UNIKASSEL
VERSITÄT

SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

Umgang mit Nebenbedingungen (Constraint handling)

Jens Kosiol

(teilweise basierend auf Folien von Eiben/Smith)

Wofür brauchen wir Nebenbedingungen?

- Viele praktische Probleme haben Nebenbedingungen:
  - Konstruiere ein Produkt, dessen Kosten einen Fixbetrag N nicht überschreiten.
  - Optimiere die Auslastung von Hörsälen, ohne dass in einem Saal zwei Veranstaltungen gleichzeitig stattfinden oder ein Dozent zwei Kurse gleichzeitig in verschiedenen Sälen halten muss.
  - …
- Wahl der Kodierung von Lösungen kann Nebenbedingungen einführen.
- Neugier: Stellt eine interessante, herausfordernde Erweiterung des reinen Optimierens von Funktionen dar.

Wiederholung: Optimierung und CSP

	Objective function	
Constraints	**Yes**	**No**
Yes	Constrained optimisation problem	Constraint satisfaction problem
No	Free optimisation problem	No problem

Beschränktes Optimierungsproblem:
- TSP mit Nebenbedingung, dass Stadt X vor Stadt Y besucht wird
- Rucksackproblem

Freies Optimierungsproblem:
- Optimieren einer mathematischen Funktion f
- Traveling Salesman Problem

Constraint Satisfaction Problem:
- SAT
- Graphfärbbarkeit

Abhängigkeit von Darstellung

Ein Problem ist (meist) nicht an sich ein freies/beschränktes Optimierungsproblem oder ein CSP!

- Die Wahl der Darstellung eines Problems hat Einfluss darauf, welche Form das Problem hat.
- Die Wahl des Suchraums (Repräsentation von Lösungen) hat Einfluss darauf, in welcher Form sich ein Problem (gut) darstellen lässt.
- Für ein Problem lassen sich zwischen den Formen häufig (annähernd) äquivalente Transformationen finden.

Beispiel: Formalisierungen Damenproblem (Wdh.)

Als Constraint Satisfaction Problem:

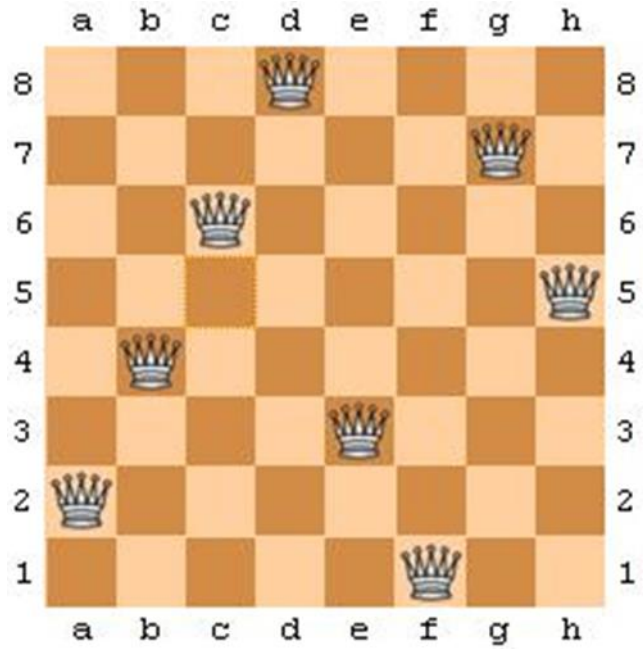
- Für jedes Feld des Schachbretts eine Formel, die sagt, dass, falls dieses Feld mit einer Dame besetzt ist, keins der durch die Dame erreichbaren Felder besetzt ist.
- Eine Formel, die ausdrückt, dass n Felder mit Damen besetzt sind

Als freies Optimierungsproblem:

- Maximiere die Funktion f, die zählt, wie viele Damen keine andere schlagen

Als beschränktes Optimierungsproblem:

- Für jedes Feld des Schachbretts eine Formel, die sagt, dass, falls dieses Feld mit einer Dame besetzt ist, keine weitere Damen in der entsprechenden Spalte und Reihe stehen.
- Funktion g, die zählt, wie viele Damen keine andere diagonal schlagen



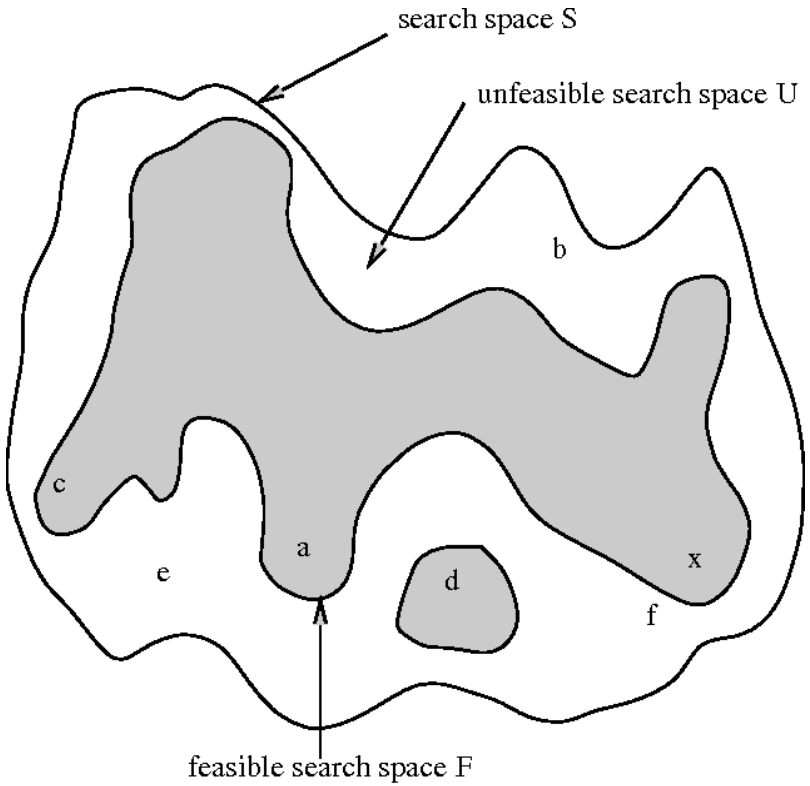
Beispiel: Kodierung TSP (Wiederholung)

Binäre Kodierung von TSP

- Städte durchnummerieren und die Nummern binär kodieren (mit fixer Länge)
- Kodierung einer Tour durch Konkatenation der einzelnen binär kodierten Nummern
- Für Anzahl der Städte $\neq 2^n$ entstehen Individuen, die (zumindest direkt) keine Lösung kodieren; Nebenbedingungen können die Ungültigkeit von Lösungen ausdrücken. (Beispiel: eine 1 an den Positionen 1 und 2 schließt eine 1 an Position 3 aus).
- Fitnessfunktion: Summe der Gewichte der Wege der Tour wird minimiert

TSP klingt wie ein freies Optimierungsproblem, durch die Wahl der Darstellung erhalten wir aber die Nebenbedingung, dass nur Binärzahlen auftauchen sollen, die auch eine Stadt kodieren.

Intuition und Nomenklatur bei eingeschränktem Suchraum



Bildquelle: [Michalewicz95]

Intuition: Bei Optimierung mit Nebenbedingungen enthält der Suchraum Regionen, die keine gültigen Lösungen enthalten.

Nomenklatur (Englisch):

- Search space (Suchraum) ist die Menge der Genotypen; die Elemente werden manchmal (candidate) solutions genannt.
- Feasible region/search space/set ist die Menge der Genotypen, die die Nebenbedingungen erfüllen.
- Genotypen, die die Nebenbedingungen nicht erfüllen, werden infeasible (solutions) genannt.

Bemerkung:

- Bei der Optimierung reeller Variablen wird die Anforderung, dass eine Variable x_i in einem Intervall $[a_i, b_i]$ liegt, normalerweise nicht als Nebenbedingung aufgefasst.

Methoden zum Umgang mit Nebenbedingungen

- Indirekter Umgang mit Nebenbedingungen: Constraints werden in Optimierungsziele transformiert
  - Berücksichtigung von Constraints in der Fitnessfunktion (z.B. Penalty für Verletzung einer Nebenbedingung)
- Direkter Umgang mit Nebenbedingungen
  - Ungültige Lösungen löschen (death penalty)
  - Reparatur
  - Variationsoperatoren entwerfen, die die Gültigkeit von Lösungen bewahren (und ggf. zusätzlich Suche mit gültigen Lösungen initialisieren)
  - Wahl einer Repräsentation, die Nebenbedingungen unnötig macht

Die Möglichkeiten schließen sich nicht aus, sondern können, gerade im Fall mehrerer Nebenbedingungen, miteinander kombiniert werden!

Indirekter Umgang mit Nebenbedingungen: Anpassung der Fitnessfunktion

- **Im indirekten Umgang mit Nebenbedingungen integriert man die Nebenbedingungen in die Fitnessfunktion und verlässt sich auf die Optimierungsfähigkeiten des evolutionären Algorithmus.** Ist f die ursprüngliche Fitnessfunktion, Φ die Konjunktion der Nebenbedingungen und x eine Lösung, so sollte für die angepasste Fitnessfunktion f_{mod} mindestens gelten:
  $f_{\mathrm{mod}}(x)$ ist optimal $\Rightarrow f(x)$ ist optimal und $\Phi(x)$ ist erfüllt
  (am besten gilt auch die Umkehrung).
- Verbreitetster Ansatz: Verletzung eines Constraints wird mit einem Penalty bestraft, der auf die Fitness addiert (Minimierungsproblem) oder von der Fitness substrahiert (Maximierungsproblem) wird.
- **Achtung:** Bei vielen beschränkten Optimierungsproblemen liegt die beste Lösung in der Nähe der Grenze zur infeasible region. Wird die Verletzung von Nebenbedingungen zu stark bestraft, wird die Grenzregion nicht (gut genug) durchsucht und gute Lösungen verpasst. (Beispiel: Rucksackproblem)

Penalty für verletzte Nebenbedingungen

Genereller Ansatz: Fitness berechnet sich als $f_{\text{mod}}(x) = f(x) + P(x)$, wobei $P(x)$ eine Strafe für die Verletzung von Nebenbedingungen berechnet (negativ für Maximierungsproblem).

- Es sollte immer gelten: $P(x) = 0 \Leftrightarrow x$ erfüllt alle Nebenbedingungen
- Hat das gegebene Problem k Nebenbedingungen, so kann sich $P(x)$ als Linearkombination $w_1 P_{1(x)} + \cdots + w_k P_k(x)$ von Penaltyfunktionen für die einzelnen Nebenbedingungen berechnen (Gewicht w_i bestimmt Bedeutung der Nebenbedingung i).
- Die Penaltyfunktion P kann linear, quadratisch, exponentiell, … sein, um den Druck auf ungültige Lösungen zu steuern.
- Die Penaltyfunktion P berücksichtigt oft, “wie stark” Nebenbedingungen verletzt sind.
  - Wie viele Constraints sind wie oft/wie schwer verletzt?
  - Wie weit ist die Entfernung (in irgendeiner passenden Metrik) zur nächsten (bekannten) gültigen Lösung?
- Feintuning: Gewichte der Penaltyfunktion können z.B. mit der Zeit adaptiert werden.

Beispiele für Penaltyfunktionen

- Binär kodiertes Damenproblem: Strafe, für Damen, die sich schlagen
- Binär kodiertes Rucksackproblem: Strafe für überschüssiges Gewicht, z.B. $cW_{\mathrm{ex}}(x)$, wobei $W_{\mathrm{ex}}(x)$ das überschüssige Gewicht einer Lösung x angibt und c eine einstellbare Konstante ist.
- Im als Graph kodierten CRA-Problem: Summiere über jedes Feature (Methoden und Attribute) auf, wie vielen Klassen dieses Feature zu viel zugewiesen wurde.
- Bei einem reellen Vektor $(v_1, \ldots, v_n)$: Euklidische Distanz zum nächsten gültigen Vektor.

Löschen ungültiger Lösungen

- Lösungen, die Nebenbedingungen verletzen, werden zurückgewiesen (death penalty), meist bei der survivor selection.
- Einfach zu programmieren, aber praktisch kaum erfolgreich:
  - Gültige Nachkommen können selten sein.
  - Manche Lösungen können unerreichbar werden.
  - Verfahren konvergiert oft mit Population nicht besonders hoher Qualität, aus der heraus nur mit geringer Wahrscheinlichkeit weitere gültige Lösungen erreichbar sind.

Reparatur

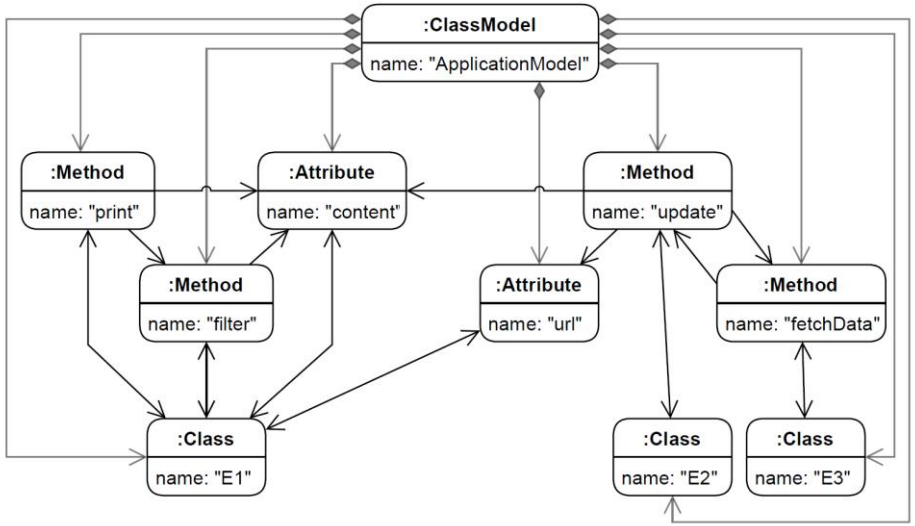
Zwei grundsätzliche Einsatzmöglichkeiten für Reparatur:

- Eine Lösung, die ein Constraint verletzt, wird auf Genotypebene repariert, also zu einer anderen Lösung transformiert (Zeitpunkt: direkt nach der Berechnung als Nachkomme).
- Für eine Lösung, die ein Constraint verletzt, wird die Dekodierung bzw. Auswertung der Fitness angepasst, sodass die Lösung doch eine gültige Lösung kodiert (Zeitpunkt: survivor selection).
  - Die adaptiert ausgewertete Lösung kann durch eine Kodierung des Phänotyps, als der sie ausgewertet wurde, ersetzt werden (Lamarckian).
  - Die adaptiert ausgewertete Lösung kann beibehalten werden (Baldwinian).

Eigenschaften:

- Wurde in beiden Varianten häufig mit Erfolg eingesetzt.
- Reparatur kann Bias erzeugen, also zur Überrepräsentation mancher Lösungen führen, wenn nur bestimmte Lösungen als Ergebnis einer Reparatur erreicht werden können.
- Wenn Reparatur nicht-deterministisch ist, ist nicht mehr eindeutig, welchen Phänotyp ein Genotyp repräsentiert.
- Für jedes Optimierungsproblem und jede Kodierung muss ein eigenes Reparaturverfahren entwickelt werden.

Beispiele Reparatur



Quelle: [KJT23]

- CRA in Graphrepräsentation: Lösche (zufällig gleichverteilt) überzählige Zuweisungen von Methoden und Attributen zu Klassen. → Reparatur auf Genotypebene
- TSP in Binärrepräsentation: Sind k Städte geben (und nicht $k = 2^n$ für irgendein n), so dekodiere die einzelnen Binärzahlen $mod\ k$. → Anpassung der Dekodierung
- Rucksackproblem in Binärrepräsentation: Lösche zufällig ausgewählte Gegenstände bis das erlaubte Gewicht nicht mehr überschritten wird. → Reparatur auf Genotypebene
- Rucksackproblem in Binärrepräsentation: Lösche Gegenstände in aufsteigender Reihenfolge ihres Wert-zu-Gewicht-Verhältnisses →Reparatur auf Genotypebene

Anpassung der Variationsoperatoren

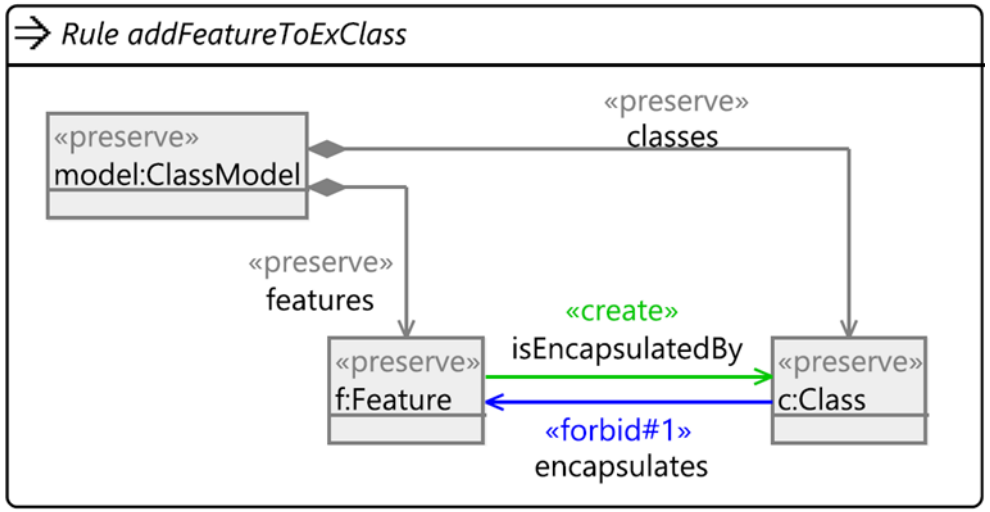
Einsatz:

- Es werden Mutations- und/oder Rekombinationsoperatoren entwickelt, die gültige Lösungen in gültige Lösungen überführen.
- Unter Umständen wird auch eine Initialisierungsmethode entwickelt, die gültige Lösungen (zufällig) generiert.

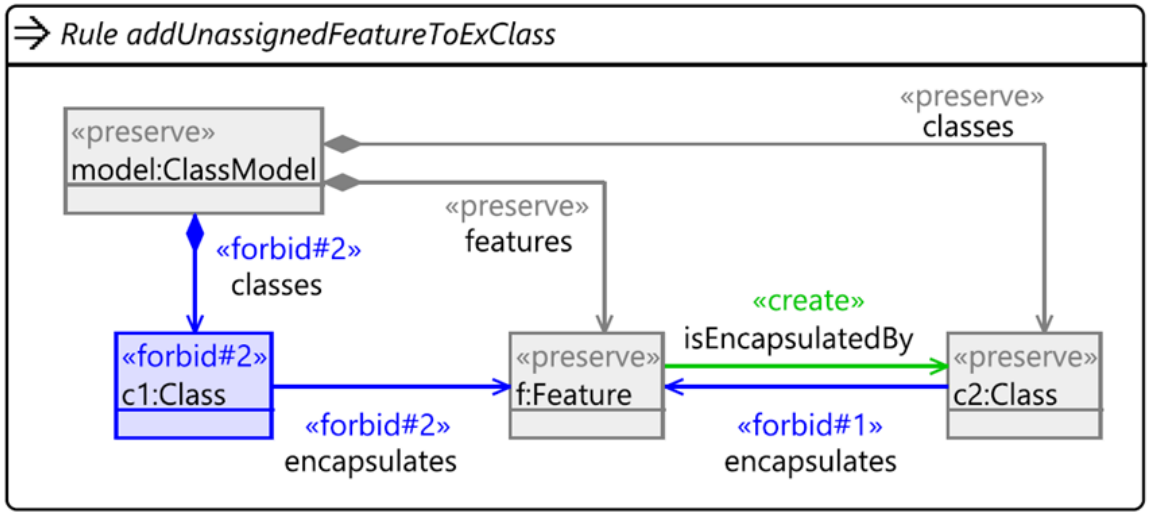
Eigenschaften:

- Wurde häufig mit Erfolg eingesetzt
- Muss für jedes Optimierungsproblem und jede Art der Kodierung neu entwickelt werden
- Die angepassten Variationsoperatoren sind oft komplexer und sorgen so für eine höhere Laufzeit pro Iteration.
- Bei Nicht-Standard-Kodierungen (z.B. Graphen), wo man ohnehin problemspezifische Operatoren entwickelt, lässt sich das gut integrieren.

Beispiel Anpassung der Variationsoperatoren CRA



Mutationsoperator, der einer Klasse ein beliebiges Feature zuweist



Mutationsoperatur, der einer Klasse ein Feature zuweist, das bisher noch nicht zugewiesen wurde

Beispiel: Anpassung des Mutationsoperators für das Rucksackproblem

Zu lösen sei das (binär kodierte) Rucksackproblem. Wir passen die Standardmutation (positionsweise unabhängiger Bitswitch gemäß gegebener Mutationswahrscheinlichkeit p_{m}) auf die folgende Art an:

1. Wir teilen die Positionen des zu mutierenden Strings disjunkt in diejenigen auf, die mit 1 besetzt sind, und diejenigen, die mit 0 besetzt sind.
2. Die Positionen mit einer 1 durchlaufen wir standardmäßig und switchen unabhängig voneinander mit Wahrscheinlichkeit p_{m} von 1 auf 0.
3. Die Positionen mit einer 0 durchlaufen wir in zufälliger Reihenfolge, switchen aber nur mit Wahrscheinlichkeit p_{m} von 0 auf 1, wenn der Erfolg nicht zu einem Überschreiten des erlaubten Gewichts führen wird.

Anpassung der Kodierung

Vorgehen:

- Das Problem wird auf eine Art und Weise kodiert, dass nur Phänotypen repräsentiert werden, die alle Nebenbedingungen erfüllen.
- Liegen für die gewählte Art der Kodierung keine Variationsoperatoren vor, müssen solche zusätzlich entworfen werden.

Eigenschaften:

- Es muss darauf geachtet werden, dass weiterhin alle (wichtigen) gültigen Lösungen
  - dargestellt werden können;
  - durch die Variationsoperatoren von anderen Lösungen aus erreicht werden können.

Beispiel: Kodierung TSP (Wiederholung)

Binäre Kodierung von TSP

- Städte durchnummerieren und die Nummern binär kodieren (mit fixer Länge)
- Kodierung einer Tour durch Konkatenation der einzelnen binär kodierten Nummern
- Für Anzahl der Städte $\neq 2^n$ entstehen Individuen, die (zumindest direkt) keine Lösung kodieren; Nebenbedingungen können die Ungültigkeit von Lösungen ausdrücken. (Beispiel: eine 1 an den Positionen 1 und 2 schließt eine 1 an Position 3 aus).
- Fitnessfunktion: Summe der Gewichte der Wege der Tour wird minimiert

Permutationskodierung von TSP

- Städte durchnummerieren
- Eine Permutation kodiert eine Tour.
- Jedes Individuum kodiert eine mögliche Tour; keine Nebenbedingungen sind nötig.
- Fitnessfunktion: Summe der Gewichte der Wege der Tour wird minimiert

Literatur

- [Michalewicz95] Zbigniew Michalewicz: A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. Evolutionary Programming 1995: 135–155
- [KJT23] Jens Kosiol, Stefan John, Gabriele Taentzer: A generic construction for crossovers of graph-like structures and its realization in the Eclipse Modeling Framework. J. Log. Algebraic Methods Program. 136 (2024). https://doi.org/10.1016/j.jlamp.2023.100909