

Automatisierte Logik und Programmierung

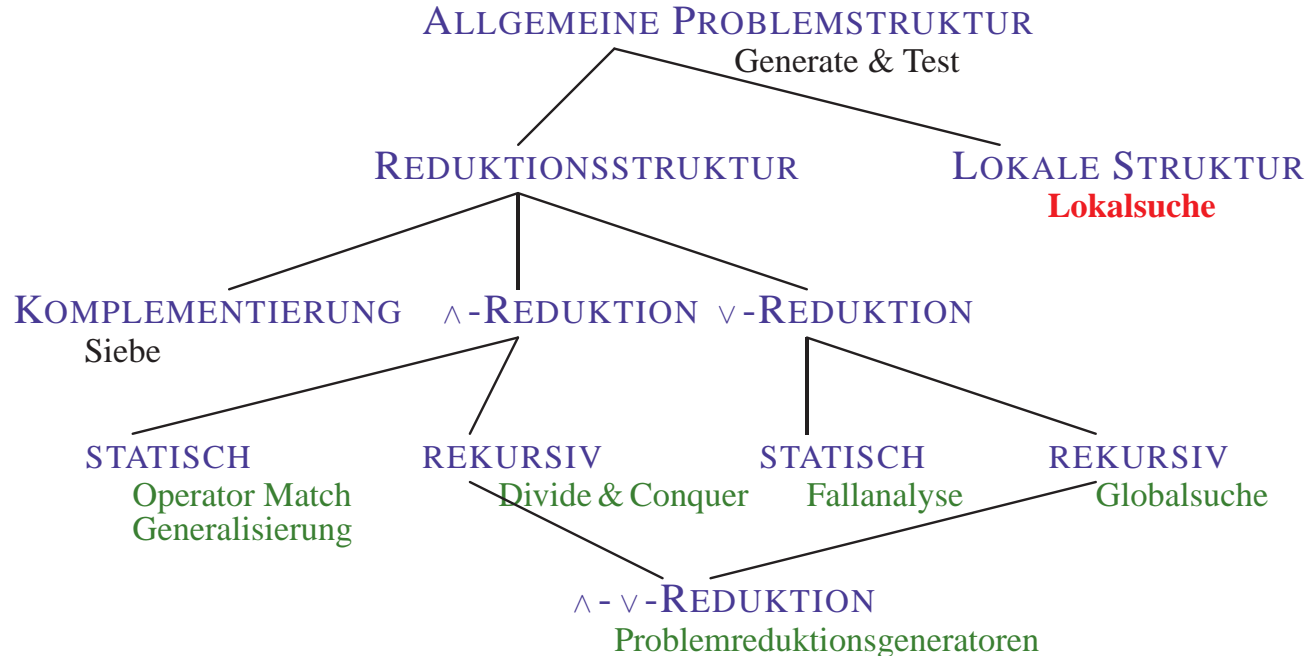
Einheit 22

Lokalsuch-Algorithmen



1. Algorithmenschema
2. Korrektheit
3. Wissensbasierte Unterstützung
4. Synthesestrategie

LOKALSUCH-ALGORITHMEN



- **Problemlösung durch kleine (lokale) Veränderungen**
 - Beginne Suche irgendwo im Raum akzeptabler Lösungen
 - Versuche Qualität der (Teil-)Lösung schrittweise zu verbessern
 - Gut für Optimierungsprobleme (Travelling Salesman, Scheduling, ...)
 - Erfordert Bewertung der Qualität von Elementen des Bildbereichs
- **Anderersartiger Ansatz ohne Problemreduktion**
 - Schwerpunkt liegt auf Durchsuchen einer lokale Nachbarschaft
 - Methode des Hillclimbing: Suchen, solange es noch aufwärts geht

EIN TYPISCHER LOKALSUCH-ALGORITHMUS

Minimiere $\sum_{i=1}^n c_i y_i$ **wobei** $\sum_{j=1}^m A_{i,j} y_i \leq b_j$ **und** $y_i \geq 0$

Unterbestimmtes System: Anzahl der Gleichungen (m) kleiner als Zahl der Variablen (n)

• Spezifikation als lineares Optimierungsproblem

FUNCTION $\text{LP}(A, b : \text{Seq}(\text{Seq}(\mathbb{Q}^+)) \times \text{Seq}(\mathbb{Q}^+)) : \text{Seq}(\mathbb{Q}^+)$

RETURNS y

SUCH THAT $\sum A_{i,j} y_i \leq b_j \wedge \forall t : \text{Seq}(\mathbb{Q}^+). \sum A_{i,j} t_i \leq b_j \Rightarrow \sum c_i y_i \leq \sum c_i t_i$

• Üblicher Ansatz: Lineare Programmierung

– **Initialisiere:** Setze $basic := \{1..m\}$, $x_{m+1}, ..x_n := 0$

löse $\sum_{i \in basic} A_{i,j} y_i = b_j$ durch Gauß-Verfahren

– **Optimiere $y, basic$ zu $z, basic'$:**

· Wähle $i \in basic, k \notin basic$ so, daß $\sum_{i \in basic'} A_{i,j} z_i = b_j$

nach Tausch von i und k lösbar ist und $\sum c_i z_i < \sum c_i y_i$

· Falls es ein solches Paar gibt, so optimiere $z, basic'$

· Ansonsten terminiere mit Ausgabe y

LOKALSUCH-ALGORITHMEN: EINHEITLICHE DARSTELLUNG

FUNCTION $LP(A, b: \text{Seq}(\text{Seq}(\mathbb{Q}^+)) \times \text{Seq}(\mathbb{Q}^+)) : \text{Seq}(\mathbb{Q}^+)$ RETURNS y
SUCH THAT $\sum A_{i,j} y_i \leq b_j \wedge \forall t: \text{Seq}(\mathbb{Q}^+). \sum A_{i,j} t_i \leq b_j \Rightarrow \sum c_i y_i \leq \sum c_i t_i$

Initialisiere: Setze $basic := \{1..m\}$, $x_{m+1}, ..x_n := 0$, löse $\sum_{i \in basic} A_{i,j} y_i = b_j$

Optimiere y , $basic$:

- Wähle $i \in basic$, $k \notin basic$ so, daß $\sum_{i \in basic'} A_{i,j} z_i = b_j$ nach Tausch von i und k lösbar ist und $\sum c_i z_i < \sum c_i y_i$
- Falls es ein solches Paar gibt, so **optimiere z , $basic'$**
- Ansonsten terminiere mit Ausgabe y

• Struktur der Spezifikation

FUNCTION $f_{opt}(x: D) : R$ WHERE $I[x]$ RETURNS y
SUCH THAT $O[x, y] \wedge \forall t: R. (O[x, t] \Rightarrow c[x, y] \leq c[x, t])$

Ergebnis y soll kostenoptimal unter den möglichen Lösungen der Spezifikation sein

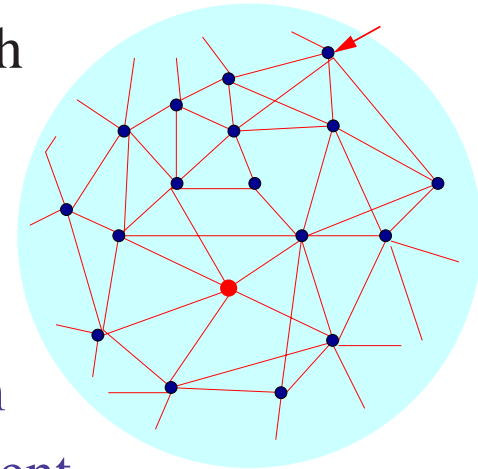
• Struktur des Lösungsalgorithmus

Optimiere Initiallösung y schrittweise durch kleine Veränderungen

```
let  $f_{LS}(x, y) =$  if locally-optimal( $y, O, cost$ ) then  $y$   
                else let  $z$  loc-arbitrary with  $O[x, z] \wedge c[x, z] < c[x, y]$   
                    in  $f_{LS}(x, z)$   
in  $f_{LS}(x, Init[x])$ 
```

Bessere Werte z werden jeweils in einer lokal beschränkten Umgebung von y gesucht

- **Optimierung als Minimierung von Kosten**
 - Spezifikation des Problems besitzt viele mögliche “Lösungen”
 - Lösungen werden bewertet nach Nutzen, Kosten, Korrektheitsgrad, ...
 - Gesucht ist Lösung mit optimaler (o.B.d.A. minimaler) Bewertung
 - Exakte Optimierung oft \mathcal{NP} -vollständig
- **Lösungsverfahren durchsucht Nachbarschaft**
 - Übergang auf Nachbarn, solange Verbesserungen möglich
bei LP z.B. auf benachbarte Mengen von Basisvariablen
 - Verfahren endet in **lokalen Optima**
 - **Nachbarschaftsstruktur** beeinflusst Güte der Lösung
 - Zu fein \Rightarrow Verfahren führt nicht zu **globalem Optimum**
 - Zu grob \Rightarrow **Suche + Test** auf lokale Optimalität ineffizient



Hauptaufgabe ist Bestimmung einer guten Nachbarschaftsstruktur

● Einfaches Programm in formaler Notation

FUNCTION $f_{opt}(x : D) : R$ WHERE $I[x]$ RETURNS y
 SUCH THAT $O[x, y] \wedge \forall t : R. (O[x, t] \Rightarrow c[x, y] \leq c[x, t])$
 \equiv let $f_{LS}(x, z)$
 = if $\forall t \in N[x, z]. (O[x, t] \Rightarrow c[x, z] \leq c[x, t])$ then z
 else $f_{LS}(x, \text{arb}(\{t \mid t \in N[x, z] \wedge O[x, t] \wedge c[x, t] < c[x, z]\}))$
 in $f_{LS}(x, \text{Init}[x])$

● 3 zentrale Komponenten der Algorithmentheorie

- $c: D \times R \rightarrow \mathcal{R}$ Kostenfunktion auf geordnetem **Kostenraum** (\mathcal{R}, \leq)
 Zusatzspezifikation des Optimierungsproblems
- $\text{Init}: D \rightarrow R$ Initiallösung für Basisspezifikation (D, R, I, O)
- $N: D \times R \rightarrow \text{Set}(R)$ **Nachbarschaftsstruktur**
 Suchraumbeschreibung für lokale Variationen

KORREKTHEIT DES LOKALSUCH-GRUNDSCHEMAS

FUNCTION $f_{opt}(x:D):R$ WHERE $I[x]$ RETURNS y
SUCH THAT $O[x,y] \wedge \forall t:R. (O[x,t] \Rightarrow c[x,y] \leq c[x,t])$
 \equiv let $f_{LS}(x,z) =$ if $\forall t \in N[x,z]. (O[x,t] \Rightarrow c[x,z] \leq c[x,t])$ then z
else $f_{LS}(x, \text{arb}(\{t \mid t \in N[x,z] \wedge O[x,t] \wedge c[x,t] < c[x,z]\}))$
in $f_{LS}(x, \text{Init}[x])$

ist korrekt, wenn 4 Axiome erfüllt sind

1. $\text{Init}[x]$ berechnet gültige Initiallösung für O

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

2. Nachbarschaftsstruktur N ist reflexiv

$\forall x:D. \forall y:R. I[x] \wedge O[x,y] \Rightarrow y \in N[x,y]$

3. Lokale Optima sind exakt

\mapsto Optimale Algorithmen

$\forall x:D. \forall y:R. I[x] \wedge O[x,y] \Rightarrow (\forall t \in N[x,y]. O[x,t] \Rightarrow c[x,y] \leq c[x,t])$
 $\Rightarrow \forall z:R. (O[x,z] \Rightarrow c[x,y] \leq c[x,z])$

4. Alle gültigen Lösungen sind endlich erreichbar

$\forall x:D. \forall y, z:R. I[x] \wedge O[x,y] \wedge O[x,z] \Rightarrow \exists k:\mathbb{N}. z \in N_O^k[x,y]$

$N_O^0[x,y] = \{y\}$ $N_O^{k+1}[x,y] = \bigcup \{ N^k[x,t] \mid t \in N[x,y] \wedge O(x,t) \}$

LOKALSUCH-SCHEMA: KORREKTHEITSBEWEIF

- **Abspalten und Spezifikation der Hilfsfunktion f_{ls}**

FUNCTION $f_{opt}(x:D) : R$ WHERE $I[x]$ RETURNS y
SUCH THAT $O[x, y] \wedge \forall t:R. (O[x, t] \Rightarrow c[x, y] \leq c[x, t])$
 $\equiv f_{LS}(x, Init[x])$

FUNCTION $f_{ls}(x, z:D \times R) : R$ WHERE $I[x] \wedge O[x, z]$ RETURNS y
SUCH THAT $O[x, y] \wedge \forall t \in N[x, y]. (O[x, t] \Rightarrow c[x, y] \leq c[x, t])$
 \equiv if $\forall t \in N[x, z]. (O[x, t] \Rightarrow c[x, z] \leq c[x, t])$ then z
else $f_{LS}(x, arb(\{t \mid t \in N[x, z] \wedge O[x, t] \wedge c[x, t] < c[x, z]\}))$

- **Korrektheit von f_{opt} folgt aus der von f_{ls} mit Axiomen 1 & 3**

- Für den Startwert $z = Init[x]$ gilt $O[x, z]$
- Für $y = f_{LS}(x, z)$ gilt $O[x, y] \wedge \forall t \in N[x, y]. (O[x, t] \Rightarrow c[x, y] \leq c[x, t])$
- Mit Axiom 3 folgt $\forall t:R. (O[x, t] \Rightarrow c[x, y] \leq c[x, t])$

- **Partielle Korrektheit von f_{ls} folgt aus Programmkörper**

- Hält f_{ls} mit Ausgabe z , so gilt $\forall t \in N[x, z]. (O[x, t] \Rightarrow c[x, z] \leq c[x, t])$

- **Terminierung von f_{ls} folgt aus Ordnung (\mathcal{R}, \leq) und Axiom 4**

- **Es sind nur 2 neue Komponenten zu bestimmen**
 - Initiaillösung *Init* für Basisspezifikation (D, R, I, O)
wird durch andere Syntheseformen konstruiert (oft trivial)
 - Geeignete Nachbarschaftsstruktur N für lokale Suche nach Optima
Kernaufgabe einer Lokalsuch-Synthese

Lösungen müssen Beweis der Axiome 1–4 liefern

- **Wie findet man gute Nachbarschaftsstrukturen?**
 - Menge von minimalen Veränderungen des aktuellen Lösungspunktes
kontrolliert durch Parameter aus einer “Verwirbelungsmenge” π
 - Dann ist $N[x, y]$ darstellbar als $\{Action[i, j, x, y] \mid i, j \in \pi[x, y]\}$
 - Bezeichnung: $\mathcal{L} = (D, R, I, O, \pi, Action)$ ist **Lokalsuchtheorie**
- **Filter können Nachbarschaftsstrukturen optimieren**
 - Nachbarschaftspunkte müssen zulässige Lösungen sein
 - Nachbarschaftspunkte müssen lokale Optima sein
 - Nachbarschaftspunkte sind, wenn möglich, globale Optima

1. Beschreibe Sortieren als **Ordnungsoptimierung**

- Suche unter den Umordnungen einer Liste L eine geordnete Liste S
Liefert Basisspezifikation $O[L, S] \equiv \text{rearranges}(L, S)$
- S ist geordnet, wenn keine Elemente in falscher Reihenfolge auftauchen
Ziel ist also Minimierung der Anzahl von Fehlstellungen $S_i > S_j$ ($i < j$)
Liefert Kostenfunktion $c[L, S] \equiv \#_>(S) : \equiv |\{(i, j) \mid i < j \wedge S_i > S_j\}|$
- Es gilt $\#_>(S) \geq 0$ und $\#_>(S) = 0 \Rightarrow \text{ordered}(S)$

2. Erstelle formale Spezifikation:

```
FUNCTION sort(L:Seq( $\mathbb{Z}$ )):Seq( $\mathbb{Z}$ ) RETURNS S  
  SUCH THAT rearranges(L, S)  
     $\wedge \forall S' : \text{Seq}(\mathbb{Z}). \text{rearranges}(L, S') \Rightarrow \#_>(S) \leq \#_>(S')$ 
```

3. Erzeuge Initiallösung für Basisspezifikation

- Einfachste Umordnung einer Liste ist die Liste selbst

Liefert $\text{Init}[L] = L$

\mapsto Axiom 1

SYNTHESE EINES LOKALSUCH-ALGORITHMUS (2)

4. Erzeuge Nachbarschaftsstruktur

- Einfache Suchstruktur vertauscht zwei beliebige Elemente S_i und S_j
Ergibt $N[L, S] \equiv \{\text{permute}_{i,j}(S) \mid i, j \leq |S|\}$
- Vertauschen ist reflexiv, Lokale Minima sind exakt \mapsto Axiom 2,3
- Alle Umordnungen erreichbar durch iteratives Vertauschen \mapsto Axiom 4

5. Instantiiere Algorithmenschema

```
FUNCTION sort(L:Seq(Z)):Seq(Z) RETURNS S  
  SUCH THAT rearranges(L, S)  
             $\wedge \forall S' : \text{Seq}(\mathbb{Z}). \text{rearranges}(L, S') \Rightarrow \#_>(S) \leq \#_>(S')$   
 $\equiv$  let sortLS(L, S)  
    = if  $\forall S' \in N[L, S]. \text{rearranges}(L, S') \Rightarrow \#_>(S) \leq \#_>(S')$  then S  
      else sortLS(L, arb{S' ∈ N[L, S] |  
                    rearranges(L, S') ∧ #>(S) > #>(S')})  
    in sortLS(L, L)
```

Ergibt Sortieren durch beliebiges Austauschen von Elementen

- Ineffizient, da zu viele Nachbarn überprüft werden (vermutlich $\mathcal{O}(n^3)$)

4. Erzeuge Nachbarschaftsstruktur

– Restriktion auf $N[L, S] \equiv \{\text{permute}_{i,i+1}(S) \mid i < |S|\}$

Nur unmittelbar benachbarte Elemente dürfen vertauscht werden

5. Verfeinere Struktur durch Filter

– Vertauschungen erhalten Umordnungseigenschaft (Filter unnötig)

– Vertauschungen bringen Verbesserungen, wenn $S_i > S_{i+1}$

Liefert $N'[L, S] \equiv \{\text{permute}_{i,i+1}(S) \mid i < |S| \wedge S_i > S_{i+1}\}$

– Es gilt $\#_>(S) > \#_>(S')$ für alle $S' \in N'[L, S]$

6. Instantiiere (und optimiere) Algorithmenschema

FUNCTION `sort`($L:\text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ RETURNS S

SUCH THAT `rearranges`(L, S)

$\wedge \forall S': \text{Seq}(\mathbb{Z}). \text{rearranges}(L, S') \Rightarrow \#_>(S) \leq \#_>(S')$

\equiv let `sortLS`(L, S)

= if $N'[L, S] = \emptyset$ then S else `sortLS`($L, \text{arb}(N'[L, S])$)

in `sortLS`(L, L)

Entspricht dem `Bubblesort` Algorithmus (ohne feste Reihenfolge)

ANALYSE LIEFERT ZWEI SUCHSTRUKTUREN FÜR `sort`

• Große Nachbarschaft: Permutation von Elementen

- Änderungsschritt: Vertauschung zweier Elemente der Ausgabeliste `S`
- Änderungsparameter: Indizes der Liste `S`

`LS_sort_wide` \equiv

D	\mapsto	<code>Seq(\mathbb{Z})</code>
R	\mapsto	<code>Seq(\mathbb{Z})</code>
I	\mapsto	<code>$\lambda L.$ true</code>
O	\mapsto	<code>$\lambda L, S.$ rearranges(L, S)</code>
π	\mapsto	<code>$\lambda L, S.$ (domain(S), domain(S))</code>
<i>Action</i>	\mapsto	<code>$\lambda i, j, L, S.$ [$S_{(i \leftrightarrow j)(k)}$ $k < S$]</code>

wobei $(i \leftrightarrow j)(k) \hat{=} \text{if } k=i \text{ then } j \text{ else if } k=j \text{ then } i \text{ else } k$

• Kleine Nachbarschaft: Permutation benachbarter Elemente

- Änderungsschritt: Vertauschung fehlgestellter Nachbarn in `S`

`LS_sort_narrow` \equiv

D	\mapsto	<code>Seq(\mathbb{Z})</code>
R	\mapsto	<code>Seq(\mathbb{Z})</code>
I	\mapsto	<code>$\lambda L.$ true</code>
O	\mapsto	<code>$\lambda L, S.$ rearranges(L, S)</code>
π	\mapsto	<code>$\lambda L, S.$ (domain(S), domain(S))</code>
<i>Action</i>	\mapsto	<code>$\lambda i, j, L, S.$ if $S_i \leq S_{i+1}$ then S else [$S_{(i \leftrightarrow i+1)(k)}$ $k < S$]</code>

Alle 4 Axiome sind erfüllt

WIE ERZEUGT MAN EFFIZIENTE SUCHSTRUKTUREN?

- **Formuliere fundamentale Nachbarschaftsstrukturen auf R**
 - Beschreibe Nachbarschaft durch inkrementelle Veränderungen in R
 - Numerik: δ -Vektoren,
 - Kombinatorik: Austausch von Komponenten (in Listen, Mengen, ...)
 - Formalisiere $N[x, y]$ als $\{Action[i, j, x, y] \mid i, j \in \pi[x, y]\}$
 - Änderung $Action[i, j, x, y]$ modifiziert Lösungspunkt $(x, y) \in D \times R$
 - Parameter $i, j \in \pi[x, y]$ sind “minimale Bestandteile” von (x, y)
 - Wissensbank speichert Lokalsuchtheorien für Grunddatentypen
Synthese spezialisiert generische Suchstrukturen auf spezifisches Problem
- **Optimiere Nachbarschaftsstruktur durch Suchfilter**
 - Verenge Suchraum durch Abschneiden unnötiger Nachbarn $t \in N[x, y]$
 - **Feasibility Constraint**: notwendige Bedingung für $O[x, t]$
 - (Local) **Optimality Constraint**: Bedingung für $c[x, t] < c[x, y]$
 - ggf. **Global Optimality Constraint** für exakte Lösungen
 - Filter können durch Vorwärtsinferenz aus Bedingungen abgeleitet werden

Spezialisiere vorformulierte Suchstrukturen

- **Lokalsuchtheorie: allgemeine Suchstruktur für R**
 - Vorgefertigte Nachbarschaftsstruktur, die *Axiome 2–4* erfüllt
 - Formalisiert als Objekt $\mathcal{L} = (D, R, I, O, \pi, \text{Action})$
 - Wissensbank speichert Lokalsuchtheorien für Grunddatentypen
- **z.B. Standard-Suche auf Teilmengen fester Größe**
 - Suche $\hat{=}$ Austausch einzelner Elemente einer Menge
 - Änderungsparameter: Elemente der Ein- und Ausgabemenge

$LS_subsets(\alpha)$	\equiv	D	\mapsto	$Set(\alpha) \times \mathbb{N}$
		R	\mapsto	$Set(\alpha)$
		I	\mapsto	$\lambda S, m. m \leq S $
		O	\mapsto	$\lambda S, m, S'. S' \subseteq S \wedge S' = m$
		π	\mapsto	$\lambda S, m, S'. (S \setminus S', S')$
		$Action$	\mapsto	$\lambda i, j, S, m, S'. (S' \cup \{i\}) - j$

ERZEUGUNG EINER SUCHSTRUKTUR FÜR `sort`

Wähle Struktur \mathcal{L} für R so daß $spec \ll spec_{\mathcal{L}}$ beweisbar

• Standard-Umordnung von Folgen über Datentyp α

- Suche $\hat{=}$ Permutation einzelner Elemente einer Folge
- Änderungsparameter: Indizes der Ausgabeliste L

<code>LS_seq_re(α)</code>	\equiv	D	\mapsto	<code>Seq(α)</code>
		R	\mapsto	<code>Seq(α)</code>
		I	\mapsto	$\lambda L. \text{true}$
		O	\mapsto	$\lambda L, S. \text{rearranges}(L, S)$
		π	\mapsto	$\lambda L, S. (\text{domain}(S), \text{domain}(S))$
		<i>Action</i>	\mapsto	$\lambda i, j, L, S. [S_{(i \leftrightarrow j)(k)} \mid k \in \text{domain}(S)]$

• Spezialisierung auf das Sortierproblem

```
FUNCTION sort(L:Seq( $\mathbb{Z}$ )):Seq( $\mathbb{Z}$ ) RETURNS  $S$   
SUCH THAT rearranges(L, S)
```

$\wedge \forall S' : \text{Seq}(\mathbb{Z}). \text{rearranges}(L, S') \Rightarrow \#_>(S) \leq \#_>(S')$

- LS-Theorie $\mathcal{L} = \text{LS_seq_re}(\mathbb{Z})$ paßt zum Bildbereich `Seq(\mathbb{Z})`
 - Wegen $O_{\mathcal{L}}(L, S) = \text{rearranges}(L, S)$ folgt $spec \ll spec_{\mathcal{L}}$ mit $\vartheta = \text{id}$
- Liefert Grundstruktur für die Lokalsuche

Elimiere überflüssige Kandidaten aus Nachbarschaft

- **Lösungs-Filter FC für \mathcal{L}** (*Feasibility Constraint*)
 - Eliminiere Punkte aus der Nachbarschaft, die keine Lösungen bzgl. O sind, aber erhalte alle akzeptablen Lösungen
 - Ist y akzeptable Lösung, dann erhalte jedes $t \in N[x, y]$ mit $O(x, t)$
 - $I[x] \wedge O[x, y] \wedge t \in N[x, y] \Rightarrow (O[x, t] \Rightarrow FC(x, y, t))$
 - Entspricht notwendigen Filtern der Globalsuche
- **Optimalitäts-Filter OC für \mathcal{L}** (*Optimality Constraint*)
 - Eliminiere kostengünstigere Punkte aus der Nachbarschaft, aber erhalte jedes bessere $t \in N[x, y]$ (mit $c[x, t] < c[x, y]$)
 - $I[x] \wedge O[x, y] \wedge t \in N[x, y] \wedge O[x, t] \Rightarrow (c[x, t] < c[x, y] \Rightarrow OC(x, y, t))$
 - y ist ein lokales Optimum, wenn $\{t \in N[x, y] \mid OC(x, y, t)\}$ leer ist
- **Exaktheits-Filter GOC** (*Global Optimality Constraint*)
 - Eliminiere suboptimale Lösungen aber erhalte alle optimalen
 - $I[x] \wedge O[x, y] \Rightarrow ((\forall t:R. O[x, t] \Rightarrow c[x, y] \leq c[x, t]) \Rightarrow GOC(x, y))$
 - Test oft aufwendig, macht Algorithmus ggf. weniger effizient

FORMULIERE FILTER ALS PARAMETERCONSTRAINTS

- **Lösungs-Filter FC für \mathcal{L}**

- $I[x] \wedge O[x, y] \wedge t \in N[x, y] \Rightarrow (O[x, t] \Rightarrow FC(x, y, t))$
- Filter auch direkt auf Parameter der Änderungsaktion angewandt werden, da $N[x, y] = \{Action[i, j, x, y] \mid i, j \in \pi[x, y]\}$
- Führt zu $I[x] \wedge O[x, y] \Rightarrow (O[x, Action[i, j, x, y]] \Rightarrow FC[i, j, x, y])$
- FC kann durch Vereinfachung von $O[x, Action[i, j, x, y]]$ im Kontext der linken Seite abgeleitet werden

- **Optimalitäts-Filter OC für \mathcal{L}**

- Umformulieren von
$$I[x] \wedge O[x, y] \wedge t \in N[x, y] \wedge O[x, t] \Rightarrow (c[x, t] < c[x, y] \Rightarrow OC(x, y, t))$$
ergibt
$$I[x] \wedge O[x, y] \wedge O[x, Action[i, j, x, y]] \Rightarrow (c[x, Action[i, j, x, y]] < c[x, y] \Rightarrow OC[i, j, x, y])$$

- **Integriere Filter in Basis-Nachbarschaftsstruktur**

- $N_c[x, y] = \{Action[i, j, x, y] \mid i, j \in \pi[x, y] \wedge FC[i, j, x, y] \wedge OC[i, j, x, y]\}$
- Exaktheits-Filter $GOC[x, y]$ wird direkt auf Lösungspunkte angewandt

● **Spezialisierungsmechanismus für LS-Theorien**

↪ §18, Folie 20

- Wähle \mathcal{L} passend zum Bildbereich der Spezifikation $spec = (D, R, I, O)$
- Beweise $spec \ll spec_{\mathcal{L}}$

$$R \subseteq R_{\mathcal{L}} \wedge \forall x: D. I[x] \Rightarrow \exists x': D_{\mathcal{L}}. (I_{\mathcal{L}}[x'] \wedge \forall y: R. O[x, y] \Rightarrow O_{\mathcal{L}}[x', y])$$

- Extrahiere Substitution $\vartheta: D \rightarrow D_{\mathcal{L}}$ aus dem Beweis
- Modifiziere \mathcal{L} mit ϑ zu wohlfundierter Lokalsuchtheorie für $spec$
Ergibt Basis-Nachbarschaftsstruktur N für Problemstellung

● **Vorwärtsinferenz (heuristisch) für Filterung**

↪ Effizienzsteigerung

- Leite Folgerungen aus den Vorbedingungen der Filter ab

$$\cdot I[x] \wedge O[x, y] \wedge O[x, Action[i, j, x, y]] \quad (FC)$$

$$\cdot I[x] \wedge O[x, y] \wedge O[x, Action[i, j, x, y]] \wedge c[x, Action[i, j, x, y]] < c[x, y] \quad (OC)$$

$$\cdot I[x] \wedge O[x, y] \wedge \forall t: R. O[x, t] \Rightarrow c[x, y] \leq c[x, t] \quad (GOC)$$

- Verwende nur Folgerungen, die auch leicht zu testen sind

SYNTHESESTRATEGIE FÜR LOKALSUCH-ALGORITHMEN

Start: FUNCTION $f_{opt}(x:D):R$ WHERE $I[x]$ RETURNS y
SUCH THAT $O[x,y] \wedge \forall t:R. (O[x,t] \Rightarrow c[x,y] \leq c[x,t])$

1. Wähle Lokalsuchtheorie \mathcal{L} mit Ausgabebetyp R (Wissensbank)

2. Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$

– Extrahiere Substitution ϑ und setze $\mathcal{L}_{\vartheta} = (D, R, I, O, \pi_{\vartheta}, Action_{\vartheta})$

3. Synthetisiere Initiaillösung $Init$ für Spezifikation (Standardsynthese)

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

4. Generiere Lösungs-Filter FC für \mathcal{L}_{ϑ} durch Vorwärtsinferenz

5. Generiere Optimalitäts-Filter OC für \mathcal{L}_{ϑ} (*Optimality Constraint*)

6. Integriere Filter in Nachbarschaftsstruktur

– Setze $N_c[x,y] \equiv \{Action[i,j,\vartheta(x),y] \mid i,j \in \pi_c[\vartheta(x),y]\}$

wobei $\pi_c[x,y] \equiv \{i,j \in \pi[x,y] \mid FC[i,j,x,y] \wedge OC[i,j,x,y]\}$

7. Instantiiere & Optimierte Schema für Lokalsuch Algorithmen

– Ggf. generiere und integriere Exaktheitsfilter

SYNTHESE EINES LOKALSUCH-SORTIERALGORITHMUS

Start: FUNCTION `sort(L:Seq(\mathbb{Z})) : Seq(\mathbb{Z})` RETURNS `S`
SUCH THAT `rearranges(L,S)`

$$\wedge \forall S' : \text{Seq}(\mathbb{Z}). \text{rearranges}(L, S') \Rightarrow \#_>(S) \leq \#_>(S')$$

1. **Wähle Lokalsuchtheorie \mathcal{L} mit Ausgabetypp R** $\mathcal{L} = \text{LS_seq_re}(\mathbb{Z})$

2. **Beweise $(D, R, I, O) \ll \text{spec}_{\mathcal{L}}$** (Folie 15) liefert $\vartheta = \text{id}$

3. **Synthetisiere Initiallösung $Init$ für Spezifikation** setze `Init(L) = L`

4. **Generiere Lösungs-Filter FC für \mathcal{L}_{ϑ}** (*Feasibility Constraint*)

$$\text{rearranges}(L, S) \Rightarrow (\text{rearranges}(L, S_{(i \leftrightarrow j)}) \Rightarrow \text{FC}[i, j, L, S])$$

Vereinfachung im Kontext ergibt $\text{FC}[i, j, L, S] = \text{true}$

5. **Generiere Optimalitäts-Filter OC für \mathcal{L}_{ϑ}** (*Optimality Constraint*)

$$\text{rearranges}(L, S) \wedge \text{rearranges}(L, S_{(i \leftrightarrow j)})$$

$$\Rightarrow (\#_>(S_{(i \leftrightarrow j)}) < \#_>(S)) \Rightarrow \text{OC}[i, j, L, S]$$

Vereinfachung im Kontext ergibt $\text{OC}[i, j, L, S] = i < j \Leftrightarrow S_i > S_j$

6. **Instantiiere & Optimiere Schema für Lokalsuch Algorithmen**

– $N_c[L, S] \equiv \{S_{(i \leftrightarrow j)} \mid i < j < |S| \wedge S_i > S_j\}$

– Kein Exaktheitstest erforderlich

SYNTHESE EINES LOKALSUCH-SORTIERALGORITHMUS

7. Instantiiertes Lokalsuch-Schema

FUNCTION *sort*(L:Seq(\mathbb{Z})):Seq(\mathbb{Z}) RETURNS S

SUCH THAT rearranges(L,S)

$\wedge \forall S' : \text{Seq}(\mathbb{Z}). \text{rearranges}(L, S') \Rightarrow \#_>(S) \leq \#_>(S')$

$\equiv \text{sort}_{LS}(L, L)$

FUNCTION *sort*_{LS}(L,S:Seq(\mathbb{Z}) \times Seq(\mathbb{Z})):Seq(\mathbb{Z})

WHERE rearranges(L,S) RETURNS Y

SUCH THAT rearranges(L,Y)

$\wedge \forall T \in \{Y_{(i \leftrightarrow j)} \mid i < j < |Y| \wedge Y_i > Y_j\}. (\text{rearranges}(L, T) \Rightarrow \#_>(Y) \leq \#_>(T))$

$\equiv \text{if } \forall T \in \{S_{(i \leftrightarrow j)} \mid i < j < |S| \wedge S_i > S_j\}. (\text{rearranges}(L, T) \Rightarrow \#_>(S) \leq \#_>(T))$

then S

else *sort*_{LS}(L, arb($\{T \in \{S_{(i \leftrightarrow j)} \mid i < j < |S| \wedge S_i > S_j\} \mid$
rearranges(L,T) $\wedge \#_>(T) < \#_>(S)\}$))

8. Nach Optimierung der Hilfsfunktion

Es gilt $\forall T \in \{Y_{(i \leftrightarrow j)} \mid i < j < |Y| \wedge Y_i > Y_j\}. \text{rearranges}(Y, T) \wedge \#_>(T) < \#_>(Y)$

FUNCTION *sort*_{LS}(L,S:Seq(\mathbb{Z}) \times Seq(\mathbb{Z})):Seq(\mathbb{Z})

WHERE rearranges(L,S) RETURNS Y

SUCH THAT rearranges(L,Y) $\wedge \{Y_{(i \leftrightarrow j)} \mid i < j < |Y| \wedge Y_i > Y_j\} = \emptyset$

$\equiv \text{if } \forall i < j < |S|. S_i \leq S_j \text{ then } S$

else *sort*_{LS}(L, arb($\{S_{(i \leftrightarrow j)} \mid i < j < |S| \wedge S_i > S_j\}$))

- **Sortieren mit engeren Suchstrukturen**
- **Minimal Spanning Tree**
 - Gegeben: Graph mit gewichteten Kanten (Zugriffszeiten, Abstände, ...)
 - Gesucht: Baum, auf dem alle Knoten mit minimalen Kosten erreichbar
 - Initialwert: Erzeuge spannenden Baum
 - Änderungsaktion: Ergänze neue Kante, entferne eine andere
 - Feasibility Constraint: Entfernte Kante muß redundant sein
 - Optimality Constraint: Hinzugefügte Kante verteuert bisherigen Weg nicht
- **Lineare Programmierung**
 - Minimiere lineare Funktion $f(x_1, \dots, x_n) = \sum c_i x_i$ unter Restriktionen $A_j[x_1, \dots, x_n]$
Standard Darstellung: Minimiere $c \star x$ unter $A \star x = b \wedge \forall i \leq n. x_i \geq 0$
 - Initiallösung: Setze $x_{m+1}, \dots, x_n := 0$ und löse $A_{1..m} \star x_{1..m}$ mit $\forall i \leq m. x_i > 0$
durch Gauß-Verfahren
 - Änderungsaktion: Setze ein $x_i := 0$, wähle neues $x_j \neq 0$
 - Feasibility Constraint: Alte + neue x -Komponenten nach Lösung positiv
 - Optimality Constraint: Relative Kosten steigen durch Veränderung nicht

Wissensbasierte Techniken zur Softwareentwicklung

- **Erzeugte Algorithmen sind korrekt und effizient**
 - Formales theoretisches Fundament sichert Korrektheit
 - Gute algorithmische Struktur liefert Effizienz
 - Nachträgliche Optimierung des schematischen Algorithmus möglich/nötig
 - Mathematische Notation übersetzbar in Programmiersprachen
- **Synthesetechniken sind automatisierbar**
 - Jeder Schritt basiert auf logischer Inferenz
 - Wissen steuert alle Strategien des Algorithmenentwurfs
 - Ähnliche Techniken für Entwurf verschiedener Algorithmenstrukturen
- **Techniken sind praktisch erfolgreich**
 - **KIDS** erzeugt korrekte Scheduling Algorithmen in wenigen Stunden
 - Erzeugter Lisp Code 2000 mal schneller als existierende ADA Software