei **`dim_experte.erg`** geschrieben.

Standardmäßig wird nur eine numerische Optimierung (Zufallssuche) durchgeführt. Bei Angabe einer **`-r`** - Option wird nur eine wissensbasierte Optimierung durchgeführt; wird zusätzlich noch die **`-n`** - Option angegeben, dann werden beide Optimierungsarten durchgeführt.

Alle Standardausgaben von DIM_EXPERTE, die denen im interaktiven Modus gleichen, werden in die Datei **`dim_experte.out`** im aktuellen Verzeichnis ausgegeben. Alle Fehlerausgaben werden auf dem Bildschirm ausgegeben.

Anhang A: Beispiel einer Aufgabenbeschreibungsdatei

Folgende Optimierungsaufgabe ist als Aufgabenbeschreibungsdatei darzustellen:

Zielfkt.:	$f(x_1,x_2) = A*x_1^2 + B*x_2^2 \rightarrow$ Minimum
	mit A = 1 , B = 10 , Zielfkt. realisiert als Programm 'aufgabe1'
Nebenbed.:	$-1 \leq x_1 \leq 1$ $-1 \leq x_2 \leq 1$ $f(x_1,x_2) < 5$
Startwert:	$x_1 = 1,\ x_2 = -1$

Aufgabenbeschreibungsdatei **`aufgabe1.dab`**:

```
datenobjekte
{ float x1, x2, fx, A, B; }

konstanten
{ A = 1; B = 10; }

parameter
{ x1 = wb(-1..1), sw(1);
  x2 = wb(-1..1), sw(-1);
}

zielfunktion
{ fx -> minimum; }

nebenbedingungen
{ fx < 5; }

programm
{ name = aufgabe1;
  art = deterministisch;
  format = i_face;
  eingabe = { x1, x2 };
  ausgabe = { fx };
}
```

Anhang B: Beispiel einer Regeldatei

Für die in Anhang A formulierte Optimierungsaufgabe sind geeignete Regeln aufzustellen. Die allgemeingültigen Regeln sind mit (*) markiert, die aufgabenspezifischen Regeln mit (#).

Aufgabenbeschreibungsdatei **`aufgabe1.rd`**:

```
regel(init_regel):  (*)
WENN { $inferenz.start == ja }
DANN { $inferenz.start := nein,
       $inferenz.prioritaet := max_erfolgsrate_spri,
       $inferenz.aktion := aendere_parameter };

regel(aendere_x1):  (#)
WENN { $inferenz.aktion == aendere_parameter,
       A > B }
DANN { aendere x1,
       $inferenz.aktion := szenario_bewerten }
CF 0.8;

regel(aendere_x2):  (#)
WENN { $inferenz.aktion == aendere_parameter,
       A < B }
DANN { aendere x2,
       $inferenz.aktion := szenario_bewerten }
CF 0.8;

regel(aendere_beide):  (#)
WENN { $inferenz.aktion == aendere_parameter }
DANN { aendere x1,
       aendere x2,
       $inferenz.aktion := szenario_bewerten }
CF 0.4;

regel(modell1):  (*)
WENN { $inferenz.aktion == szenario_bewerten }
DANN { _ := optimiere_szenario,
       $inferenz.aktion := szenario_einordnen };

regel(modell2):  (*)
WENN { $inferenz.aktion == szenario_einordnen,
       Sz := szenario_referenz(last_szenario),
       teste_nebenbedingungen(Sz),
       szenario_verbesserung(Sz) }
DANN { add_opt_szenario(Sz),
       cut,
       $inferenz.aktion := aendere_parameter };
```

Anhang C: Systemvoraussetzungen

DIM_EXPERTE läuft zur Zeit ausschließlich auf UNIX-Systemen.

Um DIM_EXPERTE für ein konkretes UNIX-System generieren zu können, muß für dieses folgende Software verfügbar sein:

- SWI-Prolog (ab Version 2.7.7)
- GNU-C/C^{++}-Compiler (ab Version 2.5.8)
- Scanner-Generator GNU-flex (ab Version 2.5.2)
- Parser-Generator GNU-bison (ab Version 1.22)

Um DIM_EXPERTE auf einem Rechner mit dem jeweiligen UNIX-System verwenden zu können, muß auf dem Rechner folgende Software verfügbar sein:

- SWI-Prolog (ab Version 2.7.7)

Die von DIM_EXPERTE verwendeten Analyseprogramme müssen auf demselben Rechner laufen wie das Optimierungssystem.

Literaturverzeichnis

/BANK-95/ Banks, J.; Carson, J.S.; Nelson, B.L.: *Discrete-Event System Simulation*. Second Edition. Prentice-Hall, 1995.

/HADE-93/ Hader, S.: *Anwendung von Methoden der KI bei der simulativen und analytischen Untersuchung speziell strukturierter Systeme*. Diplomarbeit, TU Chemnitz-Zwickau, Fachbereich Informatik, 1993.

/HADE-94a/ Hader, S.: *Ein regelbasiertes System zur automatischen Planung und Durchführung von Simulationsläufen*. 7. Workshop des ASIM-Arbeitskreises „Simulation und KI“, Braunschweig, April 1994.

/HADE-94b/ Hader, S.: *Regelbasierte Optimierung von Produktionssystemen*. 9. Symposium Simulationstechnik, Stuttgart, Oktober 1994, S. 603-606.

/HADE-95/ Hader, S.: *A Knowledge-based Approach to the Optimization of Technical Systems and Its Application to Inventory Systems*. Second International Summer School on Inventory Research, Portoroz, Slovenia, August 1995, pp. 87-92.

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