

¹

Schwarmintelligenz

²

Andreas Auer, Philipp Fuchs,
Sarah Götz, Moritz Kondmann

³

25. Februar 2025

⁴

Zusammenfassung

⁵

Diese Arbeit beschäftigt sich mit den grundlegenden Prinzipien der Schwarmintelligenz und stellt zwei darauf basierende Algorithmen vor. Im Fokus stehen der Ameisenalgorithmus (Ant Colony Optimization, ACO) und die Partikelschwarmoptimierung (Particle Swarm Optimization, PSO), die sich durch selbstorganisierte und dezentrale Entscheidungsmechanismen auszeichnen. Neben einer detaillierten Erläuterung ihrer Funktionsweise werden ihre jeweiligen Stärken und Schwächen analysiert. Zudem wird aufgezeigt, in welchen Anwendungsbereichen diese Algorithmen zum Einsatz kommen, beispielsweise in der Optimierung von Routen und in kontinuierlichen Optimierungsprozessen.

¹⁴

1 Schwarmintelligenz Allgemein

¹⁵

Schwarmintelligenz beschreibt das kollektive Verhalten von dezentralisierten, selbstorganisierten Systemen. Ein solches Kollektiv zeichnet sich durch die dezentrale Führung der Gruppe, die Selbstorganisation eines jeden Individuums und durch das Zusammenspiel der einzelnen Schwarmmitglieder untereinander aus. In der Natur zeigen Tiere diese Verhaltensweisen, um wichtige Aufgaben wie die Nahrungssuche, die Suche nach einem Rastplatz oder den Schutz vor Feinden zu bewältigen. Beispiele sind zum einen Ameisenkolonien und zum anderen Vogel- oder Fischschwärme. [vgl. KE01]

²²

In der Informatik wird das Prinzip der Schwarmintelligenz zur Entwicklung von leistungsfähigen Optimierungsalgorithmen, wie dem Ameisenalgorithmus oder der Partikelschwarmoptimierung genutzt. Diese Verfahren nutzen das koordinierte Verhalten vieler einfacher Einheiten, die durch lokale Interaktionen komplexe Probleme effizient lösen können. Sie überzeugen durch ihre Flexibilität, Widerstandsfähigkeit und Skalierbarkeit, wodurch sie für zahlreiche Optimierungsaufgaben besonders geeignet sind. [vgl. KE01, DS04]

²⁹ 2 Ameisenalgorithmus ACO

³⁰ 2.1 Einleitung

³¹ Eines der wesentlichen Probleme der Informatik ist die Optimierung komplexer Routen.
³² Dafür gibt es verschiedene Lösungsansätze. Ein recht erfolgreicher hat seinen Ursprung
³³ in der Natur: Ameisen bilden einen Schwarm und finden so gemeinsam durch eigens
³⁴ entwickelte, simple, aber clevere Mechanismen kürzeste Wege zu Nahrungsquellen oder
³⁵ Ähnlichem. Dieses Verhalten bildet die Grundlage für die sogenannte Ant Colony Opti-
³⁶ mization (ACO), eine von Ameisenkolonien inspirierte Metaheuristik, die zur bestmögli-
³⁷ chen Lösung komplexer, kombinatorischer Optimierungsprobleme eingesetzt wird. Diese
³⁸ Methode wurde in den 1990er Jahren von Marco Dorigo und seinen Kollegen entwi-
³⁹ ckelt und wird seither weiterentwickelt und für verschiedenste Probleme verwendet und
⁴⁰ adaptiert. [vgl. DS04]

⁴¹ 2.2 Biologie

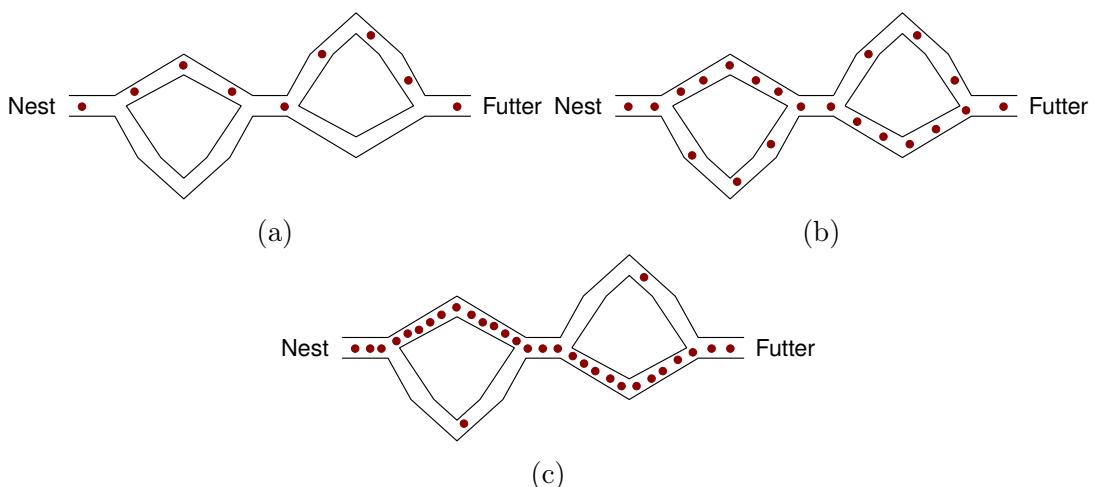


Abbildung 1: Entwicklung der Pheromonverteilung in der Natur [vgl. Eng07]

⁴² Die Strategie, nach der Ameisen in der Natur vorgehen und auf welcher auch das Prinzip
⁴³ der ACO-Algorithmen basiert, sind Pheromone. Ameisen produzieren diese Pheromone
⁴⁴ und hinterlassen sie auf ihren Routen, wodurch eine einfache Kommunikation und Ori-
⁴⁵ entierung der Kolonie möglich ist. Folgendermaßen gehen die Ameisen vor: Zu Beginn
⁴⁶ bewegt sich eine Ameise zufällig in eine beliebige Richtung. Währenddessen hinterlässt
⁴⁷ sie Pheromone, die von den folgenden Ameisen wahrgenommen werden [Abb. 1a]. Mit
⁴⁸ zunehmender Anzahl an Ameisen wird mehr Pheromon auf allen Pfaden verteilt - ins-
⁴⁹ besondere steigt die Pheromonkonzentration auf den kürzesten Pfaden, da diese schnel-
⁵⁰ ler zurückgelegt und somit häufiger markiert werden. Zudem sind Wege mit höherer
⁵¹ Pheromonkonzentration attraktiver für nachfolgende Ameisen, wodurch diese Wege mit

52 größerer Wahrscheinlichkeit gegangen werden. Währenddessen verschwindet Pheromon
53 auf weniger genutzten Routen durch Verdunstung allmählich [Abb. 1b]. Nach gewisser
54 Zeit entsteht so eine recht deutliche Pheromonverteilung. Die wenigen Ausreißer dienen
55 der Erkundung - für den Fall, dass beispielsweise ein neuer, kürzerer Pfad entdeckt wird.
56 Durch diese Mechanismen ist es den Ameisen möglich, sich auf die effizienten Wege zu
57 fokussieren, während gleichzeitig die Möglichkeit der Erkundung neuer Alternativen er-
58 halten bleibt [Abb. 1c]. Dieses Verhalten lässt sich auf ACO-Algorithmen übertragen,
59 wobei Populationen künstlicher Ameisen verwendet werden, die in einer graphischen
60 Darstellung des Problems navigieren. [vgl. DS04]

61 2.3 Problem des Handelsreisenden

62 Das Problem des Handelsreisenden (Travelling Salesman Problem, TSP) ist ein funda-
63 mentales kombinatorisches Optimierungsproblem in der Graphentheorie. Es beschreibt
64 das Problem eines Verkäufers, der ausgehend von seiner Heimatstadt, die kürzeste Tour
65 finden will um alle Städte seiner Kunden genau einmal zu besuchen und anschließend
66 zurückzukehren. Mathematisch kann das Problem als vollständig gewichteter Graph be-
67 schrieben werden, wobei die optimale Lösung, einem Hamiltonschen Kreis mit minimaler
68 Kantensumme entspricht. [Abb. 2]. Die Knoten repräsentieren dabei die Städte der Kun-
69 den und die Kantengewichte die Distanz zwischen zwei Städten. [vgl. DS04]

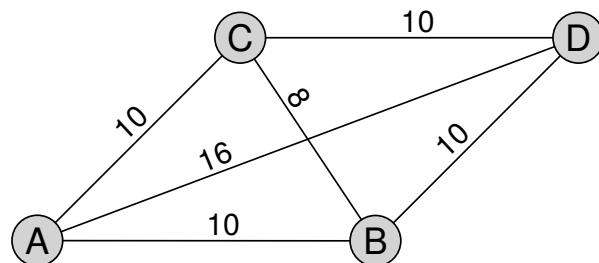


Abbildung 2: TSP Beispielgraph

70 2.3.1 Lösungsansatz: Brute Force

71 Der Brute Force Ansatz löst das Problem, indem er die Kantensummen aller möglichen
72 Permutationen berechnet, und anschließend die Tour mit der minimalen Gesamtdistanz
73 auswählt. Die Anzahl der Permutationen entspricht dabei $\frac{(n-1)!}{2}$, wobei n die Anzahl der
74 Knoten im Graph ist. [vgl. VR17]
75 Ein wesentlicher Vorteil dieser Methode ist, dass durch Berechnen aller möglichen Per-
76 mutationen garantiert eine optimale Lösung gefunden wird. Jedoch nimmt die Anzahl
77 der Permutationen für große natürliche Zahlen n extrem schnell zu, sodass dieser Ansatz
78 nur für kleine Eingabegrößen effizient berechenbar ist [Tabelle 1].

Anzahl Städte	Anzahl Touren
4	3
5	12
10	181 440
15	43 589 145 600
20	60 822 550 204 416 000

Tabelle 1: Brute Force Komplexität

79 2.3.2 Lösungsansatz: Greedy Algorithmus - Nearest Neighbor

80 Eine Alternative stellt das Verwenden eines Greedy Algorithmus (z.B. Nearest Neighbor)
 81 dar. Dieser trifft mit jedem Schritt eine lokal optimale Entscheidung und wählt somit
 82 immer die Stadt mit der geringsten Entfernung um eine Tour zu konstruieren. [vgl.
 83 DS04]

84 Am Beispielgraphen [Abb. 3] startet der Algorithmus eine Tour bei Knoten A und kann
 85 zunächst zwischen den Knoten B, C und D wählen. Aufgrund der geringeren Distanz
 86 zu Knoten B und C wird der Algorithmus einen der beiden Knoten auswählen. Fällt die
 87 Entscheidung auf B, so wird anschließend C bevorzugt, da die Kante \overline{BC} die geringeren
 88 Kosten aufweist. Nun bleibt als letzte Wahl nur noch der Knoten D übrig, von dem aus
 89 es mit \overline{DA} zurück zum Startknoten A geht.

90 Ein wesentlicher Nachteil dieses Ansatzes besteht darin, dass eine lokal optimale Ent-
 91 scheidung nicht zwangsläufig zu einem globalen Optimum führt. Es kann z.B. vorkommen
 92 dass der Algorithmus am Schluss gezwungen ist, eine deutlich teurere Kante zum Start-
 93 knoten zu wählen. Wie in Abb. 3 zu sehen führt dies zu einer Gesamtlänge der Tour von
 94 44, während eine optimale Tour (z.B. $A \rightarrow B \rightarrow D \rightarrow C \rightarrow A$) nur eine Länge von 40
 95 aufweist. [vgl. DS04]

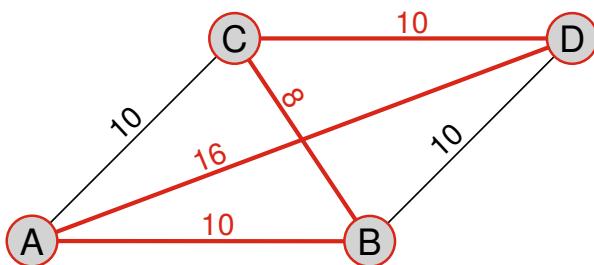


Abbildung 3: Greedy Algorithmus Lösung

⁹⁶ 2.3.3 Lösungsansatz: ACO Algorithmus

⁹⁷ Der ACO löst das TSP-Problem unter Verwendung von künstlichen Ameisen, bei denen
⁹⁸ die Entscheidungsfindung echter Ameisen mittels einer Wahrscheinlichkeitsfunktion si-
⁹⁹ muliert wird. Dabei werden neben den heuristischen Informationen wie z.B. der Distanz
¹⁰⁰ zwischen zwei Städten auch historische Informationen (Pheromone) berücksichtigt. Die-
¹⁰¹ se dienen als Gedächtnis für vergangene gute Lösungen, da Kanten die zuvor Teil einer
¹⁰² guten Lösung waren, eine höhere Pheromonkonzentration aufweisen. [vgl. DS04]

¹⁰³ Ablauf des ACO Algorithmus:

- ¹⁰⁴ 1. Pheromonwerte aller Kanten initialisieren.
 - ¹⁰⁵ 2. Ameisen zufällig auf die Knoten des Graphen verteilen.
 - ¹⁰⁶ 3. Jede Ameise konstruiert eine eigene, wahrscheinlichkeitsbasierte Lösung und er-
¹⁰⁷ mittelt deren Qualität
 - ¹⁰⁸ 4. Pheromonwerte aktualisieren
- ¹⁰⁹ Die Punkte 2 bis 4 werden so lange wiederholt, bis die Lösung konvergiert oder ein
¹¹⁰ Abbruchkriterium erreicht ist. [vgl. DS04]

¹¹¹ 2.4 Anwendung des Algorithmus

¹¹² Angewandt an dem TSP-Beispiel lässt sich der Ablauf des Algorithmus nun veranschau-
¹¹³ lichen. In dieser Veranschaulichung wird der Algorithmus für eine einzelne künstliche
¹¹⁴ Ameise durchlaufen. In der realen Anwendung des Algorithmus werden meist mehrere
¹¹⁵ Ameisen zugleich losgeschickt.

¹¹⁶ Im ersten Schritt werden die Pheromone initialisiert - eine geringe Menge an Phero-
¹¹⁷ mon wird gleichmäßig auf alle Kanten des zuvor definierten Graphen [Abb. 2] verteilt.
¹¹⁸ Beim Start der Ameise in Punkt *A* steht diese nun vor der Wahl zwischen den Kanten
¹¹⁹ \overline{AB} , \overline{AC} und \overline{AD} . Die Entscheidung über den nächstgewählten Knoten basiert auf ei-
¹²⁰ ner Wahrscheinlichkeitsregel. Die Wahrscheinlichkeit P_{ij} einer sich aktuell in Knoten *i*
¹²¹ befindenden Ameise, den Knoten *j* zu wählen ist gegeben durch

$$P_{ij} = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{k \in N_i} \tau_{ik}^\alpha \cdot \eta_{ik}^\beta}.$$

¹²² Dabei werden folgende Parameter herangezogen:

- ¹²³ • τ_{ij} repräsentiert die Pheromonkonzentration auf der Kante \overline{ij} ; diese wird nach
¹²⁴ Durchlauf aller zugleich gestarteten Ameisen jeweils aktualisiert.

125 • η_{ij} ist der heuristische Wert und entspricht typischerweise dem Kehrwert der Di-
126 stanz ($1/d_{ij}$) oder Zeit, wodurch kürzere Wege bevorzugt werden (auch Maß der
127 Attraktivität oder Sichtbarkeit genannt); diese Information ist bereits vor Beginn
128 des Algorithmus bekannt.

- 129 • α, β sind Gewichtungsfaktoren, die den relativen Einfluss von Pheromoninforma-
130 tion und heuristischem Wert steuern.
131 • N_i stellt die Menge der noch nicht von der Ameise besuchten Nachbarknoten von
132 i dar; für - unter anderem - diese Information verfügen die künstlichen Ameisen
133 über ein Gedächtnis.

134 Diese Formel zur Berechnung der Wahrscheinlichkeit wird häufig, je nach Problemstel-
135 lung und benötigter Algorithmus-Variante leicht abgeändert, Parameter ändern sich oder
136 werden hinzugefügt. Die hier beschriebene und genutzte Formel ist eine Basisversion, von
137 welcher viele Varianten abgeleitet werden - sie stammt aus dem ursprünglichen ACO-
138 Algorithmus von M. Dorigo. [vgl. DS04]

139 Unter Verwendung dieser Wahrscheinlichkeitsformel und dem initialisierten Pheromon
140 lassen sich die Wahrscheinlichkeiten der Kanten des Graphen berechnen. Anschließend
141 wird ein Zufallswert zwischen 0 und 1 bestimmt, nachdem die zu gehende Kante gewählt
142 wurde. Bei den Wahrscheinlichkeiten

$$P_{AB} = 0.38$$

$$P_{AC} = 0.38$$

$$P_{AD} = 0.24$$

144 und einem Zufallswert 0.30, welcher auf P_{AB} abbildet, wird die Kante \overline{AB} gewählt
145 [Abb. 4a]. Nach diesem Vorgehen bewegt sich die künstliche Ameise nun durch den
146 Graphen. In Stadt B hat sie die Wahl zwischen Kante \overline{BC} und \overline{BD} (zu A kann sie
147 schließlich noch nicht zurück, da alle Knoten besucht werden müssen). Nach Berechnung
148 von Wahrscheinlichkeiten

$$P_{BC} = 0.56$$

$$P_{BD} = 0.44$$

150 und einem Zufallswert 0.73, welcher auf P_{BD} abbildet, geht die Ameise nach Knoten
151 D [Abb. 4b]. Hier kommt der Vorteil der Zufallskomponente zum Tragen, da so, trotz
152 höherer Wahrscheinlichkeit der Kante \overline{BC} , die Kante \overline{BD} entdeckt und auch gegangen
153 wird. Die Wahrscheinlichkeit, den letzten noch nicht verwendeten Knoten C zu besuchen,
154 liegt trivialerweise bei eins und damit wird Kante \overline{CD} gegangen. Schlussendlich bleibt
155 noch die Kante \overline{AC} zurück zum Start und damit ist die Tour vollendet, welche in dem
156 Fall tatsächlich dem kürzesten Weg entspricht [Abb. 4c]. [vgl. DS04]

157 Im nächsten Schritt werden nun die Pheromonwerte aktualisiert, damit sich die Wahr-
158 scheinlichkeit, dass folgende Ameisen wieder diesen kürzesten Weg gehen, erhöht. Dieser

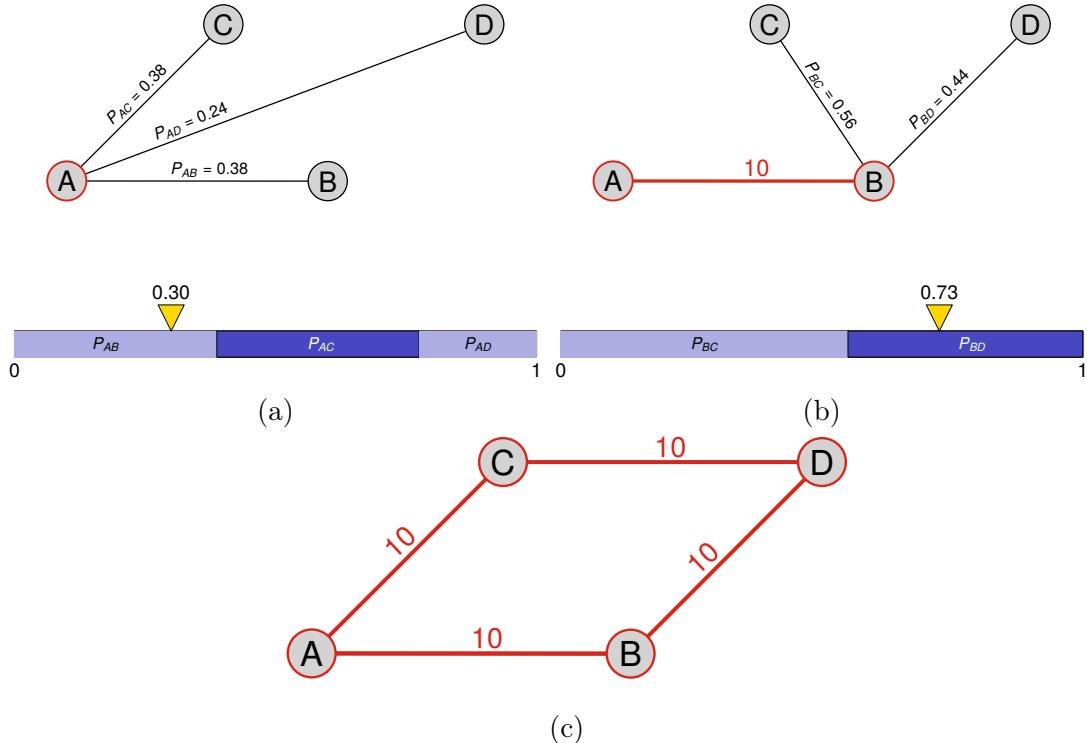


Abbildung 4: Wegfindung anhand von Wahrscheinlichkeiten und Zufallswerten

159 Schritt erfolgt in der echten Anwendung erst nachdem alle Ameisen den Iterationsschritt
 160 vollendet haben. Um die neuen Pheromonwerte zu berechnen, wird folgende zweiteilige
 161 Aktualisierungsregel, bestehend aus Verdunstung und Verstärkung, angewandt:

162 1. Die Verdunstung aller Kanten berechnet sich durch

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij},$$

163 wobei

- 164 • ρ die Verdunstungsrate und
 165 • τ_{ij} den Pheromonwert einer Kante darstellt.

166 Hierbei werden die Pheromonmengen aller Pfade um einen Faktor, abhängig von
 167 der verwendeten Verdunstungsrate reduziert. Damit ist es einerseits möglich nicht
 168 optimale Entscheidungen zu vergessen. Andererseits können sich gute Wege besser
 169 absetzen.

170 2. Die Verstärkung besuchter Kanten lässt sich durch

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^m \frac{Q}{L_k}$$

171 berechnen. Hierbei ist

¹⁷² • Q die gesamte Pheromonmenge,

¹⁷³ • L_k die Länge der Lösung der k-ten Ameisen und

¹⁷⁴ • m die Anzahl der Ameisen.

¹⁷⁵ Mit Hilfe dieser Formel wird jede Kante, die Teil einer Lösung der künstlichen
¹⁷⁶ Ameise ist, durch eine Pheromonmenge relativ zur Qualität der gefundenen Tour
¹⁷⁷ verstärkt. Angewandt bedeutet das, dass die Kanten einer kürzeren Tour mehr
¹⁷⁸ Pheromon aufweisen als jene einer längeren Tour.

¹⁷⁹ Mittels dieser Pheromon-Aktualisierung werden kurze gute Wege deutlicher verstärkt,
¹⁸⁰ was zu erhöhten und mit jeder Iteration eindeutigeren Wahrscheinlichkeiten führt. Kurze
¹⁸¹ Wege werden mit der Zeit klar bevorzugt [Abb. 5]. [vgl. DS04]

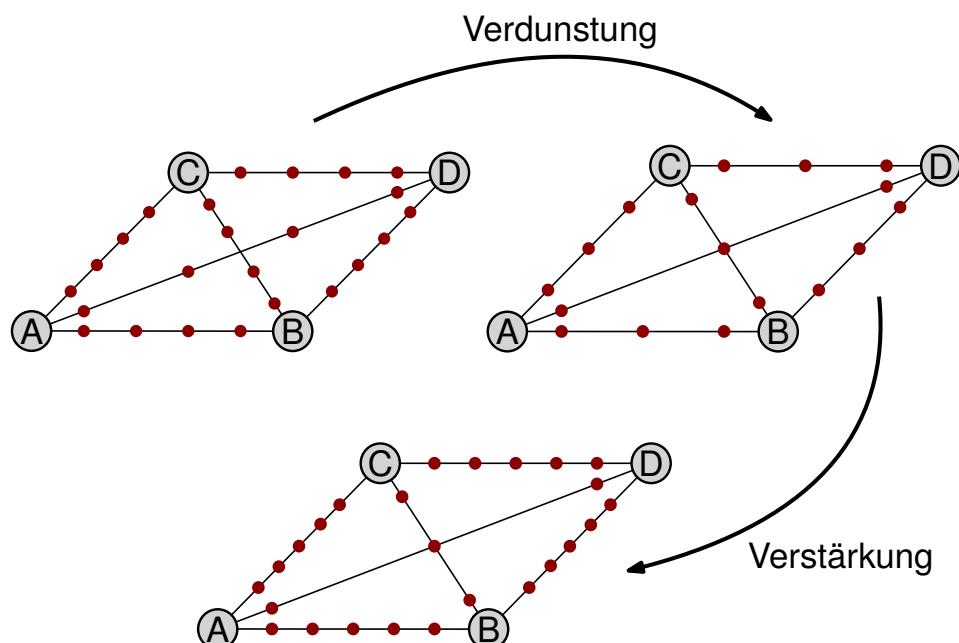


Abbildung 5: Verdunstung und Verstärkung der Kanten

¹⁸² Zusammenfassend liegt die Stärke des ACO in der Fähigkeit komplexe kombinatorische
¹⁸³ Optimierungsprobleme zu lösen. Dies findet unter anderem Anwendung im Bereich der
¹⁸⁴ Logistik, z.B. bei der Optimierung von Lieferketten und Verkehrsfluss. Aber auch im
¹⁸⁵ Netzwerk Routing können damit Datenpakete effizient geleitet werden, also z.B. Routen
¹⁸⁶ mit geringerer Latenz bevorzugt werden. Ein weiteres Anwendungsgebiet des ACO ist
¹⁸⁷ die Planung und Zuteilung knapper Ressourcen z.B. bei der Produktionsplanung. [vgl.
¹⁸⁸ DS04]

¹⁸⁹ 3 Partikelschwarmoptimierung PSO

¹⁹⁰ 3.1 Biologie

¹⁹¹ Biologisch inspiriert ist die Partikelschwarmoptimierung von Fisch- und Vogelschwärmen. Die Mitglieder eines solchen Schwarmes folgen dabei einfachen Regeln, die in ¹⁹² der Folge dann zu einem komplexen globalen Verhalten führen. Jedes Schwarmmitglied ¹⁹³ möchte bei seinem Schwarm bleiben, nicht mit den anderen Individuen zusammenstoßen und sich in die gleiche Richtung bewegen wie die anderen. Fische und auch Vögel ¹⁹⁴ nutzen dieses Verhalten um die Nahrungssuche zu optimieren aber auch um sich vor ¹⁹⁵ Fressfeinden zu schützen. [vgl. KE01]

¹⁹⁸ 3.2 Algorithmus

¹⁹⁹ Die Partikelschwarmoptimierung wird verwendet um Extremstellen in multidimensionalen Funktionen zu finden. Diese Funktionen werden auch Suchräume genannt. Der ²⁰⁰ Algorithmus setzt dabei auf die Simulation eines Schwarms von sogenannten Partikeln. ²⁰¹ Diese Partikel stellen mögliche Lösungen des Optimierungsproblems dar. Jedes Partikel ²⁰² kennt die Koordinaten seiner aktuellen Position im Suchraum, kann sich mit Hilfe eines ²⁰³ Bewegungsvektors bewegen und hat zusätzlich ein Gedächtnis für die persönlich beste ²⁰⁴ erreichte Position (kognitive Komponente) und die beste vom Schwarm erreichte Position ²⁰⁵ (soziale Komponente). Der PSO-Algorithmus bestimmt für jedes Partikel i des Schwarms ²⁰⁶ die nächste Position P_i^t . Dabei wird initial jedem Partikel des Schwarms eine zufällige ²⁰⁷ Position zugewiesen. Des weiteren wird der persönliche Bestwert $P_{pbest(i)^t}$ [Abschnitt 3.2.2] ²⁰⁸ auf die aktuelle Position gesetzt. Der globale Bestwert P_{gbest}^t [Abschnitt 3.2.3] wird für ²⁰⁹ den Schwarm berechnet und für die erste Iteration gespeichert. Bei jedem folgenden ²¹⁰ Iterationsschritt $t + 1$ des Algorithmus wird die Position für jedes Partikel neu berechnet. Um die nächste Position zu bestimmen wird ein Bewegungsvektor verwendet. Der ²¹¹ Bewegungsvektor [Abb. 6] setzt sich aus drei Komponenten zusammen:

- ²¹⁴ • Trägheitsmoment ωV_i^t ,
- ²¹⁵ • Soziale Komponente $c_1 r_1 (P_{pbest(i)^t} - P_i^t)$ und
- ²¹⁶ • Kognitive Komponente $c_2 r_2 (P_{gbest}^t - P_i^t)$.

²¹⁷ Daraus ergibt sich

$$V_i^{t+1} = \omega V_i^t + c_1 r_1 (P_{pbest(i)^t} - P_i^t) + c_2 r_2 (P_{gbest}^t - P_i^t)$$

²¹⁸ als Formel für den Bewegungsvektor. Die Position der Partikel wird so oft neu berechnet, ²¹⁹ bis eine Abbruchbedingung erreicht ist. Diese kann zum Beispiel ein für 50 Iterationen ²²⁰ unveränderter globaler Bestwert sein. [vgl. KE95, ZWJ15]

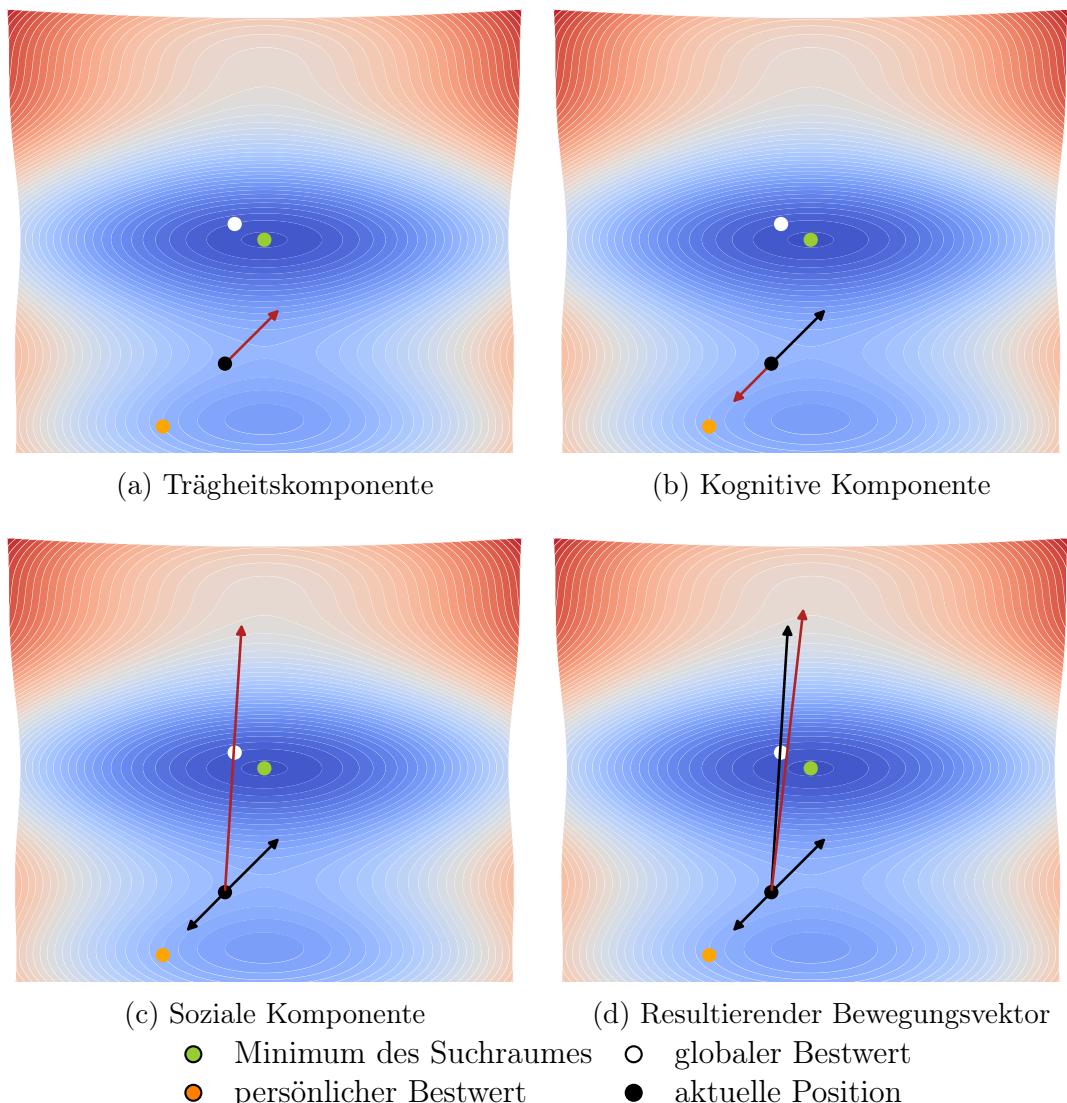


Abbildung 6: Berechnung des Bewegungsvektors

3.2.1 Trägheitskomponente

Der Algorithmus verwendet die letzte Bewegung des Partikels für die nächste Bewegung wieder. Die Bewegung wird mit einer Gewichtung ω versehen und bildet mit dem letzten Bewegungsvektor V_i^t die Trägheitskomponente [Abb. 6a] [vgl. KE01, KE95], also

$$\omega V_i^t.$$

225 **3.2.2 Kognitive Komponente**

226 Die kognitive Komponente gibt eine Gewichtung des persönlichen Bestwerts an [Abb. 6b].
227 Dabei gewichtet jedes Partikel des Schwarms den eigenen Bestwert individuell. Dies wird
228 mit Hilfe eines Richtungsvektors und zweier Gewichte realisiert. Der Richtungsvektor
229 $P_{pbest(i)^t} - P_i^t$ bildet einen Vektor, welcher vom Partikel in Richtung des persönlichen
230 Bestwertes zeigt. c_1 entspricht der kognitiven Gewichtung und ist für den gesamten
231 Schwarm identisch. r_1 ist ein Zufallsfaktor, welcher für jedes Partikel individuell ist. Die
232 zwei Gewichte werden auf den Richtungsvektor aufmultipliziert und bilden somit einen
233 Skalar. [vgl. KE01, ZWJ15] Daraus folgt

$$c_1 r_1 (P_{pbest(i)^t} - P_i^t).$$

234 **3.2.3 Soziale Komponente**

235 Die soziale Komponente ist die Gewichtung des globalen Bestwertes [Abb. 6c]. Wie die
236 kognitive Komponente aus Abschnitt 3.2.2 setzt sich die soziale Komponente aus einem
237 Richtungsvektor und zwei Gewichten zusammen. Der Richtungsvektor entspricht aber
238 $P_{gbest^t}^t - P_i^t$ und ist somit ein Vektor, welcher vom Partikel aus in Richtung des globalen
239 Bestwertes zeigt. Das erste Gewicht c_2 ist für den gesamten Schwarm gleich gewählt. r_2
240 ist wieder ein Zufallsfaktor. Die Zusammensetzung dieser Faktoren bildet den skalierten
241 Richtungsvektor [vgl. KE01, ZWJ15]

$$c_2 r_2 (P_{gbest^t}^t - P_i^t).$$

242 **3.2.4 Positionsbestimmung**

243 Die Summe der drei Komponenten der Abschnitte 3.2.1 bis 3.2.3 bilden den Bewegungs-
244 vektor

$$V_i^{t+1} = \omega V_i^t + c_1 r_1 (P_{pbest(i)^t} - P_i^t) + c_2 r_2 (P_{gbest^t}^t - P_i^t)$$

245 welcher die nächste Position P_i^{t+1} eines jeden Partikels bestimmt [Abb. 6d]. Die nächste
246 Position ist die Summe aus der aktuellen Position und dem Bewegungsvektor [vgl. KE01,
247 ZWJ15]

$$P_i^{t+1} = P_i^t + V_i^{t+1}.$$

248 **3.3 Anwendung des Algorithmus**

249 Diesen Algorithmus kann man nun auf einen ganzen Schwarm anwenden. Dabei wird
250 zuerst der Suchraum definiert. Meist ist dies eine Funktion höherer Dimension. Danach
251 wird die gewollte Menge an Partikeln ($i \in \{0, 1, \dots, N\} \quad N \in \mathbb{N}$) zufällig im Suchraum

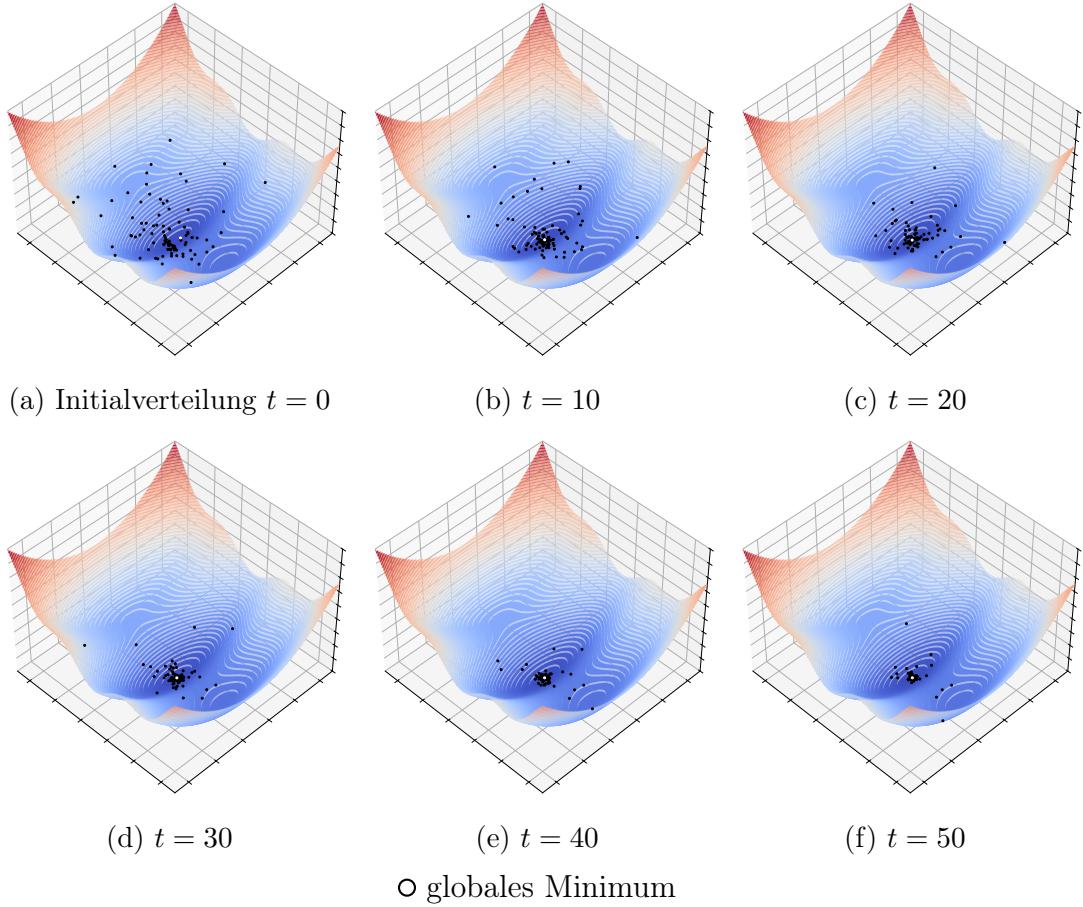


Abbildung 7: Beispiel mit 100 Partikeln

verteilt. Jedes Partikel wird dabei mit den Werten c_1, r_1, c_2 und r_2 ausgestattet. Im Anschluss wird der globale Bestwert des Schwarmes bestimmt. Dabei wird die beste Position der Partikel verwendet, welche bei der Suche nach einem Minimum der „kleinste“ Wert ist. Jedes Partikel setzt zusätzlich die aktuelle Position als persönlichen Bestwert fest. Nachdem der Schwarm initialisiert ist, wird die erste Iteration ausgeführt. Dabei wird iterativ jedes Partikel bewegt. Jedes Partikel berechnet also seinen Bewegungsvektor und bewegt sich mit diesem. Nach der Berechnung der nächsten Position wird eventuell der persönliche Bestwert neu gesetzt. Nachdem jedes Partikel die neue Position bestimmt hat, wird der globale Bestwert neu gesetzt. Dabei wird die Position jedes Partikels mit dem globalen Bestwert abgeglichen und eventuell aktualisiert. Diese Schritte werden so lange wiederholt, bis die Abbruchbedingung erreicht ist. [vgl. KE01]

Abb. 7 zeigt einen solchen Vorgang. Abb. 7a stellt dabei den ersten Schritt, die Initialisierung, dar. In den darauffolgenden Schritten Abb. 7b bis 7f werden Iterationen ausgeführt. Nach 50 Iterationen befindet sich der Schwarm bereits gut sichtbar beim Minimum des definierten Suchraums. Es befinden sich aber auch drei Partikel im lokalen Minimum neben dem globalen Minimum. Diese Partikel gewichten den persönlichen

268 Bestwert mehr als den globalen Bestwert und sind somit im lokalen Minimum „gefan-
269 gen“. Die Hürde zwischen dem lokalen und globalen Minimum ist für die Partikel nicht
270 überwindbar, wodurch sie in diesem lokalen Minimum bleiben.

271 3.4 Code

272 Der vollständige Quellcode, mit dem Abb. 6 und Abb. 7 erzeugt wurden, ist auf <https://github.com/Kneidl118/PSO> zu finden. Der Code ist in Python geschrieben und wurde
273 eigens von Andreas Auer für dieses Projekt entwickelt, um die theoretischen Konzepte
274 visuell darzustellen. Dabei werden die Formeln aus Abschnitt 3.2.4 als Basis verwendet.
275 Der Code ist unter der MIT-Lizenz und steht zur freien Verfügung.

277 3.5 Vor- und Nachteile

278 Die Partikelschwarmoptimierung stellt ein mächtiges Optimierungswerkzeug dar, das vor
279 allem durch seine Robustheit bei komplexen, mehrdimensionalen Suchräumen glänzt.
280 Der Algorithmus ist trotzdem simpel und einfach zu implementieren, da er wenige Para-
281 meter besitzt. Trotz der wenigen Parameter reagiert der Algorithmus sehr empfindlich
282 und kann bei falscher Einstellung zu schlechten Leistungen führen. Ebenso besteht das
283 Problem, die gesuchte globale Extremstelle nicht zu finden und stattdessen eine lokale
284 Extremstelle zu präferieren, wie es in Abschnitt 3.3 beschrieben ist.

285 4 Fazit

286 Das Konzept der Schwarmintelligenz zeigt wie mit einfachen, lokal befolgten Regeln
287 komplexe Probleme effizient gelöst werden können. Dies ist eine der zentralen Lektionen
288 aus der Natur, die sich in der Informatik nutzen lassen. Die vorgestellten Algorithmen
289 demonstrieren, wie eine Umsetzung in der Praxis aussehen kann. Der Ant Colony Opti-
290 mization (ACO) Algorithmus eignet sich dabei besonders gut für diskrete Optimierungs-
291 probleme, wie beispielsweise dem Suchen der kürzesten Route bei der Routenplanung.
292 Der Particle Swarm Optimization (PSO) Algorithmus hingegen eignet sich für konti-
293 nuierliche Optimierungsprobleme wie beispielsweise die Anpassung einer Route an die
294 aktuellen Verkehrsverhältnisse.

295 Abbildungsverzeichnis

<small>296</small> 1	Entwicklung der Pheromonverteilung in der Natur [vgl. Eng07]	2
<small>297</small> 2	TSP Beispielgraph	3
<small>298</small> 3	Greedy Algorithmus Lösung	4

299	4	Wegfindung anhand von Wahrscheinlichkeiten und Zufallswerten	7
300	5	Verdunstung und Verstärkung der Kanten	8
301	6	Berechnung des Bewegungsvektors	10
302	7	Beispiel mit 100 Partikeln	12

303 Literatur

- 304 [DS04] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.
- 305 [Eng07] A. P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley, 2
306 edition, 2007.
- 307 [KE95] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Pro-
308 ceedings of ICNN'95 - International Conference on Neural Networks*, pages
309 1942–1948. IEEE, 1995.
- 310 [KE01] J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.
- 311 [VR17] M. Von Rimscha. *Algorithmen kompakt und verständlich*. Springer Verlag, 4
312 edition, 2017.
- 313 [ZWJ15] Y. Zhang, S. Wang, and G. Ji. A comprehensive survey on particle swarm
314 optimization algorithm and its applications. *Mathematical Problems in Engi-
315 neering*, 2015.