

# sorting the colors

## Dimensionsbezogene Generalisierung vergleichsbasierter Sortierung

Ein Projekt von leo blume, 16 J.

Plakat 1/3: Einleitung, Definitionen und mathematischer Beweis

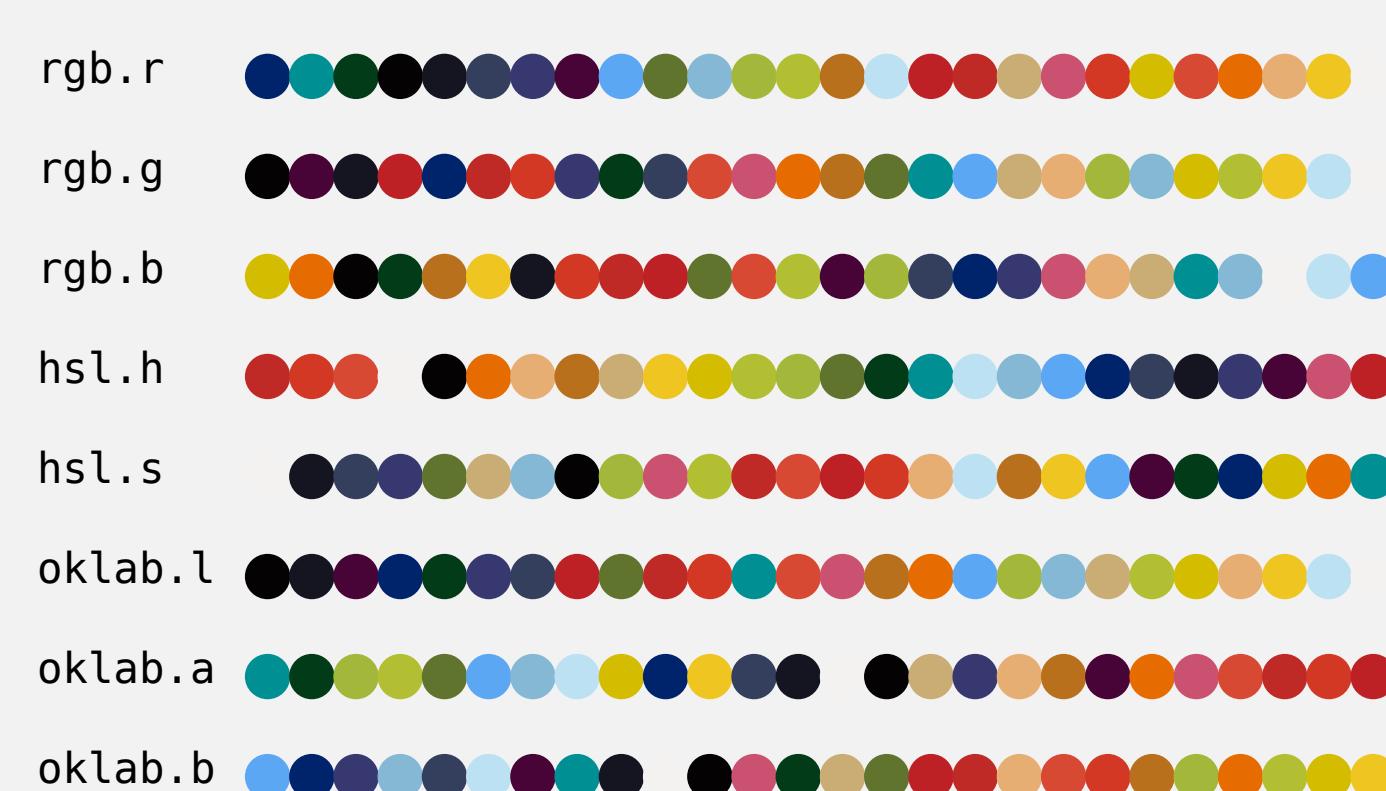


GYMNASIUM  
EssenWerden

MINTec  
Schule

### Einleitung

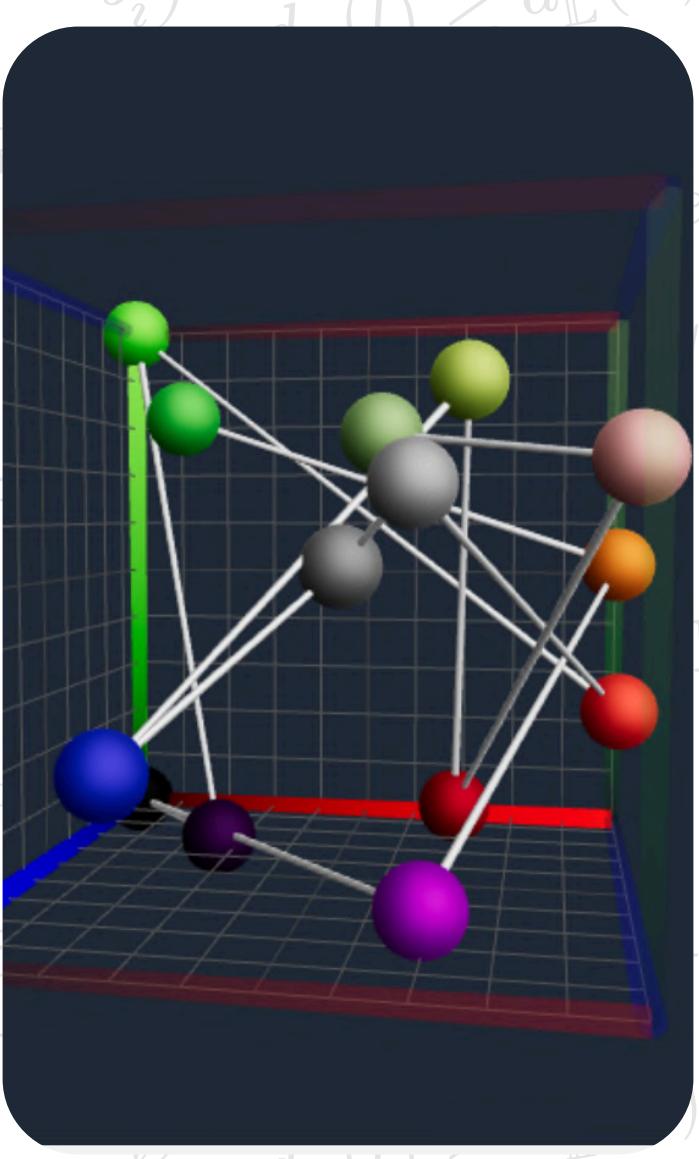
Ein Bücherregal nach **Farben** sortieren. Recht einfach? Von wegen! Ich habe in meiner Arbeit herausgefunden, dass die Farbsortierung nur ein Anwendungsfall einer neu entwickelten **mehrdimensionalen Sortierung** ist, die auch auf zahlreiche andere Objekte des Alltags angewendet werden kann.



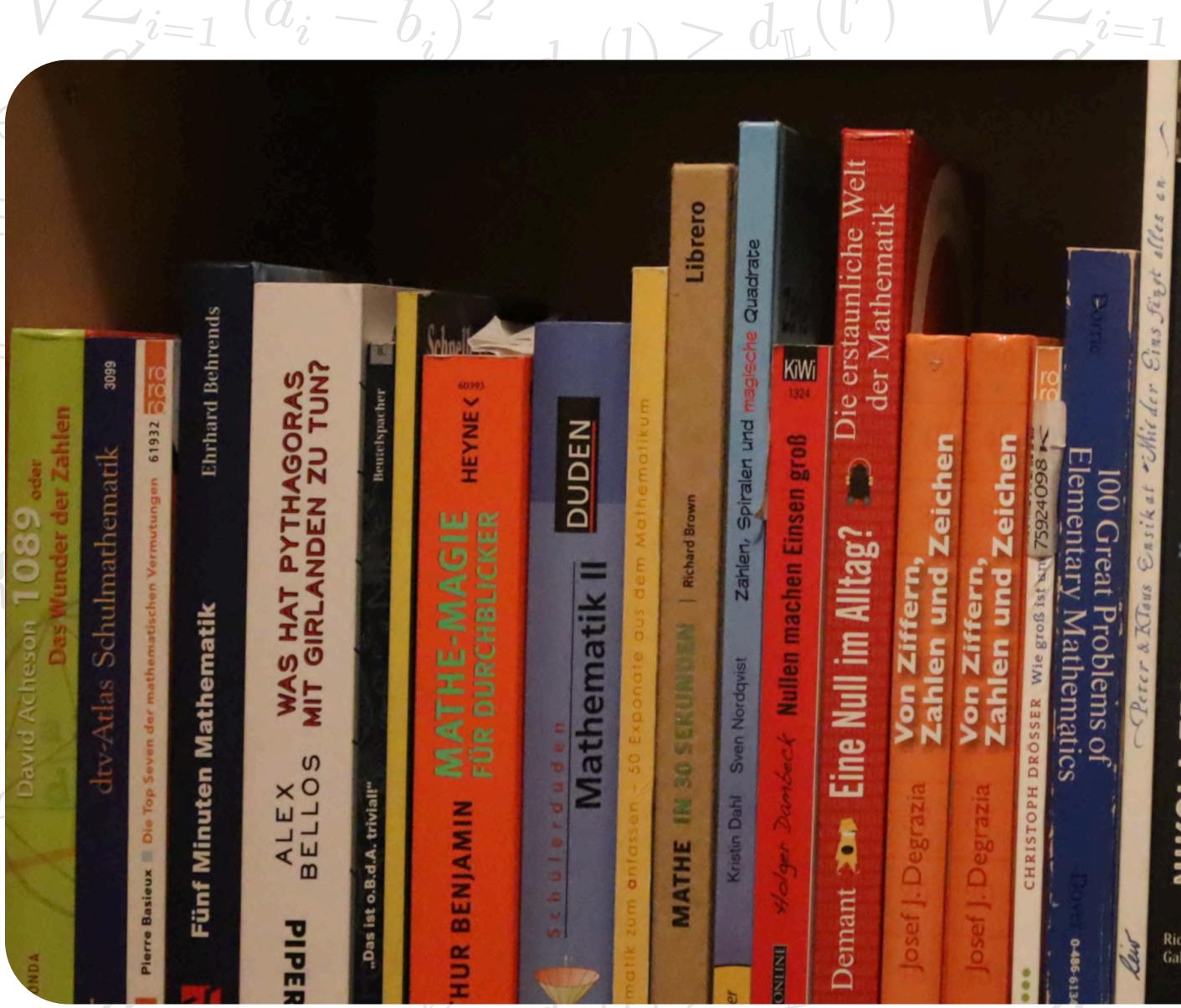
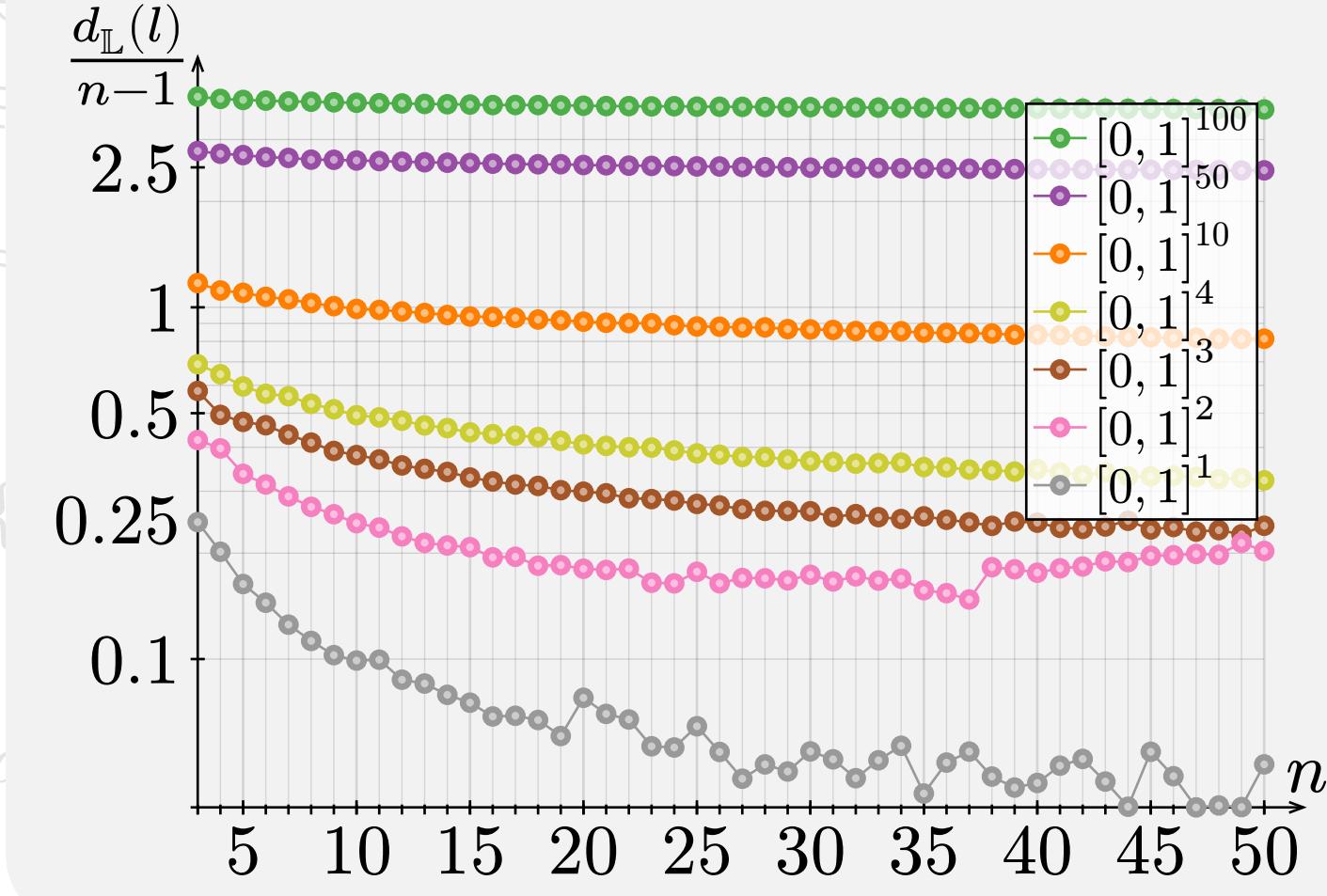
Es ist nicht möglich, die Farben nach einem einzigen numerischen Wert zu sortieren: links sind einige Versuche, so etwa Helligkeit (`oklab.l`) und Buntwert (`hsl.h`). Ästhetisch ist das jedoch kaum, da die anderen Dimensionen unberücksichtigt bleiben. Weil Menschen Trichromaten sind, brauchen Farben in jeder Repräsentation **3 Dimensionen**.

### RGB-Distanz

$\Delta x := x_a - x_b$	$d_{\text{rgb}}(a, b) := \sqrt{\Delta x^2}$
$d(\text{green}, \text{orange}) \approx 0.88$	
$d(\text{orange}, \text{purple}) \approx 0.88$	
$d(\text{purple}, \text{black}) \approx 1.05$	
$d(\text{black}, \text{purple}) \approx 0.40$	
$d(\text{purple}, \text{grey}) \approx 0.69$	
$d(\text{grey}, \text{green}) \approx 0.94$	
$d(\text{green}, \text{red}) \approx 1.26$	
$d(\text{red}, \text{grey}) \approx 0.70$	
$d(\text{grey}, \text{grey}) \approx 0.23$	
$d(\text{grey}, \text{blue}) \approx 0.51$	
$d(\text{blue}, \text{green}) \approx 0.93$	
$d(\text{green}, \text{red}) \approx 0.67$	
$d(\text{red}, \text{red}) \approx 0.39$	
$d(\text{red}, \text{brown}) \approx 0.29$	
$d(\text{brown}, \text{brown}) \approx 0.35$	
$d(\text{brown}, \text{pink}) \approx 0.36$	
$d(\text{pink}, \text{green}) \approx 0.47$	
$d(\text{pink}, \text{green}) \approx 0.47$	



### $\emptyset d_{\mathbb{L}}(l)$ kettensortierter Listen



### Definitionen 2

**Vertauschen**  $\text{Swap}(l, m, n)$

$$\text{Swap}(l, m, n)_i := \begin{cases} l_n & \text{falls } i = m \\ l_m & \text{falls } i = n \\ l_i & \text{sonst} \end{cases}$$

**Permutationen**  $\text{Perm}(l) :=$

$$\{l' \mid l'_i = l_{\sigma(i)}, \sigma : \mathbb{D} \rightarrow \mathbb{D} \text{ bijektiv}\}$$

**sortiert**

für strikte Totalordnung  $(\Sigma, >)$   
mit Ordnungsrelation  $>$

$$\forall i \in \mathbb{D} \setminus \{n\} : l_{i+1} > l_i$$

**Kettenlänge**  $d_{\mathbb{L}}(l)$

für metrischen Raum  $(\Sigma, d)$  mit  
Distanzfunktion  $d$

$$d_{\mathbb{L}}(l) := \sum_{i=1}^{n-1} d(l_i, l_{i+1})$$

**kettensortiert**

für metrischen Raum (s.o.)

$$d_{\mathbb{L}}(l) = \min_{l' \in \text{Perm}(l)} d_{\mathbb{L}}(l')$$

### Satz 1

Jede **sortierte** Liste  
reeller Zahlen ist  
unter der  
Betragssmetrik auch  
**kettensortiert**.

$$\Sigma \subseteq \mathbb{R} \wedge d(a, b) = |a - b| \wedge l \in \mathbb{L}_{\text{sort}} \Rightarrow d_{\mathbb{L}}(l) = \min_{l' \in \text{Perm}(l)} d_{\mathbb{L}}(l')$$

### Definitionen 1

**Liste**  $l$

injektive Abbildung  $\mathbb{D} \rightarrow \Sigma$ ,  
Familie  $(l_i)_{i \in \mathbb{D}}$

**Länge**  $n$

Anzahl der Elemente der Liste

**Definitionsbereich**  $\mathbb{D}$

$$[1, n] \cap \mathbb{N}, |\mathbb{D}| \in \mathbb{N}$$

**Eingabealphabet**  $\Sigma$

Typ der Listenelemente

**Index**  $i \in \mathbb{D}$

Ordinalzahl; Stelle in der Liste

**Element**  $e \in \Sigma$

In der Liste (möglicherweise)  
enthaltener Eintrag

**Teilabschnitt**  $l_{p:q}$

für  $p, q \in \mathbb{D}, p \leq q$  Liste

$$[l_p, l_{p+1}, \dots, l_{q-1}, l_q] \text{ mit } n = q - p + 1$$

**Bildmenge**  $Y$

Menge der Elemente in  $l$  mit  
 $|Y| = n$

**Konkatenation**  $l^1 \oplus l^2$

$$[l_1^1, l_1^2, \dots, l_{n_1-1}^1, l_{n_1}^1, l_1^2, \dots, l_{n_2}^2]$$

### Beweis: Jede sortierte Liste ist kettensortiert

- Liste (mit  $\Sigma \subseteq \mathbb{R}$ ) kann neben Abbildung auch induktiv definiert werden
- $\mathbb{L}$ : Menge aller solcher Listen,  $\mathbb{L}_{\text{sort}}$ : Menge sortierter Listen  $\varepsilon$ : leere Liste mit  $n = 0$ ,  $\varepsilon \oplus$  die strukturelle Konkatenation, Einfügemenge  $\text{Ins}(l, e) := \{e \mid n = |l|, i \in [1, n+1], l^e = l_{1:i-1} \oplus e \oplus l_{i:n}\}$
- Definition  $\mathbb{L}$ :  $\varepsilon \in \mathbb{L} \wedge l \in \mathbb{L} \wedge n = |l| \wedge e \in \Sigma \wedge \forall i \in [1, n] : e > l_i \Leftrightarrow \forall l^e \in \text{Ins}(l, e) : l^e \in \mathbb{L}$
- Definition  $\mathbb{L}_{\text{sort}}$ :  $\varepsilon \in \mathbb{L}_{\text{sort}} \wedge l \in \mathbb{L}_{\text{sort}} \wedge n = |l| \wedge e \in \Sigma \wedge \forall i \in [1, n] : e > l_i \Leftrightarrow l \oplus e \in \mathbb{L}_{\text{sort}}$
- $\mathbb{L}_{\text{sort}} \subset \mathbb{L}$ , da Konstruktion von  $\mathbb{L}_{\text{sort}}$  Spezialfall von  $\mathbb{L}$  handelt, bei der  $\text{Ins}(l, e) := \{l \oplus e\}$ .
- **Nebensatz**. Jede Liste mit total geordneter Zielmenge ist Element von  $\mathbb{L}$ .
- **Beweis**.
  - Man betrachte abbildende Liste  $l$
  - Iterative Konstruktion der  $\mathbb{L}$ -Liste  $w$ , beginnend mit  $\varepsilon$ 
    - Betrachtung des kleinsten nicht betrachteten Element  $e$  und Index  $i$
    - Falls in  $l$  ein  $j < i$  existiert, sodass  $l_j < l_i$ , so ist  $l_j$  bereits in  $w$  und  $e$  wird am darauffolgenden Index eingesetzt, ansonsten am Index 0.
      - Da  $l_j < l_i$  gilt, ist  $l_j < e$  (sofern  $w$  noch nicht  $i$  enthält) und  $e$  an einer Stelle eingefügt ( $\text{Ins}$ ) wird, ist Bedingung (II) erfüllt und  $w$  eine Liste
    - Eindeutige Zuordnung: alle Elemente von  $l$  sind in  $w$  enthalten, Reihenfolge wird eingehalten
      - Somit teilen  $w$  und  $l$  alle Eigenschaften und sind somit identisch
    - Konstruierbarkeit aus der Definition von  $\mathbb{L}$  als Eigenschaft der Listen festzuhalten
  - **Satz**. Jede sortierte Liste reeller Zahlen ist unter der Betragssmetrik kettensortiert.
  - **Beweis**.
    - Durch Definition:  $\forall l \in \mathbb{L}_{\text{sort}} : d_{\mathbb{L}}(l) = \min_{l' \in \text{Perm}(l)} d_{\mathbb{L}}(l')$
    - Minimum: Element, für das kein kleineres existiert
      - gleichwertige Formulierung: für keine sortierte Liste existiert eine Permutation mit kleinerer Kettenlänge
      - Betragssmetrik:  $d(a, b) = |a - b|$
  - **Induktionsbeginn**
    - Für Listen der Länge 0 und 1 existiert keine Permutationsfunktion  $\sigma$ , welche die Reihenfolge ändert  $\Rightarrow$  jede Liste sortiert und kettensortiert
    - Liste der Länge 2:  $l = l_1 \oplus l_2$ , mit  $l_2 > l_1$ 
      - keine Permutation ändert Kettenlänge, somit ebenfalls jede Liste sortiert und kettensortiert
  - **Induktionsschritt**
    - $\varepsilon$  sei sortierte und kettensortierte Liste über der Länge  $n \in \mathbb{N}, n \geq 2$
    - Induktive Definition: neues Element  $e \in \Sigma, e > l_n$  an einer beliebigen Position  $i$  in die Liste, fortan  $l^e$  genannt, eingefügt
      - Liste nur dann sortiert, wenn  $i = n+1$ , sonst wäre  $l_{i+1} < l_i$  und damit die Liste unsortiert
      - Wird gezeigt: beim Anfügen eines neuen Elements an genau dieser Stelle bleibt Liste kettensortiert

# sorting the colors

## Dimensionsbezogene Generalisierung vergleichsbasierter Sortierung

### Probleme

Bekannt:

**HAMP**: Gib den kürzesten Hamilton-Pfad eines vollständigen ungerichteten gewichteten Graphen aus.

**HAMP-ADM**: Entscheide, ob ein ungewichteter ungerichteter Graph einen Hamilton-Pfad zulässt.

Neu:

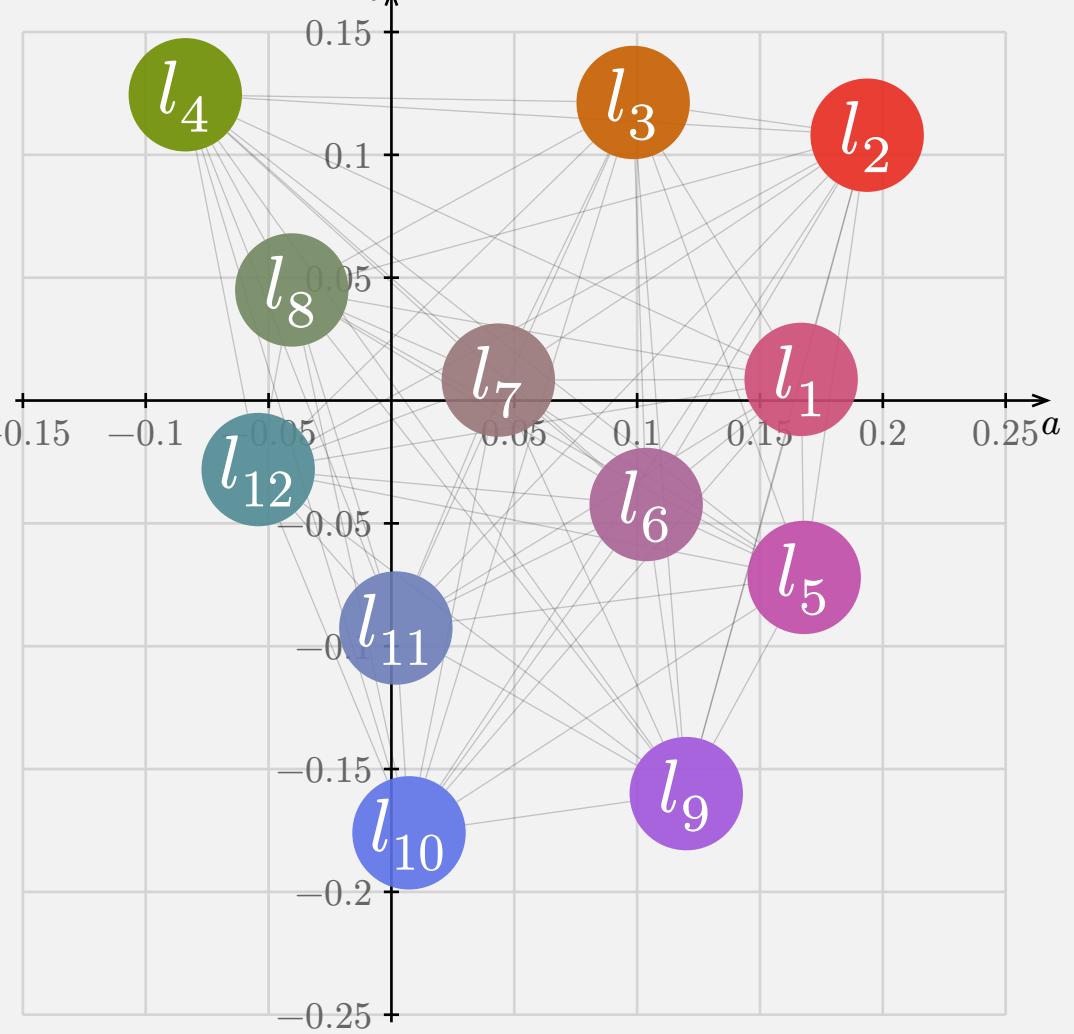
**CSORT**: Gib eine Permutation  $l'$  einer beliebigen Liste  $l$  über einen metrischen Raum  $(\Sigma, d)$  mit minimaler Kettenlänge  $d_L(l')$  zurück.

**CSORT-DEC**: Entscheide, ob für eine Liste  $l$  und eine gegebene Kettenlänge  $c$  eine Permutation  $l'$  mit  $d_L(l') < c$  existiert.

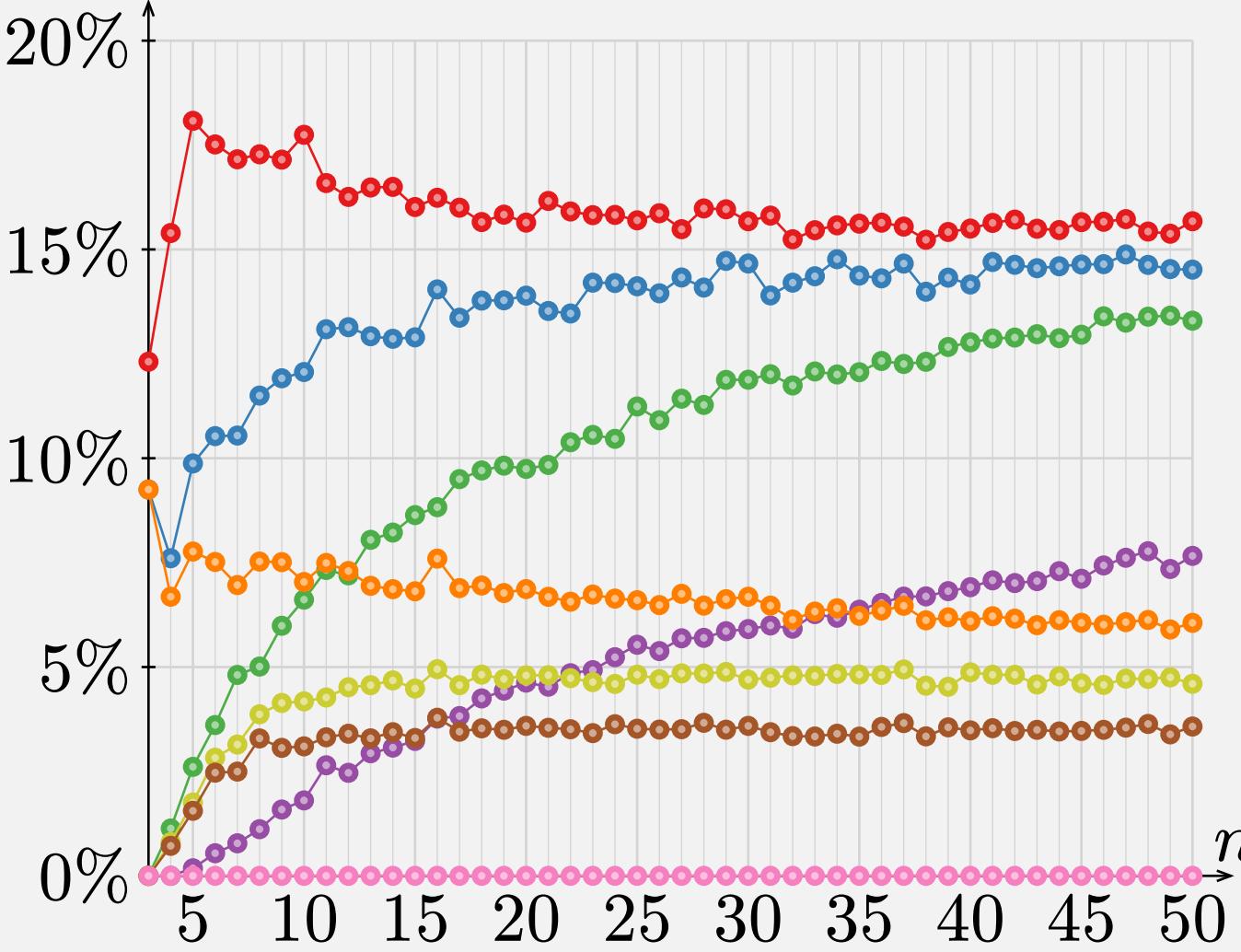
### Farbselektion

Telemagenta	Knallrot	Schokolade	Olivgrün
Orchidee	Signalviolett	Cupcake	Rosmaringrün
Violett	Königsblau	Dragonerblau	Türkisblau

### Koordinatensystem



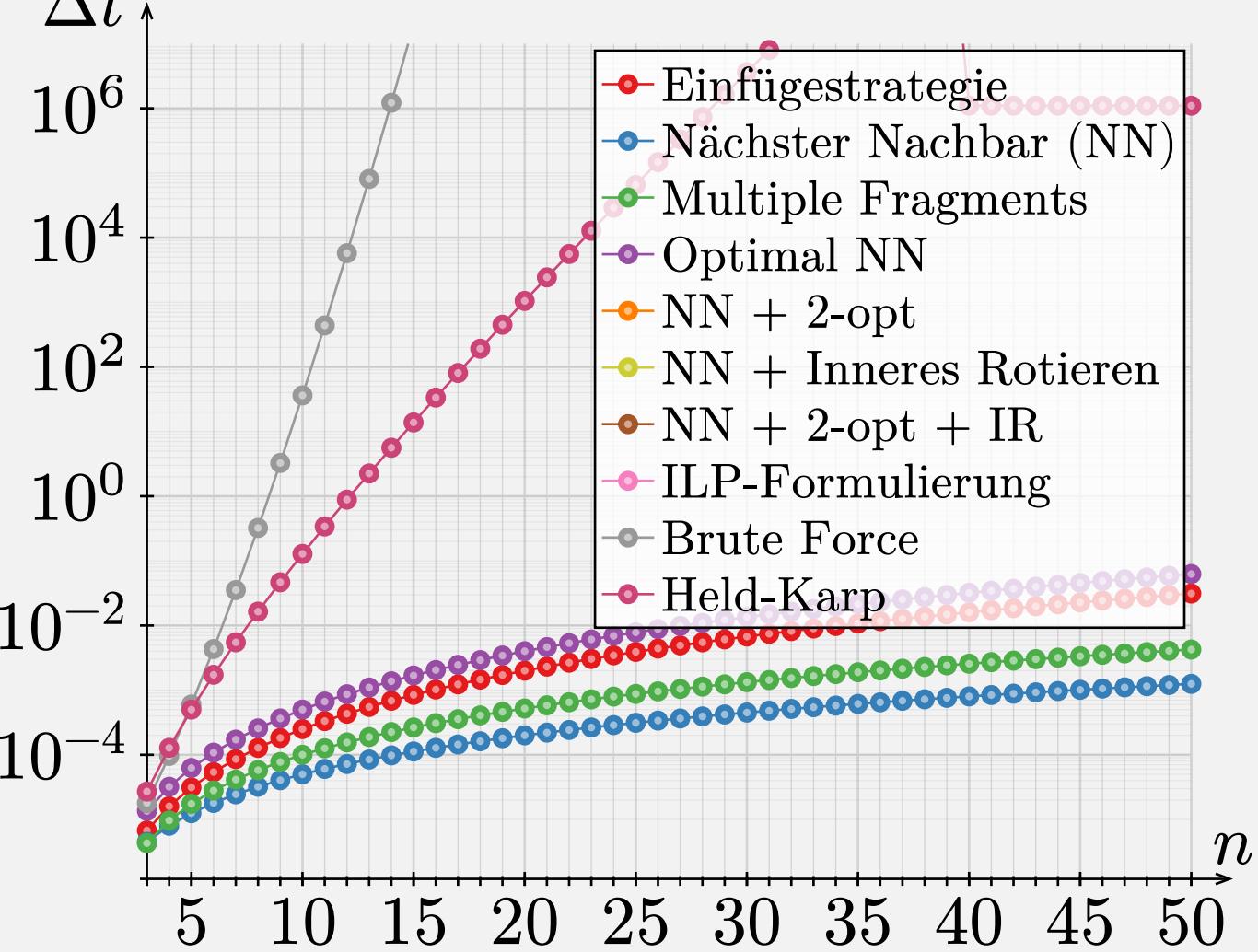
### $\emptyset$ Ungenauigkeit $\frac{d(l')}{d(l_{opt})} - 1$



### Beweis: CSORT $\geq_p$ SAT (NP-Härte durch Reduktion)

- Gegeben: HAMP-ADM  $\geq_p$  SAT
- Reduktion von HAMP-ADM auf HAMP: Man betrachte einen beliebigen (insbesondere unvollständigen) Graphen  $G = (V, E)$ . Aus diesem konstruiere man nun den neuen vollständigen gewichteten Graphen  $G' = (V, V^2)$  mit Kantenfunktion  $d(a, b) = \begin{cases} 0 & \text{falls } (a, b) \in E \\ 1 & \text{falls } (a, b) \notin E \end{cases}$ . Aus der Anwendung von  $G'$  in einen HAMP-Löser kann HAMP-ADM ermittelt werden: Man nenne den ermittelten optimalen Pfad  $p_{opt}$ . Ist  $d_{\mathbb{L}}(p_{opt}) = 0$ , so lässt  $G$  einen Hamilton-Pfad zu, da nur Kanten aus  $E$  gewählt wurden; sonst nicht. Somit  $\text{HAMP} \geq_p \text{HAMP-ADM}$ .
- HAMP ist bezüglich Translation der Kantengewichte invariant: wenn jedes Kantengewicht um ein festes  $\Delta$  verändert wird, so werden alle Pfadgewichte gleichermaßen verändert und die Eigenschaft der Optimalität eines Pfads bleibt erhalten.
- Reduktion von HAMP auf CSORT: Man betrachte die Adjazenzmatrix eines beliebigen vollständigen Graphen. Man subtrahiere von jedem Eintrag nun den Wert des niedrigsten und addiere den des nun höchsten Eintrags, sodass die Dreiecksungleichung  $\forall a, b, c : d(a, b) + d(b, c) \geq d(a, c)$  erfüllt ist. Um  $n$ -dimensionale Vektoren zu ermitteln, die diese Distanzen haben, kann ein (durch Wegfall quadratischer Terme lineares) Gleichungssystem aufgestellt werden, um die Komponenten zu berechnen. Es zeigt sich, dass dieses Gleichungssystem stets gelöst werden kann. Somit kann jeder CSORT-Algorithmus HAMP lösen und CSORT ist damit NP-schwer.
- 

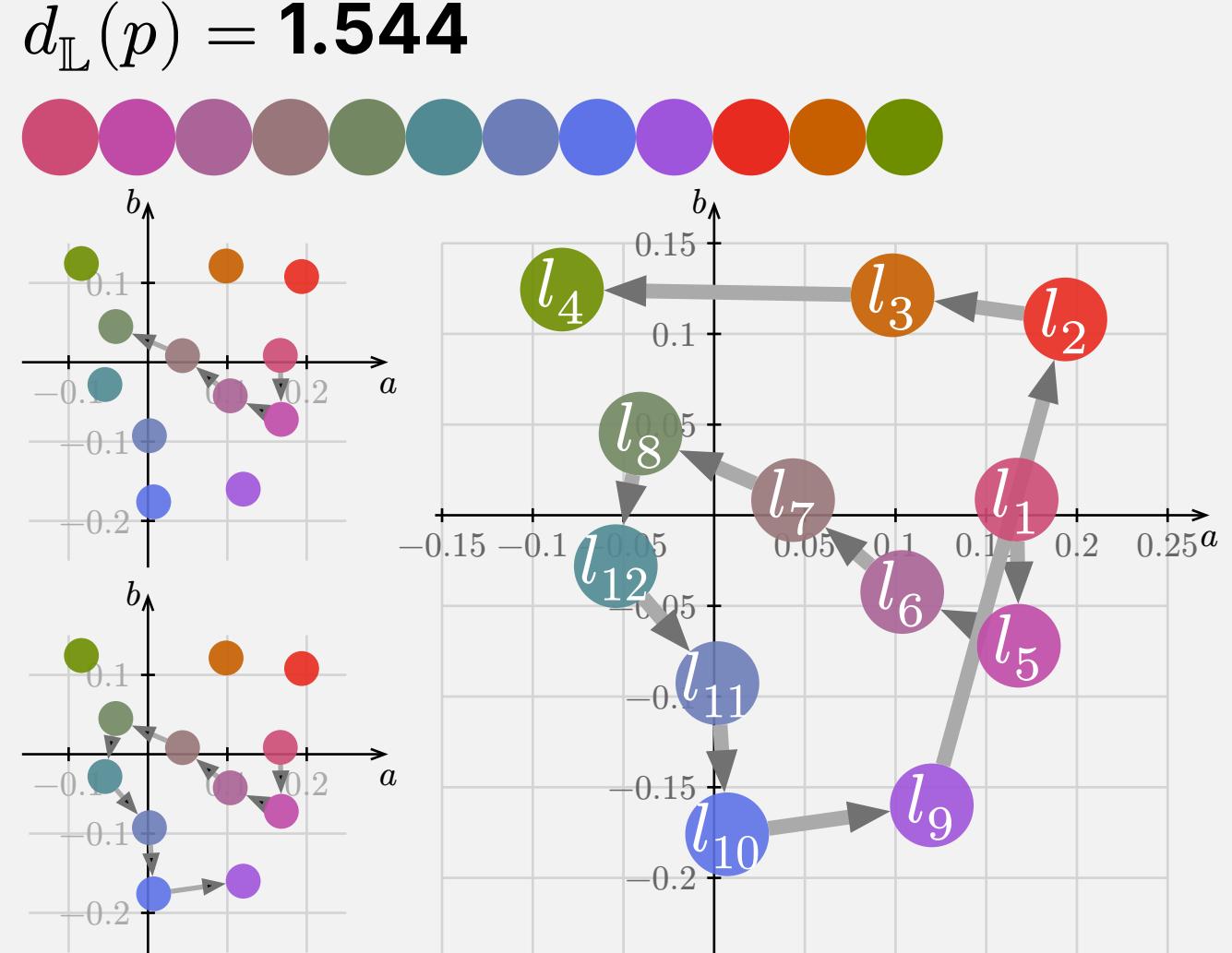
### Komplexität



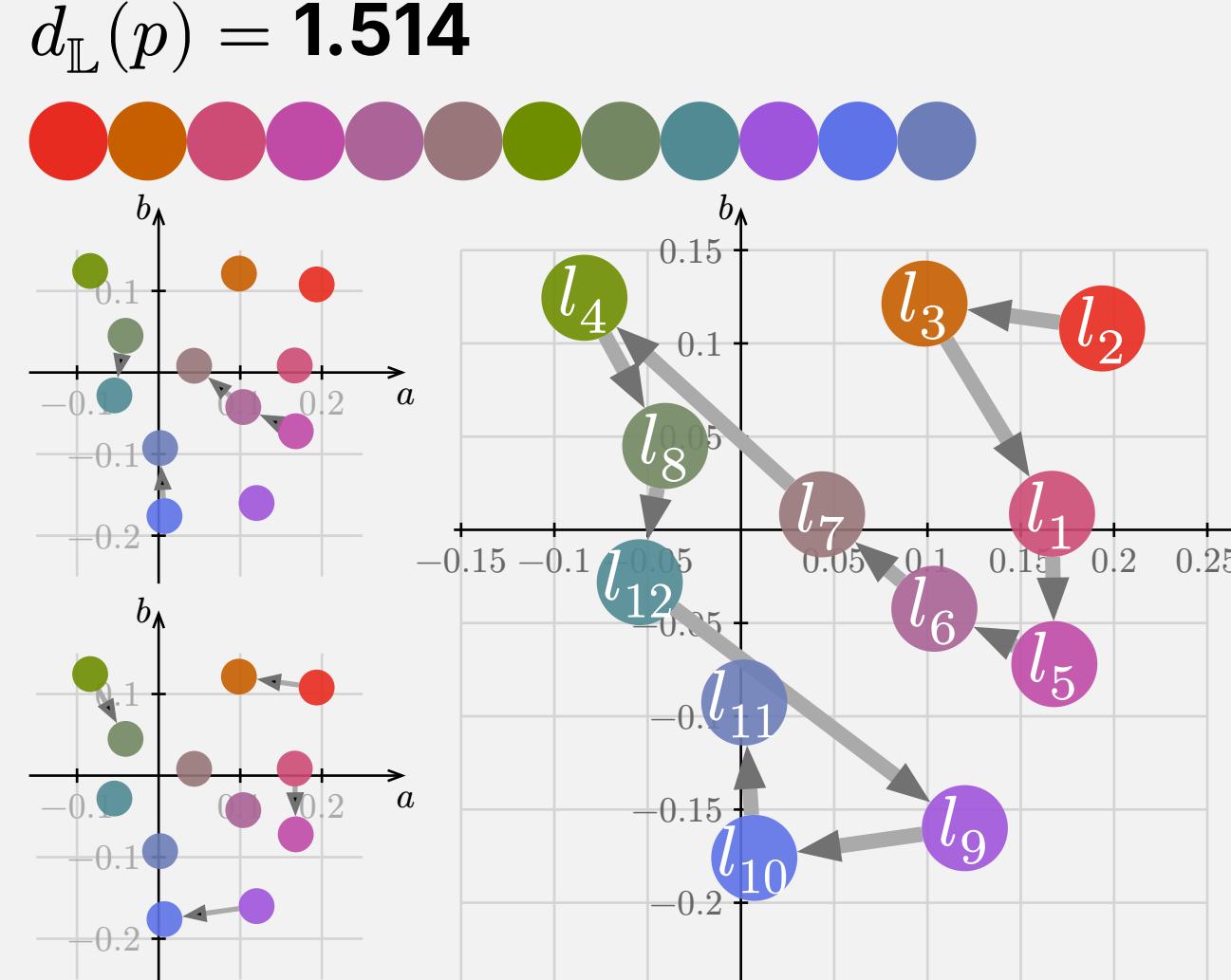
### Adjazenzmatrix

1	2	3	4	5	6	7	8	9	10	11	12
- .103 .132 .276 .081 .081 .123 .210 .175 .244 .194 .224	.103 - .096 .278 .182 .175 .180 .243 .278 .340 .278 .283	.132 .096 - .182 .205 .164 .126 .159 .282 .311 .235 .214	.276 .278 .182 - .319 .251 .172 .090 .350 .314 .233 .155	.081 .182 .205 .319 - .071 .148 .239 .100 .192 .167 .227	.081 .175 .164 .251 .071 - .079 .169 .119 .165 .114 .159	.123 .180 .166 .127 .148 .079 - .092 .185 .188 .109 .104	.210 .243 .159 .090 .239 .169 .092 - .260 .226 .144 .074	.175 .278 .282 .350 .100 .19 .185 .260 - .114 .136 .219	.244 .340 .311 .314 .192 .165 .188 .226 .114 - .084 .160	.194 .278 .235 .233 .167 .114 .109 .144 .136 .084 - .085	.224 .283 .214 .155 .227 .159 .104 .074 .219 .160 .085 -

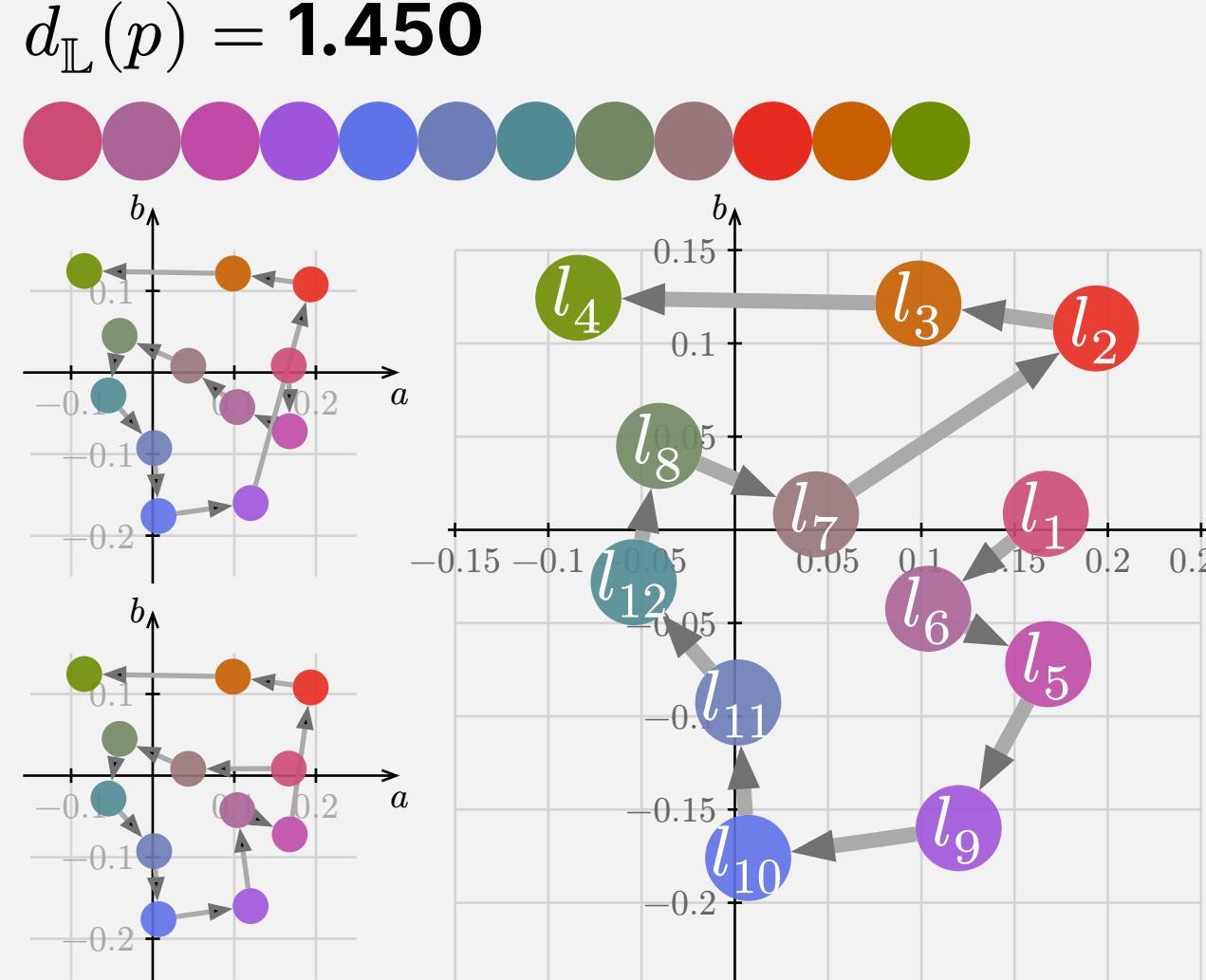
### Nearest Neighbor (NN)



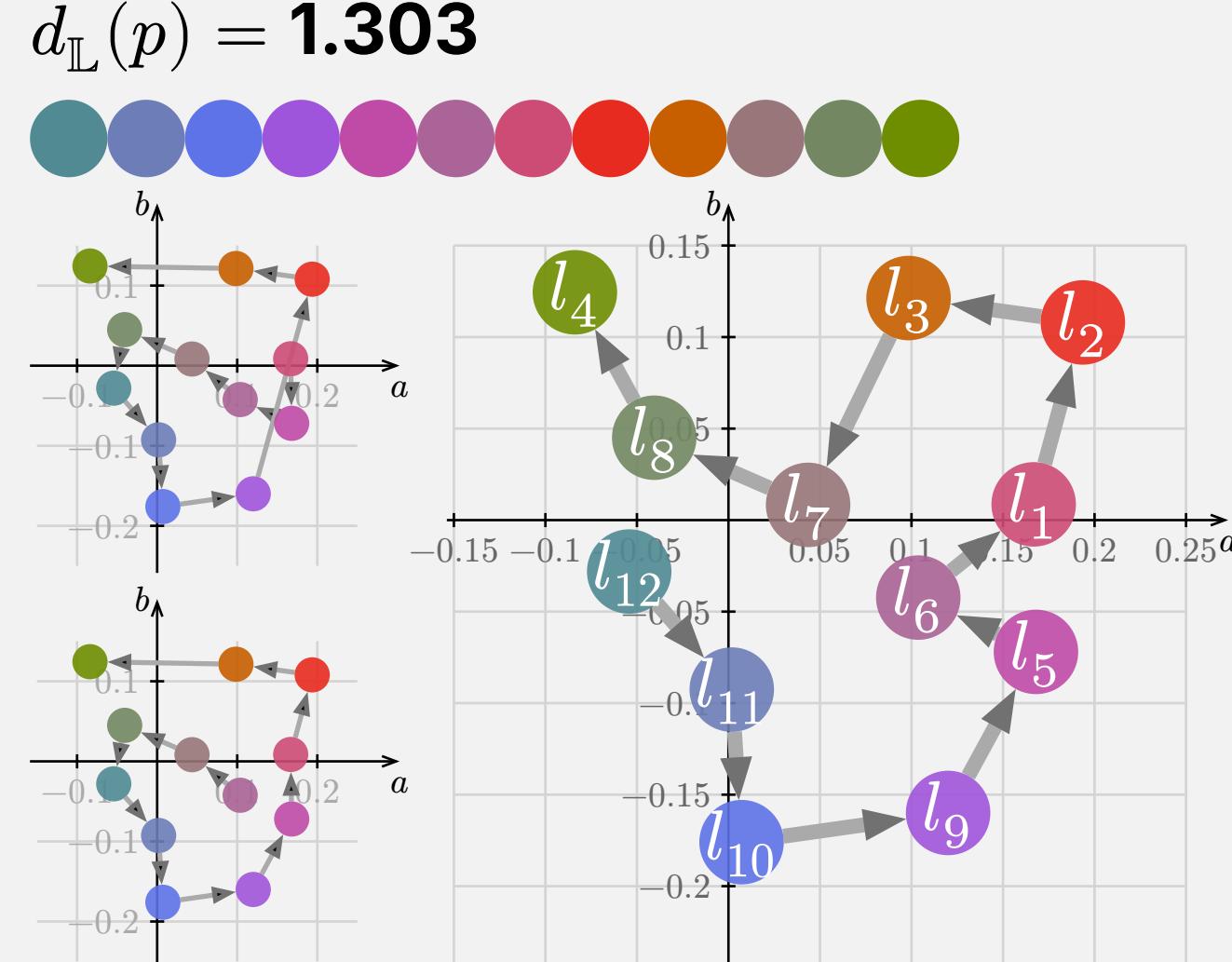
### Multiple Fragments



### NN + 2-opt



### NN + Inner Rotate



### Erläuterung

Während die Ausarbeitung eines pureren mathematischen **Modells** bereits wichtig und anspruchsvoll ist, ist es umso reizvoller, dieses Modell – und damit die mehrdimensionale Sortierung – auch **praktisch umzusetzen**. Zunächst habe ich graphentheoretisch bewiesen, dass die Ketten sortierung im allgemeinen Fall NP-schwer und damit unhandhabbar ist.

Zudem habe ich zahlreiche **Algorithmen** adaptiert, entwickelt und implementiert, die hier auch dargestellt werden. **Heuristisch** sind hier das Innere Rotieren und Simulated Annealing die besten Strategien, während meine Formulierung als ILP-Problem auch größere Instanzen noch **optimal** sortieren kann.

Rechts ist ein Quelltextausschnitt meiner Rust-Implementierung von Simulated Annealing zu sehen.

### Quelltextausschnitt

```
https://github.com/leo48/lydf2024-backend/blob/31759f694e020452782d04c27c0e9166a275/lycopath/improve.rs#L250
pub fn simulated_annealing<C: ImproveContext>(ctx: C) -> C::Path {
    let mut_path = ctx.start_path();
    let pool = ctx.options();

    let initial_temp: f64 = pool.initial_temperature.unwrap_or(0.15);
    let k: f64 = (pool
        .iteration_count
        .unwrap_or((10 / 0.000000025) as usize)
    as f64)
        .recip();
    let mut_temperature = initial_temp;
    let mut_i = 0;
    let mut_cost = ctx.cost(&mut_path);
    let mut_path_approx = path.clone();
    let mut_path_approx_cost = cost;
    let mut_improved_since_last_send = true;

    while temperature > 0.000000005 {
        // <14 Zeilen ausgelassen: Code zum Senden des aktuellen Pfads
        let index1 = fastrand::usize(..path.len());
        let index2 = fastrand::usize(..path.len());

        path.swap(index1, index2);
        let new_cost = ctx.cost(&path);
        let cost_delta = new_cost - cost;
        let thresh = f32::exp(-(cost_delta / temperature as f32));
        if new_cost < 0.0 || fastrand::f32() < thresh {
            cost = new_cost;
        } else {
            path.swap(index1, index2);
            if new_cost < path.clone().into(&mut_path_approx) {
                path_approx_cost = new_cost;
                improved_since_last_send = true;
            }
            temperature -= k;
            i += 1;
        }
        ctx.path_from_indices(path_approx.iter())
    }
}
```

$$d(\uparrow, \downarrow) = 0.120$$

$$d(\uparrow, \downarrow) = 0.157$$

$$d(\uparrow, \downarrow) = 0.098$$

$$d(\uparrow, \downarrow) = 0.101$$

$$d(\uparrow, \downarrow) = 0.101$$

$$d(\uparrow, \downarrow) = 0.125$$

$$d(\uparrow, \downarrow) = 0.142$$

$$d(\uparrow, \downarrow) = 0.104$$

$$d(\uparrow, \downarrow) = 0.107$$

$$d(\uparrow, \downarrow) = 0.093$$

$$d(\uparrow, \downarrow) = 0.113$$

$$\min \sum_{i=1}^n \sum_{j=1}^n X_{ij} \cdot d(l_i, l_j), \text{ s.t. } X_{ij} \in \{0, 1\}^{n \times n}$$

$$\wedge \forall i \in [1, n] : \sum_{j=1}^n X_{ij} + X_{ji} \geq 1$$

$$\wedge \forall i \in [1, n] : \sum_{j=1}^n X_{ij} \leq 1$$

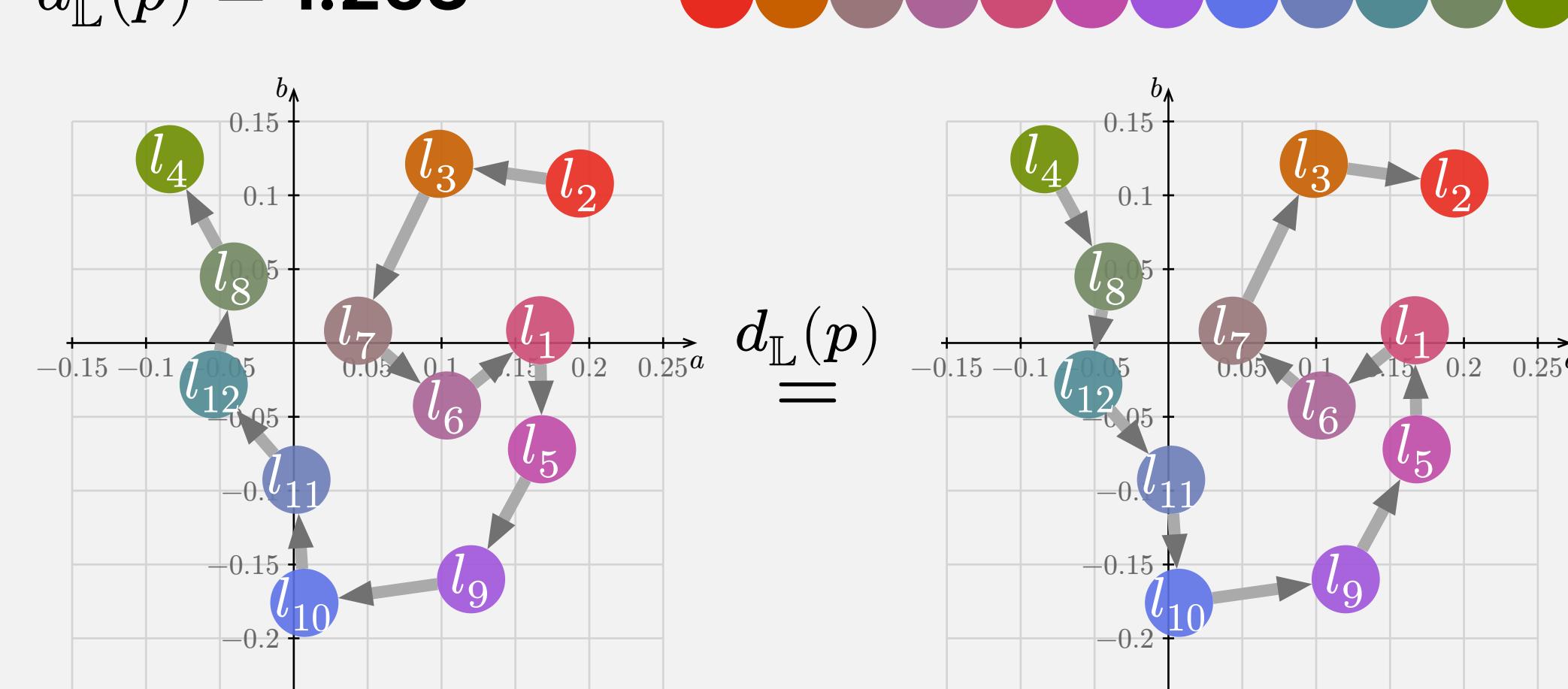
$$\wedge \forall j \in [1, n] : \sum_{i=1}^n X_{ij} \leq 1$$

$$\wedge \sum_{i=1}^n \sum_{j=1}^n X_{ij} = n - 1$$

$$\wedge \forall i \in [1, n] : \forall j \in [1, n] : X_{ij} + X_{ji} \leq 1$$

### Optimaler Pfad

$$d_{\mathbb{L}}(p) = 1.263$$

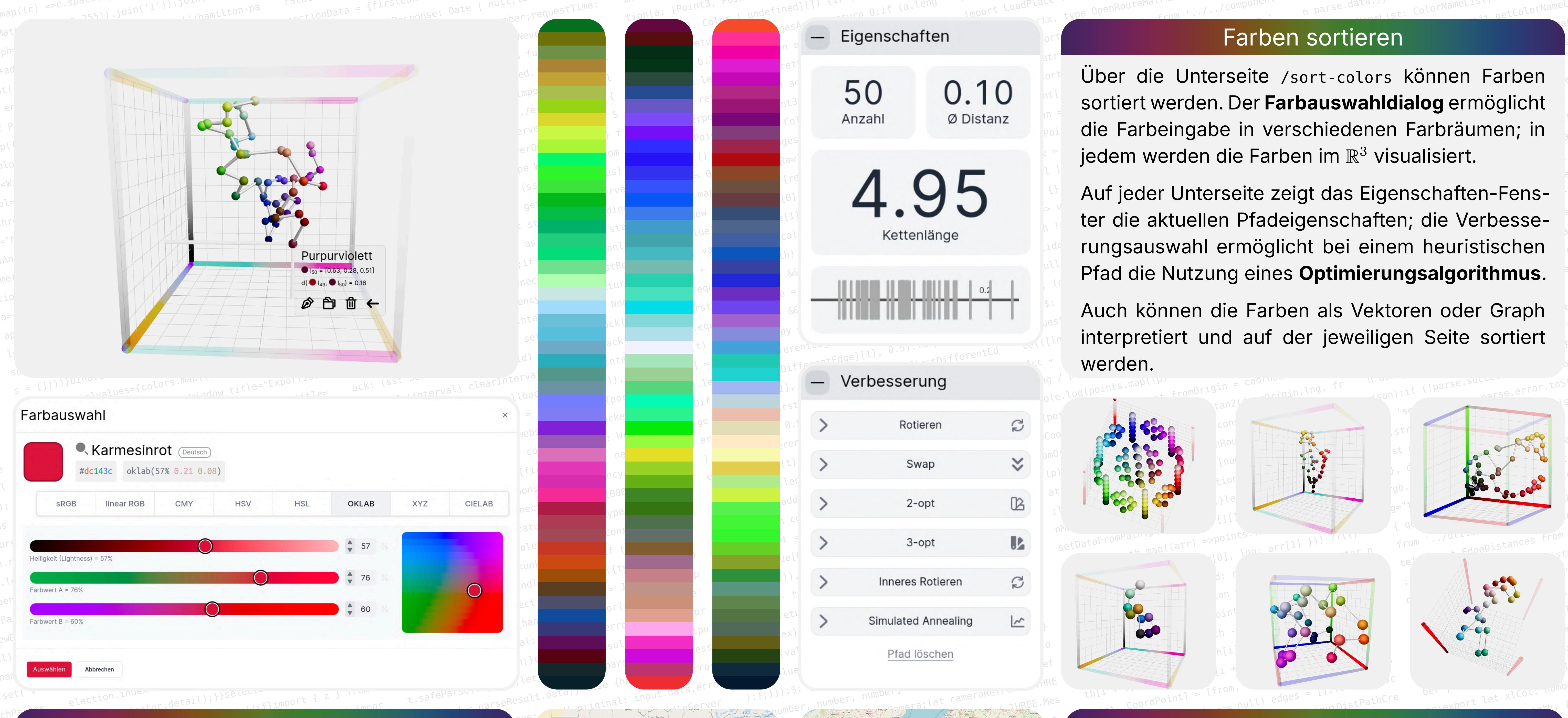
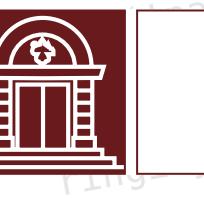
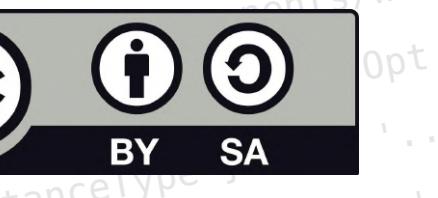


# sorting the colors

## Dimensionsbezogene Generalisierung vergleichsbasierter Sortierung

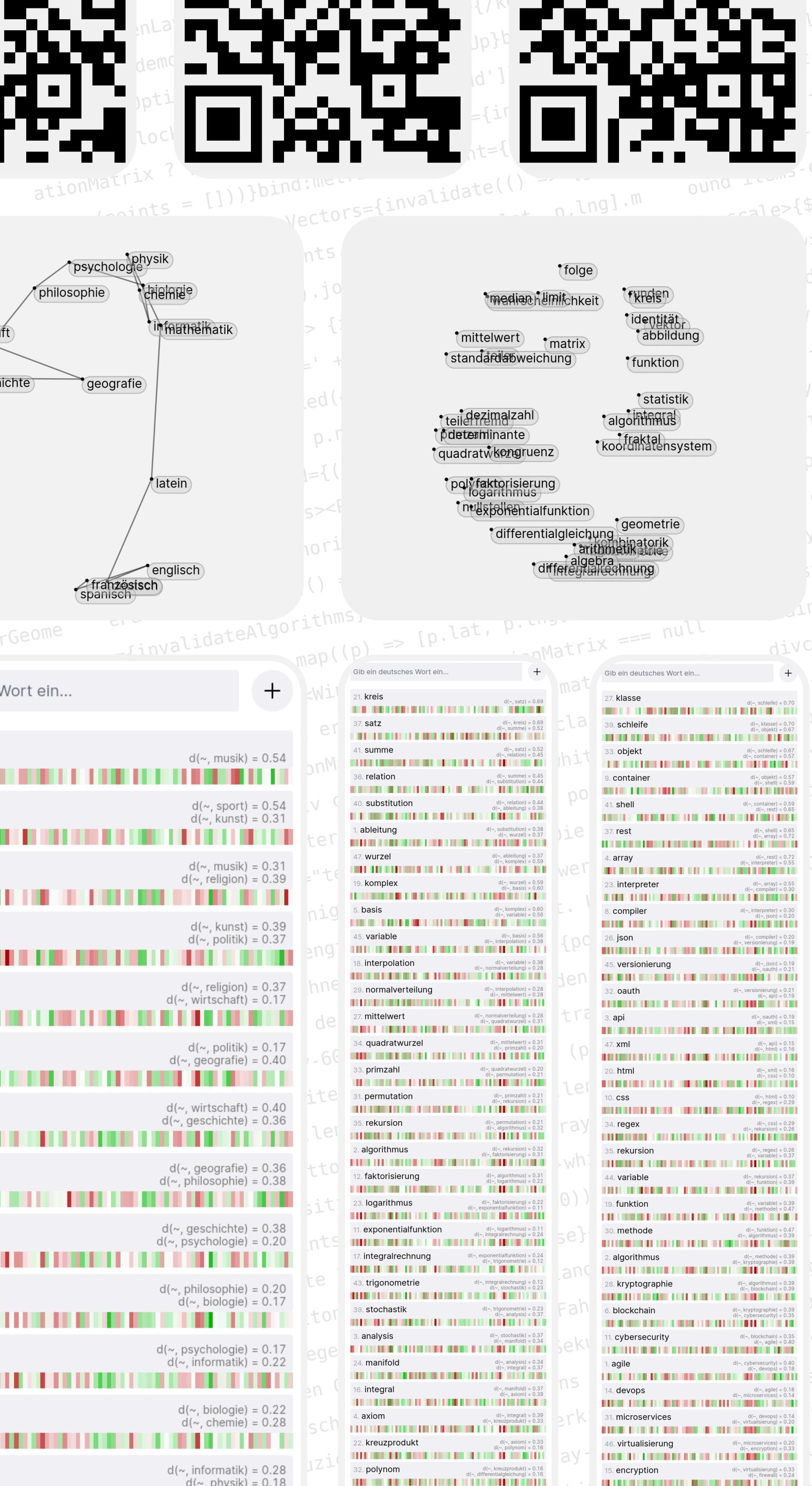
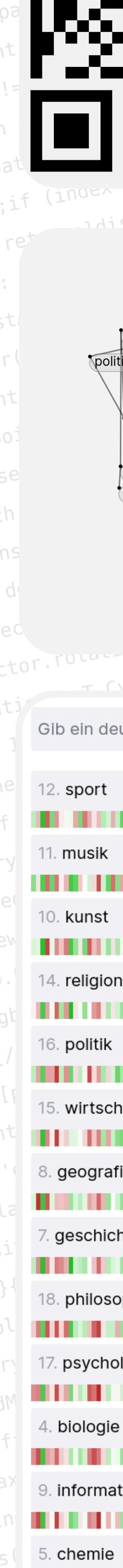
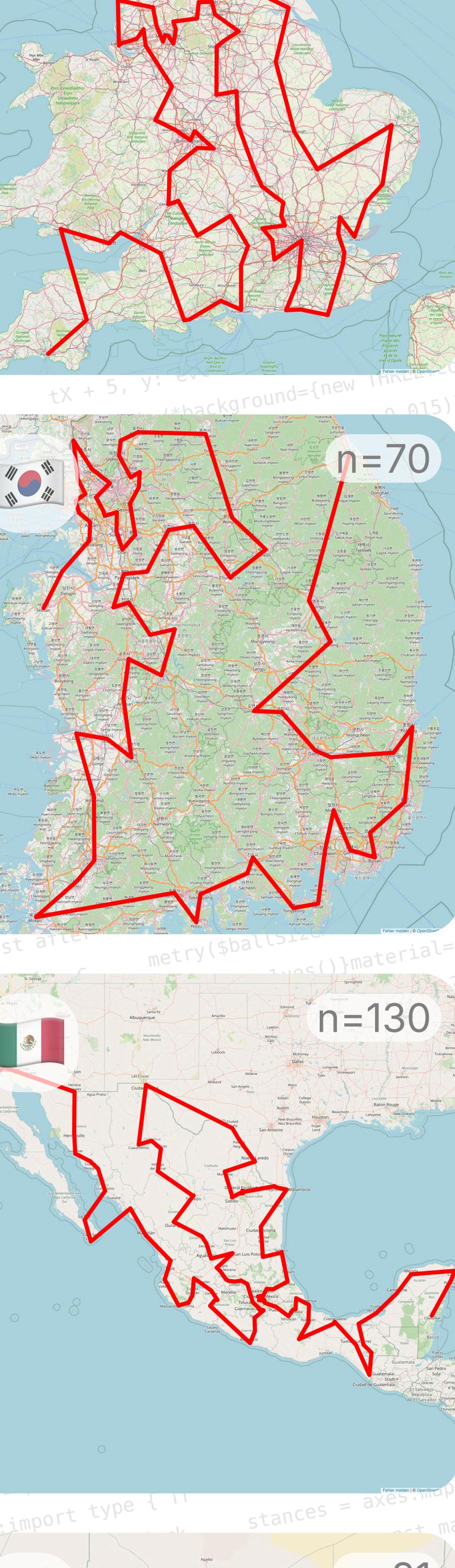
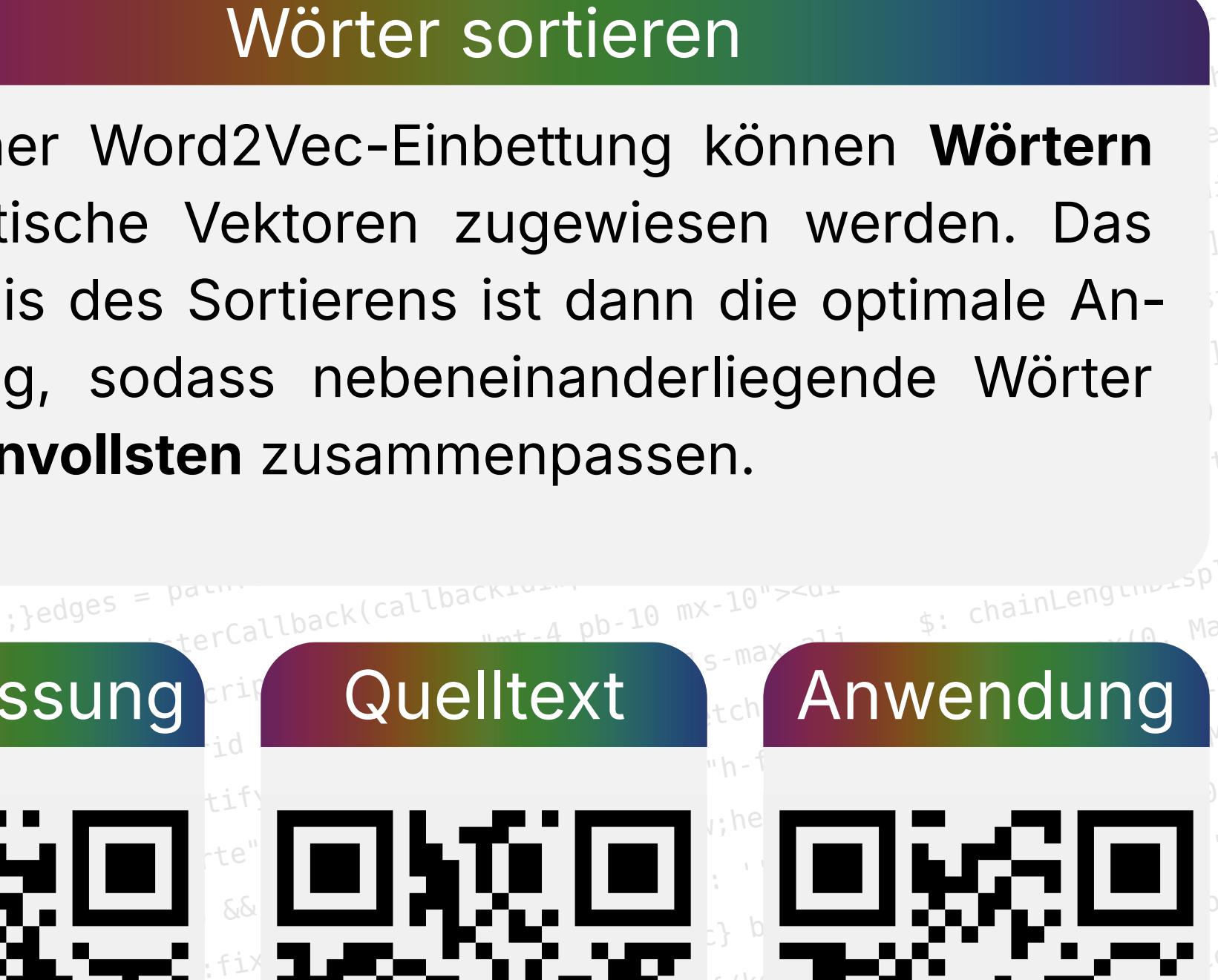
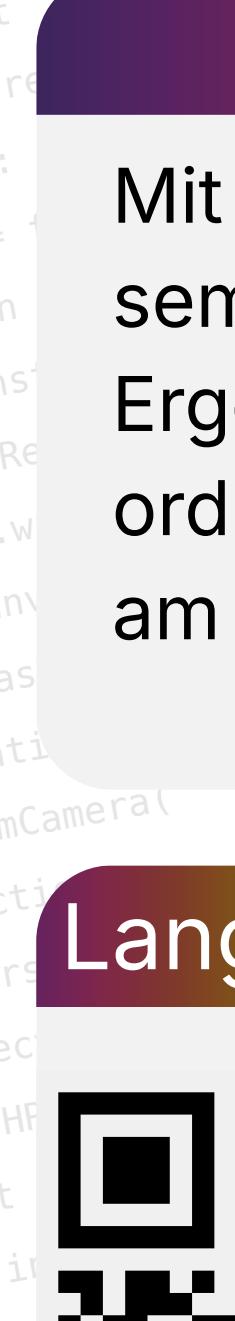
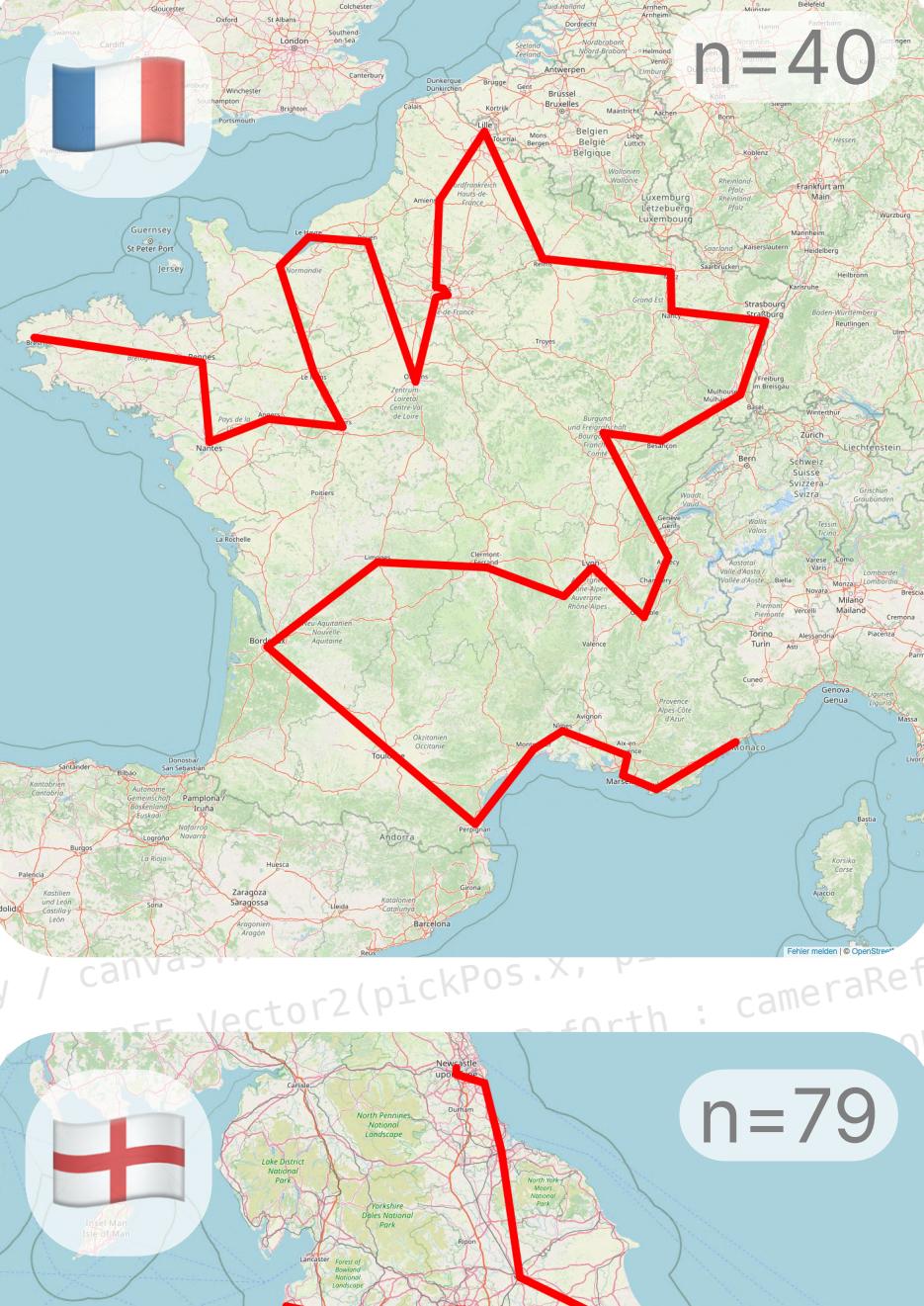
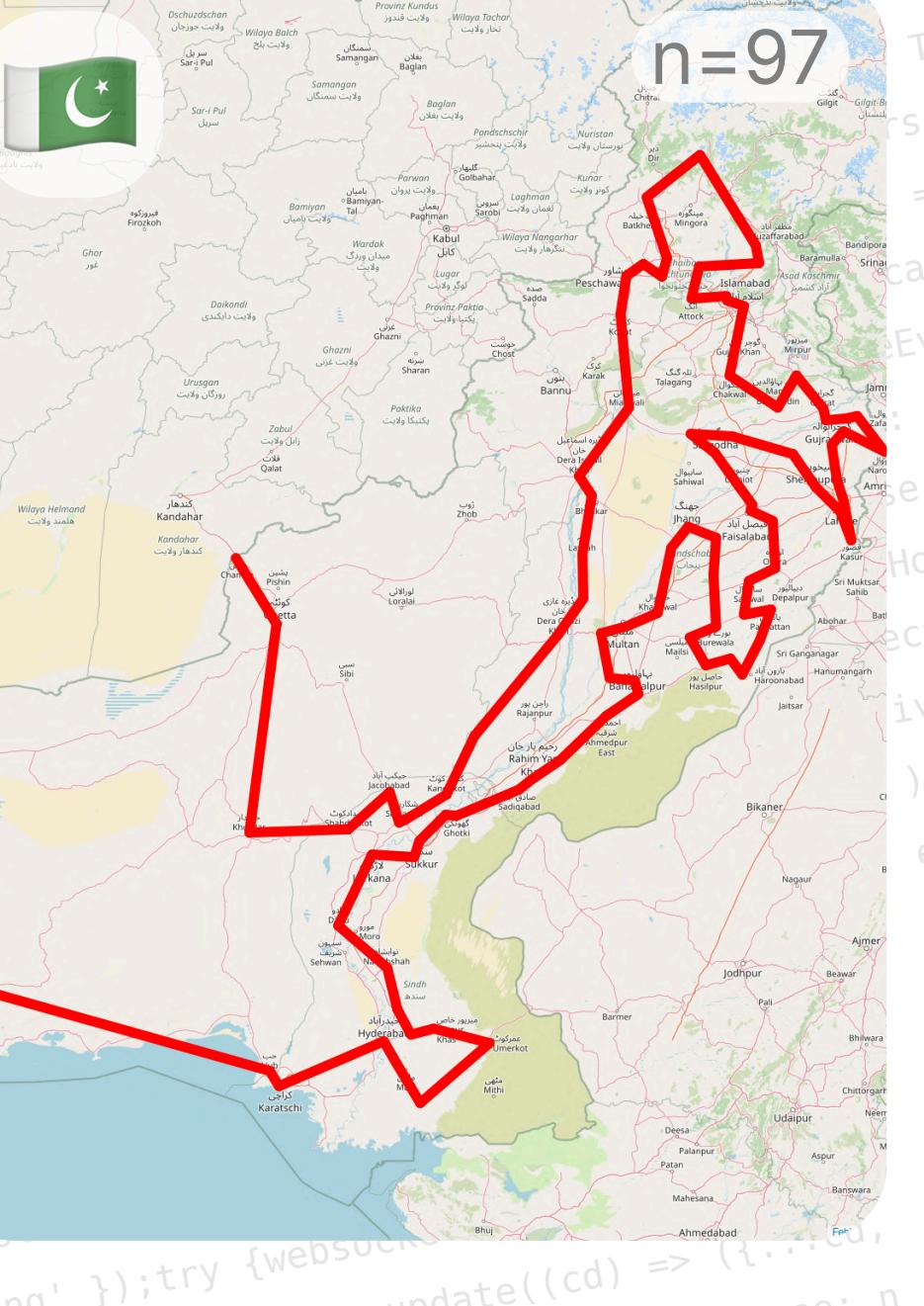
Ein Projekt von leo blume, 16. J.

Plakat 3/3: Anwendungsfelder, Visualisierung, Webanwendung



### Orte sortieren

Indem die geografischen Koordinaten von Orten als Vektoren betrachtet werden, kann eine Liste von Orten sortiert werden. Das Ergebnis ist der **kürzeste Pfad**, der alle Orte besucht: die hier dargestellten Beispiele zeigen die Großstädte ( $\geq 100\,000$  Ew.) von Deutschland 🇩🇪, Pakistan 🇵🇰, Frankreich 🇫🇷, England 🇬🇧, Südkorea 🇰🇷, Mexiko 🇲🇽 und Nigeria 🇳🇬.

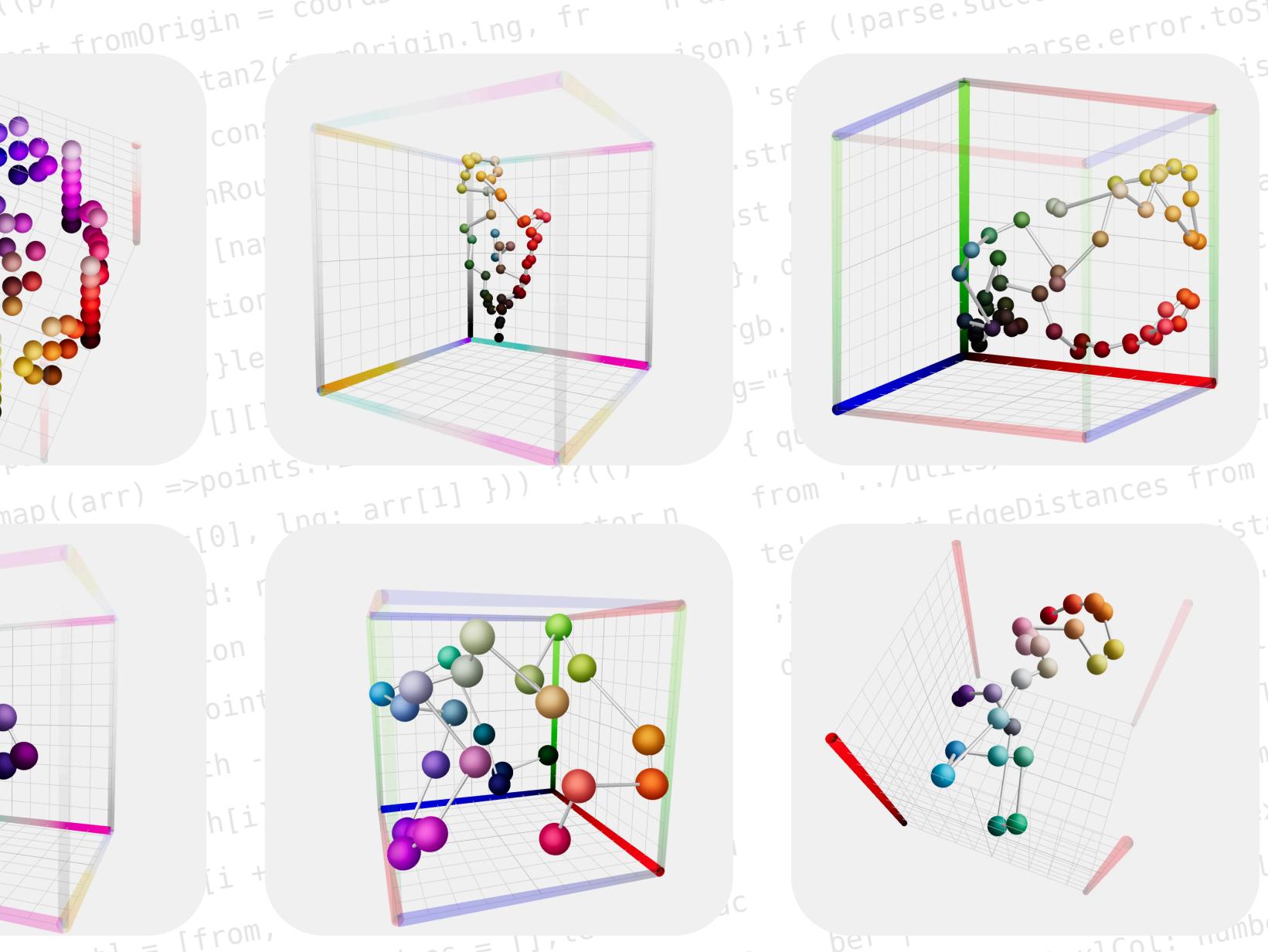


### Farben sortieren

Über die Unterseite /sort-colors können Farben sortiert werden. Der **Farbauswahldialog** ermöglicht die Farbeingabe in verschiedenen Farträumen; in jedem werden die Farben im  $\mathbb{R}^3$  visualisiert.

Auf jeder Unterseite zeigt das Eigenschaften-Fenster die aktuellen Pfadeigenschaften; die Verbesserungsauswahl ermöglicht bei einem heuristischen Pfad die Nutzung eines **Optimierungsalgorithmus**.

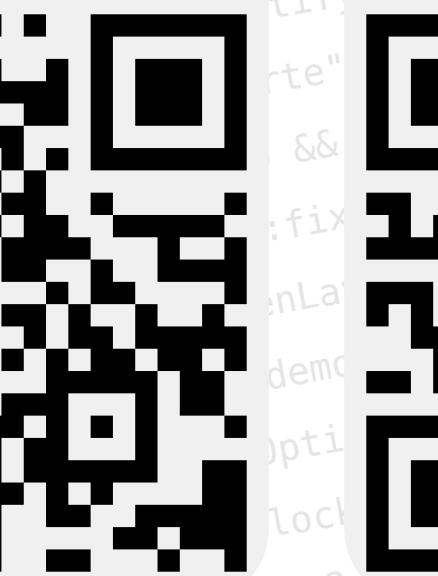
Auch können die Farben als Vektoren oder Graph interpretiert und auf der jeweiligen Seite sortiert werden.



### Wörter sortieren

Mit einer Word2Vec-Einbettung können **Wörtern** semantische Vektoren zugewiesen werden. Das Ergebnis des Sortierens ist dann die optimale Anordnung, sodass nebeneinanderliegende Wörter am **sinnvollsten** zusammenpassen.

### Langfassung



### Quelltext



### Anwendung

