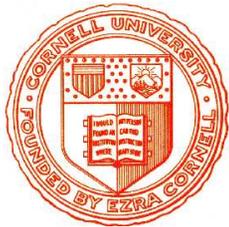


Automatisierte Logik und Programmierung

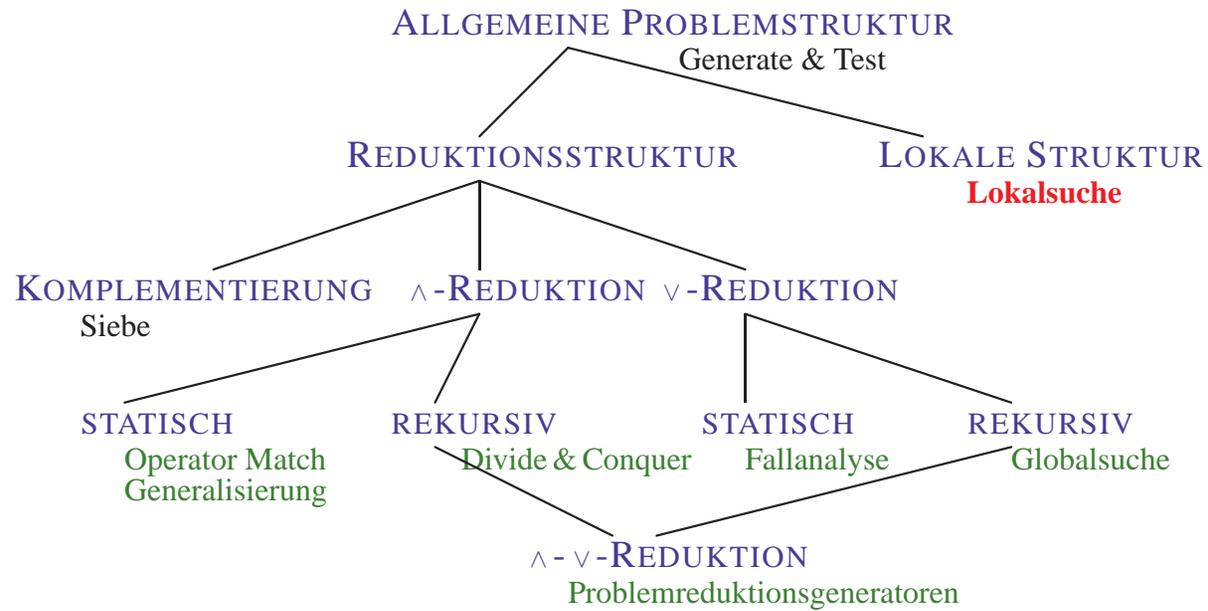
Einheit 21

Lokalsuch-Algorithmen

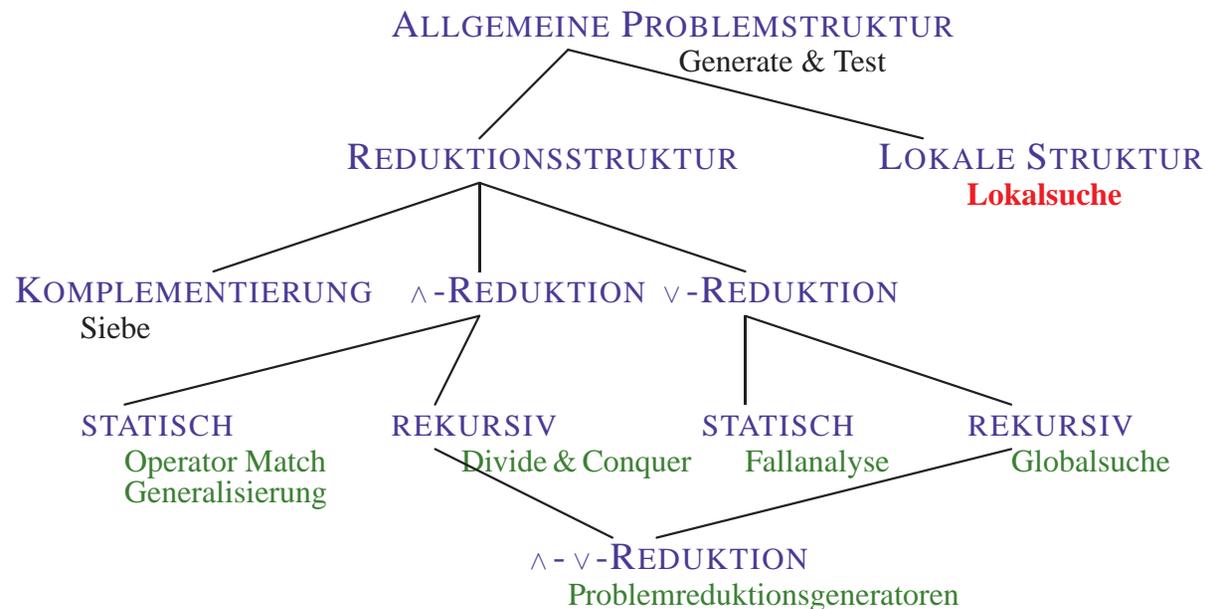


1. Algorithmenschema
2. Korrektheit
3. Wissensbasierte Unterstützung
4. Synthesestrategie

LOKALSUCH-ALGORITHMEN



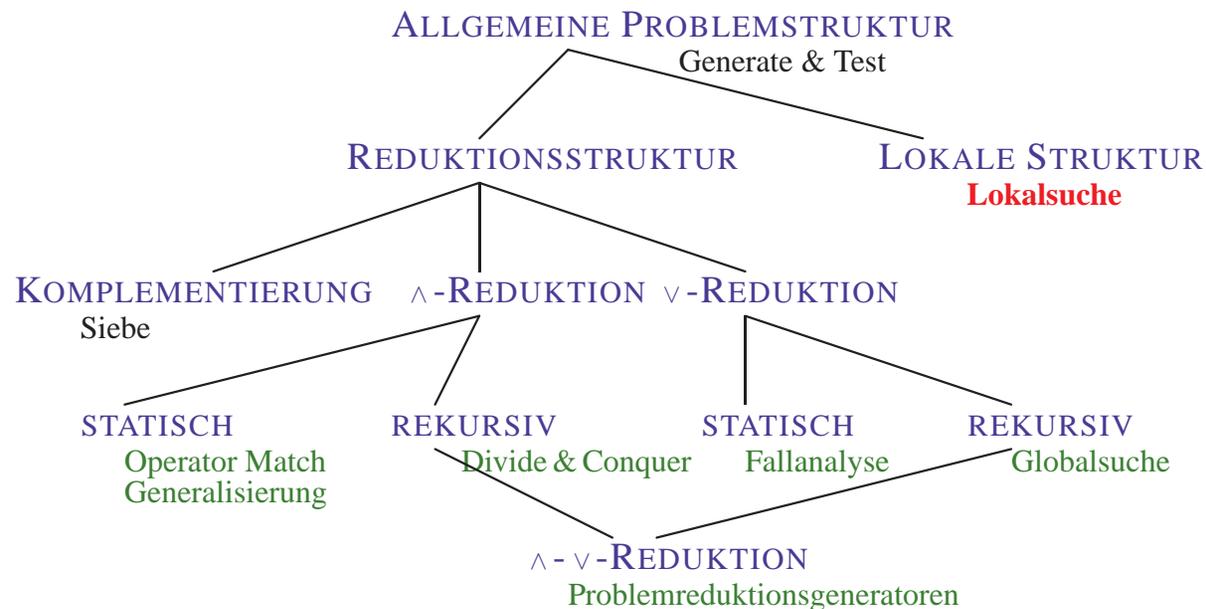
LOKALSUCH-ALGORITHMEN



● 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

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)

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

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

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 $basic := \{1..m\}$,

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

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

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 $basic := \{1..m\}$,

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

– **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 $\sum c_i z_i < \sum c_i y_i$

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

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 $basic := \{1..m\}$,

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

– **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

LOKALSUCH-ALGORITHMEN: EINHEITLICHE DARSTELLUNG

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

Initialisiere: Setze $\text{basic} := \{1..m\}$, löse $\sum_{i \in \text{basic}} A_{i,j} y_i = b_j$ durch Gauß-Verfahren

Optimiere y, basic :

- Wähle $i \in \text{basic}, k \notin \text{basic}$ so, daß $\sum_{i \in \text{basic}'} A_{i,j} z_i = b_j$ 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 $basic := \{1..m\}$, löse $\sum_{i \in basic} A_{i,j} y_i = b_j$ durch Gauß-Verfahren

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

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 $basic := \{1..m\}$, löse $\sum_{i \in basic} A_{i,j} y_i = b_j$ durch Gauß-Verfahren

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  $f_{LS}(x, \mathit{Init}[x])$ 
```

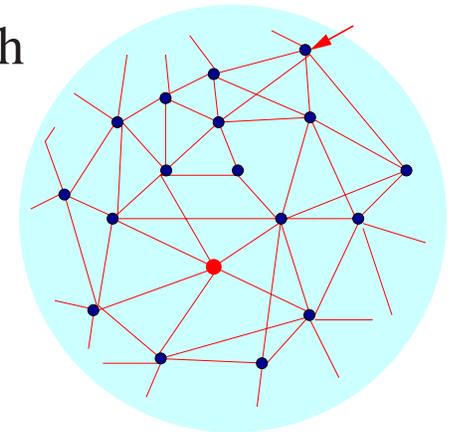
Bessere Werte z werden jeweils in einer lokal beschränkten Umgebung von y gesucht

- **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

LOKALSUCH-ALGORITHMEN: GENERELLE IDEE

- **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**



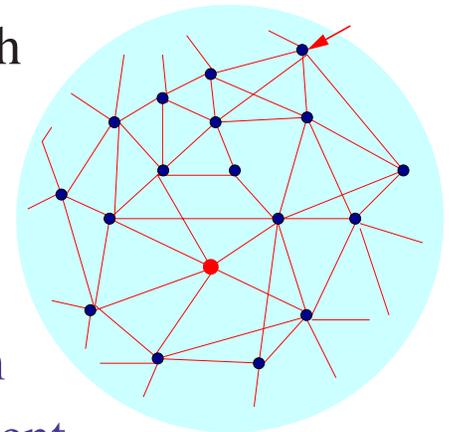
LOKALSUCH-ALGORITHMEN: GENERELLE IDEE

- **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

GRUNDSHEMA VON LOKALSUCH-ALGORITHMEN

• 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 $f_{LS}(x, \text{Init}[x])$

GRUNDSHEMA VON LOKALSUCH-ALGORITHMEN

● 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-SCHEMAS

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

KORREKTHEIT DES LOKALSUCH-SCHEMAS

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 $O[x,y]$

KORREKTHEIT DES LOKALSUCH-SCHEMAS

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

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

KORREKTHEIT DES LOKALSUCH-SCHEMAS

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

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

KORREKTHEIT DES LOKALSUCH-SCHEMAS

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 $f_{LS}(x, arb(\{t \mid t \in N[x,z] \wedge O[x,t] \wedge c[x,t] < c[x,z]\}))$

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 $\forall t:R. (O[x, t] \Rightarrow c[x, y] \leq c[x, t])$

$$\begin{aligned} \forall x:D. I[x] &\Rightarrow O[x, Init[x]] \\ \forall x:D. \forall z:R. I[x] \wedge O[x, z] &\Rightarrow O[x, f_{ls}[x, z]] \\ &\wedge \forall t \in N[x, f_{ls}[x, z]]. (O[x, t] \Rightarrow c[x, f_{ls}[x, z]] \leq c[x, t]) \\ \forall x:D. \forall y:R. I[x] \wedge O[x, y] &\Rightarrow (\forall t \in N[x, y]. O[x, t] \Rightarrow c[x, y] \leq c[x, t]) \\ &\Rightarrow \forall z:R. (O[x, z] \Rightarrow c[x, y] \leq c[x, z]) \end{aligned}$$

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 $\forall t \in N[x, z]. (O[x, t] \Rightarrow c[x, z] \leq c[x, t])$

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**

$$\forall x:D. \exists z:R. O[x, z] \wedge c[x, z] = \min\{c[x, t] \mid t \in R \wedge O[x, t]\}$$

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

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**

BEISPIEL: SORTIEREN ALS LOKALSUCHE

- **Formuliere Sortieren als Ordnungsoptimierung**

- Einfache Basisspezifikation $O[L, S] = \text{rearranges}(L, S)$

- Kostenfunktion $c[L, S] = \#_{>}(S)$: Anzahl der Fehlstellungen $S_i > S_j$ ($i < j$)

Es gilt $\#_{>}(S) \geq 0$ und $\#_{>}(S) = 0 \Rightarrow \text{ordered}(S)$

BEISPIEL: SORTIEREN ALS LOKALSUCHE

- **Formuliere Sortieren als Ordnungsoptimierung**

- Einfache Basisspezifikation $O[L, S] = \text{rearranges}(L, S)$

- Kostenfunktion $c[L, S] = \#_{>}(S)$: Anzahl der Fehlstellungen $S_i > S_j$ ($i < j$)

Es gilt $\#_{>}(S) \geq 0$ und $\#_{>}(S) = 0 \Rightarrow \text{ordered}(S)$

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

SUCH THAT `rearranges(L, S)`

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

BEISPIEL: SORTIEREN ALS LOKALSUCHE

• Formuliere Sortieren als Ordnungsoptimierung

- Einfache Basisspezifikation $O[L, S] = \text{rearranges}(L, S)$
- Kostenfunktion $c[L, S] = \#_{>}(S)$: Anzahl der Fehlstellungen $S_i > S_j$ ($i < j$)
Es gilt $\#_{>}(S) \geq 0$ und $\#_{>}(S) = 0 \Rightarrow \text{ordered}(S)$
- Spezifikation `FUNCTION sort(L:Seq(Z)):Seq(Z) RETURNS S`
 `SUCH THAT rearranges(L,S)`
 $\wedge \forall S' : \text{Seq}(\mathbb{Z}). \text{rearranges}(L, S') \Rightarrow \#_{>}(S) \leq \#_{>}(S')$

• Einfacher Lokalsuchalgorithmus

- Initillösung $\text{Init}[L] = L$ ↪ Axiom 1
- Nachbarschaft $N[S, S'] = \text{permute}_{i,j}(S, S')$: vertausche S_i und S_j
 - Vertauschen ist reflexiv, Lokale Minima sind exakt ↪ Axiom 2,3
 - Alle Umordnungen erreichbar durch iteratives Vertauschen ↪ Axiom 4

BEISPIEL: SORTIEREN ALS LOKALSUCHE

● Formuliere Sortieren als Ordnungsoptimierung

- Einfache Basisspezifikation $O[L, S] = \text{rearranges}(L, S)$
- Kostenfunktion $c[L, S] = \#_>(S)$: Anzahl der Fehlstellungen $S_i > S_j$ ($i < j$)
Es gilt $\#_>(S) \geq 0$ und $\#_>(S) = 0 \Rightarrow \text{ordered}(S)$
- Spezifikation `FUNCTION sort(L:Seq(Z)):Seq(Z) RETURNS S`
`SUCH THAT rearranges(L,S)`
 $\wedge \forall S' : \text{Seq}(\mathbb{Z}). \text{rearranges}(L, S') \Rightarrow \#_>(S) \leq \#_>(S')$

● Einfacher Lokalsuchalgorithmus

- Initillösung $\text{Init}[L] = L$ ↪ Axiom 1
- Nachbarschaft $N[S, S'] = \text{permute}_{i,j}(S, S')$: vertausche S_i und S_j
 - Vertauschen ist reflexiv, Lokale Minima sind exakt ↪ Axiom 2,3
 - Alle Umordnungen erreichbar durch iteratives Vertauschen ↪ Axiom 4
- Ergibt Sortieren durch beliebiges Austauschen von Elementen
 - Ineffizient, da zu viele Nachbarn zu prüfen (vermutlich $\mathcal{O}(n^3)$)

BEISPIEL: SORTIEREN ALS LOKALSUCHE

• Formuliere Sortieren als Ordnungsoptimierung

- Einfache Basisspezifikation $O[L, S] = \text{rearranges}(L, S)$
- Kostenfunktion $c[L, S] = \#_{>}(S)$: Anzahl der Fehlstellungen $S_i > S_j$ ($i < j$)
Es gilt $\#_{>}(S) \geq 0$ und $\#_{>}(S) = 0 \Rightarrow \text{ordered}(S)$
- Spezifikation `FUNCTION sort(L:Seq(Z)):Seq(Z) RETURNS S`
`SUCH THAT rearranges(L,S)`
 $\wedge \forall S' : \text{Seq}(\mathbb{Z}). \text{rearranges}(L, S') \Rightarrow \#_{>}(S) \leq \#_{>}(S')$

• Einfacher Lokalsuchalgorithmus

- Initillösung $\text{Init}[L] = L$ ↪ Axiom 1
- Nachbarschaft $N[S, S'] = \text{permute}_{i,j}(S, S')$: vertausche S_i und S_j
 - Vertauschen ist reflexiv, Lokale Minima sind exakt ↪ Axiom 2,3
 - Alle Umordnungen erreichbar durch iteratives Vertauschen ↪ Axiom 4
- Ergibt **Sortieren durch beliebiges Austauschen** von Elementen
 - Ineffizient, da zu viele Nachbarn zu prüfen (vermutlich $\mathcal{O}(n^3)$)

• Lokalsuchalgorithmus mit kleinerer Nachbarschaft

- Restriktion auf $N[S, S'] = \text{perm}_i(S, S') = \text{permute}_{i,i+1}(S, S')$
- Ergibt **Bubblesort** (nach algorithmischer Optimierung)

SYNTHESE VON LOKALSUCH-ALGORITHMEN

Erzeuge effiziente Nachbarschaftsstrukturen

Erzeuge effiziente Nachbarschaftsstrukturen

- **Bestimme fundamentale Nachbarschaftsstruktur auf R**
 - Beschreibe Nachbarschaft als **Perturbation** (Verwirbelung)
 - Inkrementelle Veränderung von Werten aus R
 - Numerik: δ -Vektoren,
 - Kombinatorik: Austausch von Komponenten

Erzeuge effiziente Nachbarschaftsstrukturen

- **Bestimme fundamentale Nachbarschaftsstruktur auf R**
 - Beschreibe Nachbarschaft als **Perturbation** (Verwirbelung)
 - Inkrementelle Veränderung von Werten aus R
 - Numerik: δ -Vektoren,
 - Kombinatorik: Austausch von Komponenten
 - 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)

Erzeuge effiziente Nachbarschaftsstrukturen

- **Bestimme fundamentale Nachbarschaftsstruktur auf R**
 - Beschreibe Nachbarschaft als **Perturbation** (Verwirbelung)
 - Inkrementelle Veränderung von Werten aus R
 - Numerik: δ -Vektoren,
 - Kombinatorik: Austausch von Komponenten
 - 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)
- **Optimiere Nachbarschaftsstruktur durch Suchfilter**
 - Optimiere Nachbarschaftsstruktur durch frühzeitiges Abschneiden
 - **Feasibility Constraint** für $O[x, y]$
 - **Optimality Constraint** für $\forall t \in N[x, z]. (O[x, t] \Rightarrow c[x, z] \leq c[x, t])$
 - ggf. **Global Optimality Constraint** für exakte Lösungen

Verwende vorformuliertes Programmierwissen

Verwende vorformuliertes Programmierwissen

- **Lokalsuchtheorie: allgemeine Suchstruktur für R**
 - Vorgefertigte Nachbarschaftsstruktur, die *Axiome 2–4* erfüllt
 - Formalisiert als Objekt $\mathcal{L} = (D, R, I, O, \pi, \textit{Action})$
 - Wissensbank speichert Lokalsuchtheorien für Grunddatentypen

Verwende vorformuliertes Programmierwissen

- **Lokalsuchtheorie: allgemeine Suchstruktur für R**
 - Vorgefertigte Nachbarschaftsstruktur, die *Axiome 2–4* erfüllt
 - Formalisiert als Objekt $\mathcal{L} = (D, R, I, O, \pi, Action)$
 - Wissensbank speichert Lokalsuchtheorien für Grunddatentypen
- **z.B. Umordnung von Folgen über Datentyp α**
 - Suche $\hat{=}$ *Permutation einzelner Elemente* einer Folge
 - Änderungsparameter: *Indizes* der Eingabeliste L
 - Perturbation: *Vertauschung zweier Elemente* einer Ausgabeliste S

Verwende vorformuliertes Programmierwissen

- **Lokalsuchtheorie: allgemeine Suchstruktur für R**
 - Vorgefertigte Nachbarschaftsstruktur, die *Axiome 2–4* erfüllt
 - Formalisiert als Objekt $\mathcal{L} = (D, R, I, O, \pi, Action)$
 - Wissensbank speichert Lokalsuchtheorien für Grunddatentypen
- **z.B. Umordnung von Folgen über Datentyp α**
 - Suche $\hat{=}$ *Permutation einzelner Elemente* einer Folge
 - Änderungsparameter: *Indizes* der Eingabeliste L
 - Perturbation: *Vertauschung zweier Elemente* einer Ausgabeliste S

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

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

- **Teilmengen fester Größe**

- Suche $\hat{=}$ Austausch einzelner Elemente einer Menge
- Änderungsparameter: Elemente der Ein- und Ausgabemenge
- Perturbation: Austausch zweier Elemente in Ausgabemenge

STANDARD-LOKALSUCHTHEORIE FÜR MENGEN

• Teilmengen fester Größe

- Suche $\hat{=}$ Austausch einzelner Elemente einer Menge
- Änderungsparameter: Elemente der Ein- und Ausgabemenge
- Perturbation: Austausch zweier Elemente in Ausgabemenge

$$\begin{array}{llll} \text{LS_subsets}(\alpha) \equiv & D & \mapsto & \text{Set}(\alpha) \times \mathbb{N} \\ & R & \mapsto & \text{Set}(\alpha) \\ & I & \mapsto & \lambda S, m. m \leq |S| \\ & O & \mapsto & \lambda S, m, S'. S' \subseteq S \wedge |S'| = m \\ & \pi & \mapsto & \lambda S, m, S'. (S \setminus S', S') \\ & \text{Action} & \mapsto & \lambda i, j, S, m, S'. (S' \cup \{i\}) - j \end{array}$$

Spezialisiere Lokalsuchtheorien der Wissensbank

Spezialisiere Lokalsuchtheorien der Wissensbank

- **Spezialisierungsmechanismen**

- Synthetisiere Initiallösung *Init*[x] für Spezifikation $spec = (D, R, I, O)$
- Wähle \mathcal{L} für Bildbereich R , so daß $spec \ll spec_{\mathcal{L}}$ beweisbar
- Extrahiere Substitution $\vartheta: D \rightarrow D_{\mathcal{L}}$ und spezialisiere \mathcal{L} mit ϑ
 - Ergibt Nachbarschaftsstruktur N für Problemstellung

Spezialisiere Lokalsuchtheorien der Wissensbank

● Spezialisierungsmechanismen

- Synthetisiere Initiallösung *Init*[*x*] für Spezifikation $spec = (D, R, I, O)$
- Wähle \mathcal{L} für Bildbereich R , so daß $spec \ll spec_{\mathcal{L}}$ beweisbar
- Extrahiere Substitution $\vartheta: D \rightarrow D_{\mathcal{L}}$ und spezialisiere \mathcal{L} mit ϑ
 - Ergibt Nachbarschaftsstruktur N für Problemstellung

● Beispiel einer Spezialisierung, z.B. Bubblesort

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

- Wähle LS-Theorie $\mathcal{L} = LS_seq_re(\mathbb{Z})$
- Wegen $O_{\mathcal{L}}(L, S) = rearranges(L, S)$ folgt $spec \ll spec_{\mathcal{L}}$ mit $\vartheta = id$

OPTIMIERUNG VON SUCHSTRUKTUREN DURCH FILTER

Elimiere überflüssige Kandidaten aus Nachbarschaft

Elimiere überflüssige Kandidaten aus Nachbarschaft

- **Lösungs-Filter FC für \mathcal{L}_ϑ** *(Feasibility Constraint)*
 - Eliminiert Punkte, die keine akzeptablen Lösungen bzgl. O sind
 - $I[x] \wedge O[x, y] \wedge O[x, Action[i, j, \vartheta(x), y]] \Rightarrow FC[i, j, x, y]$
 - Entspricht notwendigen Filtern der Globalsuche

Eliminiere überflüssige Kandidaten aus Nachbarschaft

- **Lösungs-Filter FC für \mathcal{L}_ϑ** (*Feasibility Constraint*)
 - Eliminiert Punkte, die keine akzeptablen Lösungen bzgl. O sind
 - $I[x] \wedge O[x, y] \wedge O[x, Action[i, j, \vartheta(x), y]] \Rightarrow FC[i, j, x, y]$
 - Entspricht notwendigen Filtern der Globalsuche
- **Lokale Optimalitäts-Filter OC für \mathcal{L}_ϑ** (*Optimality Constraint*)
 - Eliminiert kostengünstigere Punkte von weiterer Betrachtung
 - $I[x] \wedge O[x, y] \wedge O[x, Action[i, j, \vartheta(x), y]]$
 $\wedge c[x, y] \leq c[x, Action[i, j, \vartheta(x), y]] \Rightarrow OC[i, j, x, y]$
 - Lokale Optima müssen Bedingung $\forall i, j \in \pi(x, y). OC(i, j, x, y)$ erfüllen

Elimiere überflüssige Kandidaten aus Nachbarschaft

- **Lösungs-Filter FC für \mathcal{L}_ϑ** (*Feasibility Constraint*)
 - Eliminiert Punkte, die keine akzeptablen Lösungen bzgl. O sind
 - $I[x] \wedge O[x, y] \wedge O[x, Action[i, j, \vartheta(x), y]] \Rightarrow FC[i, j, x, y]$
 - Entspricht notwendigen Filtern der Globalsuche
- **Lokale Optimalitäts-Filter OC für \mathcal{L}_ϑ** (*Optimality Constraint*)
 - Eliminiert kostengünstigere Punkte von weiterer Betrachtung
 - $I[x] \wedge O[x, y] \wedge O[x, Action[i, j, \vartheta(x), y]]$
 $\wedge c[x, y] \leq c[x, Action[i, j, \vartheta(x), y]] \Rightarrow OC[i, j, x, y]$
 - Lokale Optima müssen Bedingung $\forall i, j \in \pi(x, y). OC(i, j, x, y)$ erfüllen
- **Exaktheits-Filter GOC** (*Global Optimality Constraint*)
 - Eliminiere suboptimale Lösungen (*Algorithmus wird ggf. weniger effizient*)
 - $I[x] \wedge O[x, y] \wedge O[x, Action[i, j, \vartheta(x), y]]$
 $\wedge c[x, y] \leq c[x, Action[i, j, \vartheta(x), y]] \wedge \forall t:R. O[x, t] \Rightarrow c[x, y] \leq c[x, t]$
 $\Rightarrow GOC[x, y]$

SYNTHESESTRATEGIE FÜR LOKALSUCH-ALGORITHMEN

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

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 Ausgabotyp R (Wissensbank)

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 Ausgabotyp R (Wissensbank)

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

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

$$R \subseteq R' \wedge \forall x:D. I[x] \Rightarrow \exists x':D'. (I'[x'] \wedge \forall y:R. O[x,y] \Rightarrow O'[x',y'])$$

$$\pi_{\vartheta}[x,y] = \pi[\vartheta(x), y]$$

$$Action_{\vartheta}[i,j,x,y] = Action[i,j,\vartheta(x), y]$$

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 Ausgabotyp 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 Initiallösung $Init$ für Spezifikation (Standardsynthese)

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

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 Ausgabotyp 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 Initiallösung $Init$ 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} (Feasibility Constraint)

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

– FC ergibt sich durch Vereinfachung der linken Seite (Vorwärtsinferenz)

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 Ausgabotyp 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 Initiallösung $Init$ 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} (Feasibility Constraint)

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

– FC ergibt sich durch Vereinfachung der linken Seite (Vorwärtsinferenz)

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

$I[x] \wedge O[x,y] \wedge O[x, Action[i,j,\vartheta(x),y]] \wedge c[x,y] \leq c[x, Action[i,j,\vartheta(x),y]]$
 $\Rightarrow OC[i,j,x,y]$

– OC ergibt sich durch Vereinfachung der linken Seite (Vorwärtsinferenz)

6. Instantiiere Schema für suboptimale Lokalsuch Algorithmen

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 $\forall i, j \in \pi[\vartheta(x), z]. FC[i, j, x, z] \wedge O[x, Action[i, j, \vartheta(x), y]]$
 $\Rightarrow OC[i, j, x, z] \wedge c[x, z] \leq c[x, Action[i, j, \vartheta(x), y]]$ then z
 else $f_{LS}(x, arb(\{ Action[i, j, \vartheta(x), y] \mid i, j \in \pi[\vartheta(x), z].$
 $\wedge FC[i, j, x, z] \wedge O[x, Action[i, j, \vartheta(x), y]]$
 $\wedge \neg(OC[i, j, x, z] \wedge c[x, z] \leq c[x, Action[i, j, \vartheta(x), y]]) \}))$

6. Instantiiere Schema für suboptimale Lokalsuch Algorithmen

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 $\forall i,j \in \pi[\vartheta(x),z]. FC[i,j,x,z] \wedge O[x, Action[i,j,\vartheta(x),y]]$
 $\Rightarrow OC[i,j,x,z] \wedge c[x,z] \leq c[x, Action[i,j,\vartheta(x),y]]$ then z
 else $f_{LS}(x, arb(\{ Action[i,j,\vartheta(x),y] \mid i,j \in \pi[\vartheta(x),z].$
 $\wedge FC[i,j,x,z] \wedge O[x, Action[i,j,\vartheta(x),y]]$
 $\wedge \neg(OC[i,j,x,z] \wedge c[x,z] \leq c[x, Action[i,j,\vartheta(x),y]]) \}))$

Nachgeschaltete Lösungs- und Optimalitätstests notwendig für Korrektheit

- Abarbeitung nutzt **andthen/orelse Semantik** von \wedge und \vee aus
- Algorithmus testet nur Parameter, welche die Filter FC und OC passieren

SYNTHESESTRATEGIE FÜR LOKALSUCH-ALGORITHMEN

6. Instantiiere Schema für suboptimale Lokalsuch Algorithmen

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 $\forall i,j \in \pi[\vartheta(x),z]. FC[i,j,x,z] \wedge O[x, Action[i,j,\vartheta(x),y]]$
 $\Rightarrow OC[i,j,x,z] \wedge c[x,z] \leq c[x, Action[i,j,\vartheta(x),y]]$ then z
else $f_{LS}(x, arb(\{ Action[i,j,\vartheta(x),y] \mid i,j \in \pi[\vartheta(x),z].$
 $\wedge FC[i,j,x,z] \wedge O[x, Action[i,j,\vartheta(x),y]]$
 $\wedge \neg(OC[i,j,x,z] \wedge c[x,z] \leq c[x, Action[i,j,\vartheta(x),y]]) \})$)

Nachgeschaltete Lösungs- und Optimalitätstests notwendig für Korrektheit

- Abarbeitung nutzt **andthen/orelse Semantik** von \wedge und \vee aus
- Algorithmus testet nur Parameter, welche die Filter FC und OC passieren

7. Generiere ggf. Bedingungen für Exaktheit *(Global Optimality Constraint)*

- Oft Verzicht auf Exaktheit zugunsten **effizienterer Algorithmen**

SYNTHESE EINES LOKALSUCH-SORTIERALGORITHMUS

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

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 $\mathcal{L} = \text{LS_seq_re}(\mathbb{Z})$

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 Ausgabety R $\mathcal{L} = \text{LS_seq_re}(\mathbb{Z})$
2. Beweise $(D,R,I,O) \ll \text{spec}_{\mathcal{L}}$ liefert $\vartheta = \text{id}$

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 Ausgabety R $\mathcal{L} = \text{LS_seq_re}(\mathbb{Z})$
2. Beweise $(D,R,I,O) \ll \text{spec}_{\mathcal{L}}$ liefert $\vartheta = \text{id}$
3. Synthetisiere Initiallösung *Init* für Spezifikation setze `Init(L) = L`

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 Ausgabety R $\mathcal{L} = \text{LS_seq_re}(\mathbb{Z})$
2. Beweise $(D,R,I,O) \ll \text{spec}_{\mathcal{L}}$ 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*)
`rearranges(L,S) \wedge rearranges(L,S(i \leftrightarrow j)) \Rightarrow FC[i,j,L,S]`
– Vereinfachung der linken Seite ergibt `FC[i,j,L,S]=true`

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 $(D, R, I, O) \ll \text{spec}_{\mathcal{L}}$ 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*)
`rearranges(L, S) \wedge rearranges(L, S(i \leftrightarrow j)) \Rightarrow FC[i, j, L, S]`
– Vereinfachung der linken Seite ergibt `FC[i, j, L, S] = true`
5. Generiere Optimalitäts-Filter *OC* für \mathcal{L}_{ϑ} (*Optimality Constraint*)
`rearranges(L, S) \wedge rearranges(L, S(i \leftrightarrow j))`
 `$\wedge \#_>(S) \leq \#_>(S(i \leftrightarrow j)) \Rightarrow$ OC[i, j, L, S]`
– Vereinfachung der linken Seite ergibt `OC[i, j, L, S] = i < j \Rightarrow Si \leq Sj`

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 $(D,R,I,O) \ll \text{spec}_{\mathcal{L}}$ 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*)

`rearranges(L,S) \wedge rearranges(L,S(i \leftrightarrow j)) \Rightarrow FC[i,j,L,S]`

– Vereinfachung der linken Seite ergibt `FC[i,j,L,S]=true`

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

`rearranges(L,S) \wedge rearranges(L,S(i \leftrightarrow j))`

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

– Vereinfachung der linken Seite ergibt `OC[i,j,L,S] = i < j \Rightarrow Si \leq Sj`

6. Kein Exaktheitstest erforderlich

SYNTHESE EINES LOKALSUCH-SORTIERALGORITHMUS

6. Instantiiertes 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 sortLS(L,S:Seq( $\mathbb{Z}$ ) $\times$ Seq( $\mathbb{Z}$ )):Seq( $\mathbb{Z}$ )
  WHERE rearranges(L,S) RETURNS Y
  SUCH THAT rearranges(L,Y)
             $\wedge \forall T \in \{Y_{(i \leftrightarrow j)} \mid i, j < |Y|\}$ . (rearranges(L,T)  $\Rightarrow \#_>(Y) \leq \#_>(T)$ )
 $\equiv$  if  $\forall i, j < |S|$ . rearranges(L,S(i↔j))
       $\Rightarrow i < j \Rightarrow S_i \leq S_j \wedge \#_>(S) \leq \#_>(S_{(i \leftrightarrow j)})$  then S
    else sortLS(L, arb( $\{S_{(i \leftrightarrow j)} \mid i, j < |S| \wedge \text{rearranges}(L,S_{(i \leftrightarrow j)})$ 
       $\wedge \neg(i < j \Rightarrow S_i \leq S_j \wedge \#_>(S) \leq \#_>(S_{(i \leftrightarrow j)})) \}$ )))
```

SYNTHESE EINES LOKALSUCH-SORTIERALGORITHMUS

6. Instantiiertes 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 sortLS(L,S:Seq( $\mathbb{Z}$ ) $\times$ Seq( $\mathbb{Z}$ )):Seq( $\mathbb{Z}$ )
  WHERE rearranges(L,S) RETURNS Y
  SUCH THAT rearranges(L,Y)
             $\wedge \forall T \in \{Y_{(i \leftrightarrow j)} \mid i, j < |Y|\}$ . (rearranges(L,T)  $\Rightarrow \#_>(Y) \leq \#_>(T)$ )
 $\equiv$  if  $\forall i, j < |S|$ . rearranges(L,S(i $\leftrightarrow$ j))
       $\Rightarrow i < j \Rightarrow S_i \leq S_j \wedge \#_>(S) \leq \#_>(S_{(i \leftrightarrow j)})$  then S
    else sortLS(L, arb( $\{S_{(i \leftrightarrow j)} \mid i, j < |S| \wedge \text{rearranges}(L,S_{(i \leftrightarrow j)})$ 
       $\wedge \neg(i < j \Rightarrow S_i \leq S_j \wedge \#_>(S) \leq \#_>(S_{(i \leftrightarrow j)}))$  })))
```

7. Nach Optimierung

```
FUNCTION sort(L:Seq( $\mathbb{Z}$ )):Seq( $\mathbb{Z}$ ) ...
 $\equiv \text{sort}_{LS}(L,L)$ 
```

```
FUNCTION sortLS(L,S:Seq( $\mathbb{Z}$ ) $\times$ Seq( $\mathbb{Z}$ )):Seq( $\mathbb{Z}$ ) ...
 $\equiv$  if  $\forall i < j < |S|$ .  $S_i \leq S_j$  then S
    else sortLS(L, arb( $\{S_{(i \leftrightarrow j)} \mid i < j < |S| \wedge S_i > S_j\}$ )))
```

- **Minimal Spanning Tree**

- Gegeben: Graph mit gewichteten Kanten (Zugriffszeiten, Abstände, ...)
- Gesucht: Baum, auf dem alle Knoten mit minimalen Kosten erreichbar

- **Minimal Spanning Tree**

- Gegeben: Graph mit gewichteten Kanten (Zugriffszeiten, Abstände, ...)
- Gesucht: Baum, auf dem alle Knoten mit minimalen Kosten erreichbar
- Initialwert: Erzeuge spannenden Baum
- Perturbation: Ergänze neue Kante, entferne eine andere
- Feasibility Constraint: Entfernte Kante muß redundant sein
- Optimality Constraint: Hinzugefügte Kante ist teurer als bisheriger Weg

LOKALSUCHE: ANWENDUNGSBEISPIELE

● Minimal Spanning Tree

- Gegeben: Graph mit gewichteten Kanten (Zugriffszeiten, Abstände, ...)
- Gesucht: Baum, auf dem alle Knoten mit minimalen Kosten erreichbar
- Initialwert: Erzeuge spannenden Baum
- Perturbation: Ergänze neue Kante, entferne eine andere
- Feasibility Constraint: Entfernte Kante muß redundant sein
- Optimality Constraint: Hinzugefügte Kante ist teurer als bisheriger Weg

● 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 $A \star x = b \wedge \forall i \leq n. x_i \geq 0$

LOKALSUCHE: ANWENDUNGSBEISPIELE

● Minimal Spanning Tree

- Gegeben: Graph mit gewichteten Kanten (Zugriffszeiten, Abstände, ...)
- Gesucht: Baum, auf dem alle Knoten mit minimalen Kosten erreichbar
- Initialwert: Erzeuge spannenden Baum
- Perturbation: Ergänze neue Kante, entferne eine andere
- Feasibility Constraint: Entfernte Kante muß redundant sein
- Optimality Constraint: Hinzugefügte Kante ist teurer als bisheriger Weg

● 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
- Perturbation: 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

RÜCKBLICK: SYNTHESE MIT ALGORITHMENSCHEMATA

Wissensbasierte Techniken zur Softwareentwicklung

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

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

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