

# Ausgewählte Algorithmen zur kombinatorischen Optimierung der räumlichen Relationen in Probabilistic Scene Models

Bachelorarbeit  
von

Nikolai Andreas Gaßner

An der Fakultät für Informatik  
Institut für Anthropomatik und Robotik  
Lehrstuhl Prof. Dr.-Ing. R. Dillmann

Erstgutachter:	Prof. Dr.-Ing. R. Dillmann
Zweitgutachter:	Prof. Dr.-Ing. R. Stiefelhagen
Betreuernder Mitarbeiter:	Dipl.-Inform. Pascal Meißner

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen – die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Karlsruhe, den (**Datum**)

**ToDo**

---

Nikolai Andreas Gaßner

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>vi</b>
<b>1 Einführung</b>	<b>1</b>
<b>2 Motivation und Aufgabenstellung</b>	<b>4</b>
2.1 Motivation . . . . .	4
2.2 Aufgabenstellung . . . . .	6
<b>3 Grundlagen und Stand der Technik</b>	<b>9</b>
3.1 Grundlagen . . . . .	9
3.1.1 Bayes-Netze . . . . .	9
3.1.2 Gaussian Mixture Models . . . . .	10
3.1.3 Expectation Maximization . . . . .	11
3.1.4 Bayes'sches Informationskriterium . . . . .	12
3.2 Objekt- und Szenenerkennung . . . . .	12
3.2.1 Objekterkennung . . . . .	12
3.2.2 Zweidimensionale Szenenerkennung . . . . .	17
3.2.3 Dreidimensionale Szenenerkennung . . . . .	19
3.2.4 Implicit Shape Model (ISM) . . . . .	21
3.3 Probabilistic Scene Model (PSM) . . . . .	22
3.3.1 Erstellung von Relationsbäumen . . . . .	22
3.3.2 Lernen . . . . .	24
3.3.3 Inferenz . . . . .	24
3.4 Kombinatorische Optimierung . . . . .	26
<b>4 Konzeption</b>	<b>31</b>
4.1 Relationsgraphen . . . . .	31
4.1.1 Erstellung . . . . .	33
4.1.2 Umsortierung der Graphen für neue Referenzobjekte . . . . .	36
4.1.3 Bedingte Wahrscheinlichkeiten . . . . .	39
4.2 Kombinatorische Optimierung . . . . .	43
4.2.1 Einleitung . . . . .	43
4.2.2 Optimierungsproblem . . . . .	44
4.2.3 Auswahl der Starttopologien . . . . .	44

4.2.4	Nachbarschaftsfunktion . . . . .	45
4.2.5	Kostenfunktion . . . . .	46
4.2.6	Testsets . . . . .	47
4.2.7	Anwendung der kombinatorischen Optimierung . . . . .	49
4.3	Lernen der Gaussian Mixture Models . . . . .	49
4.3.1	Einleitung . . . . .	49
4.3.2	Gültigkeit der Kovarianzmatrizen . . . . .	50
4.3.3	Bayes'sches Informationskriterium . . . . .	50
<b>5</b>	<b>Implementierung</b>	<b>51</b>
5.1	Kombinatorische Optimierung . . . . .	51
5.1.1	Klassendiagramm . . . . .	51
5.1.2	Schnittstelle zum bestehenden System . . . . .	52
5.1.3	CombinatorialOptimizer . . . . .	53
5.1.4	OptimizationAlgorithm . . . . .	54
5.1.5	WeightedSum . . . . .	55
5.1.6	TopologyManager . . . . .	55
5.1.7	TopologyEvaluator . . . . .	56
5.1.8	TestSetGenerator . . . . .	57
5.1.9	TestSetSelection . . . . .	59
5.2	Relation Graph Generator . . . . .	60
5.2.1	TopologyCreator . . . . .	60
5.2.2	TopologyTreeTrainer und TopologyTreeGenerator . . . . .	61
5.3	Lernen der Gauß-Mischverteilungen . . . . .	62
5.3.1	Anbindung ans bestehende System . . . . .	62
5.3.2	Gültigkeit der Kovarianzmatrizen . . . . .	62
5.3.3	Dokumentierung . . . . .	62
5.4	Relationsgraphen in der Inferenz . . . . .	63
5.5	ConditionalProbability . . . . .	63
<b>6</b>	<b>Evaluation</b>	<b>64</b>
6.1	False Positive . . . . .	64
6.2	Erkennungslaufzeit und -qualität . . . . .	68
6.2.1	Büroszene . . . . .	69
6.2.2	Experimente . . . . .	70
6.2.3	Fehlerkennungen . . . . .	71
6.2.4	Erkennungslaufzeit und Relationen . . . . .	71
6.2.5	Kosten . . . . .	71
6.2.6	Besuchte Topologien . . . . .	71
6.3	Modellwachstum . . . . .	72
6.4	Mehrere Szenen in einem Modell . . . . .	76
6.5	Laufzeit . . . . .	82
6.5.1	Experimente . . . . .	82
6.5.2	Lernlaufzeit . . . . .	83

6.5.3	Erkennungslaufzeit . . . . .	84
6.5.4	Tabellen . . . . .	84
<b>7</b>	<b>Ausblick</b>	<b>86</b>
<b>8</b>	<b>Zusammenfassung</b>	<b>87</b>
	<b>Literaturverzeichnis</b>	<b>88</b>

# Abkürzungsverzeichnis

**BIC:** Bayesian Information Criterion, Bayes'sches Informationskriterium (3.1.4)

**EM:** Expectation Maximization (3.1.3)

**GMM:** Gaussian Mixture Model (3.1.2)

**ISM:** Implicit Shape Model (3.2.4)

**OCM:** Object Constellation Model (3.3.2)

**PSM:** Probabilistic Scene Model (3.3)

# 1. Einführung

Nachdem Roboter in der Industrie in weiten Bereichen Einzug gehalten haben, setzt sich die Servicerobotik damit auseinander, wie auch der menschliche Alltag durch Roboter erleichtert werden kann, beispielsweise in der Altenpflege.

Für einen Serviceroboter, der autonom in einer für Menschen gemachten Umgebung agieren soll, ist die Möglichkeit, diese Umgebungen analysieren und erkennen zu können von größter Bedeutung. Dabei ist nicht nur die Erkennung von einzelnen Objekten und Interaktionsmöglichkeiten mit ihnen wichtig, sondern auch die Erkennung von Szenen, die aus mehreren Objekten zusammengesetzt sind, da diese Kontextinformationen liefern.

Zum Beispiel kann eine Schale, neben der eine Milchbüte und eine Müslischachtel stehen, dem Roboter verraten, dass er einen Frühstückstisch betrachtet.

Nicht nur das Auftreten, sondern auch die Anordnung von Objekten zueinander ist hier bedeutsam. Beispielsweise handelt es sich nur um eine Frühstücksszene, wenn die drei erwähnten Gegenstände nah beieinander angeordnet sind; steht die Schale auf dem Tisch und die Müslipackung in einem Regal, ist die Frühstücksszene wenig wahrscheinlich, selbst wenn der Roboter beide Objekte gleichzeitig beobachtet.



Abbildung 1.1: Frühstücksszenen-Objekte

Abb. 1.1 illustriert dies: Links eine als "Frühstück" interpretierbare Szene, rechts Objekte aus ihr im Regal. Wenn links eines der Objekte fehlt, könnte die Szene trotzdem noch als Frühstück aufgefasst werden, rechts ist das durch die relative Anordnung übereinander aber nicht gegeben.

Die Anordnung von Gegenständen zueinander kann noch weitere Informationen enthalten, zum Beispiel die Anordnung von Besteck zum Teller.



Abbildung 1.2: Anordnung von Besteck relativ zum Teller

In Abb. 1.2, von links nach rechts: Vor dem Essen liegt das Besteck neben dem Teller. Auf dem Teller liegt es während einer Pause überkreuzt und nach der Mahlzeit parallel.

Es wäre denkbar, dass ein Roboter den Tisch abräumt, sobald die Mahlzeit beendet ist. Er sollte dann den Unterschied zwischen der mittleren und rechten Anordnung erkennen können (vgl. [Bra15])

Um die relative Anordnung von Objekten zueinander zu beschreiben, bietet es sich an, sie in Relation zu setzen. Das *Probabilistic Scene Model (PSM)*, das die Grundlage dieser Arbeit bildet, modelliert solche Relationen mit Wahrscheinlichkeiten. Diese werden aus Objektbewegungen erlernt, die von Menschen im Rahmen des *Programmieren durch Vormachen* demonstriert wurden. Da die Erkennungszeit von der Anzahl der Relationen abhängt, ist es wichtig, diese möglichst niedrig zu halten und gleichzeitig die Szene möglichst vollständig zu beschreiben.

Die vorliegende Arbeit setzt sich mit der Verwendung von Methoden der kombinatorischen Optimierung zum Finden geeigneter Relationenmengen auseinander.

## 2. Motivation und Aufgabenstellung

### 2.1 Motivation

Um eine Szene über die Relationen zwischen den darin auftretenden Objekte möglichst vollständig zu beschreiben und zugleich die Erkennungslaufzeit niedrig zu halten, ist eine Auswahl von Relationen nötig.

Zwar könnten einfach alle möglichen Relationen gewählt werden, allerdings ist die Anzahl der Relationen ungefähr quadratisch zu der Anzahl der Objekte<sup>1</sup>. Dies führt zu einem starken Anstieg der Größe der Relationsmenge, je mehr Objekte in der Szene vorkommen. Werden hingegen Relationen weggelassen, besteht das Risiko einer Fehlerkennung, weil wichtige Beziehungen unter Umständen nicht beachtet werden.

Eine Fehlerkennung kann als *false positive* auftreten, bei dem eine ungültige Anordnung von Objekten fälschlicherweise als Repräsentation der Szene eingestuft wird, oder als *false negative*, bei dem eine gültige Szeneninstanz nicht als solche erkannt wird.

Es ist wünschenswert, eine hinsichtlich Fehlerkennungen und Erkennungszeit möglichst günstige Menge von Relationen auszuwählen. Bisher wird dazu vom *PSM* eine Heuristik benutzt:

Die zum Erlernen der Szenen benutzten Trajektorien der Objekte, die beim Programmieren durch Vormachen von einem Menschen vorgegeben wurden, werden hinsichtlich ihrer Parallelität verglichen und zu einem Baum zusammengefügt.

Anstelle dieser Heuristik soll in dieser Arbeit ein weniger problemspezifischer, auf Methoden der kombinatorischen Optimierung aufbauender Ansatz zur Auswahl vorteilhafter Relationen verfolgt werden, um auch komplexere Szenen mit komplexeren Abhängigkeiten mithilfe von Relationsgraphen präziser beschreiben zu können als mit Bäumen.

---

<sup>1</sup>Genauer: Für  $n$  Objekte ist die Anzahl der Relationen  $R = (n - 1) + (n - 2) + \dots + 2 + 1 = \frac{n(n-1)}{2}$ .



Abbildung 2.1: Idee hinter der Optimierung von Relationsmengen

Abb. 2.1 demonstriert die zugrundeliegende Idee: Die Dicke und Farbe einer Relation gebe den Wert einer definierten *Vorteilhaftigkeit* oder *Wichtigkeit* an (in dieser Arbeit konkret die Vermeidung von Fehlerkennungen). Die linke Auswahl der Relationen ist wenig vorteilhaft. In der Mitte eine aus allen Relationen bestehende *vollvermaschte* Topologie. Da sie auch die wichtige grüne Relation enthält, wird sie ein besseres Ergebnis liefern; ist allerdings die Laufzeit von der Anzahl Relationen abhängig, wird sie langsamer sein als die rechte. Diese beinhaltet gleich viele Relationen wie die linke, darunter aber im Gegensatz dazu auch die grüne. Ziel wäre es, diese Relationsmenge anstelle der anderen beiden zu finden und auszuwählen.

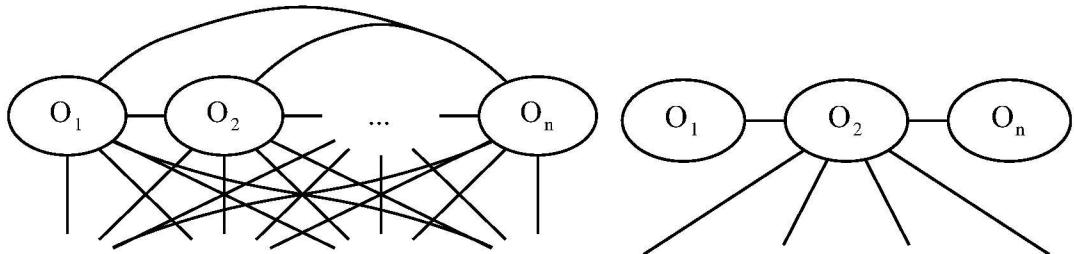


Abbildung 2.2: Komplexe Graphen

In Abb. 2.2 wird links ein komplexer Relationsgraph mit  $n$  Knoten (Objekten) und entsprechend vielen möglichen Relationen angedeutet. Eine Heuristik beinhaltet immer das Risiko, dass wichtige Relationen nicht beachtet werden, insbesondere wenn viele möglich sind. Gerade wenn nur Bäume, also eine verhältnismäßig geringe Relationszahl (für  $n$  Objekte  $n-1$  aus  $\frac{n^2-n}{2}$  möglichen) unterstützt werden. Rechts: Ein angedeuteter sternförmiger Graph für die selben Objekte, ausgehend von  $O_2$ , mit deutlich weniger Relationen.

Ein weiteres Ziel der Arbeit war es, die bislang in *PSM* zum Lernen der Gauß-Mischverteilungen, welche die Relationen im Raum darstellen, verwendete proprietäre Bibliothek durch eine

Open-Source-Lösung zu ersetzen.

## 2.2 Aufgabenstellung

Im *Probabilistic Scene Model* werden die Relationen als bedingte Wahrscheinlichkeiten einer Objektlage unter der Pose eines Elternobjekts dargestellt, sind also gerichtet.

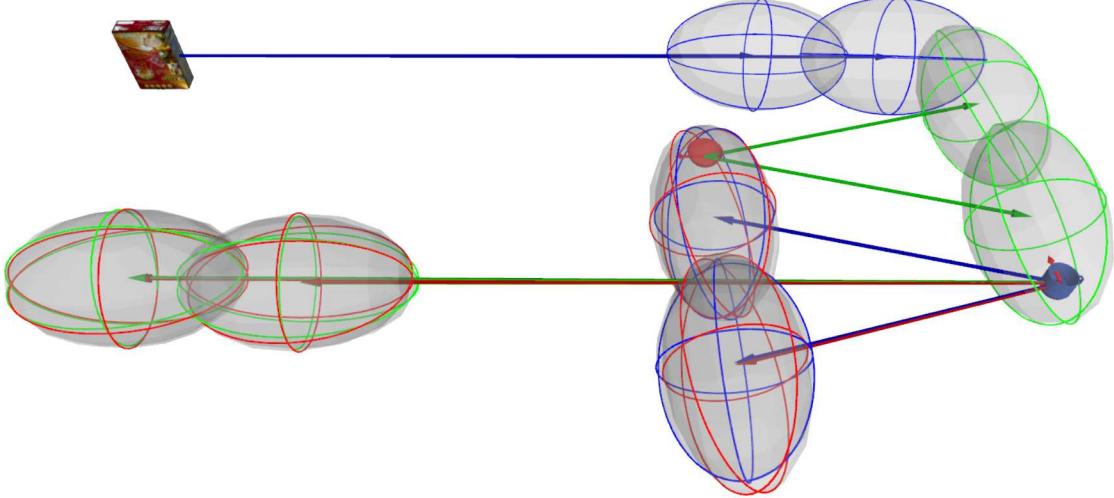


Abbildung 2.3: Probabilistic Scene Model

Ein konkretes Beispiel für ein *Probabilistic Scene Model* ist in Abb. 2.3 zu sehen. Die Szene besteht aus einer Müslischachtel (oben links), einer roten Schale (Mitte) und einer blauen Tasse (rechts). Die Relationen zwischen den einzelnen Objekten sind durch Pfeile dargestellt. Sie werden durch Gauß-Verteilungen modelliert, hier durch Ellipsoide visualisiert. Während die beiden rechten Objekte innerhalb der Ellipsoide des jeweils anderen liegen, also im erwarteten Bereich, und ihre Relation zueinander deshalb mit einer hohen Wahrscheinlichkeit bewertet wird, ist das bei der Schachtel links nicht der Fall.

Die Objekte lassen sich als Knoten  $v \in V$  eines Graphen  $G = (V, E)$  darstellen, die Relationenmenge, auch *Topologie*, als gerichtete Kanten  $e \in E$ .

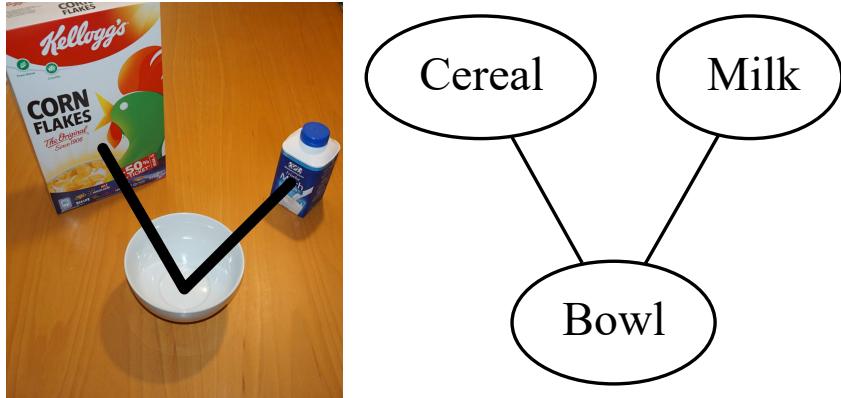


Abbildung 2.4: Relationen in einer Beispielszene

Beispielsweise ist in Abb. 2.4: links eine Szene aus den drei Objekten *Müslepackung* ("Cereal"), *Milchtröhre* ("Milk") und *Schale* ("Bowl"), zu sehen, die durch zwei Relationen verbunden sind. Rechts die Darstellung der Szene als ungerichteter Graph.

Bislang werden nur bedingte Wahrscheinlichkeiten für ein Objekt gegeben ein einzelnes Elternteil unterstützt.

Ziel ist es, eine Topologie zu finden, deren darauf aufbauendes Modell hinsichtlich Erkennungslaufzeit und Fehlerkennungen optimal ist. Dafür sollen Methoden der kombinatorischen Optimierung benutzt werden:

Sei  $M_G$  das ausgehend von  $G$  erstellte Modell und  $c(\cdot)$  eine Funktion, die einem Modell seine Kosten hinsichtlich Fehlerkennungen und Laufzeit zuordnet. Es soll ein  $G$  gefunden werden, sodass gilt

$$c(M_G) \text{ minimal}$$

Dazu müssen folgende Erweiterungen von *PSM* vorgenommen werden:

- Unterstützung von allgemeinen Graphen zusätzlich zu Bäumen.
- Möglichkeit, bedingte Wahrscheinlichkeiten gegeben mehrere Eltern darzustellen.
- Hinzufügen eines Frameworks und einer Schnittstelle zur Anwendung der für das verwandte Szenenerkennungssystem *ISM* (Kap. 3.2.4) bereits implementierten kombinatorischen Optimierungsalgorithmen.

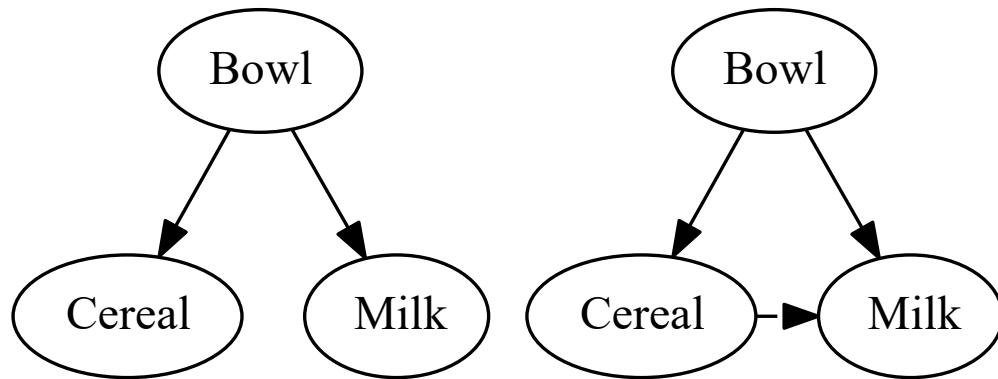


Abbildung 2.5: Bäume und Graphen

Abb. 2.5 zeigt links die bekannte Szene als Baum mit "Bowl" als Wurzel, rechts als gerichteten azyklischen Graphen.

Weiterhin ist ein Ziel, die Kompatibilität zur heuristischen Methode mit baumförmigen Relationsgraphen beizubehalten.

Zuletzt soll das Lernen der Gaußschen Mischverteilungen (Kap. 3.1.2), die zur Berechnung der bedingten Wahrscheinlichkeiten herangezogen werden, welches mithilfe von *Expectation Maximization* (Kap. 3.1.3) erfolgt, neu implementiert werden.

# 3. Grundlagen und Stand der Technik

## 3.1 Grundlagen

Die folgende Sektion behandelt die mathematischen Modelle, die dem PSM zugrunde liegen, sowie Algorithmen zu ihrer Erstellung.

### 3.1.1 Bayes-Netze

Ein Bayes-Netz ist eine Wahrscheinlichkeit über mehrere Zufallsvariablen, die durch einen gerichteten azyklischen Graphen dargestellt werden kann, dessen Knoten jeweils einer Zufallsvariablen zugeordnet werden. Die Kanten repräsentieren die Abhängigkeiten der Knoten von ihren Elternknoten innerhalb bedingter Wahrscheinlichkeiten.

Die Gesamtwahrscheinlichkeit des Netzes mit der Knotenmenge  $V$  ergibt sich als

$$P(V) = \prod_{x_i \in V} P(x_i | x_{Pa[x_i]})$$

Dabei sind die  $Pa[x_i]$  die Indizes der Elternknoten von  $x_i$ . [Pri12a]

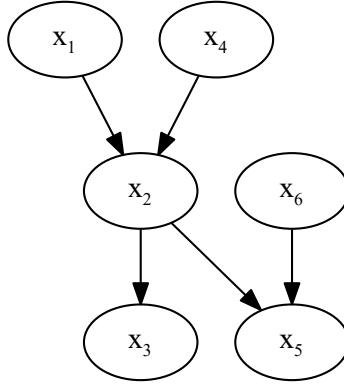


Abbildung 3.1: Beispiel für ein Bayesnetz

Abb. 3.1 zeigt ein Beispiel für ein Bayes-Netz. Die Kanten stellen Eltern→Kind-Beziehungen dar. Die durch den Graphen repräsentierte Wahrscheinlichkeit ist:

$$P(x_1, x_2, x_3, x_4, x_5, x_6) = P(x_1)P(x_2|x_1, x_4)P(x_3|x_2)P(x_4)P(x_5|x_2, x_6)P(x_6)$$

Auch der Baum und Graph aus Abb. 2.5 können als Bayes-Netze aufgefasst werden. Dabei repräsentiert der Baum die Wahrscheinlichkeit

$$P(Bowl, Cereal, Milk) = P(Bowl)P(Cereal|Bowl)P(Milk|Bowl)$$

Der Graph modelliert

$$P(Bowl)P(Cereal|Bowl)P(Milk|Bowl, Cereal)$$

### 3.1.2 Gaussian Mixture Models

**Gaussian Mixture Models (GMMs)** sind Modelle, die aus mehreren Gaußverteilungen, im Folgenden auch Gauß-Kernel genannt, zusammengesetzt sind. Sie können beispielsweise wie hier dazu benutzt werden, um einen Bereich zu beschreiben, in dem ein Objekt erwartet wird.

Ein GMM mit  $n$  Kerneln hat die Form

$$P(x|\theta) = \sum_{i=1}^n w_i \mathcal{N}_x(\mu_i, \Sigma_i)$$

mit den Gewichten  $w_i$ , wobei  $\sum_{i=1}^n w_i = 1$ , und den mehrdimensionalen Gaußverteilungen  $\mathcal{N}_x(\mu_i, \Sigma_i)$  mit Mittelwerten  $\mu_i$  und Kovarianzmatrizen  $\Sigma_i$ .

Die Parameter werden als  $\theta$  zusammengefasst. Eine multivariate (mehrdimensionale) Gauß- oder Normalverteilung im  $d$ -dimensionalen Raum ist definiert als

$$\mathcal{N}_x(\mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

[Pri12b]

### 3.1.3 Expectation Maximization

Ein GMM zur Beschreibung von Posen im Raum kann aus vorigen Beobachtungen mithilfe des **Expectation Maximization-Algorithmus** gelernt werden. Dieser kann allgemein für GMMs eingesetzt werden, hat aber einen noch breiteren Anwendungsbereich, etwa auch für *Hidden Markov Models*. [Bil97]

Ziel der Expectation Maximization ist, ein optimales  $\hat{\theta}$  zu finden:

$$\hat{\theta} = \operatorname{argmax}_{\theta} \left( \sum_{i=1}^I \log \left( \int P(x_i, h_i | \theta) dh_i \right) \right)$$

Der Algorithmus besteht aus zwei namensgebenden Schritten, dem **Maximization-** und dem **Expectation-**Schritt.

Für Wahrscheinlichkeitsverteilungen  $q_i(h_i)$  wird eine untere Schranke definiert:

$$\mathcal{B}(q_i(h_i), \theta) = \sum_{i=1}^I \int q_i(h_i) \log \left( \frac{P(x_i, h_i | \theta)}{q_i(h_i)} \right) dh_i$$

Im *Expectation*-Schritt werden die  $q(h_i)$  angepasst, um die Schranke zu verbessern, im *Maximization*-Schritt  $\theta$ . Beide Schritte werden abwechselnd wiederholt.

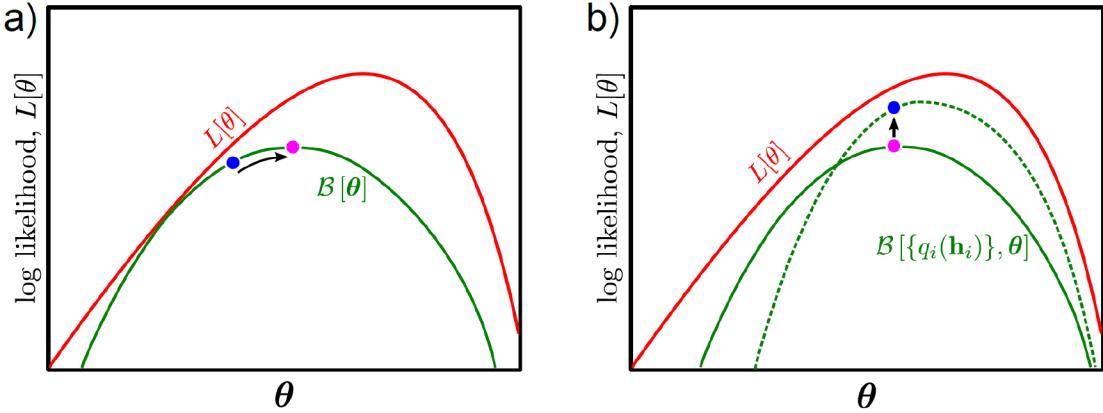


Abbildung 3.2: EM-Algorithmus [Pri12b, S.107]

Abb. 3.2 illustriert dies. Links, in (a), der Logarithmus der Wahrscheinlichkeit (Loglikelihood) aus der ersten Formel, in Abhängigkeit von den Modellparametern  $\theta$  in rot. Grün die untere Schranke. Der Maximization-Schritt erhöht die Schranke mittels  $\theta$ , um die aktuelle Schätzung (blau) zu verbessern (lila). Rechts (b) der Expectation-Schritt. Statt der Modellparameter werden die Wahrscheinlichkeitsverteilungen, von denen die Schranke ebenfalls abhängt, verändert. [Pri12b]

### 3.1.4 Bayes'sches Informationskriterium

Da das Ergebnis des EM-Algorithmus von den Ausgangswerten der Parameter wie auch von der Wahl der Kernelanzahl abhängt und dabei Punkte gesampt werden, kann es sich empfehlen, ihn mehrmals zu wiederholen. Um die dabei entstandenen unterschiedlichen Modelle zu vergleichen, lässt sich das **Bayessche Informationskriterium** verwenden, das 1978 von Gideon Schwarz eingeführt wurde [Sch78]. Für ein Modell  $\mathcal{M}$ , hier ein GMM, ist dieses folgendermaßen definiert: Sei  $k$  die Dimensionalität des Parameterraums und  $n$  die Anzahl der Datenpunkte, aus denen das Modell gelernt wurde. Dann ist das Bayes'sche Informationskriterium  $BIC$ :

$$BIC_x(\mathcal{M}) = -2 \log \left( P(x|\hat{\theta}) \right) + k \log(n)$$

[WHR12]

Das Modell wiegt also die Güte des Ergebnisses gegen die Anzahl der zur Beschreibung benötigten Parameter ab. Es kann dann das nach diesem Kriterium am besten bewertete Modell zur Beschreibung der Beobachtungen gewählt werden.

## 3.2 Objekt- und Szenenerkennung

Die Szenenerkennung versucht zu beurteilen, ob eine Beobachtung von Objekten eine Instanz einer Szene darstellt. Sie kann auf 2D-Bildern oder 3D-Eingaben durchgeführt werden. Die Erkennung von Szenen ist eng verwandt mit der von Objekten und teilt viele Algorithmen mit dieser.

### 3.2.1 Objekterkennung

Es existieren unterschiedlichste Ansätze zur Objekterkennung, wie etwa die **Deformable Part Based Models (DMPs)** von Felzenszwalb und Huttenlocher, die aussagekräftige Teile in 2D-Bildern identifizieren und sie über Federkräfte in Verbindung setzen. [FH00]

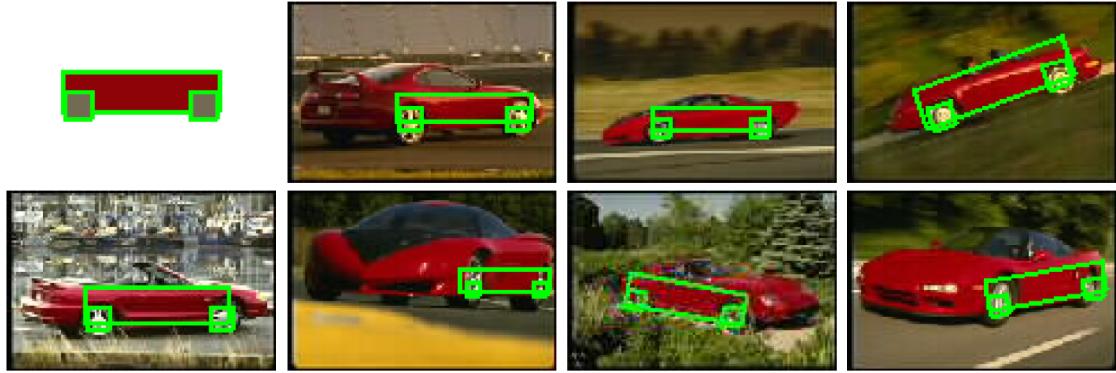


Abbildung 3.3: DMP für ein Auto. Links oben das Modell, Rest Erkennungen. Räder als prismatische Gelenke [FH00]

Da Teile die Grundlage eines DMP bilden, wird es auch als ein *teile-basiertes Modell (part-based model)* bezeichnet.

Aufbauend auf Arbeiten von *Burl et al* und *Weber et al* entwickelten *Fergus et al* das **Constellation Model (CM)**, welches Bildfeatures zur Erkennung von Objekten in 2D-Bildern benutzt. Gefundene Features, Teile des Objekts, werden mithilfe von Wahrscheinlichkeiten bewertet unter dem Blickwinkel, wie gut sie in ihrem Zusammenhang das Objekt beschreiben. Dazu werden drei Terme benutzt: *Appearance*, welcher das äußere Erscheinungsbild eines Features darstellt, *Shape* die Lage und *Occlusion* eine mögliche Verdeckung. Die Parameter der Wahrscheinlichkeitsfunktionen werden mithilfe des *EM*-Algorithmus gefunden. [FPZ03]

Das *CM* beschreibt in der ursprünglichen Form den *Shape*-Term als voll verbundenes Modell der Objektteile, was sowohl beim Lernen als auch bei der Erkennung zu einem Aufwand exponentiell zu der Teilanzahl führt, weshalb *Fergus et al* später stattdessen ein Sternmodell vorschlugen. Ist allerdings das Teil im Zentrum des Sterns verdeckt, ist bei diesem Modell die Erkennung nicht mehr möglich. [FPZ05]

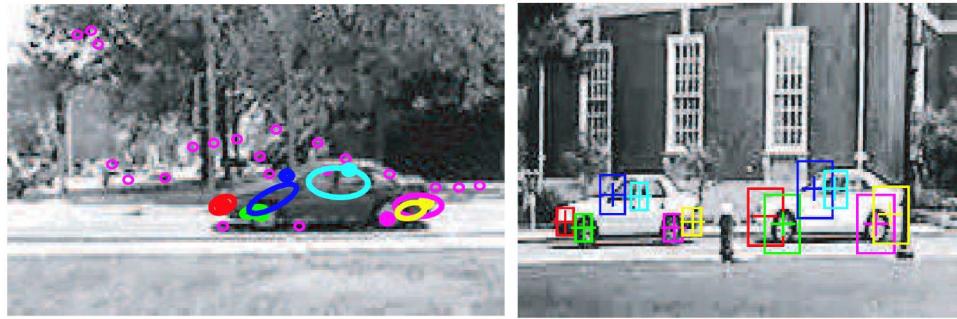


Abbildung 3.4: Links CM eines Autos. Rechts Erkennung von zweien damit [FPZ05]

Abb. 3.4 zeigt links das CM eines Autos. Als Teile (große Kreise) gewählt wurden u.a. die Fenster und Räder (insgesamt 6 Teile). Rechts wurden zwei Autos anhand des Modells im selben Bild gefunden.

Neuerdings wird Objekterkennung vornehmlich mithilfe von **Convolutional Neuronal Networks (CNNs)** durchgeführt. Inspiriert von der Funktionsweise des menschlichen Gehirns bestehen künstliche Neuronale Netze aus künstlichen Neuronen, die ihre Eingaben mittels einer sogenannten Aktivierungsfunktion mit Gewichten verrechnen und das Resultat ausgeben.

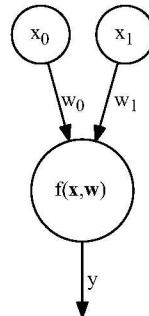


Abbildung 3.5: Künstliches Neuron

Abb. 3.5: illustriert ein künstliches Neuron mit zwei Eingaben  $\mathbf{x} = (x_0, x_1)^T$ , die es mit der Funktion  $f(\mathbf{x}, \mathbf{w})$  mit den Gewichten  $\mathbf{w} = (w_0, w_1)^T$  zur Ausgabe  $y$  verrechnet. Meist werden Schichten, die viele Neuronen enthalten, hintereinander geschaltet und miteinander verbunden. Innere Schichten werden als *hidden layers* bezeichnet. Durch Gradientenbildung und Rückpropagation des Fehlers in der Ausgabe gegenüber der erwarteten lassen sich bessere Gewichte lernen, um das Netz zum Lösen einer Aufgabe zu trainieren. [DW14]

*Convolutional Neuronal Networks* sind eine besondere Form der künstlichen Neuronalen Netze, die insbesondere bei der Verarbeitung von 2D-Bildern häufig eingesetzt wird. Als Eingabe der ersten Schicht werden die Pixel des Bildes benutzt. Nachbarschaften einer gewissen Größe sind jeweils mit einem Neuron verbunden. Dieses benutzt als Aktivierungsfunktion eine *Faltung*, engl. *convolution*.

Für diskrete Werte lautet die Formel für die Faltung von  $x$  und  $w$ :

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a) w(t-a)$$

Die Neuronen der Schicht teilen die Gewichte. Dadurch wird jeweils ein Merkmal in der Nachbarschaft gesucht, zum Beispiel eine Kante; da mehrere Nachbarschaften untersucht werden, wird Translationsinvarianz des Merkmals erreicht. Dadurch, dass nicht alle Eingaben mit allen Neuronen verbunden sind, anders als in ebenfalls benutzten voll verbundenen Neuronalen Netzen, lässt sich Rechenzeit sparen, und durch das Teilen der Gewichte auch Speicherplatz. Um die Effizienz weiter zu erhöhen, wird meist nach einem Nichtlinearisierungsschritt, um das Training durch Gradientenbildung zu erleichtern, eine *Pooling*-Stufe eingeführt, die die Ausgaben mehrerer Neuronen vereinigt, etwa durch Maximumsbildung.

Mehrere solcher Dreifachschichten werden neben- oder hintereinander geschaltet. [GBC16]

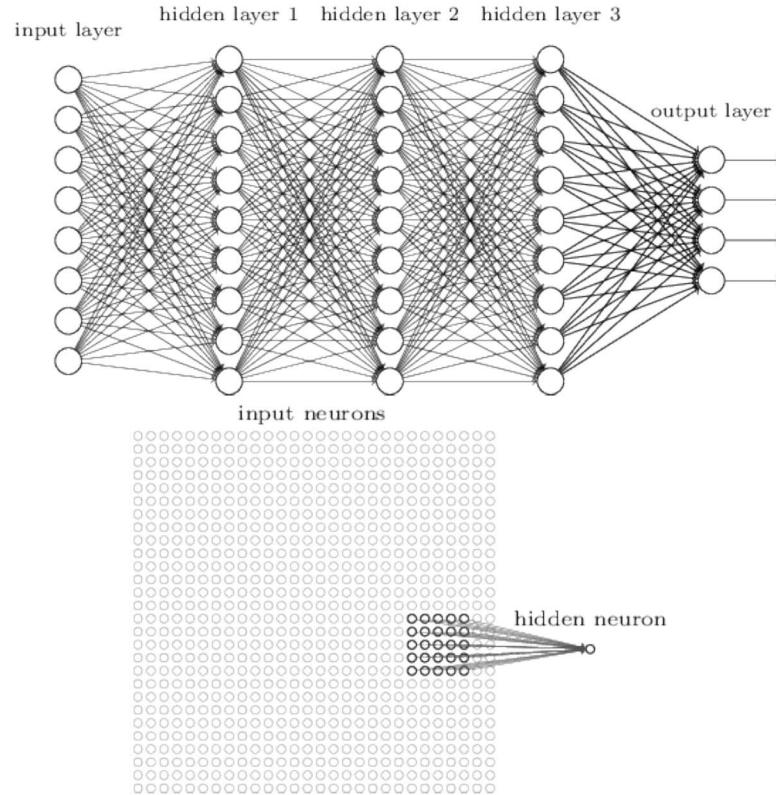


Abbildung 3.6: Convolutional Neural Nets [Nie15, Kap.6]

Oben in Abb. 3.6 ein Beispiel für ein voll verbundenes Neuronales Netz mit mehreren hidden layers. Unten die Idee hinter CNNs: statt allen Neuronen werden nur die in einem gewissen Bereich mit einem Neuron in der nächsten Schicht verbunden.

Als *Deep Neural Networks* werden Netze mit vielen inneren Schichten bezeichnet.

CNNs werden häufig auf zweidimensionalen Bildern benutzt, lassen sich aber auch anders anwenden. Beispielsweise benutzen Engelcke et al CNNs, um Objekte in 3D-Punktwolken zu erkennen, etwa Fußgänger in Straßenszenen, für Autonomes Fahren. [ERW<sup>+</sup>16]

Seit 2010 findet jährlich die **ILSVRC** statt, bei der die teilnehmenden Programme verschiedene Objekterkennungsaufgaben auf dem *ImageNet*-Dataset durchführen. Dabei handelt es sich um eine Sammlung von über 14 Millionen mittels Crowdsourcing handannotierter Bilder von Objekten in 2000 Kategorien, wovon eine Untergruppe für die Challenge verwendet wird. Die Challenge wird seit 2012 dominiert von Neuronalen Netzen.[RDS<sup>+</sup>15]

### 3.2.2 Zweidimensionale Szenenerkennung

Quattoni et al klassifizieren Innenraum-Szenen mithilfe eines modifizierten **Constellation Models** anhand der darin vorkommenden Objekte. [QT09] Parizi et al teilen Bilder von Außenszenen in ein Grid ein und betrachten die Zuweisung von Elementen, sogenannten **Regionsmodellen**, zu den Grid-Regionen, etwa "Himmel" oben, "Meer" in der Mitte und "Sand" unten für eine Strandszene.

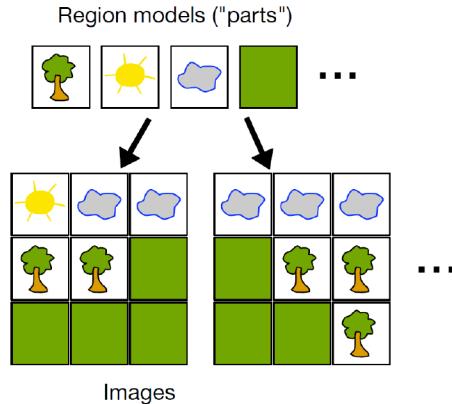


Abbildung 3.7: Zusammensetzen von Szenenbildern aus Regionsmodellen [POF12]

Verschiedene Regionsmodelle haben Ortspräferenzen, etwa in Abb. 3.7 die Sonne oben und Bäume weiter unten. [POF12]

Analog zu *ImageNet* existiert für die Szenenerkennung der **Places2**-Datensatz. Dieser enthält je nach Benchmark bis zu 10 Millionen Bilder, die Szenen aus standardmäßig 365 Kategorien darstellen. Auch auf diesem Datensatz wird eine Challenge durchgeführt. [ZLK<sup>+</sup>17]



Abbildung 3.8: Beispiele aus dem Places2-Datensatz, aus 3 Makro-Klassen [ZLK<sup>+</sup>17]

Neuronale Netze, die sowohl auf Places2 als auch auf ImageNet erfolgreich angewendet wurden, sind z.B.:

- **AlexNet:** Von Krizhevsky et al entwickelt. Gewann 2012 die ILSVRC. Es besteht aus 5 konvolutionalen und 3 vollverbundenden Schichten. Um *Overfitting*, also eine Überanpassung des Netzes an die Lerndaten, zu vermeiden, wird die *Dropout*-Technik benutzt. Dabei werden beim Lernen Neuronen zufällig für einzelne Durchgänge deaktiviert, sodass die verbliebenen Neuronen darauf trainiert werden, nicht zu stark von der Anwesenheit anderer abhängig zu sein. [KSH12]
- **VGG:** Der von Simonyan et al vorgestellte Sieger der Objektlokalisations-Aufgabe der ILSVRC 2014 (bei welcher eine *Bounding Box* um ein gefundenes Objekt gelegt werden soll) war zu seiner Zeit mit je nach Konfiguration 19 Schichten (16 konvolutional, 3 vollverbundene und nicht eingerechnete 5 max-pooling nach manchen der konvolutionalen) vergleichsweise besonders tief, benötigte 2-3 Wochen zum Training auf rund 1 Million annotierter Bilder. [SZ14]
- **GoogLeNet:** Ebenfalls 2014 sehr erfolgreich, hat dieses von Szegedy et al entworfene System 22 Schichten. Es führte die sogenannte *Inception*-Architektur ein, welche die Ausgaben von Schichten nach Korrelation zu Clustern zusammenfügt und diese als Eingabe für die nächste Schicht verwendet. [SLJ<sup>+</sup>14]

Sowohl die Gewinner der aktuell letzten ILSVRC (2017), in der Objekterkennungskategorie als auch die der Places2-Challenge 2016 verwenden Ideen aus der *Inception*-Architektur, sowie des von He et al eingeführten **Residual Learning** [ima], [pla]:

Bei Erhöhung der Schichtenanzahl kann die Fehlerrate nach anfänglichem Sinken wieder steigen (*degradation*). Um das zu vermeiden, wird die Eingabe einer unteren Schicht  $x$  unverändert weitergeleitet (sog. *shortcut connection*) und nach weiteren Schichten zu

deren Ausgabe  $F(x)$  hinzu addiert, um eine gewünschte Ausgabe  $H(x)$  mittels  $F(x) := H(x) - x$  zu trainieren. Ziel dabei ist es, den Schichten wenn nötig zu erlauben, die Identität darzustellen. Damit konnte die Anzahl der Schichten auf bis zu 152 in einem Ensemble mehrerer Netze erhöht werden, welches die ILSVRC 2015 für sich entscheiden konnte.

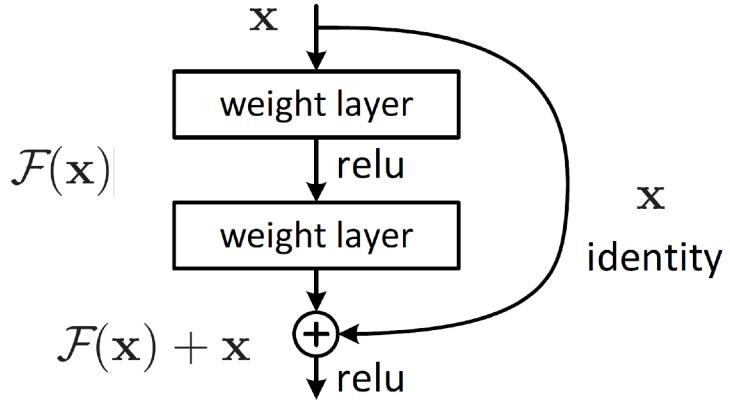


Abbildung 3.9: Residual learning [HZRS16]

Abb. 3.9 stellt die Nutzung des Residuums zur Vermeidung des *degradation*-Problems graphisch dar. Gezeigt ist ein Baustein der von *He et al* verwendeten Netze.  $x$  wird an zwei Schichten mit Gewichten vorbeigeführt (also etwa konvolutionale oder voll verbundene, nicht aber Pooling). "relu" bezeichnetet *Rectified Linear Units*, die zur Nichtlinearisierung eingesetzt werden. [HZRS16]

*Herranz et al* kombinieren DNNs für Objekterkennung auf 2D-Bildern aus der ImageNet-Datenbank und solche zur Szenenerkennung auf der Places-Datenbank, um den Einfluss von Skalierung auszugleichen. [HJL16]

CNNs haben allerdings oft Schwierigkeiten damit, mit mehreren Objekten in einem Bild umzugehen. [RDS<sup>+</sup>15] Andere Modelle können das besser behandeln.

Zudem sind Relationen zwischen Features in CNNs nichttrivial, gerade wenn die Teile weit voneinander entfernt sind, und *part-based models* damit nicht generell abbildbar.[GBC16, S.347]

Weitere Schwächen von CNNs sind die große Menge an Parametern gerade für tiefe Netze und die Struktur als *Black Box*.

### 3.2.3 Dreidimensionale Szenenerkennung

Für die Szenenerkennung gibt es zwei Ansatzpunkte.

Der eine geht davon aus, dass Szenen an sich wichtige Informationen enthalten, diesen verfolgt das PSM, auf dem diese Arbeit aufbaut.

Der andere benutzt Szeneninformation als Hilfsmittel für die Bilderkennung, da etwa, wenn ein "Badezimmer" erkannt wurde, ein "Bett" darin unwahrscheinlich ist.

Diesen Ansatz verfolgen *Lin, Fidler und Urtasun*, die in RGB-D-Bildern, d.h. Bildern mit zusätzlicher Tiefeninformation, wie sie etwa von Microsoft-kinect-Kameras gewonnen werden, Quader um Objektkandidaten legen und diese mit Szenen in Beziehung setzen [LFU13].

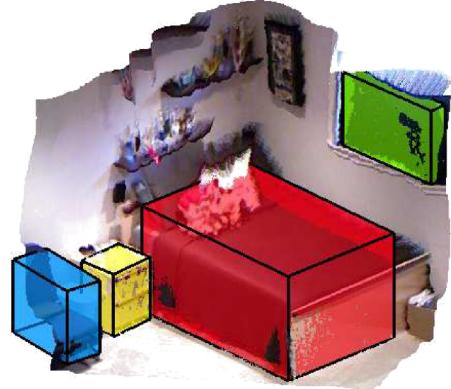


Abbildung 3.10: Quader um Schlafzimmerobjekte in einem RGBD-Bild [LFU13]

Zur Szenenerkennung in 3D, die nicht nur als Zwischenschritt für eine Objekterkennung dient, sondern selbst die Hauptaufgabe ist, um Probleme wie in der Einleitung vorgestellt zu lösen, dient das *ISM*.

### 3.2.4 Implicit Shape Model (ISM)

Das *ISM* wurde ursprünglich von *Leibe et al.* zur Objekterkennung in 2D-Bildern entwickelt. Es handelt sich um eine modifizierte Form der *Hough-Transformation*: Gefundene Teile stimmen über die vermutete Position einer Referenz ab.

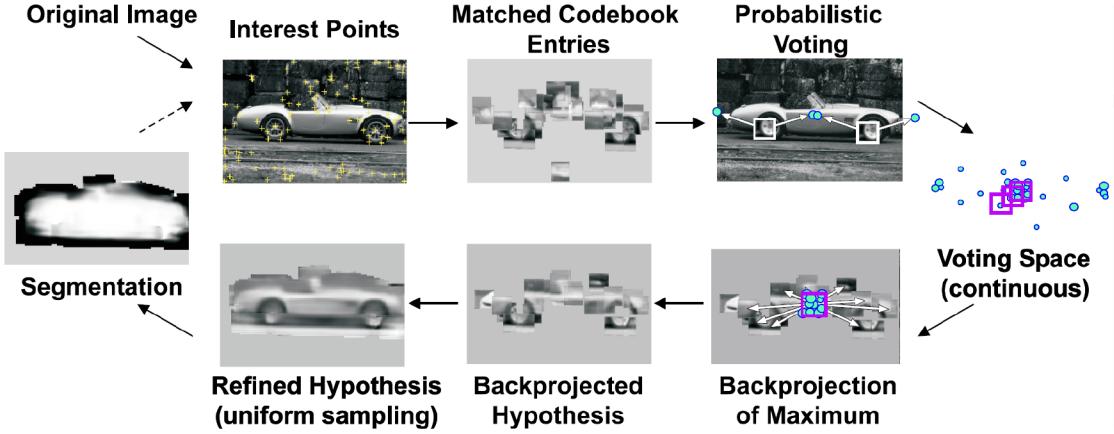


Abbildung 3.11: Implicit Shape Model [LLS04]

In Abb. 3.11 zu sehen ist der Ablauf einer Erkennung mit dem *ISM*: Bildausschnitte um *Regions of Interest* (ROI) stimmen ab und erstellen dadurch eine Objekthypothese. [LLS04]

Wenn hier von *ISM* die Rede ist, sei damit das von *Meißner et al.* entwickelte System zur Szenenerkennung in 3D gemeint. Dabei werden erkannte Objekte als Teile aufgefasst und voten auf Voxel, wo sie ihr Referenzobjekt, mit dem sie in Relation gesetzt wurden, vermuten.

Da die Modelle in ihrer Grundfassung nur sternförmige Anordnungen von Objekten unterstützen, werden mehrere Sub-Modelle zu Bäumen zusammengestellt, wobei die Sub-*ISMs* nach oben voten. [MRJ<sup>+</sup>13]

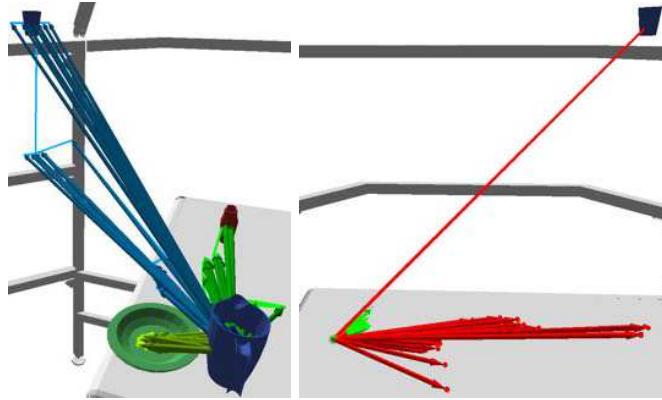


Abbildung 3.12: ISM zur Szenenerkennung in 3D [MRW<sup>+</sup>14] (leicht bearb.)

Im Beispiel, das in Abb. 3.12 zu sehen ist, links die zugrundeliegende Szene. Die Pfeile geben die relativen Posen der Trainingsdaten an. Rechts: Der Becher wurde gefunden und gibt seine Votes für andere Objekte ab, die zu der Szene gehören (rote Pfeile). [MRW<sup>+</sup>14]

### 3.3 Probabilistic Scene Model (PSM)

Wie der Name andeutet, basiert PSM im Kern auf Wahrscheinlichkeiten: Für Beobachtungen wird mithilfe von gelernten GMMs die Wahrscheinlichkeit berechnet, dass sie eine Szene darstellen.

#### 3.3.1 Erstellung von Relationsbäumen

Das Probabilistic Scene Model nimmt als Eingabe für den Lerner Objekttrajektorien. Mithilfe einer Heuristik, welche die Parallelität der Trajektorien bewertet, stellt das Programm einen Baum zusammen, der die Relationen zwischen den Objekten beschreibt. Als *parallel* werden zwei Trajektorien aufgefasst, wenn die Objekte sich für jeden Zeitschritt in Abstand und relativer Orientierung nicht wesentlich unterscheiden. Dabei tritt jedes Objekt genau einmal als Knoten in dem Baum auf. Die Heuristik dient dem Zweck, möglichst wenige Relationen auszuwählen, die aber die Szene so vollständig wie möglich beschreiben. Sie basiert auf agglomerativem Clustering.

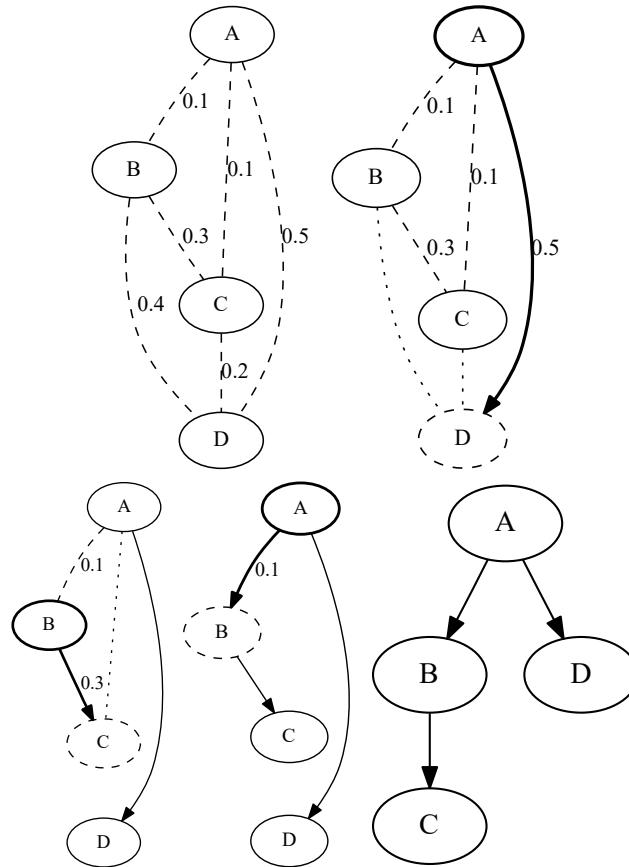


Abbildung 3.13: Agglomeratives Clustering zum Erstellen von Relationsbäumen

Die Anwendung der Heuristik und des agglomerativen Clustering wird in Abb. 3.13 anhand eines Beispiels gezeigt: Oben links sind die möglichen Relationen zwischen den vier Objekten  $A, B, C$  und  $D$  mit ihren Bewertungen durch die Heuristik als gestrichelte Linien dargestellt. Im ersten Schritt (oben rechts) wird die mit 0,5 bestbewertete Relation  $A - C$  ausgewählt. Das räumlich stabilere Objekt, hier  $A$ , wird als Referenz gesetzt, die Relation von ihm zu  $C$  gerichtet.  $C$  wird als Blatt des zukünftigen Baumes aufgefasst und deshalb alle restlichen möglichen Relationen, an denen es beteiligt wäre, gestrichen (gepunktete Linien). Hier fällt die nächstbestbewertete Relation  $B - D$  (0,4) weg. In den nächsten Schritten wird wiederum die bestbewertete verbliebene Relation zum Baum hinzugenommen und die verbleibenden möglichen zum instabileren Objekt gestrichen, bis zuletzt alle Objekte innerhalb eines Binärbaums angeordnet sind, der rechts unten dargestellt ist.

### 3.3.2 Lernen

Nach Erstellung des Relationsbaums werden für jeden Knoten, also jedes Objekt, die relativen Bewegungen jedes einzelnen Kindobjekts als Eingabe für eine Expectation Maximization benutzt, welche GMMs berechnet, die beschreiben, wo das Objekt das Kindobjekt am wahrscheinlichsten erwartet. Werden mehrere Modelle generiert, werden sie mithilfe des Bayes'schen Informationskriteriums verglichen. Dieser Teil des Modells wird als *Shape*-Modell bezeichnet, in Anlehnung an das *Constellation Model* aus der Objekterkennung, auf dem das PSM basiert (dieser Teil des Modells für den Vordergrund wird deshalb auch als *Object Constellation Model (OCM)* bezeichnet).

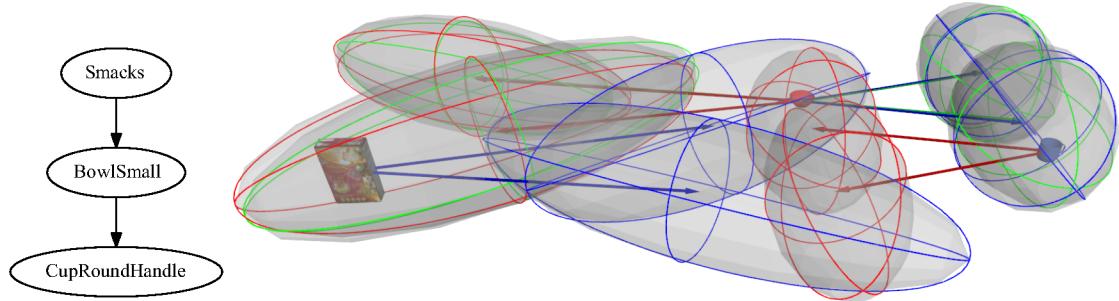


Abbildung 3.14: Lernen des PSM

Abb.3.14: Ausgehend von dem mithilfe der Heuristik konstruierten Baum (links) und seinen Umsortierungen für neue Wurzeln werden die Gaußkernel berechnet (Ellipsoide, rechts).

Es existieren noch zwei weitere Teilmodelle:

*Appearance* modelliert die Wahrscheinlichkeit, dass Objekte vom vorgeschalteten Objekterkennern miteinander oder mit dem Hintergrund verwechselt werden, die als eine Multinomialverteilung durch eine Tabelle realisiert wird.

Ebenso der *Occlusion*-Teil, der die Wahrscheinlichkeiten beinhaltet, dass Objekte in manchen Trajektorienschritten nicht beobachtet wurden, weil sie etwa verdeckt waren.

### 3.3.3 Inferenz

In der Erkennung wird eine beobachtete Objektkonstellation mit dem Szenenmodell verglichen, indem über alle möglichen Zuordnungen (*Hypothesen*) der beobachteten Objekte zu den *Objekt-Slots* im Modell iteriert wird (wodurch für mehr als 6 Objekte die Inferenz nicht mehr echtzeitfähig ist).

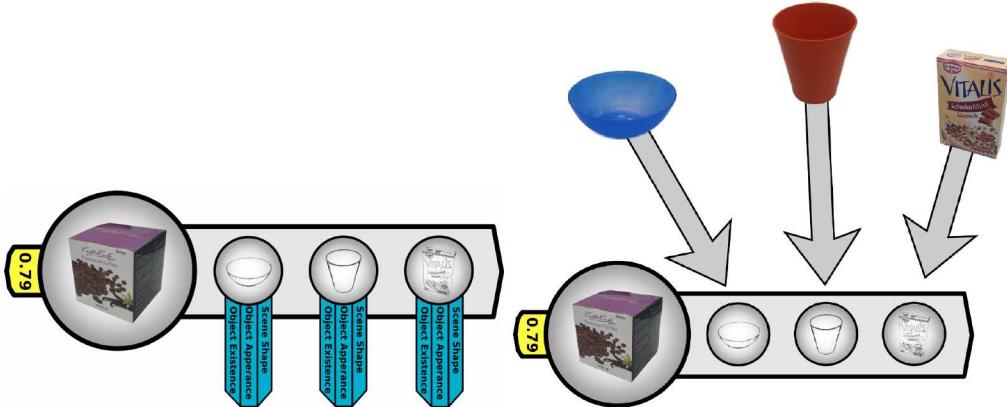


Abbildung 3.15: Slots und Hypothesen [Geh14, S.39f]

Abb. 3.15 zeigt für eine Kaffeschachtel als Referenzobjekt die Slots. Links sind darunter die Terme angedeutet, rechts eine Beispielshypothese zugewiesen. In gelb die resultierende Wahrscheinlichkeit für dieses Referenzobjekt.

Die Ergebnisse für den *Occlusion*- und den *Appearance*-Term werden aus den Tabellen geholt und multipliziert.

Für den *Shape*-Term wird mittels Tiefensuche den Relationsbaum hinabgegangen und die Posen aus den Beobachtungen, relativ zu denen der Elternknoten, mit den Kerneln verglichen, indem die *Mahalanobis-Distanz* zwischen ihnen gebildet wird:

Für den durch  $\mu, \Sigma$  beschriebenen Kernel und eine Beobachtung mit Pose  $x$  ist diese definiert als

$$Mah_x [\mu, \Sigma] = \sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu)}$$

Daraus werden Wahrscheinlichkeiten berechnet, die umso höher sind, desto näher die Beobachtung am Mittelwert  $\mu$  liegt, abhängig von der Kovarianz  $\Sigma$ . Alle Wahrscheinlichkeiten werden multipliziert.

Dieser Baum von Wahrscheinlichkeiten kann als Bayes-Netz aufgefasst werden.

Dass ein Objekt nicht beobachtet wurde, wird in diesem Term mithilfe eines *Nullobjekts* dargestellt. Wird diese einem Knoten (bzw. *Slot*) zugeordnet, wird für dieses Objekt und alle Nachfahren in den Wahrscheinlichkeiten über alle Posen marginalisiert. Die Marginalisierung ist definiert als:

$$P(x) = \int P(x, y) dy$$

Sie wird eingesetzt, um aus einer Verbundverteilung ( $P(x, y)$ ) diejenige für eine einzelne Zufallsvariable  $x$  zurückzugewinnen. Für bedingte Wahrscheinlichkeiten  $P(x|y)$  ergibt sich:

$$P(x) = \int P(x|y)P(y) dy$$

mit der a-priori-Wahrscheinlichkeit  $P(y)$ .

Die a-priori-Wahrscheinlichkeiten für jedes Objekt werden im *Shape-Term* implizit so gesetzt, dass alle derart *abgeschnittenen* Wahrscheinlichkeiten zu 1 marginalisiert werden und für den Rest der Berechnung des Terms keine Rolle mehr spielen. Um von der Wahl des Referenzobjekts unabhängig zu sein wird für jedes während des Lernens betrachtete Objekt ein Baum mit diesem als Wurzel erstellt; dazu wird der Relationsbaum umsortiert. Für jeden solchen Baum wird in der Inferenz die Wahrscheinlichkeit berechnet und zuletzt das Maximum der so entstandenen Referenzobjektwahrscheinlichkeiten als Szenenwahrscheinlichkeit ausgewählt und ausgegeben.

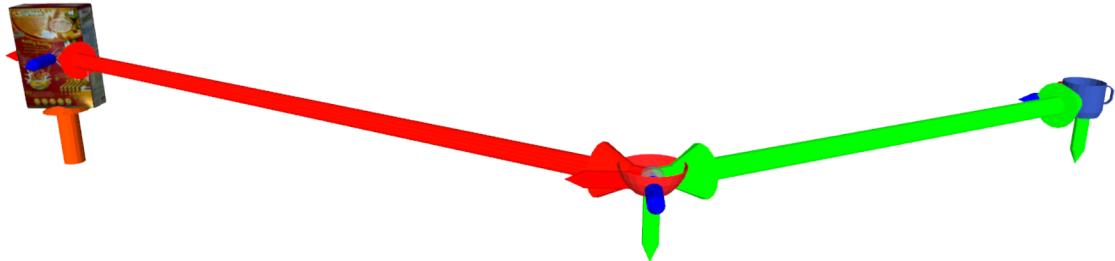


Abbildung 3.16: Inferenz mit dem PSM

Abb. 3.16: Für eine Beobachtung der Objekte aus der Szene werden ihre Relationen (Pfeile zwischen den Objekten) bewertet, hier die rechte sehr hoch (grüne Färbung), die linke sehr schlecht (rot). Der vertikale Pfeil deutet an, welches Objekt als Referenz gewählt wurde, hier die *Smacks-Müslischachtel*, seine Farbe die Gesamtbewertung der Szene, orange, also mittelmäßig. Die kleineren roten, grünen und blauen Pfeile zeigen die Koordinatensysteme und damit die Orientierung der Objekte. [Geh14, S.28–63]

### 3.4 Kombinatorische Optimierung

Die *Optimierung* hat das Ziel, in einem Zustandsraum  $\mathcal{Z}$  einen Zustand  $q \in \mathcal{Z}$  zu finden, der eine *Kostenfunktion*  $c(q)$  optimiert, also je nach Problem ein Minimum oder Maximum findet.

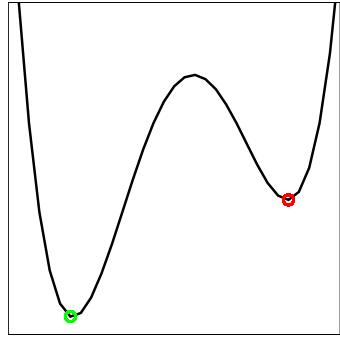


Abbildung 3.17: Optimierung (Minimum): Rot lokales, grün globales Optimum.

Bei der *kombinatorischen Optimierung* ist der Zustandsraum diskret. Ausgehend von einem *Startzustand*  $q_s \in \mathcal{Z}$  werden schrittweise mithilfe einer *Nachbarschaftsfunktion* Nachbarzustände gebildet und bewertet, ein Nachbar ausgewählt und als Ausgangszustand für den nächsten Optimierungsschritt benutzt.

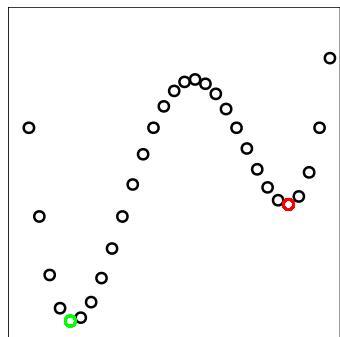


Abbildung 3.18: Kombinatorische Optimierung: diskreter Zustandsraum

Zum Finden eines Optimums gibt es verschiedene Algorithmen, von denen hier drei näher betrachtet werden sollen:

- **Hill Climbing** oder *Bergsteigeralgorithmus*: Es wird jeweils der am besten bewertete Nachbar als nächster Zustand gewählt, bis keine Verbesserung mehr möglich ist. Dieser Algorithmus hat eine hohe Tendenz, in lokalen Optima stecken zu bleiben. Um das zu vermeiden, kann zufällig ein Neustart durchgeführt werden (*Random Restart*) oder in manchen Schritten der nächste Nachbar zufällig ausgewählt werden (*Random Walk*). [Meh16, S.6f],[MS08]

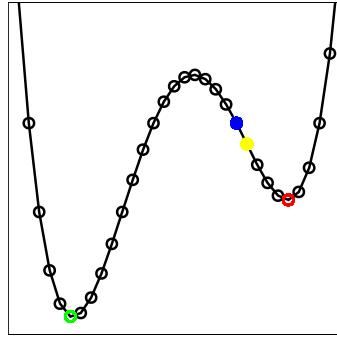


Abbildung 3.19: Hill Climbing

In Abb. 3.19: wählt *Hill Climbing* ausgehend vom blauen Startzustand den nächstbesten, gelb gekennzeichneten Nachbarn (durch Linie verbundenen Zustand). Dadurch fällt er langfristig in das lokale Optimum.

- **Record Hunt** oder *Rekordjagdalgorithmus*. Dieser merkt sich den bislang besten Zustand  $q_r$ , den *Rekord*, und erlaubt auch Schritte zu schlechter bewerteten Nachbarn, solange ihre Kosten die des Rekordhalters nicht um ein mit der Zeit abnehmendes  $\Delta$  unter- bzw. überschreiten [Goo98]. Bei einer Minimierungsaufgabe kann ein Nachbar  $q_n$  also akzeptiert werden, wenn gilt:

$$c(q_n) \leq c(q_r) + \Delta$$

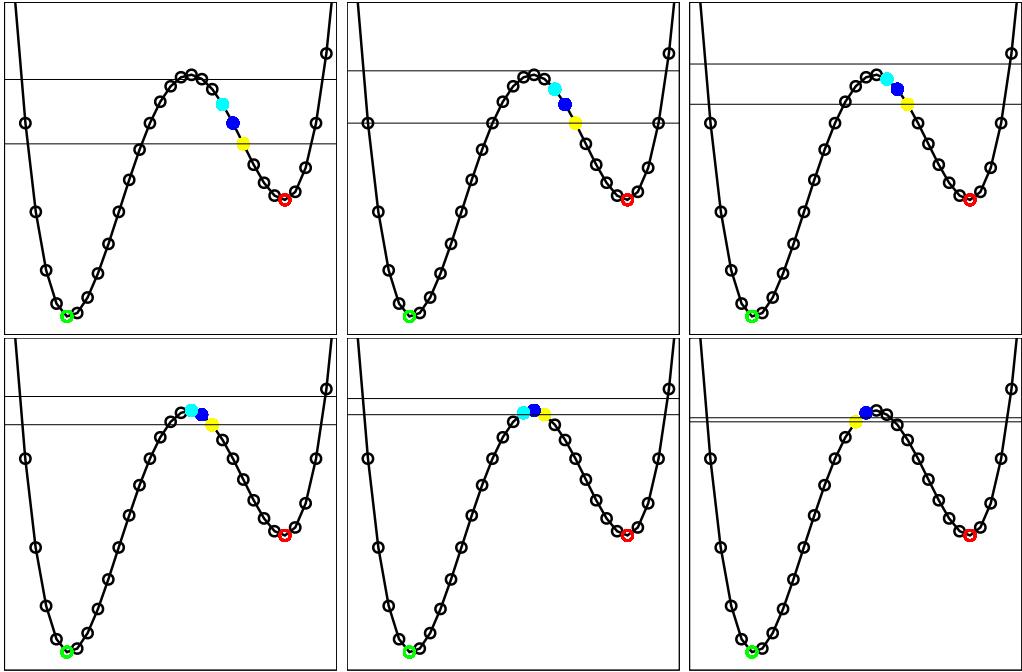


Abbildung 3.20: Record Hunt

*Record Hunt* kann in Abb. 3.20 einen schlechteren Zustand wählen (hellblau), solange dieser innerhalb des schrumpfenden Delta liegt. In diesem Beispiel überwindet der Algorithmus dadurch gerade so das lokale Maximum, welches das lokale vom globalen Minimum trennt.

- **Simulated Annealing** oder *Simuliertes Tempern* lehnt sich an einen physikalischen Vorgang an, bei dem durch kontrolliertes Abkühlen in Stoffen Kristallstrukturen erzeugt werden.

Am Anfang erlaubt der Algorithmus größere Sprünge, um lokalen Optima entkommen zu können, später werden die Sprünge durch ein Absenken einer *Temperatur*  $T$  kleiner, um das nächstliegende Optimum zu erreichen. Das wird dadurch erreicht, dass ein Nachbar  $q_n$  mit einer schlechteren Bewertung als der aktuell selektierte Zustand  $q_s$  mit einer gewissen Wahrscheinlichkeit dennoch akzeptiert wird:

$$P(\text{akzeptiert}) = \min \left( 1, \exp \left( -\frac{(c(q_n) - c(q_s))}{T} \right) \right)$$

[MS08]

Nach einer festgelegten Anzahl Schritte wird  $T$  mithilfe eines sogenannten *Cooling Schedule* abgesenkt, wofür es verschiedene Möglichkeiten gibt. Ein beliebter, der im Folgenden verwendet wird, ist der *exponentielle*: Bei der  $i$ -ten Absenkung ist  $T_i = \alpha T_0$ , mit  $T_0$  der Anfangstemperatur und  $0 < \alpha < 1$  [Meh16, S.7–10].

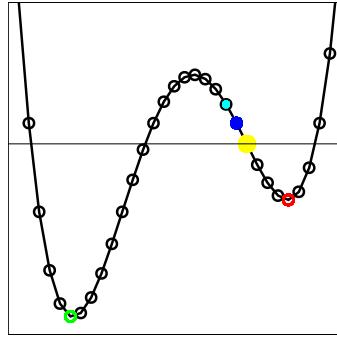


Abbildung 3.21: Simulated Annealing

Abb. 3.21: Analog zu Record Hunt akzeptiert auch Simulated Annealing Verschlechterungen, mit einer Wahrscheinlichkeit abhängig von der Temperatur und der Entfernung von der durch die Gerade markierten besten Kosten in der Nachbarschaft. Diese ist hier durch die Größe der farbigen Kreise angedeutet.  
Keiner von beiden letzteren Algorithmen umgeht das Problem der lokalen Optima vollständig, jedoch wird eher das globale gefunden, und die Laufzeit hält sich dennoch in Grenzen.

## 4. Konzeption

Bevor die kombinatorische Optimierung über Topologien angewendet werden kann, muss erst ermöglicht werden, allgemeine Graphen anstelle von ausschließlich Bäumen innerhalb des PSM zu verwenden. Dies hat Auswirkungen darauf, wie die Inferenz durchgeführt wird, namentlich dadurch, dass nun bedingte Wahrscheinlichkeiten in Abhängigkeit von mehr als einer Zufallsvariable dargestellt werden. Nachdem die Änderungen zu diesem Zweck erläutert sind, wird in der nächsten Sektion dieses Kapitels auf die Konzeption des Optimierungssystems eingegangen. Zuletzt wird das geänderte Lernen der GMMs behandelt.

### 4.1 Relationsgraphen

Um allgemeine Graphen ohne Kreise (engl. *Directed Acyclic Graphs*, DAGs) innerhalb PSM verwenden zu können, ohne die Kompatibilität zum älteren Prinzip einzuschränken, wurden *Verweisknoten* eingeführt.

Diese beinhalten einen Verweis auf den eigentlichen Knoten, den sie repräsentieren, sowie ein eigenes GMM relativ zum Elternknoten. So ist es möglich, dass ein *Objekt* mehrere Eltern haben kann, auch wenn jeder *Knoten* nur ein einziges Elternteil hat.

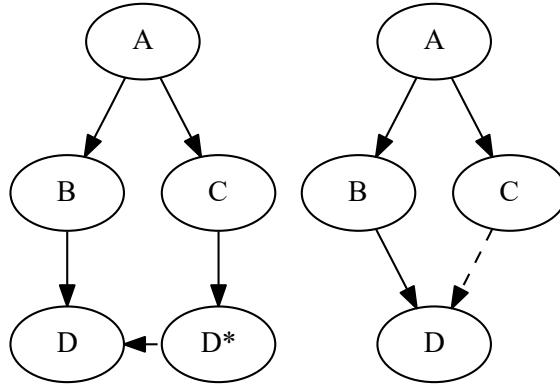


Abbildung 4.1: Verweisknoten

Im rechten Graphen von Abb. 4.1 hat  $D$  die beiden Eltern  $B$  und  $C$ , weshalb zwischen  $C$  und  $D$  ein weiterer Knoten  $D^*$  eingefügt wird, der einen Verweis auf den eigentlichen Knoten  $D$  enthält. Verkürzt lässt sich das wie im linken Graphen unter Auslassung von  $D^*$  durch eine gestrichelte *Verweiskante* darstellen.

Es ist des Weiteren zu beachten:

- Bei der Erstellung der Graphen, bzw. bei der Umwandlung von Graphen in *Bäume mit Verweisen*, muss sichergestellt werden, dass immer ein Pfad ohne Verweise zu jedem Knoten führt. Diese Pfade werden in einer Tiefensuche verfolgt, um den Nicht-Referenzknoten eindeutige Nummern zuzuweisen, über die sie angesprochen werden.
- Es sollen, gerade wenn ungerichtete Graphen in *Bäume mit Verweisen* umgewandelt werden, keine Kreise entstehen. Das lässt sich mittels einer modifizierten Breitensuche ausgehend von einem beliebigen Knoten im Ausgangsgraphen realisieren, welche Kanten zu schon einmal besuchten Knoten in Verweise umwandelt.
- Der zugrundeliegende Graph muss zusammenhängend sein, um die modifizierte Breitensuche sowie Lernen und Inferenz darauf ausführen zu können.
- Beim Umsortieren der Graphen für ein neues Referenzobjekt müssen Verweise so angepasst werden, dass keine Kreise entstehen. Beim Umsortieren werden die Vorfahren des Referenzobjekts zu Nachfahren; es ist also nötig, dass Verweise, die auf das Objekt zeigen, so umgekehrt werden, dass die Verweise stattdessen aus ihnen hinaus und auf das vorige Elternobjekt des Verweisknotens zeigen. Das muss für jeden Knoten, der seine Verwandtschaftsbeziehung infolge einer Umordnung ändert, durchgeführt werden.
- Ein Verweisknoten hat selbst keine Kinder, es sollen aber die Kinder des Knotens, auf den verwiesen wird, auch dann noch besucht werden, wenn dieser auf dem Weg über Nicht-Verweise abgeschnitten sein sollte. Dazu genügt es, den Verweisen zu

folgen, die Kinder des Zielknotens abzuarbeiten und als *von dorther besucht* zu markieren.

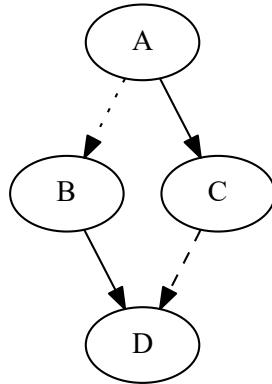


Abbildung 4.2: Erreichbarkeit von Knoten

Abb. 4.2: Ist die eine durchgezogene Nicht-Verweiskante, die in einen Knoten  $D$  führt, nicht mehr erreichbar, weil etwa wie hier eine darüber liegende Kante  $A \rightarrow B$  abgeschnitten ist (gepunkteter Pfeil), muss  $D$  trotzdem erreichbar sein, solange noch eine (gestrichelte) Verweiskante hineinführt.

Damit sind die Grundlagen geschaffen, allgemeine Topologien, wie sie von der kombinatorischen Optimierung verwendet werden, in von PSM verwendbare *Bäume mit Verweisen* umzuwandeln. Im Folgenden soll detaillierter darauf eingegangen werden.

#### 4.1.1 Erstellung

Ausgehend von einem allgemeinen, ungerichteten Graphen  $G = (V, E)$  mit Knoten  $v \in V$  und Kanten  $e \in E$  soll ein gerichteter Relationsgraph  $G_R = (V_R = V, E_R)$  erstellt werden, dessen Kantenmenge  $E_R$  für jede ungerichtete Kante  $e \in E$  eine gerichtete  $\vec{e}$  enthalten soll und für den weiterhin gilt:

- Es gibt genau einen Knoten  $v_0$  mit Eingangsgrad 0, der hier das *Referenzobjekt* oder, in Analogie zu Bäumen, die *Wurzel* genannt werden soll.
- $R_G$  enthält keine Kreise, es ist also nicht möglich, den Kanten in Kantenrichtung folgend wieder in einen zuvor besuchten Knoten zurückzukehren.
- Für die in einen Knoten  $v \in V_G$  eingehenden Kanten gilt: Genau eine wird als *Nicht-Verweiskante* aufgefasst, alle anderen als *Verweiskanten*. Letztere werden realisiert, indem ein weiterer Knoten, ein *Verweisknoten*, eingeführt wird.

Wie oben erwähnt kann das durch eine abgewandelte Breitensuche über  $G$  von einem beliebig gewählten Wurzelknoten  $v_0$  aus erreicht werden, wie in Algorithmus 1 beschrieben.

**Algorithm 1** Modifizierte Breitensuche

---

```

1: procedure BUILDTREEWITHREFERENCES(nodesByType, rootType, relations)
2:   typesToVisit  $\leftarrow$  [rootType]
3:   while relations  $\neq \emptyset$  do
4:     currentType  $\leftarrow$  typesToVisit.popFront()
5:     currentNode  $\leftarrow$  nodesByType [currentType]
6:     for relation  $\in$  relations do
7:       if relation.containsObject (currentType) then
8:         otherType  $\leftarrow$  relation.getOtherType (currentType)
9:         if not (nodesByType [otherType].inTree) then       $\triangleright$  not yet in tree
10:          currentNode.addChild (nodesByType [otherType])  $\triangleright$  add to tree
11:          nodesByType [otherType].added  $\leftarrow$  true            $\triangleright$  mark as visited
12:          typesToVisit.pushBack (otherType)
13:        else                                 $\triangleright$  Node already in tree
14:          reference  $\leftarrow$  newNode
15:          reference.referenceTo  $\leftarrow$  nodesByType [otherType]
16:          reference.isReference  $\leftarrow$  true
17:          currentNode.addChild (reference)
18:        end if
19:        relations.remove (relation)
20:      end if
21:    end for
22:  end while
23:  return nodesByType [rootType]
24: end procedure

```

---

*Verweis* ist hier als *reference* übersetzt. Das gewünschte Referenzobjekt, also die Wurzel, wird als *root* bezeichnet.

- Zeile 1: *nodesByType* enthält die Nicht-Verweisknoten, nach den Typen der ihnen zugewiesenen Objekte geordnet. Der *rootType* ist der Typ der gewünschten Wurzel. Wenn keine ausgewählt wurde, wird der Typ des ersten gefundenen Objekts benutzt, was durch einen vorhergehenden, nicht gezeigten Schritt geschieht. *relations* beinhaltet alle Relationen, die im Graphen vorkommen sollen, aus der Topologie und in ungerichteter Form.
- Zeile 2: Wie bei der Breitensuche üblich wird mit *typesToVisit* eine Liste zu besuchender Typen gehalten, hier initialisiert mit dem Typ des Wurzelobjekts.
- Zeilen 3–8: Solange noch Relationen übrig sind, die noch nicht in den Graphen aufgenommen wurden, werden der älteste Typ aus der Liste und der entsprechende Nicht-Verweisknoten aus *nonRefNodesByType* geholt. Es werden alle Relationen besucht, an denen das Objekt mit diesem Typ beteiligt ist.
- Zeilen 9–11: Ist das andere beteiligte Objekt noch nicht im *Baum mit Verweisen*, wird es als Kind dem aktuellen Knoten angefügt, die Relation also hinzugefügt und

in seine Richtung gerichtet. Das sorgt dafür, dass ein einzelner Pfad über Nicht-Verweisknoten in jeden Nicht-Verweisknoten führt. Es ist anzumerken, dass dieser Algorithmus nicht sicherstellt, dass alle Knoten besucht werden; er verlässt sich darauf, dass die Relationen einen zusammenhängenden Graphen beschreiben, was in einem vorhergehenden Schritt geprüft werden muss.

- *Zeile 12:* Der Typ des anderen beteiligten Knotens wird der Liste der zu besuchenden hinzugefügt, damit später die übrigen Relationen, an denen er beteiligt ist, von ihm aus dem Graphen hinzugefügt werden.
- *Zeilen 13–18:* Ist der andere an der Relation beteiligte Knoten bereits im Graphen, wird stattdessen ein neuer Knoten angelegt, der als *Verweisknoten* deklariert wird. Er hat keine Kinder und verweist auf den eigentlichen Knoten seines Typs, der sich bereits im Graphen befindet. Der Verweisknoten wird als Kind an den momentan betrachteten angehängt und die Relation so eingefügt.
- *Zeile 19:* Die dem Graphen hinzugefügten Relationen werden aus der Liste der Relationen entfernt.
- *Zeile 23:* Nachdem der *Baum mit Verweisen* fertig gestellt ist, wird der Nicht-Verweisknoten mit dem Wurzel-Typ als Wurzel zurückgegeben; zu diesem Zeitpunkt hängt an diesem der gesamte Relationsgraph.

Nicht im Pseudocode notiert ist davor ein Schritt, welcher ausgehend von dem Wurzelknoten eine Tiefensuche über den Graphen durchführt und den Nicht-Verweisknoten ihre Nummer in der Besuchsreihenfolge als ID zuweist. Dies gehört nicht zur eigentlichen modifizierten Breitensuche. Die Tiefensuche wird hier in Anlehnung an die Inferenz gewählt, welche die Knoten zur Berechnung des *Shape*-Terms ebenfalls in der Reihenfolge einer Tiefensuche besucht.

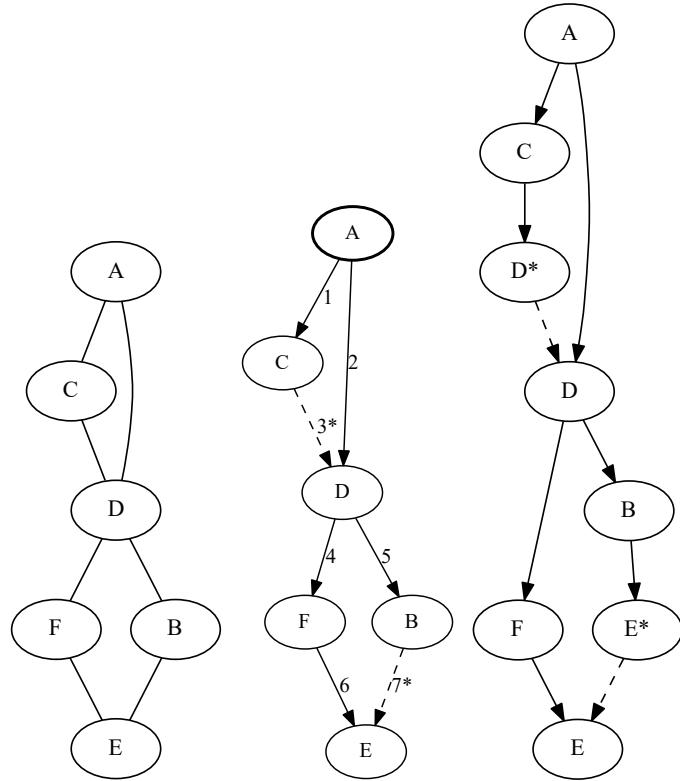


Abbildung 4.3: Modifizierte Breitensuche

Abb. 4.3 zeigt die Anwendung des Algorithmus auf eine Beispieltopologie: Aus dem linken ungerichteten Graphen wird, wie in der Mitte zu sehen, durch Besuchen der Knoten ausgehend von  $A$  in der durch die Zahlen angegebenen Reihenfolge ein gerichteter azyklischer *Baum mit Verweisen*. Gestrichelte Kanten, deren Nummern mit einem Stern versehen sind, führen in schon einmal besuchte Knoten und werden als *Verweiskanten* behandelt. Rechts ist zu sehen, wie sie in *Verweisknoten* umgewandelt werden.

#### 4.1.2 Umsortierung der Graphen für neue Referenzobjekte

Um bei der Rearrangierung von Relationsbäumen für neue Referenzobjekte (*Wurzeln*), Verweise auf Knoten, die durch die Neuordnung auf eine höhere Verwandtschaftsebene verschoben wurden, anzupassen, werden, um Kreise zu vermeiden, die Verweise umgekehrt.

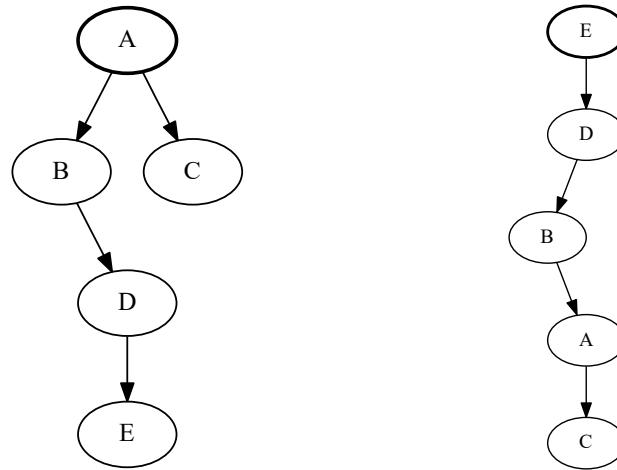
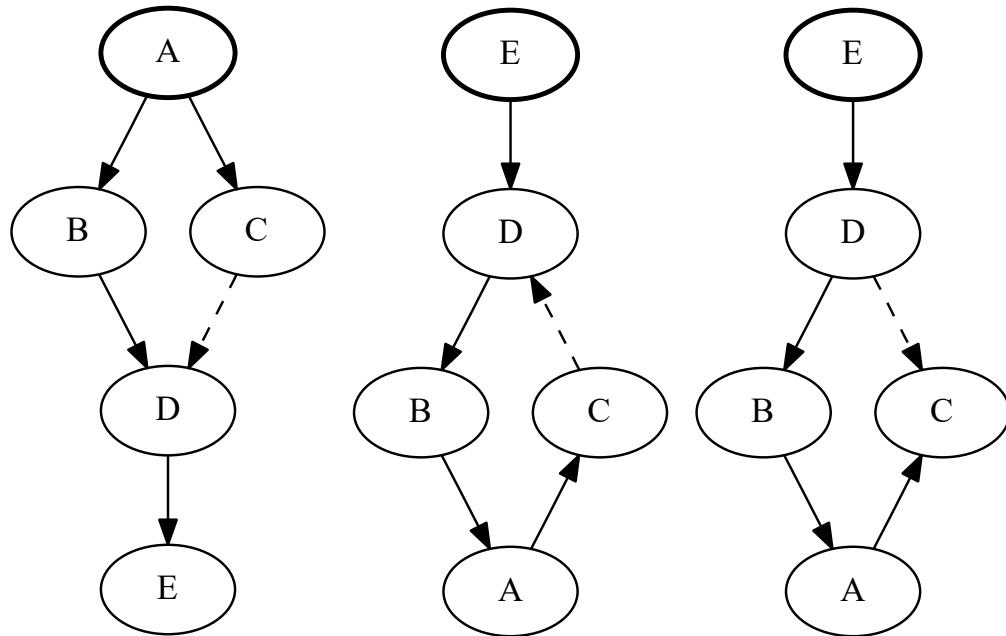


Abbildung 4.4: Umsortierung eines Relationsbaums für eine neue Wurzel

Die Umsortierung für einen Baum, wie bisher benutzt, ist in Abb. 4.4 zu sehen: Links ist Knoten *A* die Wurzel des Baums. Rechts wurde *E* zur neuen Wurzel erklärt und die Kanten, die in der Verwandtschaftshierarchie über *E* standen, umgedreht, um wieder einen Baum zu erhalten.

Abbildung 4.5: Umsortierung eines *Baums mit Verweisen* für eine neue Wurzel

Anders als zuvor gibt es in Abb. 4.5 *Verweiskanten*, wie  $C \rightarrow D$ ; statt eines Baums liegt also nun ein gerichteter azyklischer Graph vor (links). Da  $A$  keine Eltern hat, wird es es hier als *Wurzel* bezeichnet (aufgrund der modifizierten Breitensuche kann in einem zusammenhängenden Graphen nur ein einziger Knoten ohne Eltern vorliegen, nämlich der, von dem aus sie durchgeführt wurde). Wieder soll statt  $A$  nun  $E$  zum Referenzobjekt gemacht werden. Im mittleren Graphen ist dadurch, dass die Richtung von  $C \rightarrow D$  nicht umgekehrt wurde, ein Kreis entstanden; rechts ist die korrekte, azyklische Lösung zu sehen.

Es wird eine Erweiterung des in 2 beschriebenen Algorithmus (vereinfacht nach dem von Joachim Gehrung entwickelten [Geh14]) durch die Funktion *reverseReferences()* in Zeilen 3, 9 vorgenommen, die Verweis-Umkehrung 3.

---

**Algorithm 2** Rearrangierung für neue Wurzel

---

```

1: newRoot.parent.reassignParent(newRoot)
2: newRoot.parent ← None
3: newRoot.children.append (newRoot.reverseReferences (newRoot))
4: procedure REASSIGNPARENT(newParent)
5:   this.parent.reassignParent(this)
6:   this.parent ← newParent
7:   newParent.children.pushBack (this)
8:   this.children.remove (newParent)
9:   newParent.children.append (newParent.reverseReferences (newParent))
10: end procedure
```

---

**Algorithm 3** Verweis-Umkehrung

---

```

1: procedure REVERSEREFERENCES(root)
2:   if not (this.isReference) then
3:     for child ∈ children do
4:       if child.referenceTo = root then
5:         reversedReference ← Node
6:         reversedReference.referenceTo ← this
7:         reversedReference.parent ← root
8:         reversedReferences.pushBack (reversedReference)
9:         children.remove (child)
10:      else
11:        reversedReferences.append (child.reverseReferences (root))
12:      end if
13:    end for
14:  end if
15:  return reversedReferences
16: end procedure
```

---

Die gerichteten Relationen sind hier so realisiert, dass jeder Knoten außer den Blättern und Verweisknoten eine Liste seiner Kindknoten hat sowie jeder Knoten außer der Wurzel jeweils einen einzelnen Elternknoten. Wird eine neue Wurzel *newRoot* bestimmt, dreht *reassignParent* ausgehend von deren früherem Elternknoten die Relationen von Eltern zu Kindern um, indem es das jeweilige ehemalige Kind aus der Kinderliste entfernt und stattdessen als Elternknoten setzt, während der vorige Elternknoten den Kindern hinzugefügt und selbst rekursiv dieser Prozedur unterzogen wird. Da dies allerdings nur die Eltern und Kinder, nicht aber die Verweise berücksichtigt, wurde die Funktion *reverseReferences* hinzugefügt. Diese liefert eine Liste von Knoten mit umgekehrten Verweisen zurück, die der Liste der Kinder hinzugefügt werden kann, nachdem sie die Kinder, welche die umgekehrten Kanten darstellten, bereits entfernt hat (diese Trennung ist notwendig, um sicherzustellen, dass alle Knoten rekursiv betrachtet werden).

Sie ist folgendermaßen konzipiert:

Es werden alle Kinder des aktuellen Knotens durchgegangen. Da Verweisknoten nach Konstruktion keine Kinder haben, werden sie nicht weiter betrachtet.

Handelt es sich bei dem Kind nicht um einen Verweisknoten, muss auch kein Verweis umgekehrt werden, es wird dann rekursiv zu ihm und seinen Kindern weiter gegangen (die Prüfung auf Verweis findet wiederum beim Aufruf statt). Da als neuer Referenzknoten jeweils der dem gewählten Objekttyp zugeordnete Nicht-Verweisknoten ausgewählt wird, beginnt der Algorithmus immer bei einem solchen, weshalb ein solcher für jeden Typ im Graphen vorliegen muss, was durch die oben beschriebene Konstruktion gegeben ist.

Handelt es sich dagegen bei dem Kind um einen Verweis auf den aktuellen Wurzelknoten (der nicht unbedingt identisch sein muss mit dem neuen Referenzknoten, sondern auch diejenige eines Unterbaumes mit Verweisen sein kann, etwa Knoten *D* in Abb. 4.5), muss der Verweis umgekehrt werden. Es wird dazu ein neuer Verweisknoten, die *reversedReference*, angelegt. Sie zeigt auf den aktuellen Knoten. Dieser ist dem Eingangs-Check zufolge selbst kein Verweisknoten.

Als Elternknoten der *reversedReference* wird die Wurzel gesetzt und das Kind entfernt. Es wird also für jeden Verweis, der den aktuellen Knoten als Elternknoten hatte und der auf die neue Wurzel zeigte, stattdessen dem Wurzelknoten ein Verweis als Kind hinzugefügt, der auf den aktuellen Knoten zeigt.

#### 4.1.3 Bedingte Wahrscheinlichkeiten

Dadurch, dass ein Knoten jetzt mehrere Eltern haben kann, ist eine Änderung des Inferenz-Algorithmus nötig.

Für Bäume wird bisher im *Shape*-Term jedem Knoten eine Objektbeobachtung zugewiesen und anhand der Objektposen eine Wahrscheinlichkeit  $P(x|p)$  für jeden Knoten *x* anhand des einzelnen Elternteils *p* berechnet:

$$P(x|p) = \sum_{k=1}^n w_k \exp \left( -\sqrt{(\text{rel}(x, p) - \mu_k)^T \Sigma_k^{-1} (\text{rel}(x, p) - \mu_k)} \right) =: f(x, p)$$

Dabei ist  $rel(x, p)$  die relative Pose von  $x$  zu  $p$ .  $n, w_k, \mu_k, \Sigma_k$  sind die zuvor gelernten Parameter der GMMs. Die Wurzel beschreibt die Mahalanobis-Distanz von  $rel(x, p)$  zu  $\mu_k$ . Die einzelnen bedingten Wahrscheinlichkeiten werden multipliziert. Sie stellen also ein Bayes-Netz dar.

Wird  $x$  oder einer seiner Vorfahren in der Hierarchie des Baums mit dem *Nullobjekt* belegt, welches für die Möglichkeit steht, dass das zum Knoten gehörige Objekt nicht beobachtet wurde, nennen wir den Knoten, wie auch den gesamten am mit Null belegten Knoten hängenden Teilbaum *abgeschnitten*. Für jeden abgeschnittenen Knoten wird über alle Posen marginalisiert, was aufgrund der für das Modell gewählten a-priori-Wahrscheinlichkeiten  $P(x|p) = 1$  ergibt. [Geh14, S.45–48].

Ziel war es nun, eine Darstellung für eine bedingte Wahrscheinlichkeit in Abhängigkeit von  $m$  Eltern  $p_1, p_2 \dots p_m$  zu finden, welche für Knoten mit  $m = 1$  das selbe Ergebnis liefert wie bisher.

Es wurde dafür das Minimum der  $f(x, p_i)$  ausgewählt:

$$P(x|p_1, p_2, \dots, p_m) = \min_{i \in \{1, 2, \dots, m\}} f(x, p_i)$$

Für  $m = 1$  ergibt sich wie zuvor

$$P(x|p_1) = \min_{i \in \{1\}} f(x, p_i) = f(x, p_1)$$

Die resultierende Wahrscheinlichkeit ist von  $m$  unabhängig und alle einzelnen bedingten Wahrscheinlichkeiten miteinander vergleichbar.

Durch das Minimum wird eine Relation zwischen mehreren Objekten gleichzeitig dargestellt, die nur so stark ist wie ihr schwächstes Glied. Man kann sich auch vorstellen, dass es eine Art Schnitt zwischen den Ergebnissen für die einzelnen Elternteile repräsentiert.

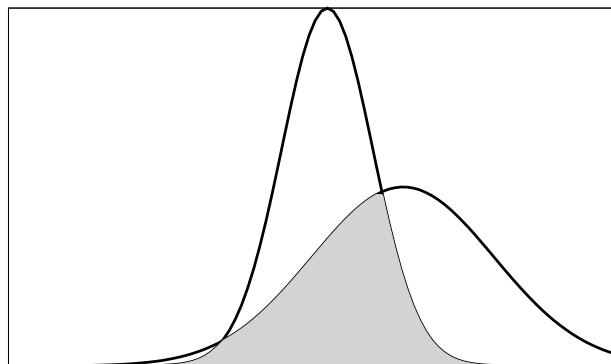


Abbildung 4.6: Minimumsbildung bei Gaußkurven

Abb. 4.6 stellt dies beispielhaft für Gaußkurven im zweidimensionalen Raum dar: Das Minimum der beiden fett gezeichneten Kurven ist mit einer dünnen Linie und flächig

markiert. Es ergibt sich, was man als Schnitt der Kurven auffassen kann.

Da wie zuvor jedes der  $f(x, p)$  für sich betrachtet eine gültige Wahrscheinlichkeit darstellt, ist auch das Minimum mehrerer eine solche.

Ein abgeschnittener Elternknoten wird zugunsten noch erreichbarer ignoriert

(da  $f_{\text{nichtabgeschnitten}} \leq f_{\text{abgeschnitten}} = 1$ ); nur, wenn alle Pfade zu  $x$  abgeschnitten bzw.  $x$  selbst mit Null belegt ist, wird auch das Minimum 1.

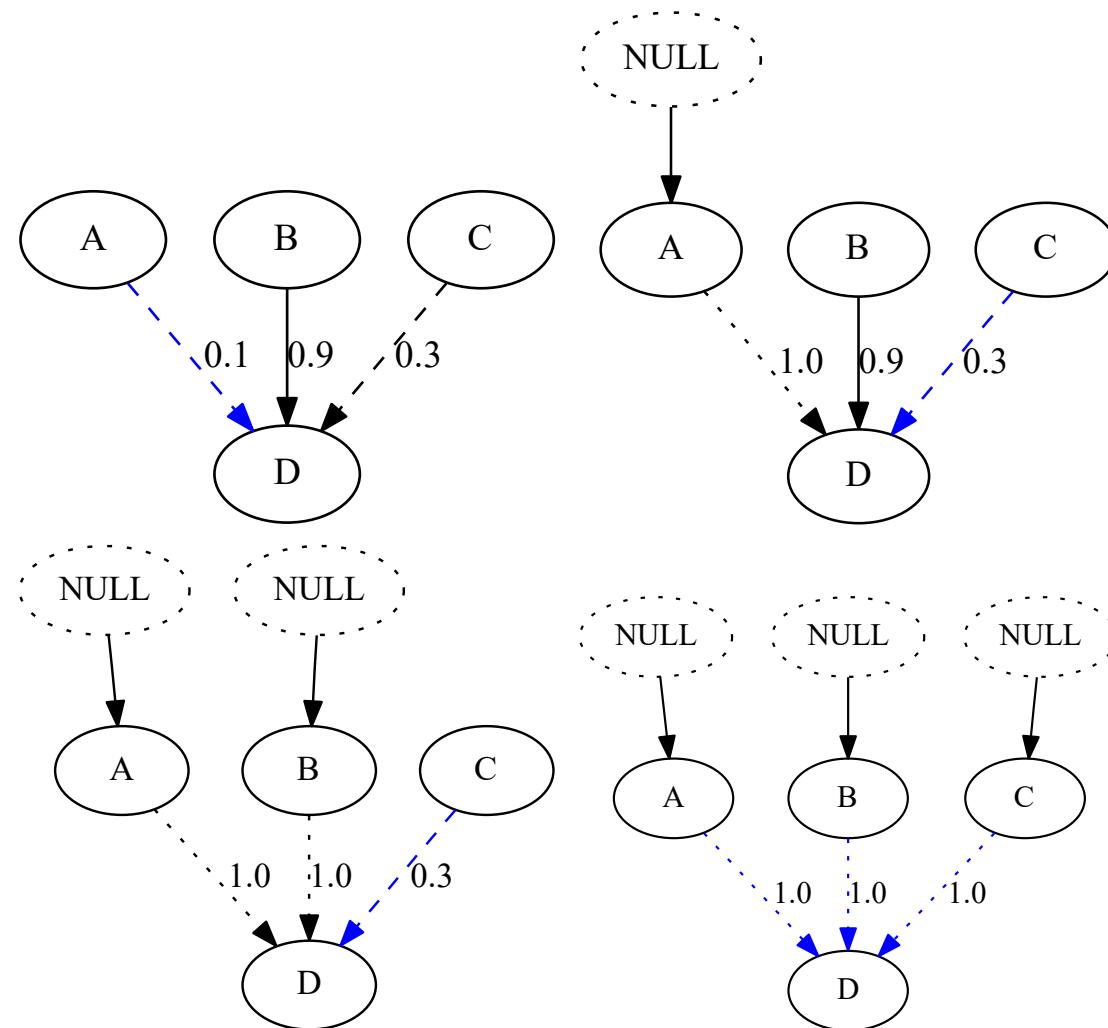


Abbildung 4.7: Erreichbarkeit von Knoten über Elternknoten

Abb. 4.7 enthält ein Beispiel dafür: Oben links seien allen Knoten  $\text{Objekte} \neq \text{NULL}$  zugeordnet, die Knotenobjekte also als beobachtet aufgefasst, und seien die  $f(D|A), f(D|B), f(D|C) < 1$ , also auch  $P(D|A, B, C) = \min_{Pa \in \{A, B, C\}} f(D|Pa) < 1$ . Wird nun, wie oben rechts, einem Knoten (hier  $A$ ) das Nullobject  $\text{NULL}$  zugeordnet, wird in Anlehnung an die

Marginalisierung der Wahrscheinlichkeit für ein einzelnes Elternteil  $f(D, A) = 1$ . Dennoch ist weiterhin  $P(D|A, B, C) < 1$ . Erst wenn, wie unten rechts, alle Eltern mit *NULL* belegt sind, wird  $P(D|A, B, C) = \min\{1, 1, 1\} = 1$ . Wie gehabt stellen gestrichelte Pfeile Verweiskanten dar (was hier keinen Einfluss auf das Ergebnis hat) und gepunktete durch das Nullobjekt abgeschnittene Kanten. Neben den Kanten Beispielwahrscheinlichkeiten, blau das Minimum.

Die aus den drei zentralen Termen des PSM berechneten Gesamtwahrscheinlichkeiten  $P(s)$  für die jeweiligen Referenzobjekte  $s \in S$  mit  $S$  der Menge der Szenenobjekte wird durch Maximumsbildung fusioniert, es wird das Objekt  $\hat{s}$  mit der höchsten Wahrscheinlichkeit gewählt und dessen Wahrscheinlichkeit  $P(\hat{s})$  ausgegeben [Geh14, S.51]:

$$\hat{s} = \arg \max_{s \in S} P(s)$$

Die Minimumsbildung hier ist vergleichbar dazu.

Wie sich die Änderungen konkret äußern, soll an einem Beispiel illustriert werden.

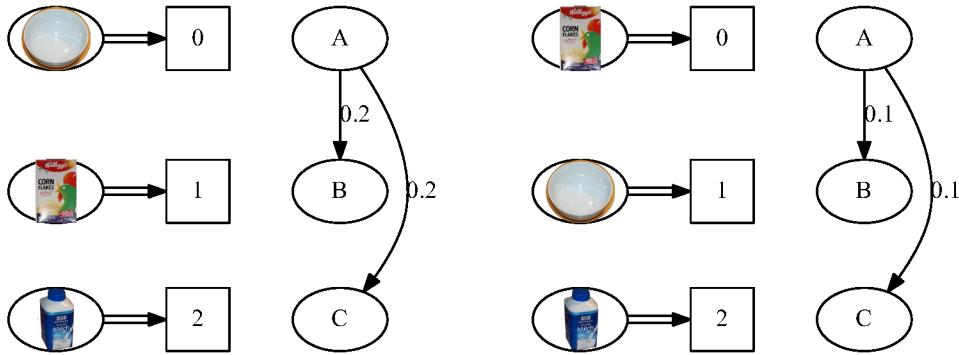


Abbildung 4.8: Zuweisung von Evidenzen zu Slots für Bäume

Abb. 4.8 zeigt die Zuweisung von Beobachtungen (Evidenzen) zu Slots über einem Baum: Die durch Bilder repräsentierten Beobachtungen werden den quadratischen, durchnummurierten Slots zugeordnet. Daneben der zugrundeliegende Baum. Als Bayes-Netz aufgefasst ergibt sich, wenn der *i*te Slot mit  $x_i$  bezeichnet wird,  $P(x_0, x_1, x_2) = P(x_0)P(x_1|x_0)P(x_2|x_0)$ . Wird zur Vereinfachung die a-priori-Wahrscheinlichkeit für das Auftreten des Referenzobjekts an  $x_0$  jeweils mit 1 angenommen, ergibt sich für die linke Belegung:

$$P_0 := P(Bowl, Cereal, Milk) = P(Cereal|Bowl)P(Milk|Bowl)$$

Sei nun z.B.

$$P(Cereal|Bowl) = P(Milk|Bowl) = 0,2$$

Dann ergibt sich  $P_0 = 0,04$ .

Die rechte Belegung ergibt

$$P_1 := P(Cereal, Bowl, Milk) = P(Bowl|Cereal)P(Milk|Cereal)$$

Die Objekte sollen hier an anderen Stellen auftreten als erwartet, die bedingten Wahrscheinlichkeit sind dann niedriger, hier z.B. beide 0,1. Damit ist  $P_1 = 0,01$ .

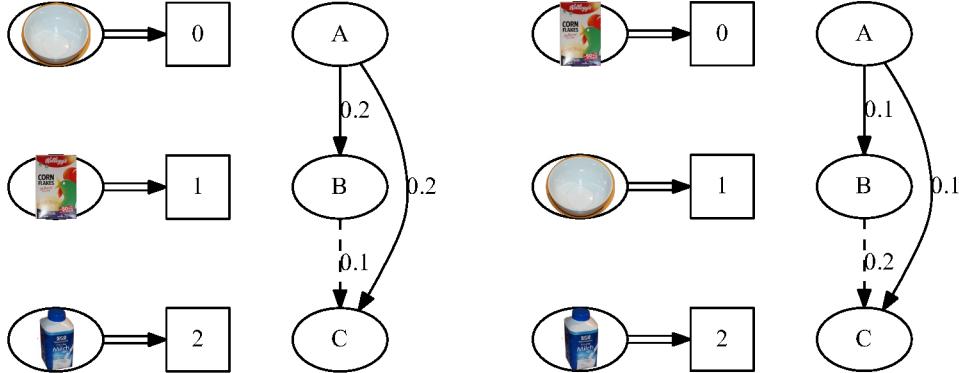


Abbildung 4.9: Zuweisung bei Graphen

Für einen *Baum mit Verweisen* ergibt sich das in Abb. 4.9 gezeigte Bild: Mit den Werten von oben ist links

$$\begin{aligned}
 P_2 &:= P(\text{Cereal}|\text{Bowl}) P(\text{Milk}|\text{Bowl}, \text{Cereal}) \\
 &= 0,2 \cdot \min(P(\text{Milk}|\text{Bowl}), P(\text{Milk}, \text{Cereal})) \\
 &= 0,2 \cdot \min(0,1, 0,2) = 0,02
 \end{aligned}$$

Die schlechter bewertete Relation ist also eingerechnet. Rechts ist

$$P_3 := P(\text{Bowl}|\text{Cereal}) P(\text{Milk}|\text{Bowl}, \text{Cereal}) = 0,1 \cdot 0,1 = 0,01$$

Der letzte Term ist derselbe wie links, die unterschiedliche Zuweisung der oberen Objekte spielt durch die Verknüpfung von *Milk* mit den beiden also keine Rolle mehr.

## 4.2 Kombinatorische Optimierung

### 4.2.1 Einleitung

Die Anwendung der Kombinatorischen Optimierung orientiert sich an derjenigen aus der Bachelorarbeit von Jonas Mehlhaus ([Meh16]), in welcher Relationsmengen für das *Implicit Shape Model* optimiert werden. Dieses ist, wie erwähnt ein Szenenmodell für den selben Anwendungsbereich wie PSM, das allerdings nicht auf probabilistischer Modellierung, sondern auf einer modifizierten *Hough-Transformation* beruht, bei der Objekte auf in Relation stehende voten. Deshalb werden einige Anpassungen sowie Erweiterungen vorgenommen.

### 4.2.2 Optimierungsproblem

Die Instanzen, über die optimiert werden soll, sind hier Topologien, welche Objektrelationen darstellen. Das Optimierungsproblem besteht darin, eine Topologie zu finden, deren zugehöriges PSM möglichst wenige Fehlerkennungen bei möglichst geringer Laufzeit liefert.

Es wird über jede betrachtete Topologie ein PSM gelernt und damit auf einer Reihe von Testsets, die gültige und ungültige Szeneninstanzen darstellen, eine Inferenz durchgeführt, und die Anzahl an Fehlerkennungen gezählt, sowie der Durchschnitt über die Erkennungslaufzeiten für die Sets gebildet.

### 4.2.3 Auswahl der Starttopologien

Als Starttopologien können zufällig generierte zusammenhängende Graphen benutzt werden. Des weiteren soll hier auch die von der Kostenfunktion am besten bewertete Topologie aus der vollvermaschten und den Sterntopologien als Starttopologie herangezogen werden können. Diese Topologien werden zur Berechnung der maximalen erlaubten Laufzeit und Anzahl von Fehlerkennungen benutzt, mithilfe derer die Ergebnisse für andere Topologien innerhalb der Gewichtsfunktion normalisiert werden. Deshalb werden diese Topologien schon vor Beginn der eigentlichen Optimierung ausgewertet. Ihre Kosten können dann mit geringem Aufwand berechnet werden.

Die Laufzeit für die vollvermaschte Topologie wird als Maximum gesetzt. Jede Topologie mit höherer Laufzeit kann durch diese ersetzt werden, die dann auf jeden Fall vorzuziehen ist, da sie als restriktivste keine Fehlerkennungen liefert.

Ebenso kann anstelle einer Topologie mit mehr Fehlerkennungen als einer Sterntopologie diese stattdessen verwendet werden, da sie ein Minimum an Relationen aufweist.

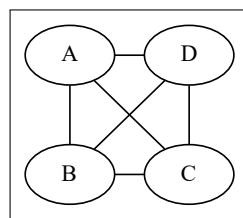


Abbildung 4.10: Vollvermaschte Topologie für 4 Objekte  $A, B, C$  und  $D$

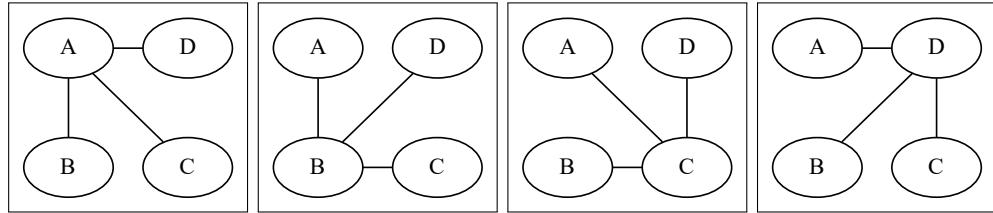


Abbildung 4.11: Sterntopologien

#### 4.2.4 Nachbarschaftsfunktion

Die Nachbarschaftsfunktion setzt sich aus drei Operationen zusammen:

- Hinzufügen von Relationen.
- Entfernen von Relationen.
- Vertauschen von Relationen.

Mit diesen Operationen sind alle Topologien von jeder Starttopologie aus erreichbar [Meh16, S.30–34].

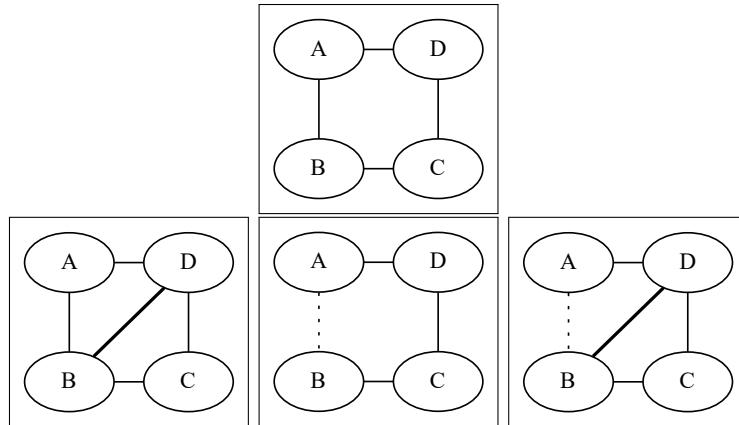


Abbildung 4.12: Nachbarschaftsfunktion

Aus der oberen Topologie in Abb. 4.12 entstehen die unteren Nachbarn durch, von links nach rechts:

**Hinzufügen** einer Kante  $B - D$  (fett).

**Entfernen** einer Kante  $A - B$  (gepunktet).

**Tauschen** einer Kante  $A - B$  mit  $B - D$ .

### 4.2.5 Kostenfunktion

Für die Kostenfunktion wurde hier, wie in ISM, eine gewichtete Summe von normalisierter Laufzeit und Fehlerkennungen gewählt. Allerdings treten im ISM Fehlerkennungen nur in Form von *false positives* auf, also fälschlicherweise als Szeneninstanzen erkannten Objektanordnungen. Hingegen kann es beim PSM auch zu *false negatives* kommen, bei denen korrekte Szeneninstanzen nicht als solche erkannt werden.

Dass false negatives auftreten können, liegt daran, dass anders als das deterministische ISM das PSM eine nichtdeterministische Komponente hat: den EM-Algorithmus. Dadurch wird dieselbe Relation nach einer Umkehrung durch Neuordnung des Graphen für ein anderes Wurzelobjekt nicht notwendigerweise gleich modelliert, ist also nicht unbedingt symmetrisch. Deshalb kann es durch die Bildung des Minimums bei mehreren Elternteilen und die des Maximums bei der Fusion über die Wurzelobjekte vorkommen, dass eine Wahrscheinlichkeit unter diejenige für die vollvermaschte Topologie sinkt. Dieses Problem ist zwar durch kombinatorische Optimierung über die Relationstopologien nicht direkt behebbar, es soll aber versucht werden, eine Topologie zu finden, die dagegen robust ist.

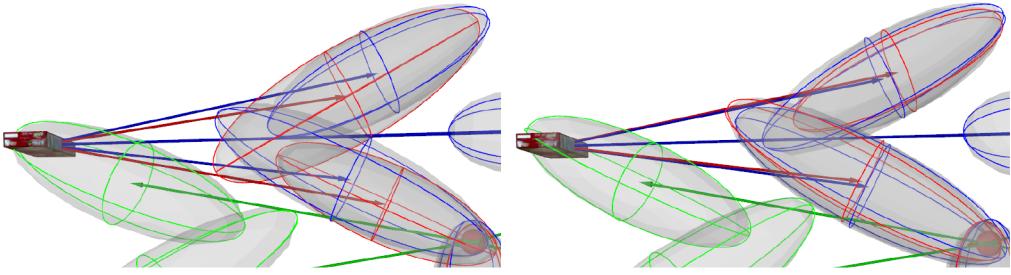


Abbildung 4.13: Unterschiedliche Gauß-Kernel für die selbe Relation

In Abb. 4.13 zu sehen sind ausschnittsweise zwei verschiedene Beschreibungen der selben Szene. Obwohl die roten bzw. blauen Kernel dieselbe Bewegung der roten Schale unten rechts relativ zur Schachtel links darstellen, unterscheiden sie sich sowohl untereinander (Asymmetrie) als auch zwischen den beiden Bildern. Das linke Modell würde mit dem roten Ellipsoid eine Beobachtung der roten Schale in der Bildmitte aufgrund des Überhangs eher akzeptieren als die rechte. Wird die Einteilung in true positives und negatives anhand der vollvermaschten Topologie also mithilfe dieses Modells vorgenommen, produziert das rechte in dem Fall ein false negative.

Es existieren also zwei Terme für Fehlerkennungen:

Sei  $T$  die normalisierte durchschnittliche Erkennungslaufzeit für die Testsets,  $f_p$  die normierte Anzahl von false positives und  $f_n$  die von false negatives, und seien  $w_t, w_p, w_n$  die Gewichte, dann ist die zu minimierende Kostenfunktion für die Topologie  $t$ :

$$c(t) = w_t T(t) + w_p f_p(t) + w_n f_n(t)$$

Über die Gewichte lässt sich bestimmen, welche Bedeutung den jeweiligen Termen bei gemessen wird.

#### 4.2.6 Testsets

Analog zu ISM erfolgt die **Generierung von Testsets**, indem aus jeder Trajektorie eine Beobachtung an einem zufälligen Zeitschritt ins Koordinatensystem eines zufällig gewählten Referenzobjekts transformiert wird [Han14]. Es kann sein, dass zufällig gültige relative Anordnungen gewählt werden. Dadurch entstehen insgesamt gültige Szeneninstanzen und ungültige (aber gültigen ähnlichen) Objektanordnungen, da die Modelle nicht alle Beobachtungen abdecken.

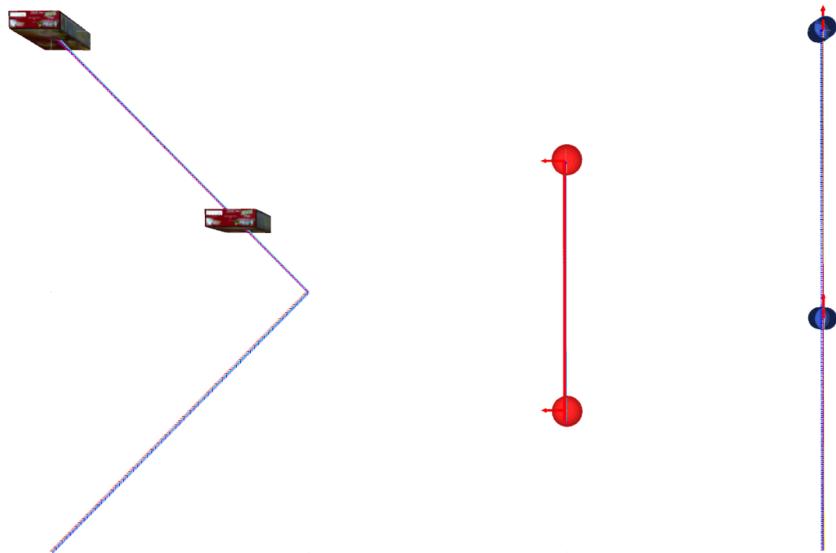


Abbildung 4.14: Testset

Abb. 4.14: Ein Beispiel für ein Testset, projiziert auf die Trajektorien, aus denen es gegriffen wurde (vgl. dazu auch Kap. 6.1). Die Objekte oben am Anfang der Trajektorien repräsentieren diejenigen, die sich darauf bewegen, die Objekte unterhalb davon das Testset. Dieses Set wurde mit einer Erkennungsschwelle (s.u.) von  $(10^{-4})^{3-1} = 10^{-8}$  als gültig eingestuft.

Es lässt sich hier zudem eine gewisse Wahrscheinlichkeit einstellen, mit der ein Objekt nicht in ein Testset aufgenommen wird, da das PSM explizit die Möglichkeit modelliert, dass auch eine unvollständige Szene als gültig aufgefasst werden kann. Die Einteilung in gültige und ungültige Testsets erfolgt mithilfe der vollvermaschten Topologie, welche die stärksten Einschränkungen darstellt.

Da die Inferenz des PSM Wahrscheinlichkeiten als Ergebnisse liefert, also weiche Entscheidungsgrenzen aufweist, im Gegensatz zu den harten des ISM, war es, um eine Szene binär als *erkannt* oder *nicht erkannt* einstufen zu können, nötig, einen **Erkennungsschwellwert (recognition threshold)** für die Wahrscheinlichkeit zu definieren, oberhalb dessen eine Szene als erkannt betrachtet wird.

In einem Binärbaum oder Sterngraphen für  $n$  Knoten (Objekte) gibt es  $n - 1$  Kanten, was die minimale Anzahl für einen zusammenhängenden Graphen darstellt. Da durch die Minimumsbildung auch für mehr Relationen nur die Wahrscheinlichkeiten über  $n - 1$  nicht verworfen werden, wird ausgehend von einer Grund-Erkennungsschwelle  $r_r$  der tatsächliche Wert der eigentlichen Erkennungsschwelle  $r$  als  $r = r_r^{n-1}$  berechnet.  $r_r$  stellt also etwas wie eine Schwelle für eine einzelne Relation dar (wobei durch die Multiplikation eine Relation die Schwäche einer anderen ausgleichen kann). Dadurch soll es ermöglicht werden, mit  $r_r$  eine von der Objektanzahl unabhängige Schwelle auszuwählen.

Testsets sollen auch aus Datenbanken geladen werden können.

Um den Fall abzudecken, dass die Testsets mithilfe eines anderen zugrundeliegenden Modells, etwa dem *ISM*, erstellt wurden und nicht vollständig mit dem *PSM* kompatibel sind, wird wenn nötig eine **Auswahl von Testsets** vorgenommen, was im *ISM* nicht der Fall ist. Dabei werden *gültige* Testsets, deren Wahrscheinlichkeit unter der höchsten eines *ungültigen* liegt und *ungültige*, die über der niedrigsten eines *gültigen* liegen, schrittweise entfernt.

Dabei wird das nächste zu entfernende folgendermaßen gewählt: Ist der Anteil der gültigen Testsets unterhalb der Schwelle an der Gesamtanzahl der gültigen Testsets höher als der Anteil falsch eingestufter ungültiger Testsets, wird das ungültige Testset mit der höchsten Wahrscheinlichkeit entfernt, ansonsten das gültige Testset mit der niedrigsten. Das wird solange wiederholt, bis keine Überschneidungen mehr vorliegen. Durch diese Heuristik soll die Anzahl entfernter Testsets möglichst gering gehalten werden.

Die Mitte zwischen den Wahrscheinlichkeiten der zuletzt noch übrigen höchstbewerteten ungültigen und niedrigstbewerteten gültigen Testsets kann als flexible Erkennungsschwelle ausgewählt werden (die dann nicht noch zusätzlich potenziert wird).

Sei in Algorithmus 4 die Menge der gültigen (validen) Testsets  $V$ , die der ungültigen (invaliden)  $I$  und  $r_{flex}$  die flexible Erkennungsschwelle. Sei außerdem  $P_{V,min}$  die niedrigste Wahrscheinlichkeit eines gültigen Sets und  $P_{I,max}$  die höchste eines ungültigen.

**Algorithm 4** Selektion von Testsets

---

```

1: procedure SELECTTESTSETS( $V, I$ )
2:   while  $V \neq \emptyset \wedge I \neq \emptyset$  do
3:      $P_{V,min} = \min_{v \in V} P(v)$ 
4:      $P_{I,max} = \max_{i \in I} P(i)$ 
5:     if  $P_{V,min} > P_{I,max}$  then ▷ done.
6:        $r_{flex} = P_{I,max} + (|P_{V,min} - P_{I,max}|/2)$ 
7:       return  $r_{flex}$ 
8:     end if
9:      $b \leftarrow a \leftarrow 0$  ▷ below and above
10:    for  $v \in V$  do ▷ count sets below
11:      if  $P(v) < P_{I,max}$  then
12:         $b \leftarrow b + 1$ 
13:      end if
14:    end for
15:    for  $i \in I$  do ▷ count sets above
16:      if  $P(i) > P_{V,min}$  then
17:         $a \leftarrow a + 1$ 
18:      end if
19:    end for
20:    if  $\frac{b}{|V|} > \frac{a}{|I|}$  then ▷ remove worse rated set
21:       $I \leftarrow I \setminus \arg \max_{i \in I} P(i)$ 
22:    else  $V \leftarrow V \setminus \arg \min_{v \in V} P(v)$ 
23:    end if
24:  end while
25: end procedure

```

---

**4.2.7 Anwendung der kombinatorischen Optimierung**

Für das ISM sind die drei oben beschriebenen Optimierungsalgorithmen *Hill Climbing*, *Record Hunt* und *Simulated Annealing* implementiert. In PSM werden diese Implementierungen unverändert mit den angepassten Funktionen wie oben erläutert verwendet.

**4.3 Lernen der Gaussian Mixture Models****4.3.1 Einleitung**

Zum Lernen der GMMs wird in PSM eine proprietäre Implementierung des Expectation-Maximization-Algorithmus verwendet. Diese wird durch diejenige in *OpenCV* ersetzt. „OpenCV (Open Source Computer Vision Library) ist eine Open-Source-Bibliothek für Computer Vision und Maschinenlernen.“ [ope]

Sie wird an anderen Stellen des Systems, zu dem PSM gehört, bereits verwendet, weshalb sie sich hier anbietet.

Genauer wird der in ihr implementierte EM-Algorithmus auf den Lerndaten aufgerufen. Das Ergebnis wird auf Gültigkeit überprüft und das BIC berechnet.

### 4.3.2 Gültigkeit der Kovarianzmatrizen

Eine gültige Kovarianzmatrix für einen Gaußkernel ist *positiv definit* [Pri12b], was für die vom *EM*-Algorithmus erzeugten Matrizen überprüft werden muss. Dieses Kriterium ist erfüllt, wenn die Eigenwerte  $\lambda_i$  der Matrix größer als Null sind (die *Eigenwerte* einer reellen Matrix  $\Sigma \in \mathbb{R}^{n \times n}$  sind  $\lambda_i$ , sodass es Vektoren  $v_i \in \mathbb{R}$  gibt mit  $\Sigma v_i = \lambda_i v_i$ ); deshalb werden die Eigenwerte berechnet und untersucht. Da die Matrizen bei der Berechnung der Mahalanobis-Distanz zudem invertiert werden, muss sichergestellt werden, dass sie regulär, ihre Eigenwerte also ungleich null sind. [Kü13]

Es muss also sichergestellt werden, dass für eine Matrix, die als Kovarianzmatrix verwendet werden soll, gilt:

$$\forall \lambda_i : \lambda_i > 0$$

Sollte das nicht der Fall sein, muss der EM-Algorithmus neu gestartet werden, solange, bis eine gültige Matrix gefunden wurde oder eine Obergrenze an Wiederholungen erreicht ist.

### 4.3.3 Bayes'sches Informationskriterium

Zur Berechnung des BIC müssen die freien Parameter der Matrizen, Mittelwerte und die Gewichte betrachtet werden. Eine quadratische, symmetrische Matrix  $\Sigma$  mit Dimension  $n \times n$  hat beispielsweise  $n + (n - 1) + \dots + 2 + 1 = \frac{n(n+1)}{2}$  freie Einträge, eine Diagonalmatrix (mit 0 überall außer auf der Diagonalen)  $n$ , ebenso ein Mittelwertvektor  $\mu$  mit Dimension  $n$ . Da die Gewichte sich zu 1 aufsummieren sollen, ist eines durch die anderen festgelegt, für  $m$  Gewichte sind also  $m - 1$  frei.

# 5. Implementierung

Im Folgenden wird auf die Klassen eingegangen, die zur Umsetzung des vorgestellten Konzepts implementiert wurden, in etwa in der Reihenfolge, wie sie im Programm benutzt werden.

## 5.1 Kombinatorische Optimierung

### 5.1.1 Klassendiagramm

Das Herzstück der kombinatorischen Optimierung bildet der *CombinatorialOptimizer*, der mittels des *CombinatorialTrainer* über die Schnittstelle des *AbstractTrainer* an das bestehende System angebunden wird.

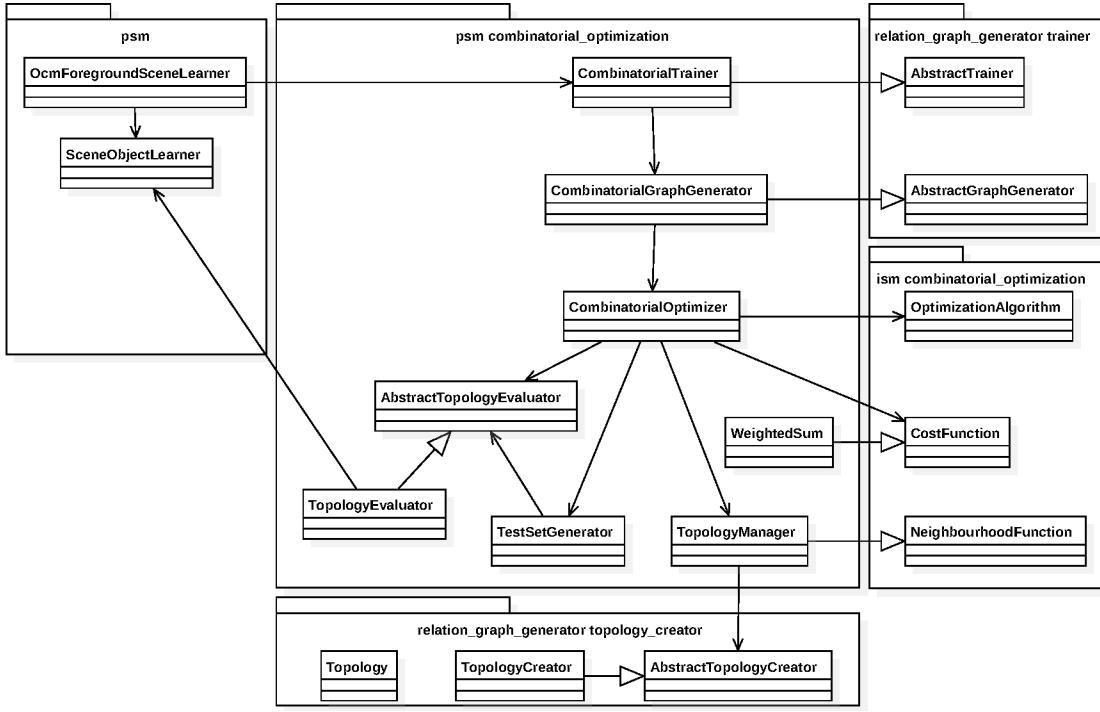
Abbildung 5.1: Klassendiagramm *CombinatorialOptimizer*

Abb. 5.1: Ausschnitt aus dem Klassendiagramm für den *CombinatorialOptimizer*. Bereits bestehende Klassen in *psm*, *ism combinatorial\_optimization* und *relation\_graph\_generator\_trainer*.

Neu hinzugekommene in *psm combinatorial\_optimization* und *relation\_graph\_generator\_topology\_creator*.

### 5.1.2 Schnittstelle zum bestehenden System

PSM bietet die Möglichkeit, mehrere Szenen in einem Modell darzustellen. Ein Modell beinhaltet aber immer mindestens zwei Szenen: Den Hintergrund, der Rauschen im Objekterkenner darstellt und den Vordergrund, die eigentliche aus Objekten bestehende Szene. Der **OcmForegroundSceneLearner** lernt ein Object Constellation Model des Vordergrunds über eine bestimmte Szene. Dafür stößt er in der Funktion `learn()` die Erstellung eines Relationsgraphen an und benutzt das Ergebnis für die Lerner für die einzelnen Objekte, **SceneObjectLearner**. Zusätzlich zu der heuristischen Erstellungsmethode, die mittels eines Parameters weiterhin ausgewählt werden kann, wurde die von **CombinatorialTrainer** durchgeführte kombinatorische Optimierung implementiert. Zudem ist als dritte Möglichkeit eine vollvermaschte Relationstopologie auswählbar, falls etwa die Rechenzeit in der Inferenz gegenüber der Erkennungsgenauigkeit einmal keine Rolle spielen sollte. Der **FullyMeshedTrainer**, der dafür verantwortlich ist, und der alte

**PSMTrainer** sind Unterklassen von **AbstractTrainer**, ebenso der **CombinatorialTrainer**.

Intern benutzt dieser über den *CombinatorialGraphGenerator* den *CombinatorialOptimizer*. Der **CombinatorialTrainer** nimmt als Quelle für die Lerndaten, die er an den *CombinatorialOptimizer* weiterreicht, eine *ExamplesListSource*, welche die übergebenen Daten ohne Änderungen übergibt; sie implementiert *AbstractSource*. Diese dient dazu, die Schnittstelle des **AbstractTrainer** zu bedienen.

### 5.1.3 CombinatorialOptimizer

Der **CombinatorialOptimizer** bereitet die Optimierung vor und stößt sie an. Er erstellt einen *TopologyEvaluator*, der Topologien auswertet, den *TopologyManager*, welcher Topologien erstellt, verwaltet und als Nachbarschaftsfunktion für die eigentlichen Optimierungsalgorithmen dient, sowie den *TestSetGenerator*.

Der *CombinatorialOptimizer* benutzt den *TopologyEvaluator*, um die vollvermaschte Topologie und die Sterntopologien auszuwerten, die, wie oben erläutert, Schranken für Laufzeit und Fehlerkennungen liefern. Des Weiteren erstellt er die Starttopologien. Es sind zwei Möglichkeiten implementiert: zufällige Starttopologien, wie in ISM, und die am besten bewertete aus den zuvor betrachteten.

Zuletzt erstellt der **CombinatorialOptimizer** die *WeightedSum* als Kostenfunktion und einen der drei für ISM implementierten Optimierungsalgorithmen.

Mit dem Aufruf der Funktion *runOptimization()* wird die Optimierung gestartet; als Ergebnis liefert sie die am besten bewertete Topologie zurück. Die Klasse *Topology* beinhaltet dann den zugehörigen Relationsbaum.

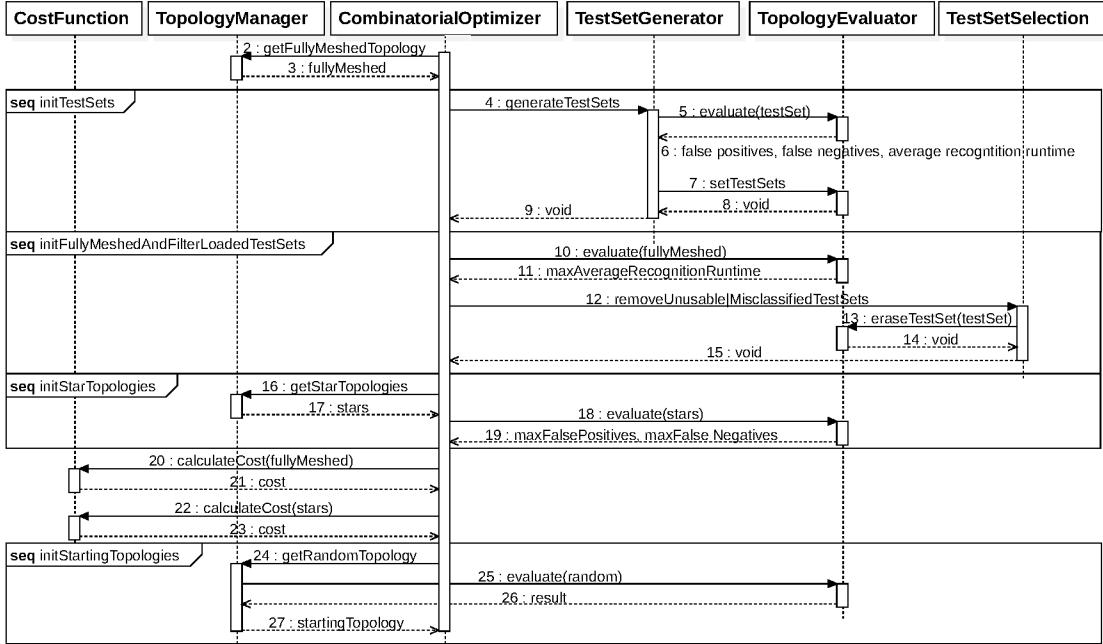
Abbildung 5.2: Zeitlicher Ablauf der Initialisierung des *CombinatorialOptimizer*

Abb. 5.2 stellt verkürzt den Ablauf der Initialisierung des *CombinatorialOptimizer* dar. Die Unter-Sequenzen repräsentieren Funktionen des *CombinatorialOptimizer*. Nach der Erstellung der vollvermaschten Topologie werden mit ihrer Hilfe die Testsets erstellt (abgebildet) oder geladen. Danach wird die vollvermaschte Topologie, im Fall, dass die Testsets geladen wurden, über den Testsets ausgewertet und ungültige aussortiert, entweder mit der Funktion *removeUnusableTestSets()* oder *removeMisclassifiedTestSets()*. Da der hier präsentierte Fall nur eintreten kann, wenn die Testsets aus einer Datei geladen wurden, ist er komplementär zu dem darüber gezeigten der Erstellung. Es werden dennoch beide gezeigt, da sie die jeweils aufwändigere Interaktion zur Folge haben. Nach der Initialisierung der Sterntopologien zum Finden der maximalen Fehlererkennungszahlen kann damit die Kostenfunktion initialisiert werden. Zuletzt ist die Initialisierung von zufälligen Starttopologien gezeigt; andernfalls wird als solche einfach die am besten bewertete aus denen gewählt, deren Kosten bereits berechnet wurden. Nicht im Diagramm zu sehen ist die darauf folgende Initialisierung des *TopologyManagers* als Nachbarschaftsfunktion sowie die des Optimierungsalgorithmus.

### 5.1.4 OptimizationAlgorithm

**OptimizationAlgorithm** ist eine von ISM bereitgestellte abstrakte Oberklasse über die drei implementierten Algorithmen *HillClimbing*, *RecordHunt* und *SimulatedAnnealing*. Es wurde daran nichts geändert und keine weiteren Algorithmen implementiert, was aber

theoretisch möglich wäre, wodurch weitere Algorithmen hinzugefügt werden können. Jeder **OptimizationAlgorithm** hat eine *CostFunction* und eine *NeighbourhoodFunction*, mit einem dem Problem entsprechenden Instanzentyp templatisiert (hier: *Topology*), und eine Funktion *optimize()*, welche die Optimierung durchführt und das Ergebnis zurückgibt.

### 5.1.5 WeightedSum

Die Klasse **WeightedSum** stellt eine Implementierung der ISM-Oberklasse *CostFunction* dar. Sie implementiert eine Funktion *calculateCost(Topology)*, welche die Kosten der Topologie anhand der normalisierten Erkennungslaufzeit und Anzahl von Fehlerkennungen als gewichtete Summe berechnet und zurückgibt.

### 5.1.6 TopologyManager

Der **TopologyManager** erbt von der abstrakten Klasse *NeighbourhoodFunction* aus ISM. Er implementiert die Funktionen *setReferenceInstance(Topology)* sowie *hasNextNeighbour()* und *getNextNeighbour()*. Beim Aufruf der erstenen werden die Nachbarn der übergebenen Topologie erstellt, aber noch nicht ausgewertet. Das geschieht erst beim Aufruf von *getNextTopology()*, da abhängig vom Optimierungsalgorithmus nicht alle Nachbarn einer Topologie untersucht werden und somit Zeit gespart werden kann. Des Weiteren dient der **TopologyManager** als zentrale Stelle der Verwaltung der Topologien und bietet als solche Funktionen zur Erstellung der vollvermaschten Topologie, der Stern- sowie von zufälligen Topologien an, in denen er den *TopologyCreator* benutzt. Zudem führt er die Erstellung von Bäumen aus den Topologien mithilfe des *TopologyTreeTrainer* durch.

Zuletzt führt er über alle besuchten Topologien Buch und erlaubt dadurch, eine Historie des Optimierungsverlaufes auszugeben, entweder in Textform oder als eine svg-Grafik, in der die untersuchten Topologien als Kreise dargestellt werden, die Informationen über die Erkennungslaufzeit, Anzahl von Fehlerkennungen und die daraus errechneten Kosten visualisieren. Dazu wird ein von ISM bereits zur Verfügung gestelltes Framework herangezogen.

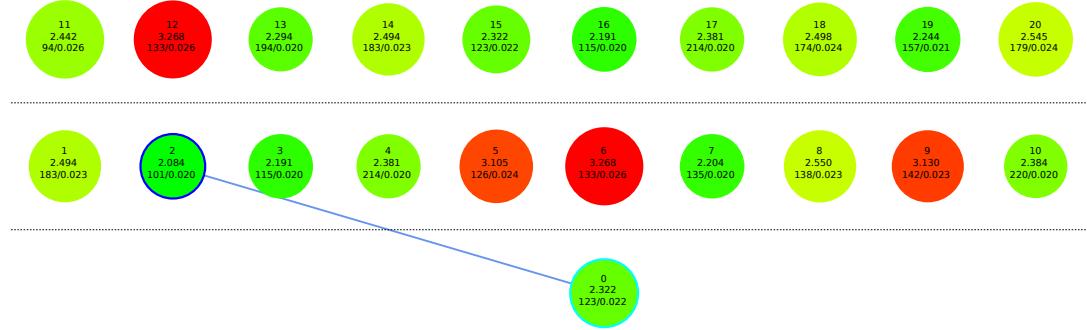


Abbildung 5.3: Optimierungsverlauf

In Abb. 5.3 ist ein Beispiel zu sehen. Die Kreise stellen untersuchte Topologien dar. Die oberste Zahl bezeichnet die Nummer der Topologie. Die mittlere die Kosten, die sich aus den Fehlerkennungen (Summe der false positives und false negatives, da letztere in der zugrundeliegenden Implementierung aus dem ISM nicht gesondert ausgegeben werden) und der durchschnittlichen Erkennungslaufzeit in Sekunden, welche darunter angegeben sind, errechnet. Die Größe der Kreise hängt von der Laufzeit ab (je größer desto länger die Laufzeit), die Farbe von den Kosten. Die von einem hellblauen Kreis umgebene Topologie (hier nur eine) ist die im jeweiligen (durch gepunktete Linien abgetrennten) Schritt ausgewählte Topologie. Der dunkelblaue Kreis bezeichnet die beste, am Ende ausgewählte Topologie. Die durchgezogene blaue Linie ist der Pfad zwischen den ausgewählten Topologien. Im letzten Schritt wird keine Verbesserung mehr gefunden (weshalb, da es sich hierbei um eine Historie von Hill Climbing handelt, die beste aus dem vorigen Schritt ausgewählt wird).

### 5.1.7 TopologyEvaluator

Der **TopologyEvaluator**, welcher die Schnittstelle *AbstractTopologyEvaluator* implementiert, benutzt die Testsets aus dem *TestSetGenerator*, die ihm über die Funktionen *setValidTestSets(Testsets)* und *setInvalidTestSets(Testsets)* übergeben werden, um Topologien auf Erkennungslaufzeit und Anzahl von Fehlerkennungen auszuwerten. Dazu dient *evaluate(Topology)*.

Der **TopologyEvaluator** erstellt ausgehend vom *Baum mit Verweisen* der Topologie ein Teilmodell des PSM, das nur das OCM des Vordergrunds enthält. Ein vollständiges Modell benötigt Informationen von außerhalb des *OcmForegroundSceneLearners*. Zudem ist das Programm in seiner Implementierung so strukturiert, dass die Verwendung von anderen Teilmodellen außer OCM möglich wäre, die unter Umständen andere Relationsgraphen oder gar keine verwenden könnten. Aufgrund dieses Mangels an Information wird nur der an dieser Stelle bekannte Teil des Modells benutzt. Zum Lernen verwendet der **TopologyEvaluator** dieselben *SceneObjectLearners* wie der *OcmForegroundSceneLearner*. Die partiellen Modelle werden demjenigen Teil des Inferenzsystems übergeben,

der für ihre Auswertung zuständig ist, *ForegroundSceneContent*. Sie können auch als .xml-Dateien ausgegeben werden, analog zur Ausgabe des vollständigen Modells am Ende des Lernvorgangs.

```

- <partial_model run="0" topology_id="111">
  - <object name="CupRoundHandle" type="ocm" priori="1">
    <slots number="3"/>
    + <shape></shape>
    + <appearance></appearance>
    + <occlusion></occlusion>
  </object>
  + <object name="BowlSmall" type="ocm" priori="1"></object>
  + <object name="Smacks" type="ocm" priori="1"></object>
</partial_model>
```

Abbildung 5.4: Teilmodell als .xml-Baum

Dargestellt ist in Abb. 5.4 das erste betrachtete Teilmodell eines Optimierungslaufs (*run* = "0"). Dabei handelt es sich immer um das für die vollvermaschte Topologie. Hier für eine Frühstücksszene mit 3 Objekten (daher auch 3 Slots). Die *topology.id* entspricht dem Bitvektor (da alle Relationen vorhanden, 111). Die Terme entsprechen denen aus den vollständigen PSM-xmlds und sind aus Platzgründen eingeklappt, ebenso wie die weiteren Objekte.

Dann werden die Testsets mit dem Modell von der Inferenz ausgewertet. Dies kann analog zur Visualisierung der gewöhnlichen Inferenz graphisch dargestellt werden.

Abb. 5.5 zeigt die Visualisierung der Inferenz gegen zwei Testsets für ein dreiecksförmiges Teilmodell, Pfeile geben Relationen und Farben deren Bewertung an (grün-hoch, rot-niedrig).

Die Laufzeit wird gemessen und zuletzt über die Testsets gemittelt. Die Anzahl der anhand der zuvor aus einem übergebenen Parameter berechneten Erkennungsschwelle falsch als valide oder invalide charakterisierten Testsets wird gezählt, d.h als false positives diejenigen ungültigen Testsets, deren Wahrscheinlichkeit über der Erkennungsschwelle lag, als false negatives die gültigen darunter. Die Ergebnisse werden innerhalb der *Topology* gespeichert.

### 5.1.8 TestSetGenerator

Der **TestSetGenerator** erstellt beim Aufruf der Funktion *generateTestSets(Observations, testSetCount)* aus den Beobachtungen *testSetCount* zufällige Testsets, also Objektan-

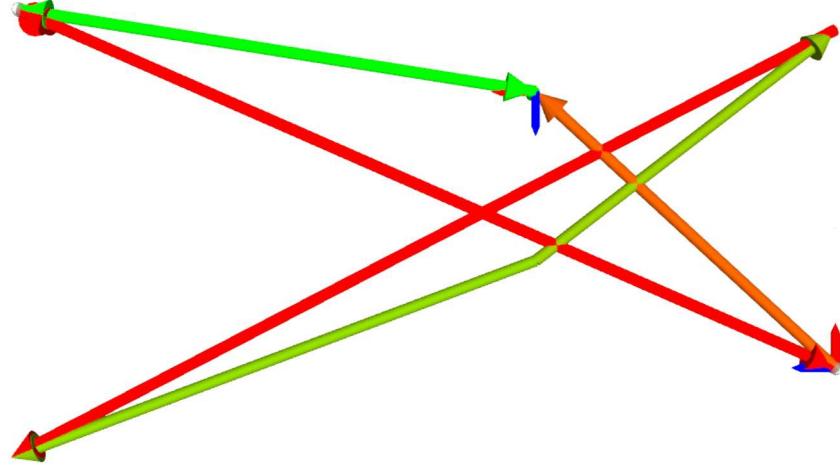


Abbildung 5.5: Teilmodell-Visualisierung

ordnungen, die nicht unbedingt gültige Szenen darstellen, mithilfe des oben erläuterten Algorithmus. Die Testsets werden über der vollvermaschten Topologie vom *Topology-Evaluator* ausgewertet und in gültige und ungültige eingeteilt.

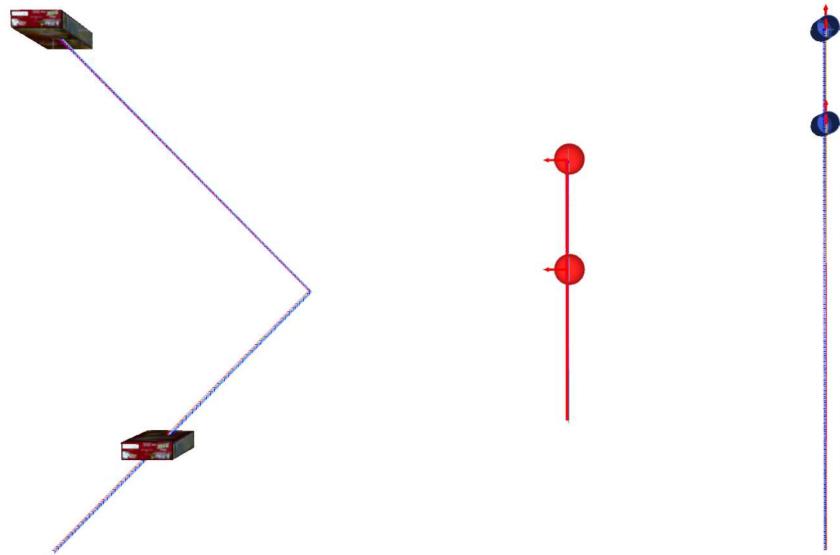


Abbildung 5.6: Ungültiges Testset

Abb. 5.6: Wiederum ein Testset (vgl. Abb. 4.14), dieses Mal als *ungültig* eingestuft, da die Objekte weiter auseinander liegen als beim Training je der Fall.

Um Zeit zu sparen kann der **TestSetGenerator** auch Testsets aus sqlite-Datenbanken

laden, die er zuvor geschrieben hat, aber auch aus solchen, die vom *ISM-TestSetGenerator* geschrieben wurden, um sie vergleichen und austauschbar verwenden zu können.

Die eigentliche Erstellung der Testsets findet in Unterklassen statt, die *generateRandomSets()* implementieren. **RelativeTestSetGenerator** nimmt für jedes Objekt eine Beobachtung von einem zufälligen Zeitpunkt aus den Lerndaten, wählt zusätzlich dazu ein Objekt als Referenz aus und transformiert alle Beobachtungen in dessen Koordinatensystem, in Anlehnung an *ISM*. Er implementiert also das oben beschriebene Konzept.

### 5.1.9 TestSetSelection

Werden Testsets aus Datenbanken geladen, werden diejenigen, die anhand der gegebenen Erkennungsschwelle falsch eingestuft werden (etwa, weil sie mit einer anderen generiert wurden) nach dem Laden entfernt. Der *CombinatorialOptimizer* stößt das bei der Initialisierung der vollvermaschten Topologie an, wenn nötig. Dabei wird die Funktion *removeMisclassifiedTestSets(recognitionThreshold)* der Klasse **TestSetSelection** benutzt.

Alternativ kann eine flexible Erkennungsschwelle eingestellt werden, die so gewählt wird, dass sie die geladenen Testsets möglichst richtig klassifiziert. Die Entfernung von Testsets, deren Wahrscheinlichkeiten zu Überschneidungen zwischen gültigen und ungültigen führen würden, geschieht mithilfe der Funktion *TestSetSelection::removeUnusableTestSets()*. Diese implementiert den in Kap. 4.2.6 erläuterten Algorithmus.

Die Funktion gibt die zuletzt noch übrige höchste ungültige bzw. niedrigste gültige Wahrscheinlichkeit zurück (auch wenn keine Sets entfernt werden mussten). Als der flexible (nicht zusätzlich potenzierte) recognition threshold wird die Mitte zwischen den beiden gewählt.

Durch einen Parameter kann ausgewählt werden, dass das Programm danach beendet wird, was nützlich sein kann, um anhand anderweitig erstellter Testsets einen recognition threshold zu bestimmen. Wenn die Testsets nicht geladen sondern erstellt werden, endet das Programm dann nach der Erstellung.

## 5.2 Relation Graph Generator

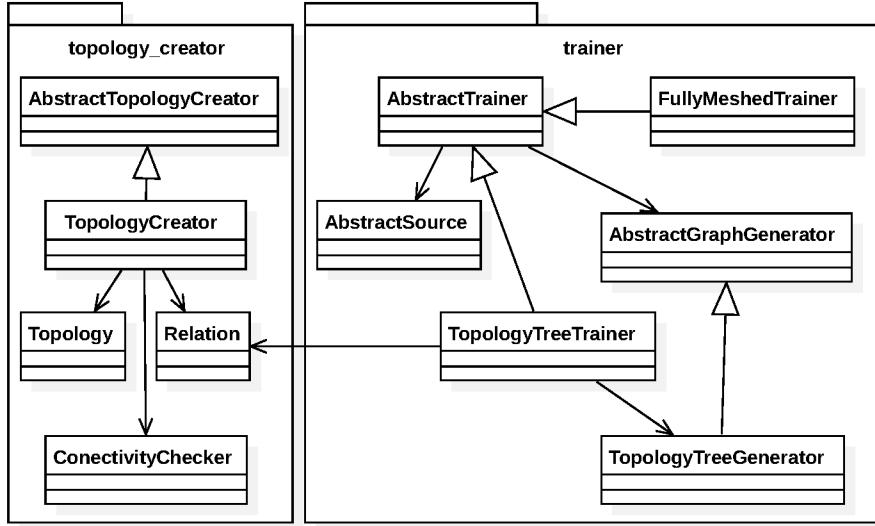


Abbildung 5.7: Klassendiagramm *RelationGraphGenerator*

Abb. 5.7 ist ein Ausschnitt aus dem Klassendiagramm für die beiden Teile des Pakets *Relation Graph Generator*, *topology\_creator* und *trainer*.

### 5.2.1 TopologyCreator

Der **TopologyCreator**, Unterklasse des *AbstractTopologyCreator*, wird für eine gewisse Objektmenge initialisiert, auf der er die möglichen Topologien erstellen kann. Er implementiert Funktionen zur Ausgabe der vollvermaschten Topologie, aller möglichen Stern-topologien sowie zufälliger Topologien.

Die zentrale Funktion ist *generateNeighbours(Topology)*, welche die Nachbarn der Topologie zurückgibt und vom *TopologyManager* benutzt wird.

Intern stellt der **TopologyCreator** wie in ISM die Topologien als Bitvektoren dar. Jedes Bit steht für eine der möglichen Relationen zwischen den gegebenen Objekten, eine 1 gibt an, dass die Relation in der dargestellten Topologie auftritt, eine 0, dass nicht. So ist etwa die vollvermaschte Topologie durch einen Vektor dargestellt, der nur Einsen enthält [Meh16, S.32].

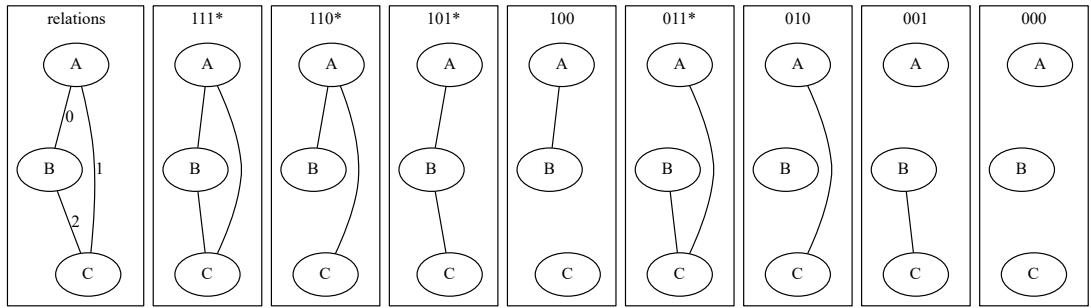


Abbildung 5.8: Darstellung der Relationen durch Bitvektoren

Abb. 5.8 illustriert ganz links alle möglichen Relationen in einer Szene aus drei Objekten, durchnummieriert ("relations"). Rechts alle möglichen Kombinationen von Relationen, durch Bitvektoren dargestellt (Bits in der Reihenfolge der Nummerierung). Ein Stern kennzeichnet zusammenhängende Graphen.

Um bei der Nachbarbildung Relationen hinzuzufügen bzw. zu entfernen genügt es also, ein Bit umzukehren.

Der **TopologyCreator** verwendet die Klasse *ConnectivityChecker* um sicherzustellen, dass erstellte Nachbarn zusammenhängende Graphen darstellen, da nur solche vom PSM ausgewertet werden können. Dazu wird eine Breitensuche durchgeführt und überprüft, ob alle Knoten besucht werden konnten. [Meh16, S.34]

Wurde durch einen Parameter die Anzahl von Nachbarn, die in einem Schritt höchstens betrachtet werden sollen, eingeschränkt, aber davon mehr gefunden, gibt der **TopologyCreator** eine entsprechende Anzahl zufällig ausgewählter zurück.

### 5.2.2 TopologyTreeTrainer und TopologyTreeGenerator

Der **TopologyTreeTrainer**, der allgemeine Graphtopologien in *Bäume mit Verweisen* umwandelt, implementiert den *AbstractTrainer*, wie der heuristische *PSMTrainer*. Er kann also auch an dessen Stelle eingesetzt werden, wenn eine vorher feststehende Topologie, bestehend aus *Relations* zwischen je zwei Objekten, als Grundlage für den Relationsgraphen benutzt werden soll, was hier allerdings nicht der Fall ist. Stattdessen wird der TopologyTrainer vom *TopologyManager* benutzt, um Topologien ihre Bäume zuzuordnen.

Als *AbstractTrainer* hat der **TopologyTreeTrainer** eine *AbstractSource* und einen *AbstractGenerator*. Als Quelle benutzt er, wie der *PSMTrainer*, eine *PbdSceneGraphSource*, welche die beobachteten Trajektorien liefert, die den Baumknoten den repräsentierten Objekten entsprechend zugeordnet werden. Von dort werden sie später von Lerner geholt. Der Generator ist ein **TopologyTreeGenerator**. Dieser übernimmt die Hauptarbeit. Mittels der Funktion *setRelations(vector<Relation>)* werden ihm vom **TopologyTreeTrainer** die Relationen weitergereicht und mittels *buildTree()* daraus

ein Baum aufgebaut. Diese Funktion wiederum wird beim Aufruf von *AbstractTrainer::loadTrajectoriesAndBuildTree()* aufgerufen. Es existieren zwei Varianten davon, wobei die eine zusätzlich einen Parameter nimmt, der den Objekttyp angibt, welcher die Wurzel bilden soll. Das wird durch zwei entsprechende Versionen der Funktion *buildTree()* von **TopologyTreeGenerator** unterstützt. Wird kein Parameter übergeben, wird der Knoten für das erste gefundenen Objekt als Wurzel gewählt.

Der Aufbau des Baums mit Verweisen erfolgt mittels des oben beschriebenen auf Breitensuche basierenden Algorithmus 1.

## 5.3 Lernen der Gauß-Mischverteilungen

### 5.3.1 Anbindung ans bestehende System

Die Ersetzung der alten Bibliothek wurde in der Klasse **GMMParameterEstimator** vorgenommen, wo sie benutzt wurde. Im Kern steht die Funktion *learn()* dieser Klasse, in der mehrere GMMs gelernt und anhand des BIC verglichen werden. Dazu wird die Funktion *runExpectationMaximization()* für jede Anzahl Kernel zwischen den durch Parameter vorgegebenen Minima und Maxima und für eine vorgegebene Anzahl Wiederholungen aufgerufen und die Ergebnisse mithilfe des BIC verglichen.

*runExpectationMaximization()* erstellt intern eine OpenCV-Klasse, *EM*, und führt durch Aufruf von *train()* auf den Lerndaten die Expectation Maximization durch. Danach berechnet die Funktion die Anzahl der freien Parameter, die für das BIC benötigt wird, berechnet dieses, und gibt die Ergebnisse und den gelernten Kernel zurück.

### 5.3.2 Gültigkeit der Kovarianzmatrizen

Um die Gültigkeit der Kovarianzmatrizen zu überprüfen, werden die Eigenwerte mithilfe der *Eigen*-Algebra-Bibliothek berechnet und darauf untersucht, ob sie größer 0 sind.

Gelingt es dem Algorithmus nicht, eine gültige Matrix zu finden, was insbesondere für kleine Lerndatenmengen häufig vorkommt, wird er erneut gestartet. Die maximale Anzahl der Versuche kann durch einen Parameter eingestellt werden. Nach der Hälfte der Versuche wird als Ziel des Algorithmus anstelle einer generischen Matrix eine Diagonalmatrix gewählt, die leichter zu finden, allerdings auch weniger präzise ist.

### 5.3.3 Dokumentierung

Zudem bietet die Klasse *GMMParameterEstimator* die Funktion *plotModel()* an, die das ausgewählte GMM in der Orientierung visualisiert (im PSM werden Position und Orientierung separat behandelt). Dazu werden zufällig mit der durch das GMM dargestellten Verteilung Werte gezogen und für die Rotationsachsen (Roll, Pitch und Yaw) jeweils ein Histogramm gebildet (wofür die Funktion *MathHelper::calcHistogram()* implementiert wurde). Mithilfe der neuen Klasse *GMMGnuplotVisualization*, genauer der Funktion *plotOrientationHistogram()*, werden die Histogramme über eine Hilfsbibliothek ausgegeben.

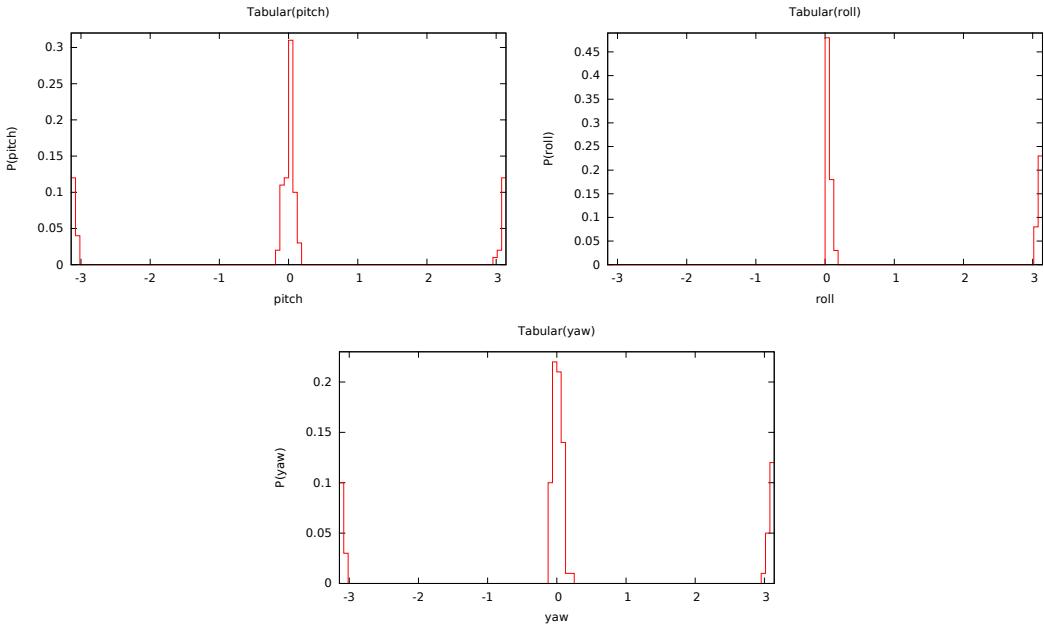


Abbildung 5.9: Plots für einen Gauß-Kernel in der Orientierung

Die Funktionalität, Kernel wie in Abb. 5.9 ausgeben zu lassen, wurde in Anlehnung an die alte Bibliothek (*ProBT* [pro]) neu implementiert.

## 5.4 Relationsgraphen in der Inferenz

Innerhalb des als .xml-Datei gespeicherten Modells wird ein Verweis als ein optionales Attribut realisiert, welches die Nummer des Knotens, auf den verwiesen wird, in der Reihenfolge einer Tiefensuche ohne Beachtung von Verweisknoten beinhaltet. Diese wird in der Methode *HierarchicalShapeModel::load()* erst durchgeführt, um den Knoten IDs zuzuweisen, und dann die entsprechenden Verweise gesetzt. Die bereits vorhandene Klasse wurde dazu entsprechend erweitert.

## 5.5 ConditionalProbability

Um die Möglichkeit anzubieten, bedingte Wahrscheinlichkeiten für mehrere Eltern zu unterstützen, wurde eine abstrakte Klasse **ConditionalProbability** angelegt.

Mit der Funktion *setProbability(parentId, probability)* wird eine Elternwahrscheinlichkeit  $f$  hinzugefügt. Jedes Elternteil hat höchstens eine Wahrscheinlichkeit; die ID dient zur Zuordnung und zur Dokumentation mit *printParentProbabilities()*.

Wird die Funktion *getProbability()* aufgerufen, ruft *ConditionalProbability* die von Unterklassen implementierte Funktion *calculateProbability()* auf, welche aus den Elternwahrscheinlichkeiten eine einzelne Wahrscheinlichkeit berechnet. Die Minimumsbildung wie oben erläutert wurde als **MinimumConditionalProbability** implementiert.

# **6. Evaluation**

Alle im Folgenden beschriebenen Experimente wurden auf einem Rechner mit 4-Kern Intel Core i5-4460-Prozessor mit 3.20GHz und 7,7 GiB RAM durchgeführt.

## **6.1 False Positive**

Es kann vorkommen, dass durch die bisherige Heuristik wichtige Relationen nicht beachtet werden und es dadurch zu Fehlerkennungen kommt. Dazu wird z.B. eine modifizierte Frühstücksszene betrachtet, die aus einer Müslischachtel, einer Müslischale in der Mitte und einer Tasse besteht: Während des Lernens bewegen sich diese aufs Aufzeichnungssystem zu, die Schale und die Tasse parallel zueinander, aber die Schale ein Stück weniger. Das Müsli bewegt sich zusätzlich auf die Schale zu, zum Einfüllen, und dann wieder davon weg. Müsli und Tasse befinden sich vom Beobachter aus immer auf der selben Höhe.

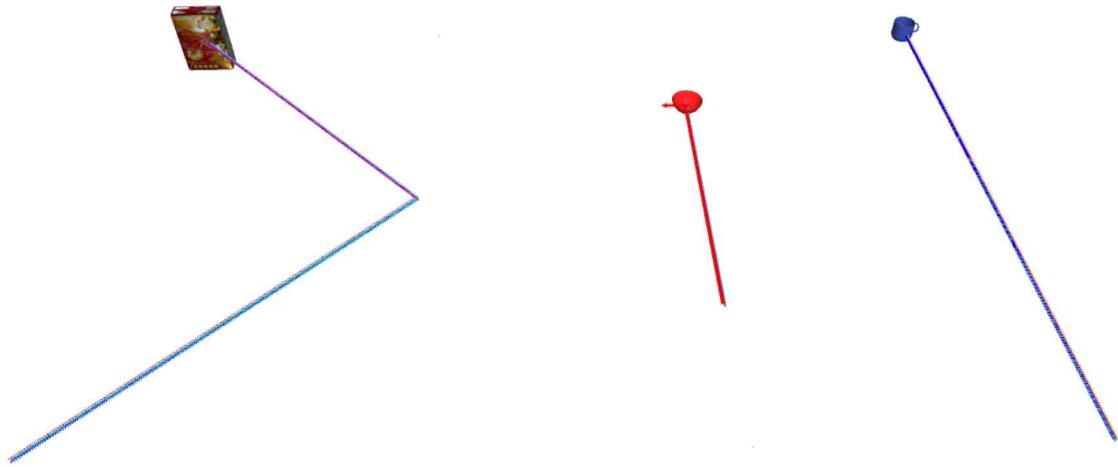


Abbildung 6.1: Frühstücksszene für false positive

In Abb. 6.1 zu sehen sind die Objekte und ihre Bewegung zum Beobachter, mit dem Knick in der Trajektorie der Müslischachtel links.

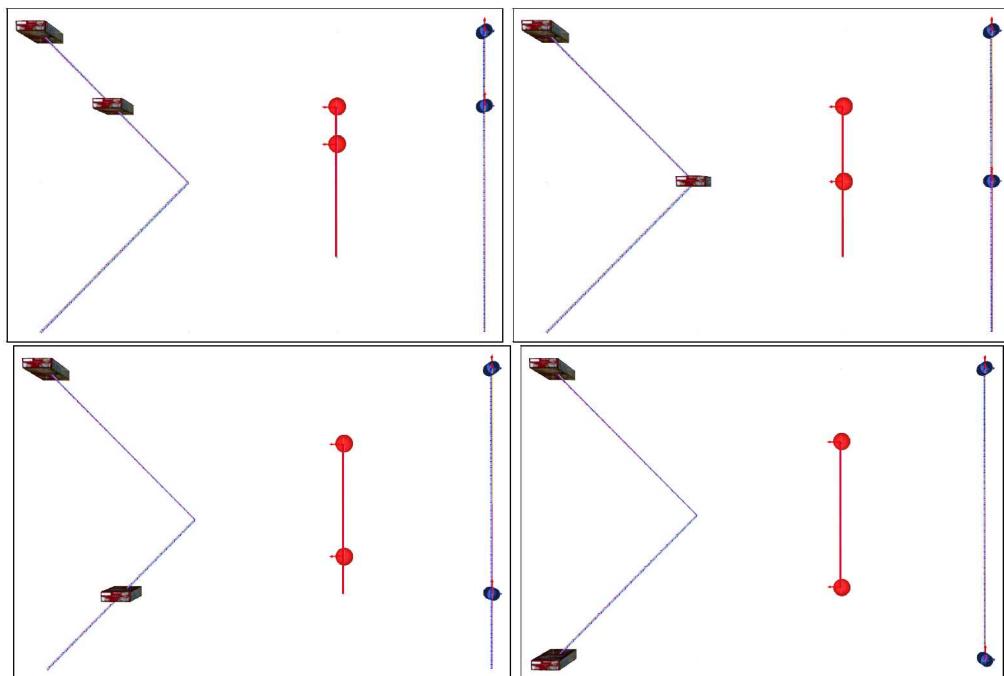


Abbildung 6.2: Zeitleiste der Bewegung der Frühstücksszene

In Abb. 6.2 gezeigt sind vier Schritte aus der Bewegung der Objekte. Müslischachtel links und Becher rechts haben jeweils die selbe Höhe.

Da die Trajektorien der Schale und der Tasse parallel sind, wird diese und nicht die abknickende Bewegung der Schale als Grundlage einer Relation ausgewählt. Die sich weniger bewegende Schale wird zum Referenzobjekt erklärt, an welches dann noch die übrig gebliebene Relation zum Müsli angehängt wird (da an die als Blatt deklarierte Tasse keine weiteren angefügt werden dürfen).

Dadurch wird aber die Beziehung zwischen Schachtel und Tasse nicht beachtet, die immer gleich weit vom Beobachter entfernt sein sollten.

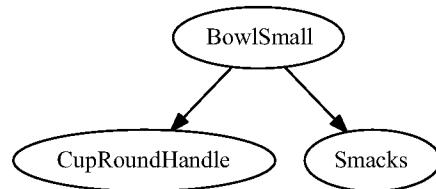


Abbildung 6.3: Baum aus Heuristik. Relation *Cup - Smacks* nicht repräsentiert

Deshalb führt eine Anordnung, bei welcher die Schachtel ober- und die Tasse unterhalb der Schale innerhalb der relativ dazu gültigen Bereiche sind, zu einer fälschlichen hohen Bewertung, einem *false positive*.

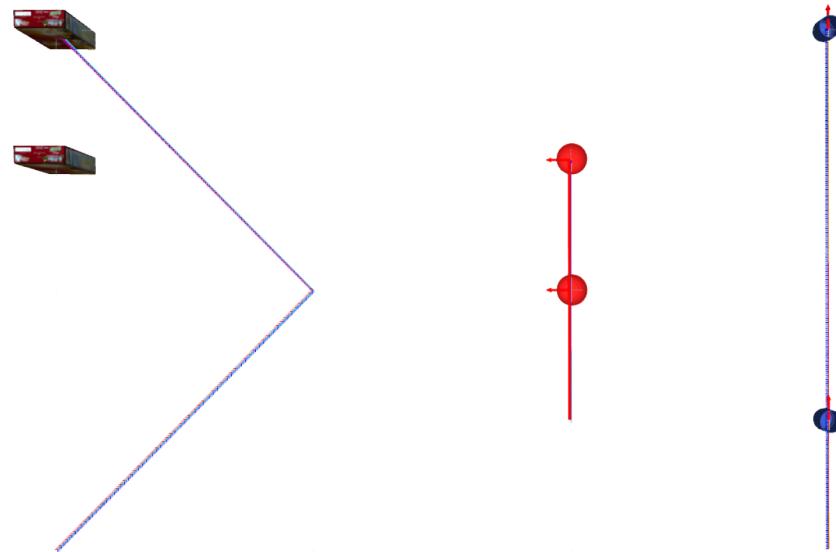


Abbildung 6.4: False positive erzeugende Evidenzen

Abb. 6.4: Die Evidenzen, für welche der von der Heuristik gefundene Baum zu einem

false positive führt, relativ zu den Trajektorien, über denen gelernt wurde.

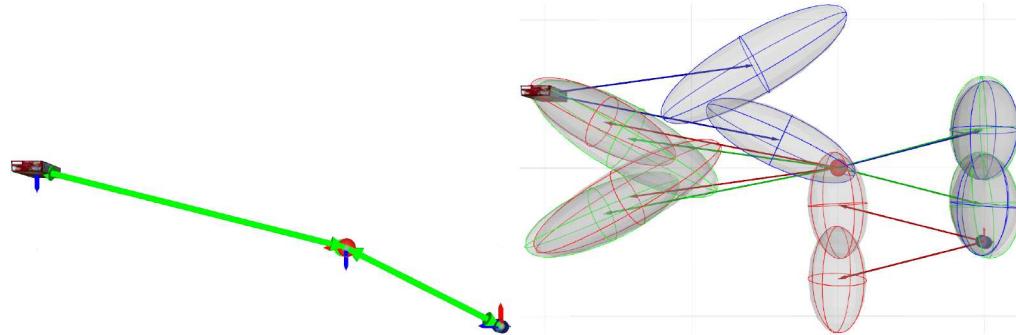


Abbildung 6.5: False positive

Links in Abb. 6.5 sind beide Relationen erfüllt (grüne Pfeile), deshalb wird die Beobachtung hoch bewertet. Rechts ist zu sehen, weshalb: Jedes Objekt liegt innerhalb eines einen Kernel repräsentierenden Ellipsoids von einem anderen, haben also eine niedrige Mahalanobis-Distanz. Die Wahrscheinlichkeit für die Szene rechts ist höher als 0,5, was hier als *akzeptiert* aufgefasst wird.

Durch der kombinatorische Optimierung wird die Bedeutung der Relation zwischen Schachtel und Tasse erkannt und entsprechend in den Relationsgraphen mit aufgenommen.

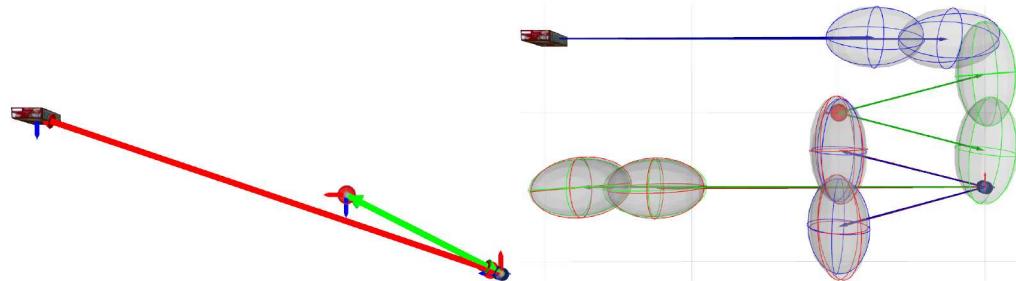
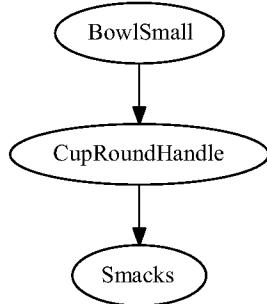


Abbildung 6.6: Vermeidung des false positive durch optimierte Topologie

Abb. 6.6: Die Relation zwischen Müsli und Becher ist nicht erfüllt (roter Pfeil), weshalb die Beobachtung abgelehnt wird. Müslipackung und Becher liegen weit entfernt von den ihnen zugeordneten Ellipsoiden des jeweils anderen Objektes.

Heuristik	Optimiert	Vollvermascht
0,592314	0,000702	0,000664

Tabelle 6.1: Durchschnittliche Wahrscheinlichkeit für Frühstücksszene

Abbildung 6.7: Optimierter Graph. Relation *CupRoundHandle - Smacks* repräsentiert

Die Bewertung durch den optimierten Graphen ist der durch die vollvermaschte Topologie aufgrund der Minimumsbildung, durch welche die zusätzliche Relation in der vollvermaschten nicht eingerechnet wird, sehr ähnlich, allerdings ist die Erkennungslaufzeit für den optimierten Graphen mit durchschnittlich 0,0024 s gegenüber 0,0028 s kürzer.

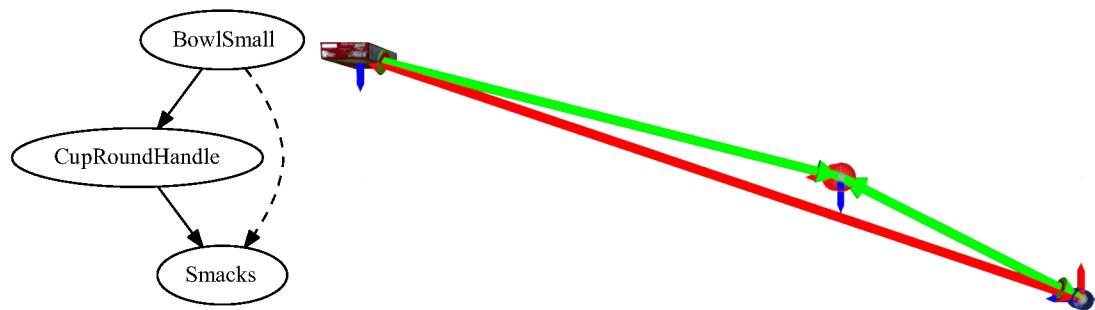


Abbildung 6.8: Vollvermaschte Topologie

Abb. 6.8: Auch die vollvermaschte Topologie vermeidet das false positive. Zwar hat sie die beiden erfüllten Relationen der Heuristik ebenfalls, lehnt aufgrund der Minimumsbildung aber eine davon zugunsten der nicht erfüllten ab (im gegebenen Beispielgraphen die gestrichelte Verweiskante zwischen *BowlSmall* und *Smacks*).

## 6.2 Erkennungslaufzeit und -qualität

Im Folgenden wird die Laufzeit und Qualität der Erkennung für Modelle derselben Szene verglichen, die von verschiedenen Algorithmen gefunden wurden.

### 6.2.1 Büroszene

Die hier benutzte Szene stellt einen Büroarbeitsplatz dar, bestehend aus zwei höhenverstellbaren Bildschirmen, Maus und Tastatur, die gegeneinander bewegt werden (Abb. 6.9). Die selbe Szene wurde schon in der Arbeit von *Jonas Mehlhaus* zum Vergleich verschiedener Algorithmen im ISM benutzt. [Meh16, S.52–55]

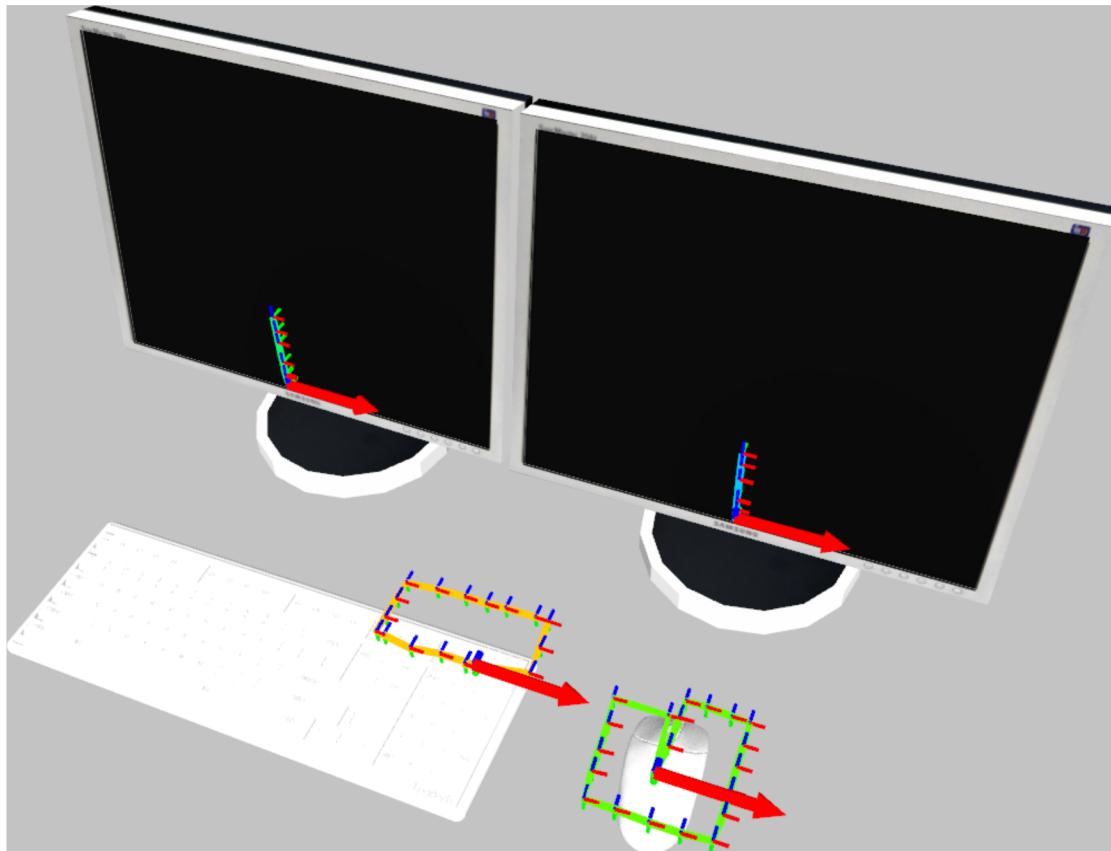


Abbildung 6.9: Lerndaten der Büroszene

Es wurden 600 zuvor mit dem *RelativeTestSetGenerator* erstellte Testsets benutzt, 243 gültige und 357 ungültige bei einer Erkennungsschwelle von  $2,0 \cdot 10^{-02}$  ( $\rightarrow 8 \cdot 10^{-6}$ ). Aufgrund des Zufallseinflusses auf die Kernel wurden in jedem Lauf durchschnittlich 61 gültige und 37 ungültige Sets entfernt, die vom jeweils konkreten, aus der vollvermaschten Topologie generierten Modell anders bewertet wurden. Diese unterschiedliche Bewertung kommt dadurch zustande, dass, um die geringe Anzahl Lerndaten für die Szene (51 pro Objekt) für den EM-Algorithmus zu kompensieren und aussagekräftig große Kernel zu generieren künstliche Samples im Umkreis der tatsächlichen zufällig erzeugt

werden (zur sog. *sample relaxation*), hier jeweils eines pro wirklich beobachtetem Datenpunkt. Ohne die *sample relaxation* werden für geringe Bewegung die Objekte exakt an der Stelle erwartet, wo sie beim Lernen beobachtet wurden, was der Intuition, dass das Programm wie ein Mensch eine abweichende Szeneninstanz mit einer gewissen Toleranz wiedererkennen sollte, zuwiderläuft ([Geh14, S.61]).

Werden die künstlichen Samples ausgeschaltet, werden zwar keine Testsets mehr nachträglich von der vollvermaschten Topologie anders eingestuft als zuvor, allerdings werden die Kernel dann sehr klein und die Wahrscheinlichkeiten sehr niedrig: um überhaupt noch eine Trennung zwischen gültigen und ungültigen Testsets vornehmen zu können, muss innerhalb des Lerners die Erkennungsschwelle auf  $2,0 \cdot 10^{-9}$  gesetzt werden. In der Inferenz ergibt sich nach der Normalisierung mit dem Hintergrundterm für gültige Sets eine Wahrscheinlichkeit in der Größenordnung von  $10^{-6}$  (für maximal 2 Kernel, wie beim Rest der Experimente. Für 5, was die Genauigkeit der Beschreibung noch erhöht, bei  $10^{-13}$ ), für ungültige im Bereich von  $10^{-23}$  ( $10^{-22}$ ), also verschwindend kleine Wahrscheinlichkeiten. Deshalb wird hier ein künstliches Sample pro Datenpunkt hinzugefügt und die dadurch eingeführte leichte Inkonsistenz zugunsten deutlicherer Wahrscheinlichkeiten toleriert. Für größere Lerndatensätze sind weniger bis keine künstlichen Samples nötig, weshalb das Problem dort von geringerer Relevanz ist.

### 6.2.2 Experimente

Für den Vergleich der Ergebnisse der Algorithmen unter Erkennungslaufzeit und Anzahl der Fehlerkennungen, also Summe von false positives und negatives, wurden die folgenden Parameter gewählt (die, sofern nicht anders angegeben, auch für die folgenden Kapiteln verwendet werden):

- Als Starttopologie die bestbewertete aus Sternen oder Vollvermaschter.
- Für Hill Climbing eine Random-Restart-Wahrscheinlichkeit von 0,2, was im Schnitt zu je zwei Durchläufen führte.
- Record Hunt: Initiales  $\Delta$  zur Kostenakzeptanz: 0,02. Senkungsfaktor: 0,01.
- Simulated Annealing: Starttemperatur 1. Endtemperatur 0,005. 8 Wiederholungen zwischen Temperaturupdates. Änderungsfaktor 0,9.

Es ergibt sich:

Algorithmus	false pos.	neg.	Summe	Zeit	Relationen	c
Vollvermascht	0,0	0,0	0,0	0,0234	6	2,5
RecordHunt	69,0	4,2	73,2	0,0161	4	2,194
HillClimbing	111,4	6,4	117,8	0,0141	3	2,145
SimulatedAnnealing	127,6	0,2	127,8	0,0134	3	1,999
Heuristik	148,4	5,96	154,4	0,0138	3	2,311
Stern	206,2	5,8	211,96	0,0158	3	2,675

Tabelle 6.2: Vergleich der Algorithmen, geordnet nach Summe der durchschnittlichen Fehlerkennungen

### 6.2.3 Fehlerkennungen

Die vollvermaschte Topologie hat die wenigsten Fehlerkennungen, danach das mithilfe von Record Hunt erzeugte Modell, vor denen der beiden anderen Optimierungsalgorithmen. Die Heuristik liefert mehr Fehlerkennungen als die optimierten, durch die Optimierung ist also ein Gewinn erzielt. Diejenige Sterntopologie mit den meisten Fehlerkennungen liegt, wie zu erwarten, abgeschlagen hinten.

Es treten deutlich weniger false negatives als positives auf, da diese, wie an früherer Stelle erläutert, nur durch zufallsbedingte Unterschiede in den Kerneln Zustände kommen (wobei auch hier die künstlichen Samples eine Rolle spielen).

### 6.2.4 Erkennungslaufzeit und Relationen

Die Erkennungszeit ist für die Topologien außer der vollvermaschten vergleichbar und korrespondiert, wie in der Tabelle zu sehen, mit der Anzahl Relationen. Da für vier Objekte maximal 6 Relationen möglich sind, und die Verweise, da sie durch Pointer realisiert sind, schnell abzuarbeiten sind, ist der Unterschied zwischen den kleineren Topologien und der vollvermaschten gering, aber sichtbar.

### 6.2.5 Kosten

Die letzte Zeile der Tabelle gibt die jeweiligen durchschnittlichen Kosten  $c$  der gewählten Topologien an. Am günstigsten bewertet wurde, trotz der vergleichsweise vielen Fehlerkennungen, die von Simulated Annealing gefundene Topologie, aufgrund ihrer niedrigen Laufzeit. Das ist dadurch verursacht, dass das Gewicht für die normalisierte Laufzeit mit 2,5 gegenüber jeweils 1 für false positives und false negatives recht hoch gewählt wurde, da für höhere Gewichtung der Fehlerkennungen die Optimierungsalgorithmen für diese Szene rasch zur Auswahl der vollvermaschten Topologie tendieren.

### 6.2.6 Besuchte Topologien

Für die Optimierungsalgorithmen lässt sich zusätzlich betrachten, wie viele Topologien während der Optimierung besucht wurden.

Algorithmus	Topologien	Schritte	Topologien pro Schritt
Hill Climbing	26.8	3.5	7.608333333333334
Record Hunt	151.4	48.0	3.2699438772609506
Simulated Annealing	415.2	140.2	3.03172860853792

Es wurden dabei für jeden Versuch erst die Topologien pro Schritt berechnet und dann darüber gemittelt.

Hill Climbing hat am meisten Topologien pro Schritt besucht, da dieser Algorithmus jeweils alle Nachbarn betrachtet. Interessanterweise besucht Simulated Annealing mehr Topologien als Record Hunt, schafft es aber dennoch nicht, zu einem entsprechend guten Ergebnis zu gelangen. Das deutet darauf hin, dass in diesem konkreten Experiment die Starttemperatur zu niedrig gewählt war bei gleichzeitigem geringen Abkühlen, so dass Bereiche um ein lokales Optimum lange abgesucht wurden.

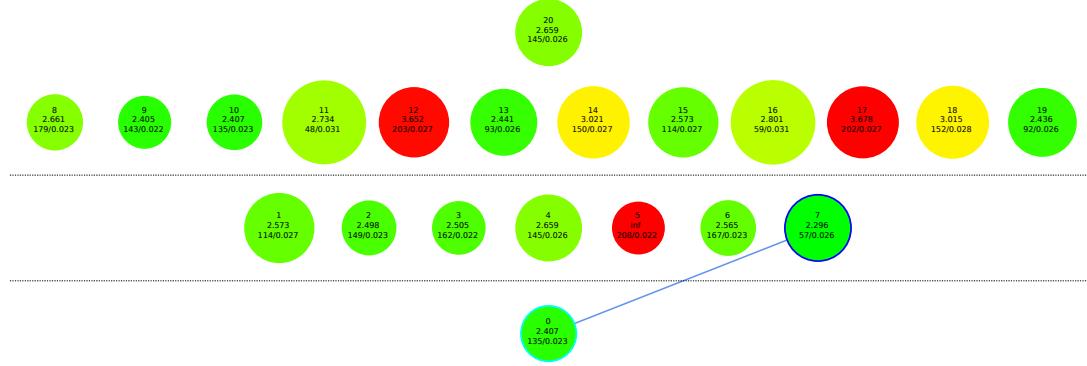


Abbildung 6.10: Beispiel für einen Optimierungsverlauf von Record Hunt

Abb. 6.10 zeigt eine Optimierungshistorie für *Record Hunt*. Aus Platzgründen ist ein uncharakteristisch kurzer Optimierungsverlauf dargestellt, ausgehend von der bestbewerteten Sterntopologie. Eine bessere Topologie wird aus weniger Nachbarn ausgewählt (im ersten Schritt also nicht alle besucht), danach die möglichen Nachbarn komplett durchsucht. Es wird allerdings keine gefunden, die im Auswahlbereich liegen würde. Ebenfalls aus Raumgründen ist keiner der langen Optimierungsverläufe für Simulated Annealing gezeigt. Für einen kurzen von Hill Climbing siehe Abb. 5.3, ausgehend von einer zufälligen Topologie im Rahmen eines Random Restart. Dort werden jeweils alle Nachbarn untersucht.

## 6.3 Modellwachstum

Werden von der kombinatorischen Optimierung mehr Relationen zur Beschreibung einer Szene ausgewählt, als von der Heuristik in einem Binärbaum, wächst auch das generierte Modell. Um das Wachstum zu evaluieren, wurde eine künstliche Szene verwendet, die der entspricht, mit der in der Arbeit von *Joachim Gehrung* die Laufzeit der ursprünglichen Fassung des Programms getestet wurde. Eine wachsende Anzahl von *Vitalis-Schoko-Schachteln* bewegt sich dabei in 200 Schritten parallel zueinander von einer imaginären Startlinie weg. Eine räumlich weitestgehend stationäre *Smacks-Schachtel* bleibt auf dieser Linie. Dabei wird zusätzlich für jedes Objekt ein geringes zufälliges Rauschen in der Pose hinzugefügt. Die Trajektorien lassen sich generell mit je einem Kernel beschreiben.

Da in der Arbeit von *Joachim Gehrung* bereits gezeigt wurde, dass das Modell mit der Komplexität der Szene wächst, soll das hier nicht weiter betrachtet werden, sondern stattdessen auf die Modellgrößen für Topologien, die von verschiedenen Algorithmen gefunden wurden, fokussiert werden. [Geh14, S.93–98]

Für 2 Objekte ergibt sich, da es nur eine mögliche Relation gibt, für jeden Algorithmus dasselbe Ergebnis, deshalb wird mit 3 Objekten begonnen. Für 5 und mehr wird die Laufzeit des Trainings sehr hoch (vergleiche dazu auch das Kapitel zur Evaluation der Laufzeit 6.5), weshalb bis zu einer Szene aus 5 Objekten untersucht wird.

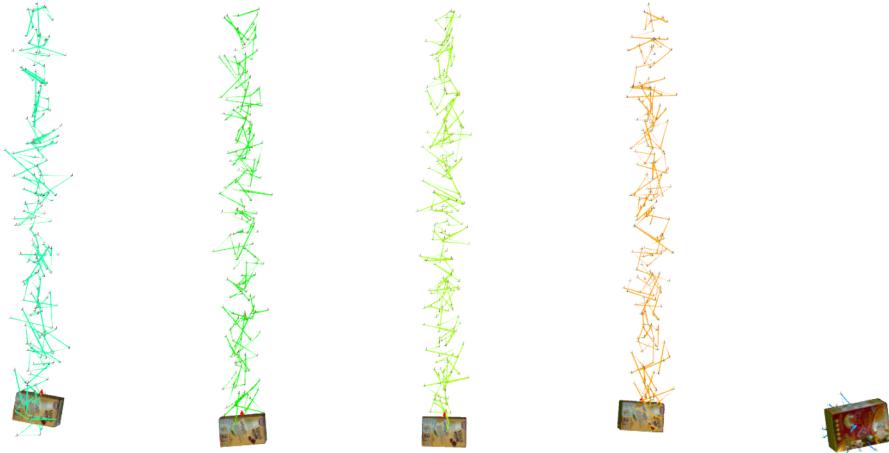


Abbildung 6.11: Lerndaten der Szene für Modellwachstums-Experimente

In Abb. 6.11 gezeigt ist die 5-Objekte-Szene. Die Schachtel rechts bewegt sich mit gewissem Jitter auf der Stelle, die restlichen sich mit zufälligem Zittern nebeneinander nach oben. Im 3-Objekt-Szenario sind nur die rechten drei Schachteln vorhanden, es kommt dann bei jedem neuen Szenario links jeweils eine dazu.

Es sollen die Ergebnisse für verschiedene Gewichte betrachtet werden.

Die verschiedenen Szenarien sind, mit  $w_p$  dem Gewicht der false positives,  $w_n$  dem der false negatives und  $w_t$  dem der durchschnittlichen Erkennungslaufzeit:

- 151:  $w_p = 1, w_t = 5, w_n = 1$  (Schnelligkeit bevorzugt).
- 111:  $w_p = 1, w_t = 1, w_n = 1$  (gleiche Gewichte).
- 515:  $w_p = 5, w_t = 1, w_n = 5$  (Korrektheit bevorzugt).

Als Starttopologien wurde jeweils eine zufällige verwendet, die Wahrscheinlichkeit für einen Random Restart auf 0 gesetzt.

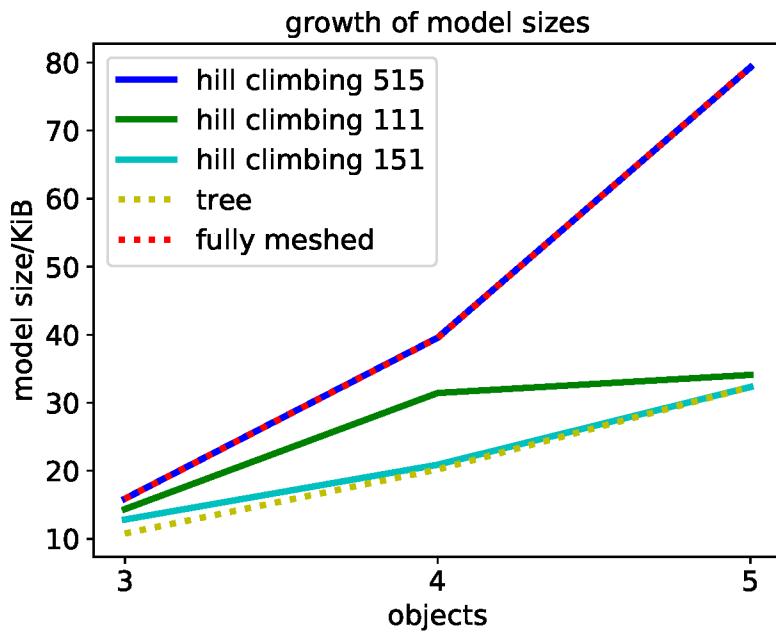


Abbildung 6.12: Modellgrößen in Abhängigkeit von der Objektanzahl

Algorithmus	3 Objekte	4 Objekte	5 Objekte
Heuristik	10, 80	20, 20	32, 32
Hill Climbing 151	12, 84	20, 90	32, 32
Hill Climbing 111	14, 37	31, 44	34, 10
Hill Climbing 515	15, 90	39, 57	79, 30
Vollvermascht	15, 90	39, 53	79, 35

Tabelle 6.3: Vergleich der Modellgrößen in KiB für unterschiedliche Gewichte

Da für mehr Objekte mehr Relationen möglich sind, wächst die Modellgröße mit der Objektanzahl. Wird die Laufzeit stärker gewichtet (151), nähern sich die Modellgrößen der Heuristik an, da diese eine minimale Anzahl Relationen aufweist, welche sich auch auf die Laufzeit auswirkt.

Werden die Gewichte der Fehlerkennungen hoch gesetzt (515), nähert sich das Modell, wie zu erwarten, dem vollvermaschten an, damit auch seine Größe.

Für gleiche Gewichte (111) befinden sich die durchschnittlichen Modellgrößen ungefähr in der Mitte zwischen Minimum (einem Baum) und Maximum (der vollvermaschten Topologie). Bei 3 Objekten kommt das dadurch zustande, dass mal die vollvermaschte, mal eine baumförmige Topologie gewählt wird. Bei 4 Objekten variiert die Anzahl der gewählten Relationen, in den Experimenten zwischen 20, 20 KiB (dem minimalen

Baum) und 39,80 KiB (der vollvermaschten Topologie), doch auch Werte dazwischen sind abgedeckt, die lokale Optima der Topologien repräsentieren.

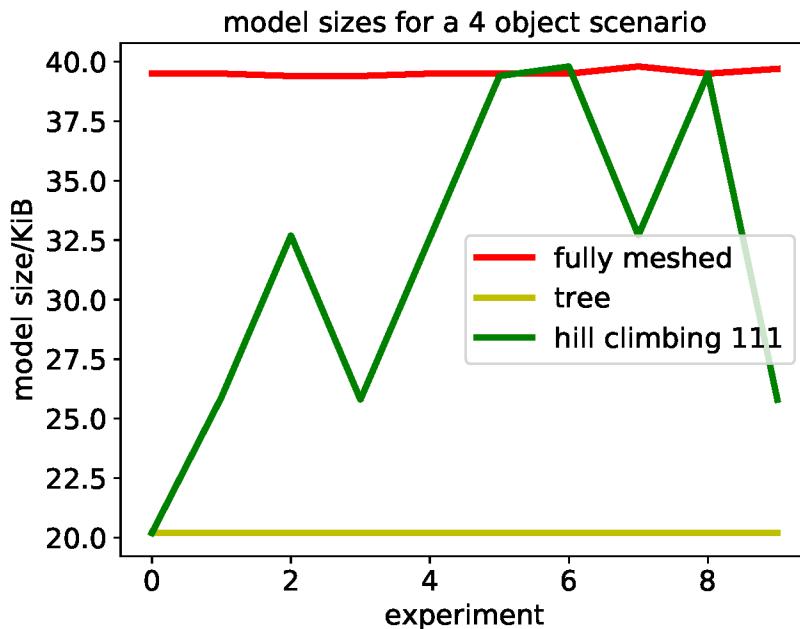


Abbildung 6.13: Modellgrößen aus unterschiedlichen Experimenten

Dies ist in Abb. 6.13 gezeigt: Der Start von zufälligen Topologien aus ergibt für die Modelle, die von *HillClimbing 111* erzeugt wurden, Schwankungen der Modellgröße zwischen denjenigen für die vollvermaschte Topologie (*fully meshed*) und der auf dem von der Heuristik gefundenen Baum (*tree*). Zwischen 4 Objekten sind maximal 6 Relationen möglich; hier ist jede Größe zwischen dem Minimum von 3 und 6 abgedeckt.

Lauf 0	Lauf 1	Lauf 2	Lauf 3	Lauf 4	Lauf 5	Lauf 6	Lauf 7	Lauf 8	Lauf 9
20, 2	25, 9	32, 7	25, 8	32, 6	39, 4	39, 8	32, 7	39, 5	25, 8

Tabelle 6.4: Modellgrößen in KiB bei Hill Climbing 111 der einzelnen Testläufe

Die Modellgröße ist annähernd linear zur Anzahl der Relationen.

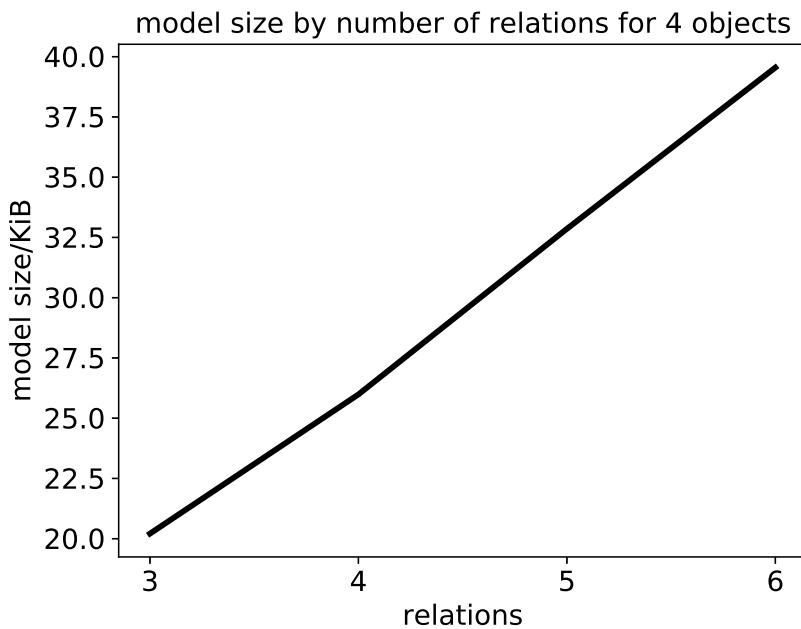


Abbildung 6.14: Relationsanzahl und Modellgrößen. Annähernd linear korreliert.

## 6.4 Mehrere Szenen in einem Modell

Um den Umgang der von der Optimierung gefundenen Relationsgraphen mit Clutter und fehlenden Objekten mit dem durch die Heuristik zu vergleichen, wurde das zu diesem Zweck von *Joachim Gehrung* für das originale PSM entwickelte Experiment noch einmal durchgeführt.

Es wurden zwei Frühstücksszenen in ein Modell gelernt. Szene *A* besteht aus einem tiefen Teller (*PlateDeep*), einem Becher (*Cup*) und zwei Müslischachteln, *VitalisSchoko* und *Smacks*. Szene *B* umfasst neben dem Becher und dem Teller stattdessen eine Kaffekiste (*CoffeeBox*) und einen Messbecher (*MeasuringCup*). Um ein Fehlen der Objekte zu erlauben, was das gegenwärtig implementierte System nicht lernen kann, wurden die Wahrscheinlichkeiten im *Occlusion*-Term nachträglich von Hand angepasst: alle Objekte haben eine Verdeckungswahrscheinlichkeit von 0,3, außer dem als zentral für beide Szenen aufgefassten Teller mit einem Wert von 0,1 (d.h. die komplementäre Wahrscheinlichkeit, dass die Objekte nicht verdeckt sind, beträgt 0,7 bzw. 0,9).

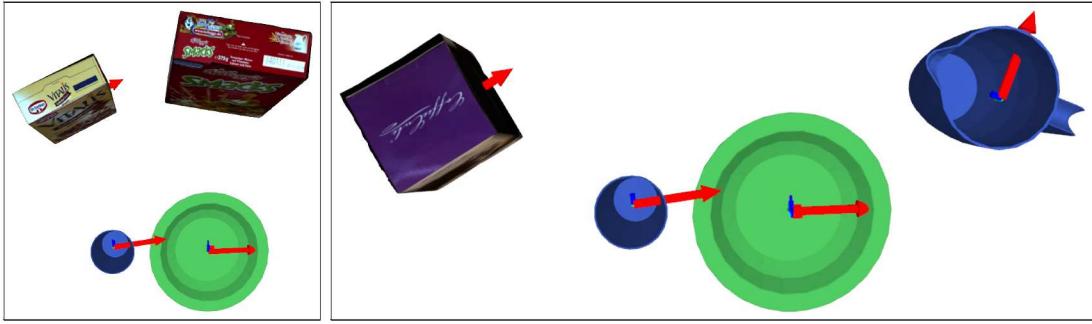


Abbildung 6.15: Frühstücksszenen

Abb. 6.15 zeigt die Lerndaten für die hier benutzte Frühstücksszene: Links Szene A, rechts Szene B. Das Experiment besteht aus dem Übergang von der einen zur anderen. Beide teilen sich den Teller und den Becher. Die roten Pfeile zeigen die Orientierung der Objekte.

Bei der Inferenz werden folgende Beobachtungen betrachtet:

1. Szene A komplett vorhanden, gültige Anordnung.
2. *VitalisSchoko* entfernt.
3. *Smacks* entfernt: Nur noch die gemeinsamen Objekte.
4. Messbecher hinzugefügt: Beginn des Aufbaus von Szene B.
5. Kaffekiste hinzugefügt: Szene B komplett, gültige Anordnung.
6. Nur die einzigartigen Objekte, nicht Becher und Teller.
7. Beide Szenen, zusätzlich dazu als Clutter eine Schale (*Bowl*).

[Geh14, S.85–88]

Gewichte der Optimierung sind  $w_p = 1$ ,  $w_t = 5$ ,  $w_n = 1$ . Es ergeben sich daraus Erkenntnisse über das Verhalten zweier Szenen in einem Modell.

Algorithmus	Hill Climbing Bsp.		Heuristik	
	Szenario	$P(A)$	$P(B)$	$P(A)$
A komplett	0,813522	0,184799	0,766020	0,232020
A reduziert	0,889369	0,101014	0,623863	0,345532
gemeinsame	0,748431	0,150012	0,378193	0,381468
B reduziert	0,782264	0,202809	0,193935	0,788751
B komplett	0,401808	0,597460	0,125569	0,873362
einzigartige	0,650858	0,344572	0,498251	0,498251
Clutter	0,372414	0,627586	0,323640	0,676359

Tabelle 6.5: Vergleich der Szenenwahrscheinlichkeiten für eine Beispielstopologie aus der Optimierung und den Durchschnitt über die Heuristik

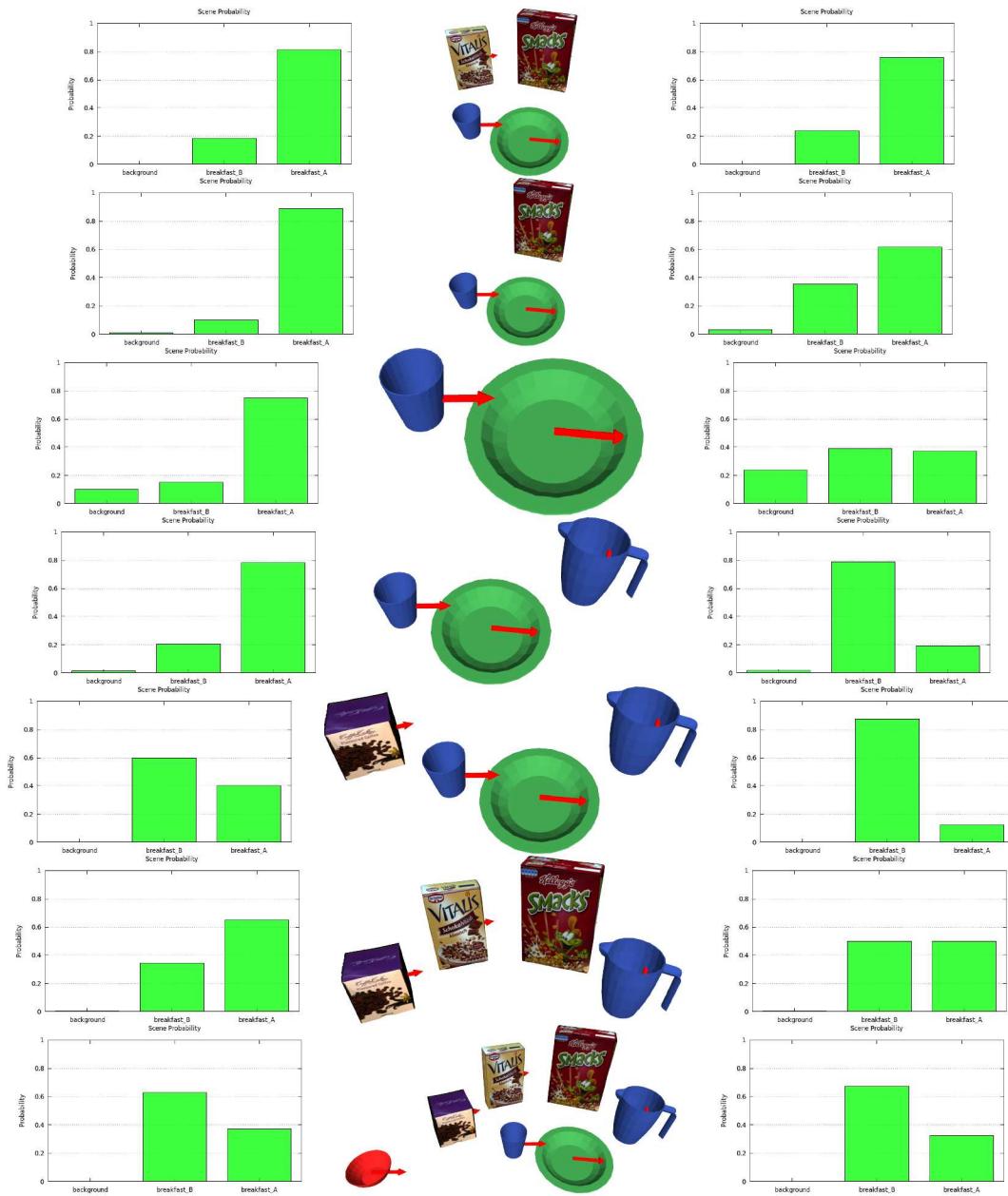


Abbildung 6.16: Vergleich Inferenz Optimierte - Heuristik für 2 Szenen

Abb. 6.16 gibt beispielhaft einen Vergleich der Ergebnisse der Inferenz für das Ergebnis eines Laufs der kombinatorischen Optimierung mit Hill Climbing bei zufälligem Start

(links) mit denen der Heuristik (rechts) für die verschiedenen Evidenzen (Mitte) wieder. In den Balkendiagrammen steht ganz links die Hintergrundwahrscheinlichkeit, mittig die für Szene B und rechts die für A.

Auffällig ist im Vergleich, dass die Wahrscheinlichkeit für Szene A (rechter Balken) bei der optimierten Topologie generell höher ist als die für das heuristisch erstellte Modell. Das liegt daran, dass die Relation zwischen den gemeinsamen Objekten *Cup* und *Plate-Deep* in der optimierten Topologie für Szene A nicht vorkommt.

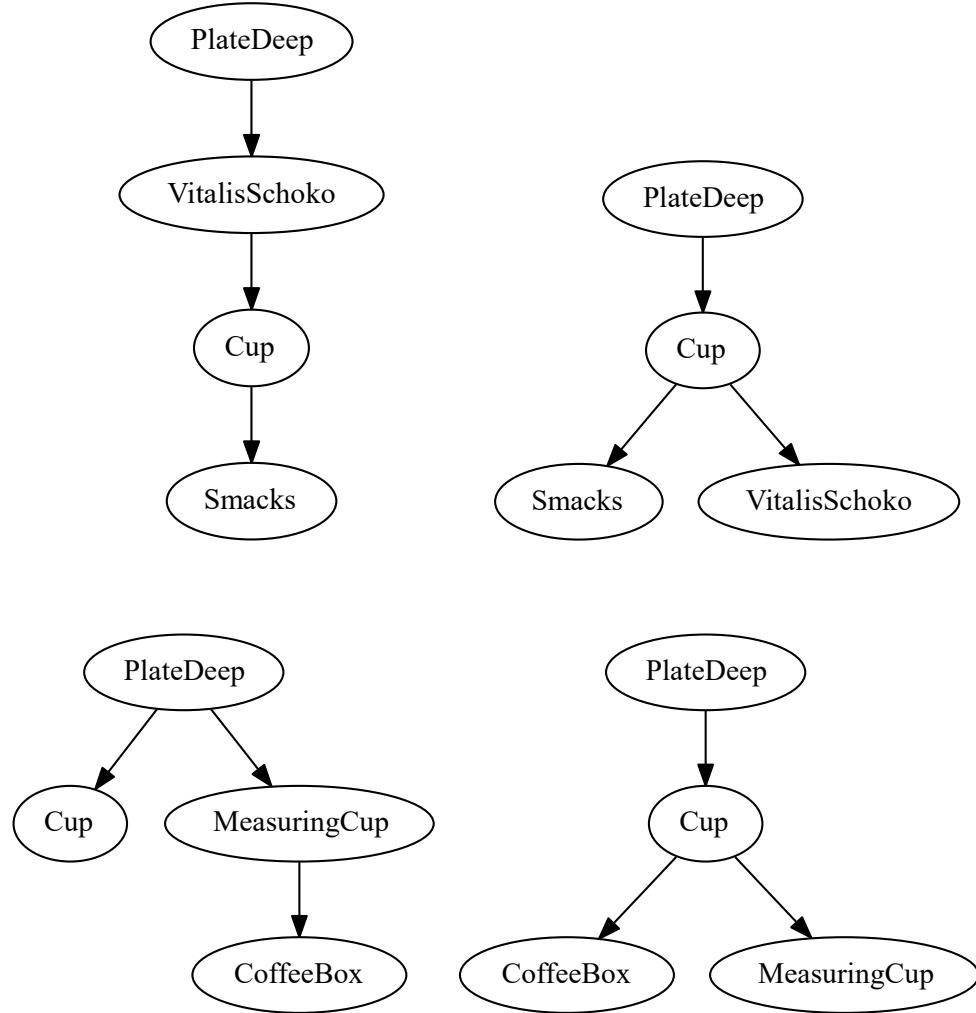


Abbildung 6.17: Graphen der optimierten (l) und heuristisch erstellten Topologien (r)

Abb. 6.17 zeigt oben die Graphen für Szene A, unten die für Szene B, links die optimierten, rechts die heuristisch erstellte. Die Relation *Plate Deep - Cup* kommt in allen vor

außer dem optimierten für Szene *A*.

Am deutlichsten zeigt sich der Effekt der nicht aufgenommenen Relation ab dem Punkt, wo nur noch die beiden beteiligten Objekte vorhanden sind: solange nicht Szene *B* vollständig ist, wird aufgrund der erfüllten Relation *Cup-PlateDeep* Szene *A* höher bewertet. Dieses auf den ersten Blick widersprüchliche Verhalten röhrt daher, dass durch die fehlende Relation der Weg zwischen *Cup* und *PlateDeep* über *VitalisSchoko* abgeschnitten ist und daher über alle Relationen marginalisiert wird, weshalb in diesem Fall die niedrigere *Occlusion*-Wahrscheinlichkeit (welche die Gesamtwahrscheinlichkeit maximiert), hier die von *PlateDeep*, den wesentlichen Anteil an der Gesamtwahrscheinlichkeit hat.

Da, wenn alle Objekte (zuzüglich *Clutter*, der sich allerdings, wie *Joachim Gehrung* gezeigt hat, nur auf den von der Objektanzahl abhängigen Hintergrundterm auswirkt [Geh14, 85-88], was auch für die optimierte Topologie der Fall ist), vorhanden sind, keine Pfade mehr abgeschnitten werden, verhalten sich beide Topologien im letzten Fall identisch.

Im Durchschnitt nähert sich, wie in Tabelle 6.6 zu sehen das Verhalten der optimierten an das der mittels Heuristik erstellten Topologie an. Die Relation zwischen den gemeinsamen Objekten ist also generell als wichtig eingestuft, das Beispiel ist ein lokales Optimum. Es illustriert allerdings, dass es, wenn mehrere Szenen, die Objekte teilen, in ein Modell gelernt werden, zu unintuitivem Verhalten kommen kann, wenn die Relationen zwischen den geteilten Objekten unterschiedlich behandelt werden. Namentlich ist das im Beispiel, dass selbst wenn ein zusätzliches Objekt, das nur in Szene *B* gefunden wurde, zu den gemeinsamen hinzukommt (4.), dennoch Szene *A* als wahrscheinlicher eingestuft wird. Das ist nicht auf die Optimierung beschränkt, sondern kann auch mit der Heuristik vorkommen.

	HillClimbing		Heuristik	
	$P(A)$	$P(B)$	$P(A)$	$P(B)$
1.	0,6216063	0,3767550	0,7660201	0,2320202
2.	0,5118714	0,466336	0,6238632	0,3455323
3.	0,3249858	0,5027912	0,3781934	0,3814683
4.	0,3614389	0,6112566	0,1939350	0,7887508
5.	0,1447096	0,8543131	0,1255689	0,8733621
6.	0,650858	0,344572	0,498251	0,498251
7.	0,3037903	0,6962096	0,3236404	0,6763594

Tabelle 6.6: Vergleich der durchschnittlichen Wahrscheinlichkeiten für Hill Climbing und Heuristik bei fehlenden Objekten und Clutter

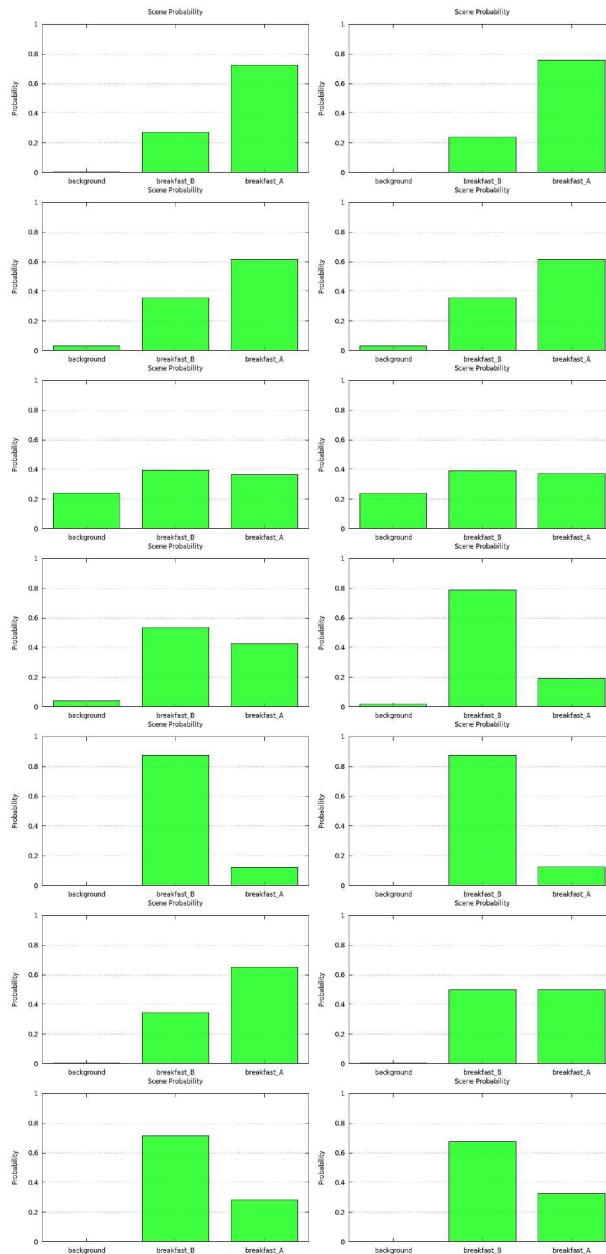


Abbildung 6.18: Vergleich von unterschiedlichen Topologien

Abb. 6.18: links optimierte, rechts heuristisch erstellte Topologie. Die hier beispielhaft gewählten Topologien liefern dem Durchschnitt ähnliche Ergebnisse.

## 6.5 Laufzeit

### 6.5.1 Experimente

Angelehnt an die entsprechenden Tests für das ISM aus der Arbeit von *Jonas Mehlhaus* wird die Trainingslaufzeit mit künstlich erzeugten Szenen getestet. Für eine gewisse Anzahl Objekte wird eine bestimmte Zahl zufälliger Posen als Trajektorien erzeugt. Da die Laufzeit der Inferenz für PSM exponentiell von der Anzahl Objekte abhängt und die Inferenz während der kombinatorischen Optimierung für jede untersuchte Topologie auf jedem Testset durchgeführt wird, steigt auch die Laufzeit des Lerners exponentiell, wie gleich zu sehen sein wird. Deshalb wurden anders als bei den Tests für ISM, wo Szenen mit 5, 10 und 15 Objekten untersucht wurden, hier nur 3, 4 und 5 Objekte verwendet, die für Trajektorien mit je 100, 200, 300 und 400 Schritten benutzt wurden. [Meh16, S.70–76]

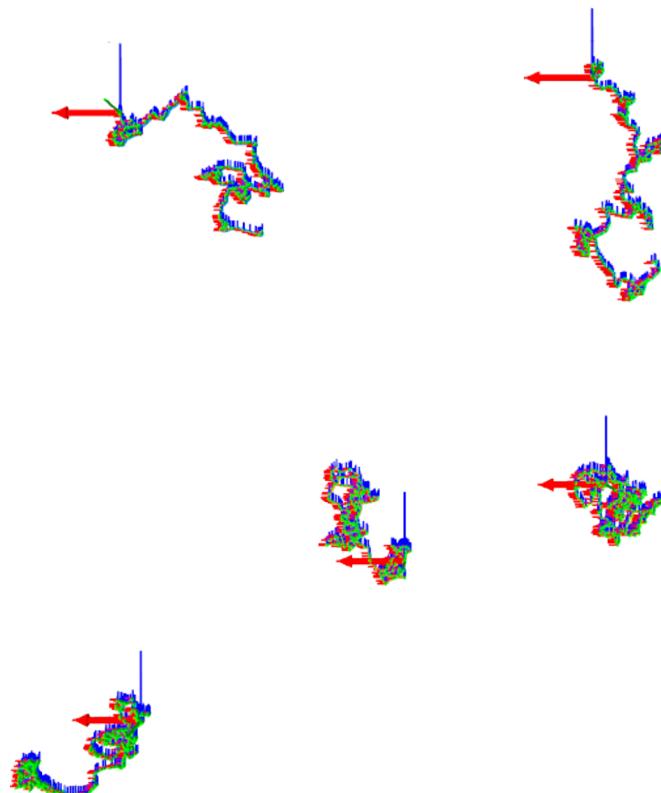


Abbildung 6.19: Lerndaten für Laufzeittests

Gezeigt sind in Abb. 6.19 als Beispiel die zufällig generierten Trajektorien für 5 Objekte mit je 400 Schritten. Sie befinden sich alle in einer Ebene. Durch ihre chaotische Gestalt sind unter Umständen viele Kernel zu ihrer Beschreibung nötig, hier als Maximum 5 eingestellt. Die Objekte selbst sind nicht dargestellt, da es sich um abstrakte Testobjekte

handelt.

Gewichte sind  $w_p = 5, w_t = 1, w_n = 5$ . Die Laufzeit wird also gering gewichtet, um sie unter ungünstigen Bedingungen zu testen.

### 6.5.2 Lernlaufzeit

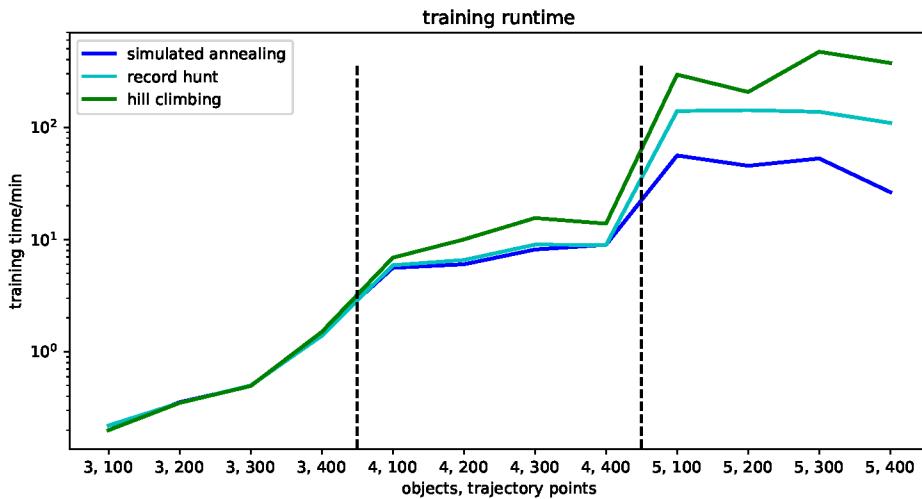


Abbildung 6.20: Trainingslaufzeit der Optimierungsalgorithmen in Minuten nach Objektanzahl, Trajektorienlänge. Logarithmische Zeitachse (y)

Für die Trainingslaufzeit ist in Abb. 6.20 zu erkennen, dass die in der Objektanzahl exponentielle Erkennungslaufzeit sich durch die während der Optimierung durchgeführte Inferenz auf die Laufzeit des Lerners überträgt. Da es sich beim PSM um ein parametrisches Modell handelt, das, einmal gelernt, von der Anzahl der Lerndaten unabhängig ist, ist auch die Erkennungszeit von der Menge der Trajektorienpunkte unabhängig [Geh14, S.93-98].

Hill Climbing benötigt am längsten, da es wiederum jeweils alle Nachbarn besucht. Dadurch, dass das Gewicht für die Laufzeit niedrig gewählt ist, wählt der Algorithmus tendenziell eher langsamere Topologien mit wenigen Fehlerkennungen aus. Record Hunt und für diese Szenen insbesondere Simulated Annealing akzeptieren hingegen auch Verschlechterungen, in diesem Fall also schnellere Topologien mit mehr Fehlerkennungen, wodurch sie insgesamt schneller sind.

### 6.5.3 Erkennungslaufzeit

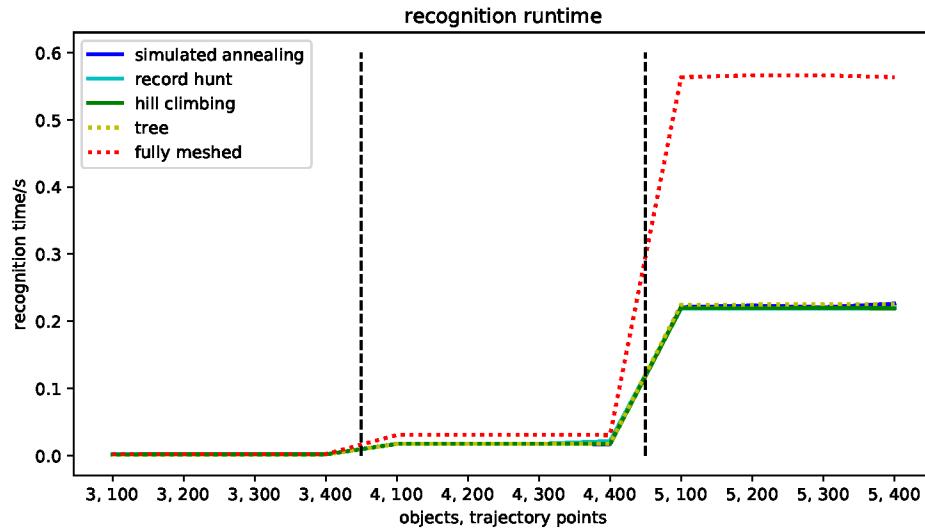


Abbildung 6.21: Erkennungslaufzeit der Optimierungsalgorithmen in Sekunden nach Objektanzahl, Trajektorienlänge

Für die Erkennungslaufzeit lässt sich in Abb. 6.21 ebenfalls deutlich der exponentielle Anstieg erkennen. Die gefundenen Topologien, obwohl die Laufzeit bei der Optimierung weniger stark gewichtet wurde als Fehlerkennungen, ist dennoch mit derjenigen des jeweils von der Heuristik gefundenen Baums vergleichbar und deutlich unter derjenigen der vollvermaschten Topologie, insbesondere mit zunehmender Objektanzahl. Wie erwähnt ist die Erkennungslaufzeit beim PSM von der Anzahl der Lerndaten unabhängig, wie zu sehen ist.

### 6.5.4 Tabellen

Als Anhang Tabellen mit den Daten, die den Graphiken zugrunde liegen.

Szenario	Sim. Ann.	Rec. Hunt	Hill Climb.
3, 100	0,200	0,220	0,200
3, 200	0,350	0,350	0,355
3, 300	0,495	0,500	0,495
3, 400	1,500	1,387	1,500
4, 100	6,890	5,897	5,590
4, 200	10,015	6,567	6,010
4, 300	15,540	9,000	8,140
4, 400	13,845	8,940	8,975
5, 100	294,850	139,550	56,045
5, 200	206,355	142,050	45,375
5, 300	471,350	137,410	52,755
5, 400	373,950	109,405	26,420

Tabelle 6.7: Trainingszeiten der Optimierungsalgorithmen in Minuten nach Objektanzahl, Trajektorienlänge

Szenario	Vollv.	Heur.	Sim. Ann.	Rec. Hunt	Hill Climb.
3, 100	0,002133265	0,001711925	0,00170678	0,00171461	0,00172309
3, 200	0,002136925	0,001727765	0,00172595	0,001715005	0,0017335
3, 300	0,00214782	0,001719405	0,00172631	0,00172263	0,00173023
3, 400	0,002140675	0,001725285	0,001730175	0,00172332	0,001735195
4, 100	0,03055645	0,01730945	0,01730785	0,0172327	0,01742445
4, 200	0,0306066	0,01735395	0,01742265	0,01733515	0,0174265
4, 300	0,03060075	0,01770875	0,01734955	0,0173214	0,01739175
4, 400	0,0305109	0,017651	0,01719695	0,0214937	0,01709595
5, 100	0,563272	0,2247595	0,220025	0,2192595	0,2209405
5, 200	0,5662165	0,22493	0,2199165	0,219193	0,2232105
5, 300	0,5661455	0,22531	0,2198305	0,219057	0,2207225
5, 400	0,5634435	0,2253665	0,21918	0,219408	0,226089

Tabelle 6.8: Erkennungszeiten der Algorithmen in Sekunden

## 7. Ausblick

Dass die Inferenz von PSM für mehr als 6 Objekte nicht mehr echtzeitfähig ist, wirkt sich dadurch, dass während der Optimierung nun Inferenz durchgeführt wird, auch nachteilig auf die Laufzeit des Lerners aus. Eine Verschnellerung in diesem Bereich würde also auch für die Optimierung von Vorteil sein.

Statt eines festen Erkennungs-Schwellwerts wäre auch denkbar, ein Intervall um die Wahrscheinlichkeit festzulegen, die für die vollvermaschte Topologie errechnet wurde, innerhalb dessen eine Wahrscheinlichkeit als gültig bewertet wird. Es würden also die Wahrscheinlichkeiten für unvollständige Topologien mit der korrekten Wahrscheinlichkeit verglichen und dementsprechend bewertet. Es wäre noch näher zu untersuchen, ob dieses Vorgehen einen Vorteil gegenüber dem festen Erkennungsschwellwert bieten könnte.

Um das nichtdeterministische Verhalten für niedrige Lerndatenanzahlen durch die zufälligen synthetischen Samples zu lindern, wäre denkbar, die Samples deterministisch zu erzeugen.

## 8. Zusammenfassung

In dieser Arbeit wurde der Einsatz ausgewählter Algorithmen innerhalb des Probabilistic Scene Models zur Szenenerkennung implementiert, um mithilfe kombinatorischer Optimierung im Sinne von Erkennungslaufzeit und Fehlerkennungen vorteilhafte Mengen räumlicher Objektrelationen auszuwählen.

Dazu wurde das bestehende System so erweitert, dass neben Bäumen auch allgemeine Graphen zum Lernen und zur Erkennung benutzt werden können und dass dementsprechend ein Knoten darin auch mehr als ein Elternteil haben kann, was sich auf die Berechnung der Wahrscheinlichkeiten in der Inferenz, deren Ergebnisse die Ausgabe des PSM darstellen, auswirkt. Gelöst wurde dieses Problem, die bedingte Wahrscheinlichkeit in Abhängigkeit von mehreren Eltern darzustellen, mittels Minimumsbildung über die einzelnen Eltern.

Zudem wurde die Berechnung von Gaussian Mixture Models zur Darstellung einzelner Relationen innerhalb des Lerners mittels des Expectation-Maximization-Algorithmus neu implementiert, um eine proprietäre durch eine *Open Source*-Bibliothek zu ersetzen.

Es wurde empirisch gezeigt, dass trotz gewisser Schwächen bei geringer Lerndatenanzahl gute Ergebnisse in Hinsicht auf Erkennungslaufzeit und -qualität damit erzielbar sind, wie zu erwarten im Austausch gegen eine höhere Trainingslaufzeit.

# Literaturverzeichnis

- [Bil97] Jeff Bilmes: *A Gentle Tutorial of the EM algorithm and its application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models.* (TR-97-021), 1997.
- [Bra15] Kai Braun: *Prädiktion von Objektposen mittels räumlicher Relationen aus hierarchischen Constellation Models zur mobilen Szenenexploration*, Seite 11. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), Karlsruhe, 2015.
- [DW14] Prof. R. Dillmann und Prof. A. Waibel: *Vorlesung Kognitive Systeme*, 2014.
- [ERW<sup>+</sup>16] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong und Ingmar Posner: *Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks.* CoRR, abs/1609.06666, 2016. <http://arxiv.org/abs/1609.06666>.
- [FH00] Pedro Felzenszwalb und Daniel Huttenlocher: *Efficient Matching of Pictorial Structures.* IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2000. <http://cs.brown.edu/~pff/papers/blobrec2.pdf>.
- [FPZ03] Robert Fergus, Pietro Perona und Andrew Zisserman: *Object class recognition by unsupervised scale-invariant learning.* In: *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, Band 2, Seiten II–II. IEEE, 2003.
- [FPZ05] Robert Fergus, Pietro Perona und Andrew Zisserman: *A sparse object category model for efficient learning and exhaustive recognition.* In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, Band 1, Seiten 380–387. IEEE, 2005.
- [GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville: *Deep Learning*, Seiten 330–372. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [Geh14] Joachim Gehrung: *Probabilistische Szenenerkennung durch hierarchische Constellation Models über räumliche Relationen aus demonstrierten Objekttrajektorien.* Masterarbeit, Karlsruher Institut für Technologie (KIT), Karlsruhe, 2014.
- [Goo98] Gerhard Goos: *Vorlesungen über Informatik Band 4: Parallelität und nicht-analytische Lösungsverfahren*, Seite 180. Springer, Jan 1998.

- [Han14] Fabian Hanselmann: *Szenenmodellierung mit automatisch selektierten, räumlichen Relationen aus beobachteten Objektkonfigurationen*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), Karlsruhe, 2014.
- [HJL16] Luis Herranz, Shuqiang Jiang und Xiangyang Li: *Scene recognition with CNNs: objects, scales and dataset bias*. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Seiten 571–579, 2016.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren und Jian Sun: *Deep Residual Learning for Image Recognition*. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [ima] *Large Scale Visual Recognition Challenge 2016 (ILSVRC2016)*. <http://image-net.org/challenges/LSVRC/2016/results>. Gesehen 27.7.2017.
- [Kü13] PD Dr. Stefan Kühnlein: *Vorlesung Lineare Algebra I+II*, 2012–2013.
- [KSH12] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton: *Imagenet classification with deep convolutional neural networks*. In: *Advances in neural information processing systems*, Seiten 1097–1105, 2012.
- [LFU13] Dahua Lin, Sanja Fidler und Raquel Urtasun: *Holistic Scene Understanding for 3D Object Detection with RGBD cameras*. 2013.
- [LLS04] Bastian Leibe, Ales Leonardis und Bernt Schiele: *Combined object categorization and segmentation with an implicit shape model*. In: *Workshop on statistical learning in computer vision, ECCV*, Band 2, Seite 7, 2004.
- [Meh16] Jonas Mehlhaus: *Komparative Analyse ausgewählter Algorithmen zur kombinatorischen Optimierung der räumlichen Relationen in hierarchischen Implicit Shape Models*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), Karlsruhe, 2016.
- [MRJ<sup>+</sup>13] Pascal Meißner, Reno Reckling, Rainer Jäkel, Sven R. Schmidt-Rohr und Rüdiger Dillmann: *Recognizing scenes with hierarchical implicit shape models based on spatial object relations for programming by demonstration*. International Conference on Advanced Robotics, 2013.
- [MRW<sup>+</sup>14] Pascal Meißner, Reno Reckling, Valerij Wittenbeck, Sven R. Schmidt-Rohr und Rüdiger Dillmann: *Active Scene Recognition for Programming by Demonstration using Next-Best-View Estimates from Hierarchical Implicit Shape Models*. IEEE International Conference on Robotics and Automation, 2014.
- [MS08] Kurt Mehlhaus und Peter Sanders: *Algorithms and Data Structures - The Basic Toolbox*, Seiten 250–255. Springer Verlag, Berlin Heidelberg, 2008.
- [Nie15] Michael A. Nielsen: *Neural Networks and Deep Learning*. <http://neuralnetworksanddeeplearning.com/chap6.html>, 2015. Gesehen 08.08.2017.

- [ope] *About - OpenCV library.* <http://opencv.org/about.html>. Gesehen 2.5.2017.
- [pla] *Places2 Challenge 2016.* <http://places2.csail.mit.edu/results2016.html>. Gesehen 27.7.2017.
- [POF12] Sobhan Naderi Parizi, John G Oberlin und Pedro F Felzenszwalb: *Reconfigurable models for scene recognition*. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, Seiten 2775–2782. IEEE, 2012.
- [Pri12a] Simon J.D. Prince: *Computer vision: models, learning and inference*, Seiten 219–223. Cambridge University Press, New York, 2012.
- [Pri12b] Simon J.D. Prince: *Computer vision: models, learning and inference*, Seiten 106–115. Cambridge University Press, New York, 2012.
- [pro] *ProBT.* <http://www.probayes.com/fr/recherche/probt/>. Gesehen 16.08.2017.
- [QT09] Ariadna Quattoni und Antonio Torralba: *Recognizing indoor scenes*. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, Seiten 413–420. IEEE, 2009.
- [RDS<sup>+</sup>15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg und Li Fei-Fei: *ImageNet Large Scale Visual Recognition Challenge*. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015.
- [Sch78] Gideon Schwarz: *Estimating the Dimension of a Model*. Ann. Statist., 6(2):461–464, März 1978. <http://dx.doi.org/10.1214/aos/1176344136>.
- [SLJ<sup>+</sup>14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke und Andrew Rabinovich: *Going Deeper with Convolutions*. CoRR, abs/1409.4842, 2014. <http://arxiv.org/abs/1409.4842>.
- [SZ14] Karen Simonyan und Andrew Zisserman: *Very deep convolutional networks for large-scale image recognition*. arXiv preprint arXiv:1409.1556, 2014.
- [WHR12] Ernst Wit, Edwin van den Heuvel und Jan Willem Romeijn: ‘All models are wrong...’: an introduction to model uncertainty. Statistica Neerlandica, 66(3):217–236, 2012, ISSN 1467-9574. <http://dx.doi.org/10.1111/j.1467-9574.2012.00530.x>.
- [ZLK<sup>+</sup>17] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva und Antonio Torralba: *Places: A 10 million Image Database for Scene Recognition*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017.