UNI KASSEL VERSITÄT

SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

# Repräsentationen

Jens Kosiol

# Datenstrukturen für evolutionäre Algorithmen

- Auf welchen Datenstrukturen wollen wir evolutionäre Algorithmen durchführen?
  - Strings über endlichem Alphabet (Verallgemeinerung von Bitstrings)
  - Permutationen
  - Bäume und Graphen
  - Vektoren natürlicher oder reeller Zahlen
- Im Folgenden: Einführung dieser Datentypen und typischer Probleme, für die wir die entsprechende Datenstruktur als Kodierung benutzen
- Im Anschluss: Variationsoperatoren für die verschiedenen Datenstrukturen

# Gewünschte Eigenschaften Repräsentation

Die folgenden Gesichtspunkte sind bei der Wahl einer Kodierung von Bedeutung:

- Passen Genotypraum und Phänotypraum gut zueinander?
  - Lässt sich jeder Phänotyp als ein (eindeutiger) Genotyp kodieren?
  - Kodiert jeder Genotyp einen Phänotypen?
  - Lokalität: Kodieren ähnliche Genotypen auch ähnliche Phänotypen und werden ähnliche Phänotypen durch ähnliche Genotypen kodiert?
- Ist die Dekodierung bzw. Fitness effizient berechenbar?
  - Während eines evolutionären Algorithmus wird diese Operation dauernd durchgeführt.
- Stehen (effiziente) Variationsoperatoren für die gewählte Datenstruktur zur Verfügung oder können mit vertretbarem Aufwand entworfen werden?

# Beispiel

Wir betrachten das TSP Problem und wollen Lösungen binär kodieren. Unser Vorgehen:

- Städte durchnummerieren und die Nummern binär kodieren (mit fixer Länge)
- Kodierung einer Tour durch Konkatenation der einzelnen, binär kodierten Nummern

**Repräsentiert jeder Bitstring passender Länge eine Rundtour?**

# Beispiel Nicht-Lokalität

**Hamming-Distanz**

Für zwei Bitstrings $v_1, v_2$ gleicher Länge zählt die Hamming-Distanz die Anzahl ihrer unterschiedlichen Stellen.

**Beispiele für Nicht-Lokalität**

- Als natürliche Zahlen haben 0 und $2^n$ den Abstand $|2^n - 0| = 2^n$; in binärer Kodierung hingegen eine Hamming-Distanz von 1.
- Umgekehrt haben $011 \dots 1$ und $100 \dots 0$ eine Hamming-Distanz ihrer Länge aber als natürliche Zahlen einen Abstand von 1.

# Strings

Gegeben ist eine endliche Zeichenmenge $\Sigma$. Dann sind die Strings (Zeichenketten) über $\Sigma$, notiert als $\Sigma^*$, die Menge aller endlichen Zeichenketten mit Zeichen aus $\Sigma$ (mit leerem Wort als Kette der Länge 0). Mit $\Sigma^k$, für eine natürliche Zahl $k$, bezeichnen wir die Menge aller Strings der Länge $k$.

Wichtigstes Beispiel: $\Sigma = \{0,1\}$

Größe Suchraum: $|\Sigma|^k$

# Einsatz von Stringkodierung

Historisch ist die Kodierung als (Binär)String einer der verbreitetsten Genotypen für den Einsatz evolutionärer Algorithmen. Natürliche Anwendung:

$\mathbf{\Sigma = \{0, 1\}}$:

- „Auswahlprobleme“
- (Tupel von) Zahlen

$\mathbf{|\Sigma| \geq 2}$:

- Spielstrategien (mit konstant $|\mathbf{\Sigma}|$ möglichen Spielzügen pro Runde)
- Klassifikation von Elementen in $|\mathbf{\Sigma}|$ Klassen

# Auswahlprobleme

Bei einem Auswahlproblem ist eine (linear geordnete) Menge von $k$ Elementen gegeben und es muss eine optimale Auswahl unter ihnen getroffen werden.

Kodierung als Bitstrings aus $\{0,1\}^k$: Ein Bitstring $\sigma \in \{0,1\}^k$ kodiert die Auswahl derjenigen Elemente, an deren Position in $\sigma$ eine 1 steht.

**Beispiele:**

- Damenproblem: Nummeriere die Felder des $k \times k$-Schachbretts durch; ein $\sigma = (\sigma_1, \dots, \sigma_{k^2}) \in \{0,1\}^{k^2}$ kodiert, dass genau dann eine Dame auf dem $i$-ten Feld positioniert ist, wenn $\sigma_i = 1$.
- Cliquenproblem: Nummeriere die $k$ Knoten eines gegebenen Graphen $G$ durch; ein $\sigma = (\sigma_1, \dots, \sigma_k) \in \{0,1\}^k$ kodiert, dass derjenige Untergraph betrachtet wird, der von den Knoten induziert wird, an deren Position in $\sigma$ eine 1 steht.

*Die Länge der Kodierung wächst linear in der Anzahl der Elemente, die Größe des Suchraums exponentiell.*

# Kodierung von Zahlen

- Für natürliche Zahlen:
  - Unäre Kodierung (Zähle Anzahl der 1en)
  - Binäre Kodierung
  - Gray-Kodierung
- Für reelle Zahlen:
  - Floating point numbers
  - Gleichmäßige Aufteilung des betrachteten Intervalls und binäre Durchnummerierung der entstehenden Abschnitte (wie im Kapitel „SGA“ verwendet)
- Für Tupel von Zahlen: Die individuellen Kodierungen der Einträge werden konkateniert.

*Die Länge der Kodierung wächst linear (unär) bzw. logarithmisch (binär/Gray) mit der Größe der Zahl bzw. mit der geforderten Präzision; der Suchraum wächst exponentiell.*

# Gray-Code

Der Gray-Code ist eine binäre Kodierung natürlicher Zahlen für die gilt, dass benachbarte natürliche Zahlen in Gray-Kodierung immer eine Hamming-Distanz von 1 haben.

**Konstruktion:**

1. Kodiere 0 als 0 und 1 als 1.
2. Wiederhole bis benötigte Größe erreicht:
   1. Ergänze die existierende Liste von Binärstrings um sich selbst in umgekehrter Reihenfolge.
   2. Setze vor die alten Strings eine 0 und vor die neuen eine 1.

**Konstruktion aus Binärzahlen:**

1. Kodiere eine natürliche Zahl $n$ als Dualzahl $b_1$.
2. Berechne $b_2$ als Rechts-Shift von $b_1$.
3. Berechne die Gray-Kodierung $b_3$ von $n$ als bitweises Xor von $b_1$ und $b_2$.

Als Formel: $b_3 = (n)_{\text{dual}} \oplus ((n)_{\text{dual}} \gg 1)$

**Dekodierung zu Binärzahl:** Übersetzung einer Zahl $g_1 \dots g_k$ im Gray-Code zu einer Binärzahl $b_1 \dots b_k$ kann wie folgt rekursiv erfolgen:

$$b_1 = g_1$$
$$b_i = g_i \oplus b_{i-1}, \qquad \text{für } i \geq 2$$

# Unäre Codierung

In der unären Kodierung gibt die Anzahl der Einsen eines Bitstrings an, welche natürliche Zahl kodiert ist.

Kodierung: Wenn eine natürliche Zahl $k$ zu kodieren ist, schreibe $k$ Einsen hintereinander und fülle bis zur benötigten Länge mit Nullen auf.

Dekodierung: Zähle Anzahl der Einsen eines Bitstrings.

**Eigenschaften:**

- Länge der Kodierung wächst linear mit der höchsten Zahl, die kodiert werden muss.
- Ungleichmäßige Repräsentation von Phänotypen im Genotypraum: Ist $n$ die höchste natürliche Zahl, die kodiert werden muss, und $0 \leq k \leq n$, so gibt es $\binom{n}{k}$ Genotypen, die die Zahl $k$ repräsentieren.
- Nicht-Lokalität: Sind $0 \leq k_1, k_2 \leq n$ zwei natürliche Zahlen, so haben unterschiedliche Genotypen, die $k_1$ und $k_2$ repräsentieren, unterschiedliche Hamming-Distanzen zueinander.

# Kodierung von Spielstrategien

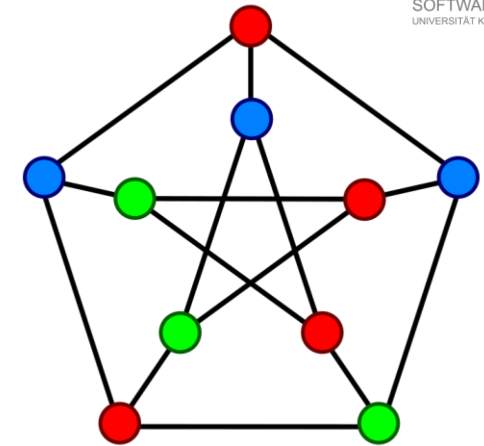
Wir betrachten ein Spiel, in dem zwei Spieler gegeneinander antreten und in jeder Runde jeweils einer von $l$ Zügen ausgewählt werden kann (Beispiel: Schere, Stein, Papier mit $l = 3$). Wähle ein Alphabet mit $|\Sigma| = l$; z.B. $\Sigma = \{1, \dots, l\}$. Wähle ein Gedächtnis von $d$ Zügen. Dann gibt es $l^{2d}$ Möglichkeiten, welche Wahlen die beiden Spieler in den letzten $d$ Runden getroffen haben können; diese Möglichkeiten können wir durchnummerieren.

https://bigbangtheory.fandom.com/de/wiki/Stein,_Papier,_Schere,_Echse,_Spock

Ein String $\sigma = (\sigma_1, \dots, \sigma_{l^{2d}})$ der Länge $l^{2d}$ über $\Sigma$ kodiert dann auf folgende Weise eine Spielstrategie: Ein Spieler mit Strategie $\sigma$ spielt Zug $\sigma_i$, wenn die letzten $d$ Spielzüge der $i$-ten Möglichkeit entsprechen ($1 \leq i \leq l^{2d}$).

*Die Länge der Kodierung wächst exponentiell in der Größe des Gedächtnis.*

# Klassifikation von Elementen

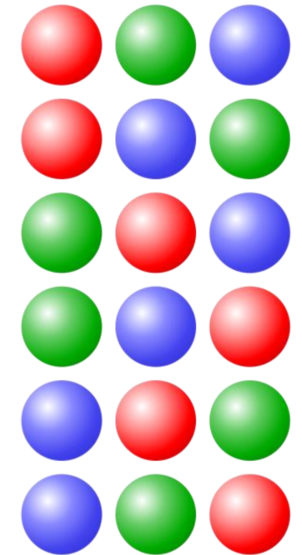


Um Versuche, einen Graphen in $n$ Farben zu färben, zu kodieren, können wir Strings aus $\{1, \dots, n\}^k$ verwenden (wobei $k$ die Anzahl der Knoten ist): Wir nummerieren die Knoten und für ein $\sigma = (\sigma_1, \dots, \sigma_k) \in \{1, \dots, n\}^k$ kodiert $\sigma_i = j$, dass der $i$-te Knoten in der Farbe $j$ gefärbt werden soll.

*Die Länge der Kodierung wächst linear in der Anzahl der Knoten, die Größe des Suchraums exponentiell.*

# Permutation

Gegeben sei eine (endliche) Menge $M$. Dann ist eine Permutation über $M$ eine Anordnung der Elemente von $M$. Schreibweise: Liste die Elemente aus $M$ in der gewünschten Reihenfolge in Klammern auf.



**Beispiel:** Für $M = \{R, B, G\}$ gibt es die Permutationen $(RGB), (RBG), (GRB), (GBR), (BRG)$ und $(BGR)$.

**Anzahl:** Von einer endlichen Menge $M$ gibt es $|M|!$ Permutationen.

# TSP per Permutationen kodiert

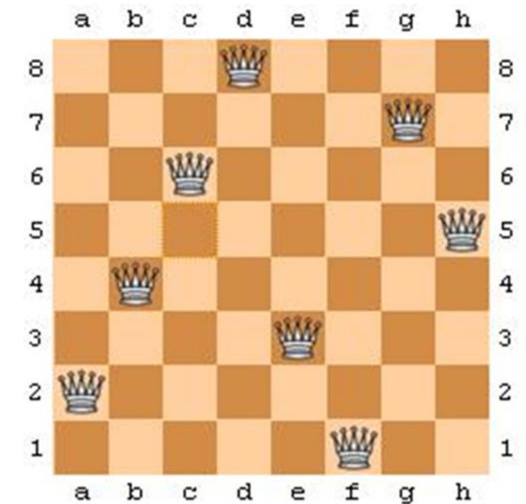
- Als Menge $M$ dient die Menge der zu besuchenden Städte.
- Eine Permutation kodiert direkt, in welcher Reihenfolge die Städte besucht werden sollen.
- Für numerische Kodierung: Ordne jeder Stadt eine Zahl zu
- Die Kodierung nutzt Domänenwissen: in einer Tour wird jede Stadt genau einmal besucht.

**Beispiel:**

| | | |
|---|---|---|
| Berlin: | 1 | Permutation (231) kodiert die |
| München: | 2 | Tour München → Hamburg → |
| Hamburg: | 3 | Berlin |

# Damenproblem per Permutationen kodiert

- Für ein $n \times n$-Schachbrett kodieren wir Positionierungen der Damen als Permutationen über der Menge $\{1, \dots, n\}$.
- Steht die Zahl $i$ an der Stelle $j$ einer solchen Permutation ($1 \leq i, j \leq n$), so kodiert das, dass die Dame in der $j$-ten Spalte auf dem $i$-ten Feld steht.
- Diese Kodierung nutzt Domänenwissen aus, nämlich, dass in einer Lösung des Problems in jeder Spalte genau eine Dame steht.




**Beispiel:**

Die Position aus dem Bild rechts wird als (75316824) kodiert.

# Workflow-Optimierung

Gegeben seien $n$ Aufgaben, die abgearbeitet werden müssen. Jede Reihenfolge, diese Aufgaben zu bearbeiten, ist mit spezifischen Kosten verbunden, die optimiert werden sollen.

Permutationen bieten eine natürliche Kodierung für Lösungen:

- Nummeriere die Aufgaben durch.
- Jede Permutation über $\{1, \dots, n\}$ kodiert eine Reihenfolge, die Aufgaben abzuarbeiten.

# Reihenfolge vs. Nachbarschaft

Bei Verwendung einer Kodierung als Permutation die Reihenfolge oder die Nachbarschaft von Elementen eine entscheidende Rolle spielen.

**Reihenfolge**

- Die Reihenfolge, in der die Elemente auftauchen, ist entscheidend.
- Elemente $(1234)$ und $(2341)$ können eine höchst unterschiedliche Fitness haben.
- Beispiel: Optimierung von Workflows.

**Nachbarschaft**

- Entscheidend ist, welche Elemente nebeneinander auftauchen.
- Elemente $(1234)$, $(2341)$ und vielleicht sogar $(4321)$ kodieren den gleichen Phänotyp.
- Beispiel: (symmetrisches) TSP – Startpunkt einer Tour hat keinen Einfluss auf Länge.

# Vergleich Permutation vs. Bitstrings

**Permutation**

- Nutzt oft Domänenwissen (Wissen über das Optimierungsproblem)
- Oft kompakt
  - Damenproblem und TSP: Kodierung der Länge $n$ mit Suchraum der Größe $n!$
- Variationsoperatoren?

**Bitstrings**

- Oft relativ einfach zu entwerfen
- Oft nicht kompakt und Suchraum enthält ungeeignete Lösungskandidaten
  - Damenproblem: Kodierung der Länge $n^2$ mit Suchraum der Größe $2^{2n}$; TSP: $n\log(n)$ bzw. $2^{n\log(n)}$
  - Viele Lösungskandidaten ungültig
- Einfach zu implementierende und effiziente Variationsoperatoren bekannt

# Symbolische Regression

Gesucht ist eine unbekannte Funktion $f: \mathbb{R}^n \to \mathbb{R}$. Gegeben ist ein Orakel, das auf Anfragen $r = (r_1, \dots, r_n) \in \mathbb{R}^n$ den Funktionswert $f(r)$ an der Stelle $r$ zurückgibt. Es werden keine weiteren Annahmen über $f$ (linear, exponentiell, …) gemacht.

**Formalisierung als Optimierungsproblem**

Phänotypraum: $\mathbb{R}^{\mathbb{R}^n}$ (Raum der Funktionen von $\mathbb{R}^n$ nach $\mathbb{R}$)

Fitnessfunktion: Wähle $k$ Stützstellen $r_1, \dots, r_k \in \mathbb{R}^n$ und erfrage $f(r_1), \dots, f(r_k)$. Die Fitness einer Funtion $g: \mathbb{R} \to \mathbb{R}$ ergibt sich als

$$\sum_{i=1}^{k} \left(g(r_i) - f(r_i)\right)^2$$

und soll minimiert werden (Methode der kleinsten Quadrate).

# Frage

Wie würdet ihr den Phänotypraum für eine symbolische Regression kodieren?

# Phänotypraum symbolische Regression

Im Vergleich zu den bisher betrachteten Optimierungsproblemen ist der Phänotypraum für die symbolische Regression überabzählbar ($|\mathbb{R}^{\mathbb{R}^n}|$).

**Konsequenz:**

- Wir müssen überlegen, welchen Teil des Phänotypraums wir überhaupt kodieren können müssen oder wollen.
- Wenn der zu kodierende Teil des Phänotypraums zumindest abzählbar unendlich sein soll, muss eine Kodierung mit flexibler Größe gewählt werden.

# Stringkodierung für symbolische Regression

Wir wählen ein Alphabet

$$\Sigma = \{\mathrm{add}, \mathrm{sub}, \mathrm{mul}, \mathrm{exp}, \sin, \dots, x_1, \dots, x_n\} \cup \mathbb{R},$$

in dem benötigte mathematische Funktionssymbole (add, sub, ...), $n$ Variablen und die reellen Zahlen vorkommen.

Als Genotypraum wählen wir $\Sigma^*$, also die Menge aller Strings über $\Sigma$.

**Dekodierung (Prefix-Notation):**

- Das erste Symbol des Strings wird die „Wurzel" des zu konstruierenden Ausdrucks.
- Das nächste Zeichen wird Eingabeparameter des letzten Funktionssymbols, das noch einen Eingabewert benötigt.

**Beispiel:**

add exp 3 sin mul $x_1$ mul 5 $x_2$ entspricht add(exp(3), sin(mul($x_1$, mul(5, $x_2$)))), also $e^3 + \sin(5x_1x_2)$.

# Probleme der Stringkodierung für symbolische Regression

- Viele Strings aus $\Sigma^*$ kodieren keine mathematische Funktion.
- Selbst wenn man eine initiale Population von Strings generiert, die tatsächlich Funktionen kodieren, ist es schwierig Variationsoperatoren zu entwerfen, die
  - Strings, die Funktionen kodieren, in Strings, die Funktionen kodieren, überführen;
  - den Suchraum umfassend durchsuchen (Strings verlängern und verkürzen)

# Graphen und Bäume

Ein (gerichteter) Graph $G = (V, E, s, t)$ besteht aus einer Menge von Knoten (vertices) $V$, einer Menge von Kanten (edges) $E$, und zwei Inzidenzfunktionen $s, t: E \to V$, die jeder Kante ihren Start ($s$) bzw. Zielknoten ($t$) zuweisen.

Ein gerichteter azyklischer Graph (directed acyclic graph – DAG) ist ein gerichteter Graph ohne Kreise (gerichtete Pfade von einem Knoten zu sich selbst).

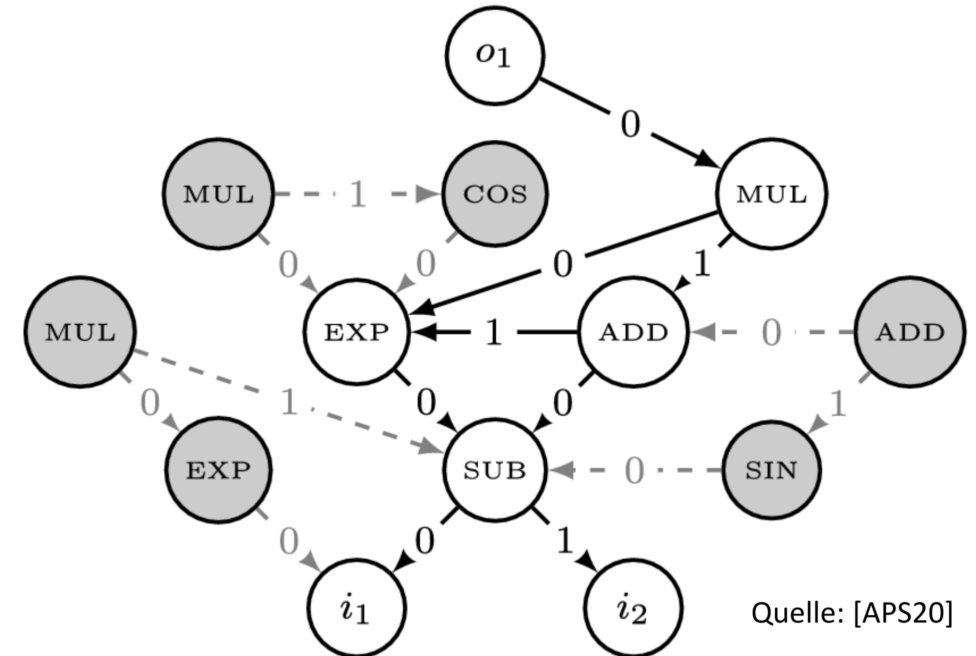
Ein Baum ist ein Graph, in dem jede zwei Knoten durch einen eindeutigen Pfad verbunden sind.

# Daten in Graphen

Gegeben sei ein Graph $G = (V, E, s, t)$.

Dann erlauben wir, dass Knoten und Kanten

- gelabelt (getypt) sind und/oder
- Attribute und deren Werte speichern.




Quelle: [APS20]

**Formal:**

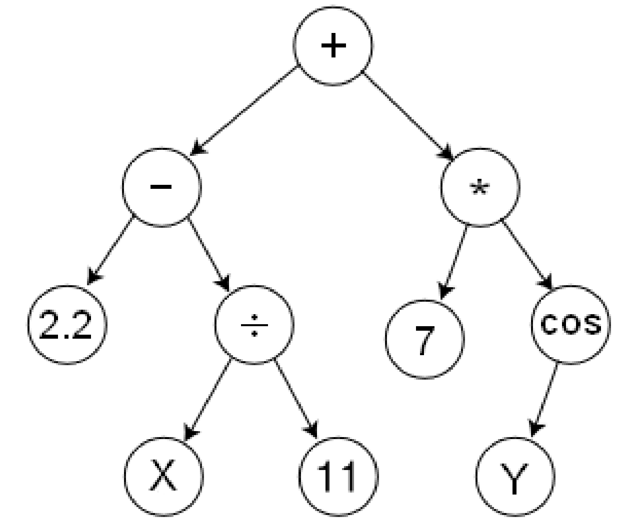
- Es gibt ein Alphabet $L_V$ und eine (partielle) Funktion $l_V: V \rightarrow L_V$, die Knoten Label zuweist.
- Es gibt ein Alphabet $L_E$ und eine (partielle) Funktion $l_E: E \rightarrow L_E$, die Kanten Label zuweist.
- Es gibt partielle Funktionen von den Knoten/Kanten, die ausgewählten Knoten/Kanten Datenwerte (von Typen `Int`, `Bool`, `String`, …) zuweisen.

# Kodierung als Baum

Gesucht sei eine Funktion $f: \mathbb{R}^n \to \mathbb{R}$.

Kodierung von Lösungen als expression trees:

- Wähle ein Alphabet mit Symbolen für grundlegende mathematische Funktionen, $n$ Variablen und Zahlen, z.B. $L = \{\mathrm{add}, \mathrm{sub}, \mathrm{mul}, \mathrm{exp}, \sin, \dots, x_1, \dots, x_n\} \cup \mathbb{R}$. Diese dienen als Label für Knoten.
- Wähle das Alphabet $L' = \{0, \dots, l\}$ als Label für Kanten (hierbei ist $l + 1$ die höchste Stelligkeit einer Funktion aus $L$).
- Nebenbedingungen:
  - Jeder Knoten und jede Kante hat ein Label; Kantenlabel kodieren die Reihenfolge der Kinder.
  - Hat ein Knoten ein Funktionssymbol $s$ als Label, so entspricht die Anzahl der ausgehenden Kanten der Stelligkeit von $s$. Diese haben die Label $0, \dots, t - 1$, wobei $t$ die Stelligkeit der Funktion $s$ ist.
  - Ein Knoten hat genau dann eine Variable oder Zahl als Label, wenn er ein Blatt des Baums ist. (Ergibt sich aus obiger Anforderung.)




(Reihenfolge durch „links vor rechts“ statt durch Nummerierung der Kanten kodiert)

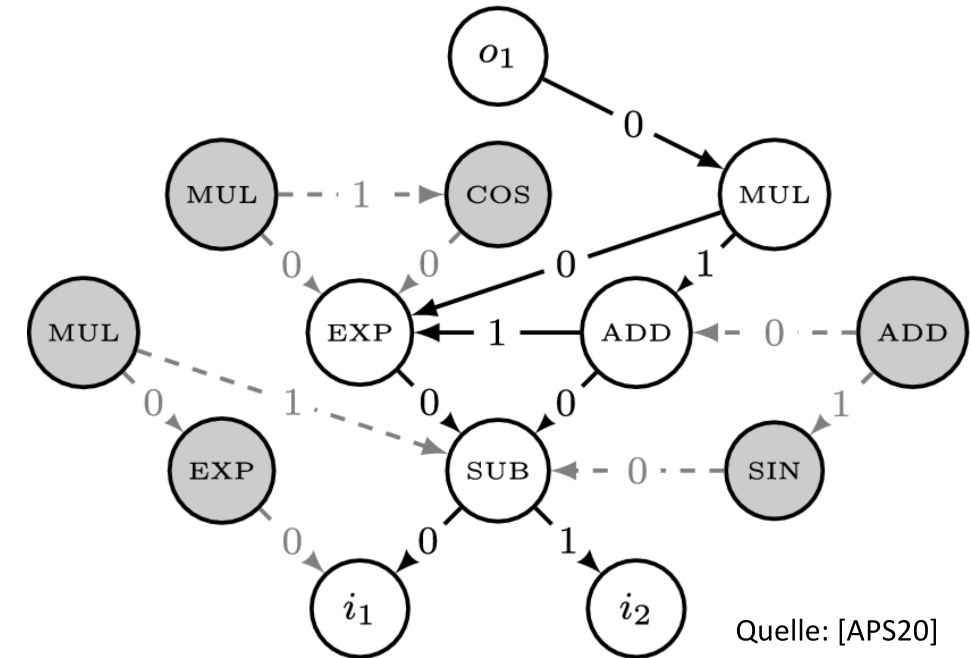
SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

# Kodierung als Graph

Gesucht sei eine Funktion $f: \mathbb{R}^n \to \mathbb{R}$.

Kodierung von Lösungen als Graphen:

- Wähle ein Alphabet mit Symbolen für grundlegende mathematische Funktionen, $n$ Variablen, ein Output-Symbol und Zahlen, z.B.
  $L = \{\text{add}, \text{sub}, \text{mul}, \text{exp}, \text{sin}, \dots, i_1, \dots, i_n, o_1\} \cup \mathbb{R}$.
  Diese dienen als Label für Knoten.
- Wähle das Alphabet $L' = \{0, \dots, l\}$ als Label für Kanten (hierbei ist $l + 1$ die höchste Stelligkeit einer Funktion aus $L$).
- Nebenbedingungen:
  - Jeder Knoten und jede Kante hat ein Label.
  - Hat ein Knoten ein Funktionssymbol $s$ als Label, so entspricht die Anzahl der ausgehenden Kanten der Stelligkeit von $s$.
  - Der Graph ist zykelfrei.
  - Der Outputknoten hat genau eine ausgehende und keine eingehende Kante.




Quelle: [APS20]

Dekodierung: Der Graph kodiert die Funktion, die von dem expression tree repräsentiert wird, den man ausgehend von dem Knoten erhält, auf den der Output-Knoten zeigt.

# Vergleich Kodierungen symbolische Regression



Gegeben sei ein gemeinsames Alphabet

$$\Sigma = \{\mathrm{add}, \mathrm{sub}, \mathrm{mul}, \mathrm{exp}, \sin, \dots, x_1, \dots, x_n\} \cup \mathbb{R},$$

das wir für jede der Kodierungen benutzen (für die Graphen kommt ein Symbol zum Markieren des Output-Knotens hinzu).

- Alle drei Kodierungen haben exakt die gleiche Ausdrucksstärke, erlauben also, die gleichen mathematisichen Funktionen zu kodieren: expression trees und Ausdrücke in Prefix-Notation entsprechen einander bijektiv.
- Alle drei Kodierungen erlauben Individuen verschiedener Größen und können (zumindest theoretisch) sogar überabzählbar viele Individuen kodieren (durch überabzählbar viele Symbole im Alphabet).
- Im Fall der String-Kodierung enthält der Suchraum viele Genotypen, die keinem Phänotypen (keiner Funktion) entsprechen.
- Im Fall der Kodierung als Baum oder Graph kodieren alle Individuen, die die Nebenbedingungen einhalten, auch einen Phänotypen.
- Im Fall der Kodierung als Graph wird der gleiche Phänotyp durch (unendlich viele) verschiedene Genotypen kodiert.
- Für jede der Kodierungen müssen wir spezielle Variationsoperatoren entwickeln und überlegen, wie wir mit den Nebenbedingungen umgehen wollen.

Wir werden sehen:

- Die visuelle Syntax von Bäumen und Graphen erleichtert die Entwicklung von Variationsoperatoren, die die Gültigkeit der Nebenbedingungen bewahren.
- Die zusätzlichen Knoten im Graphen (im Vergleich zur Baum-Kodierung) erlauben interessantere Variationsoperatoren.

# Class Responsibility Assignment

Gegeben sind Features (Methoden und Attribute) eines Klassenmodells und deren Abhängigkeiten (Methoden können andere Methoden aufrufen und Attribute als Parameter haben).

Gesucht ist eine Zuordnung der Methoden zu Klassen, sodass die Kohäsion *hoch* ist und die Kopplung *niedrig*.

Kohäsion: Gemeinsam in einer Klasse gruppierte Feature gehören zusammen (Methoden sind abhängig voneinander und/oder operieren auf gemeinsamen Daten)

Kopplung: Klassen sind gekoppelt, wenn es Abhängigkeiten zwischen ihnen gibt.

# Frage

Wie würdet ihr den Phänotypraum für das Class Responsibility Assignment Problem kodieren?

# Formalisierung von Kopplung und Kohäsion

(Definition nach [MJ14])

Kohäsionsrate: $\sum_{c \in C}\left(\frac{\mathrm{MAI}(c,c)}{|\mathrm{M}(c)| \times |\mathrm{A}(c)|} + \frac{\mathrm{MMI}(c,c)}{|\mathrm{M}(c)| \times |\mathrm{M}(c)-1|}\right)$

Kopplungsrate: $\sum_{\substack{c_i, c_j \in C \\ c_i \neq c_j}}\left(\frac{\mathrm{MAI}(c_i,c_j)}{|\mathrm{M}(c_i)| \times |\mathrm{A}(c_j)|} + \frac{\mathrm{MMI}(c_i,c_j)}{|\mathrm{M}(c_i)| \times |\mathrm{M}(c_j)|}\right)$

**Hierbei gilt:**

$$\mathrm{MMI}(c_i, c_j) = \sum_{\substack{m_i \in \mathrm{M}(c_i) \\ m_j \in \mathrm{M}(c_j)}} \mathrm{DMM}(m_i, m_j)$$

$$\mathrm{DMA}(m_i, a_j) = \begin{cases} 1, \text{falls Abhängigkeit zw.}\ m_i \text{und}\ a_j \\ 0, \text{sonst} \end{cases}$$

$$\mathrm{MAI}(m_i, a_j) = \sum_{\substack{m_i \in \mathrm{M}(c_i) \\ a_j \in \mathrm{A}(c_j)}} \mathrm{DMA}(m_i, a_j)$$

$$\mathrm{DMM}(m_i, m_j) = \begin{cases} 1, \text{falls Abhängigkeit zw.}\ m_i \text{und}\ m_j \\ 0, \text{sonst} \end{cases}$$

$\mathrm{A}(c)$: Die Attribute einer Klasse $c$

$\mathrm{M}(c)$: Die Methoden einer Klasse $c$

SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

# Phänotypraum CRA und Kodierung als Strings natürlicher Zahlen

Der Phänotypraum für das CRA-Problem besteht aus allen möglichen Zuteilungen der gegebenen $n$ Feature auf Klassen. Schließen wir leere Klassen aus, gibt es dafür

$$\binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n} = 2^n - 1$$

Möglichkeiten.

**Kodierung als Strings natürlicher Zahlen:**

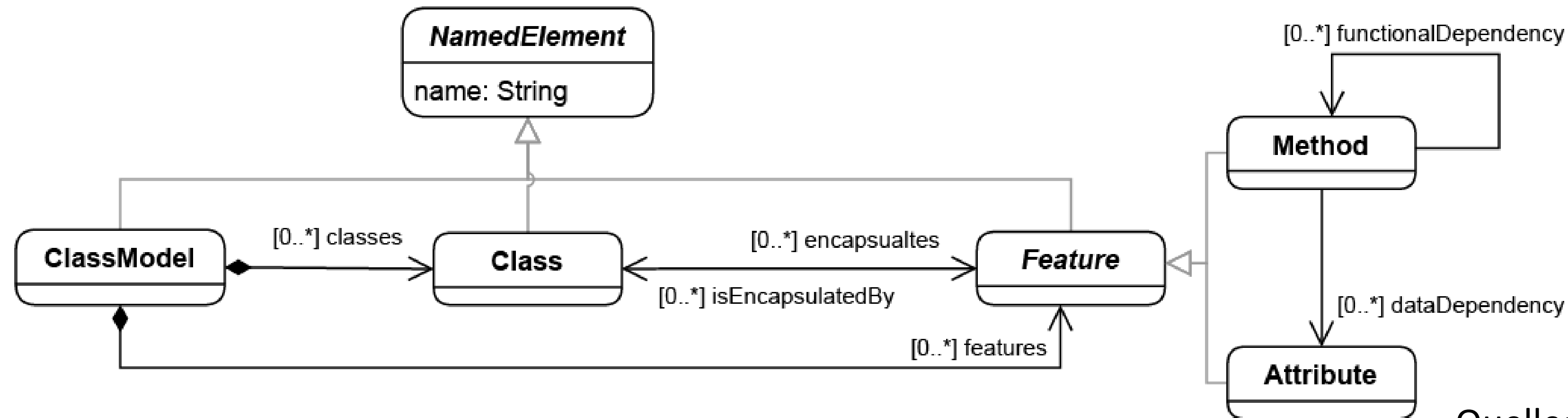
Wir nummerieren die Feature von 1 bis $n$ durch und kodieren Lösungen als Strings der Länge $n$ über dem Alphabet $\Sigma = \{1, \dots, n\}$, erhalten also den Suchraum $\{1, \dots, n\}^n$.

Ein String $v = (v_1, \dots, v_n) \in \{1, \dots, n\}^n$ kodiert, dass das $i$-te Feature der Klasse $v_i$ zugeordnet ist.

Nachteil: Suchraumgröße $n^n$ statt $2^n - 1$

SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

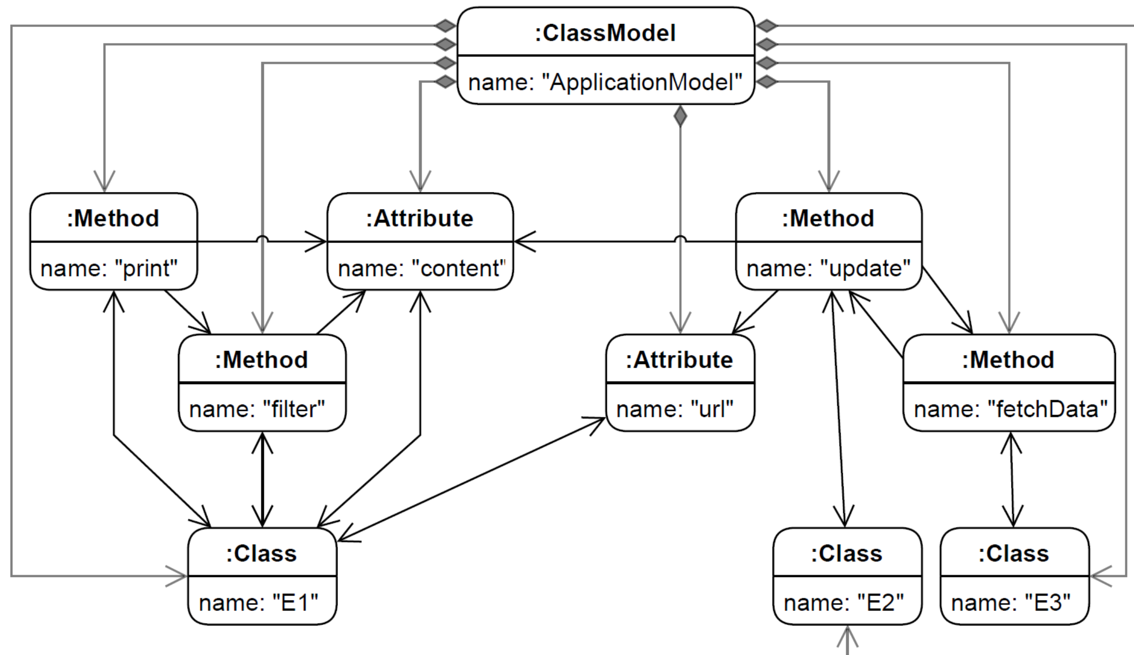
# Kodierung über Graphen I




Quelle: [KJT23] nach [FTW16]

Ein Klassendiagramm stellt uns die Syntax zur Verfügung, konkrete Instanzen vom CRA-Problem zu entwerfen.

# Kodierung über Graphen II




Quelle: [KJT23]

Objektdiagramme (aufgefasst als Graphen) kodieren

- konkrete Probleminstanzen und
- deren Lösungen.

Nebenbedingung: In einer validen Lösung ist jedes Feature (Methode oder Attribut) genau einer Klasse zugeordnet.

Eigenschaften der Kodierung: Bis auf Isomorphie und Attributwerte (Namen der Klassen) gibt es $2^n - 1$ valide Lösungen.

# Vektoren/Strings von Zahlen

- Vektoren bzw. Strings (gleicher Länge) von Zahlen sind eine vielbenutzte Repräsentation für evolutionäre Algorithmen.
- Der Zahlenraum (natürliche, ganze, reelle Zahlen) hängt von der Problemstellung ab.
- Für jede Position im Vektor kann es einen Definitionsbereich geben, also $(v_1, \ldots, v_n) \in [a_1, b_1] \times \cdots \times [a_n, b_n]$
- Typische Einsatzfälle:
  - Maximierung/Minimierung von Funktionen
  - Parameter eines Konstruktionsproblems optimieren
  - …

Diese Art der Kodierung ist häufig eine direkte Kodierung: Genotyp und Phänotyp stimmen überein.

# Ordinale vs. kardinale Bedeutung natürlicher Zahlen

**Ordinale Bedeutung**

- Die einzelnen Einträge $v_i$ in einem String natürlicher Zahlen $v_1 \dots v_n$ stehen tatsächlich für natürliche Zahlen.
- Beispiel: Optimierung einer Funktion mit $n$ Parametern
- Die Einträge $v_i$ dürfen oft unbeschränkt sein oder zumindest sehr große Werte annehmen.
- Der Entwurf von Variationsoperatoren, die mathematische Operatoren $(+, -, \times, \dots)$ auf die einzelnen Einträge anwenden, kann ein vielversprechender Ansatz sein.

**Kardinale Bedeutung**

- Die einzelnen Einträge $v_i$ in einem String natürlicher Zahlen $v_1 \dots v_n$ repräsentieren etwas anderes, zum Beispiel Zugehörigkeit zu einer Klasse.
- Beispiele: Farben, Städte, Optionen für Spielzüge werden durchnummeriert und (speicherplatzeffizient) als Integer gespeichert.
- Die Einträge $v_i$ dürfen oft nur in einem ausgewählten Bereich $[0, \dots, k-1]$ liegen ($k$ Klassen).
- Variationsoperatoren sollten nicht ausnutzen, dass die Einträge Zahlen sind.

# Literatur


- [APS20] Timothy Atkinson, Detlef Plump, Susan Stepney: Horizontal gene transfer for recombining graphs. Genet. Program. Evolvable Mach. 21(3): 321–347 (2020)
- [FTW16] Martin Fleck, Javier Troya, Manuel Wimmer: The Class Responsibility Assignment Case. TTC@STAF 2016: 1–8
- [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.
- [MJ14] Hamid Masoud, Saeed Jalili: A clustering-based model for class responsibility assignment problem in object-oriented analysis. J. Syst. Softw. 93: 110–131 (2014)