CARL VON OSSIETZKY universität OLDENBURG

Von der Natur lernen – Optimierungsverfahren der Informatik

Dr. Ute Vogel
Universität Oldenburg
Department für Informatik
Umweltinformatik

Motivation

- Starre Strukturen – Selbstorganisation
- Programmierung solcher selbst organisierender Prozesse

→ agentenbasierte Programmierung (Multiagentensysteme)

  - am Beispiel der Sprache NetLogo

- Einsatz solcher selbstorganisierender Methoden zur Lösung schwerer Probleme

  - Wie beschreiben Informatiker die Komplexität von Problemen?

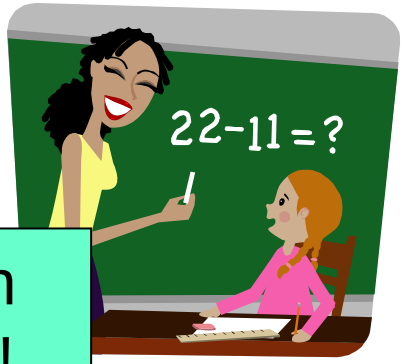
→ Ant Colony Algorithmen

Organisation – klare Sache!

Kultusminister
Chef

Lehrer
(Unterchef)

Lehrer
(Unterchef)



Schüler

Die baden
alles aus!

CARL VON OSSIETZKY universität OLDENBURG

Aussicht eines Fisches im Fischschwarm

Selbstorganisation

- Schwarm verhält sich wie eine Einheit:
  emergentes Verhalten ergibt sich aus Verhalten der einzelnen Fische.

Wie kann das funktionieren?

- Alle Fische sind gleich –

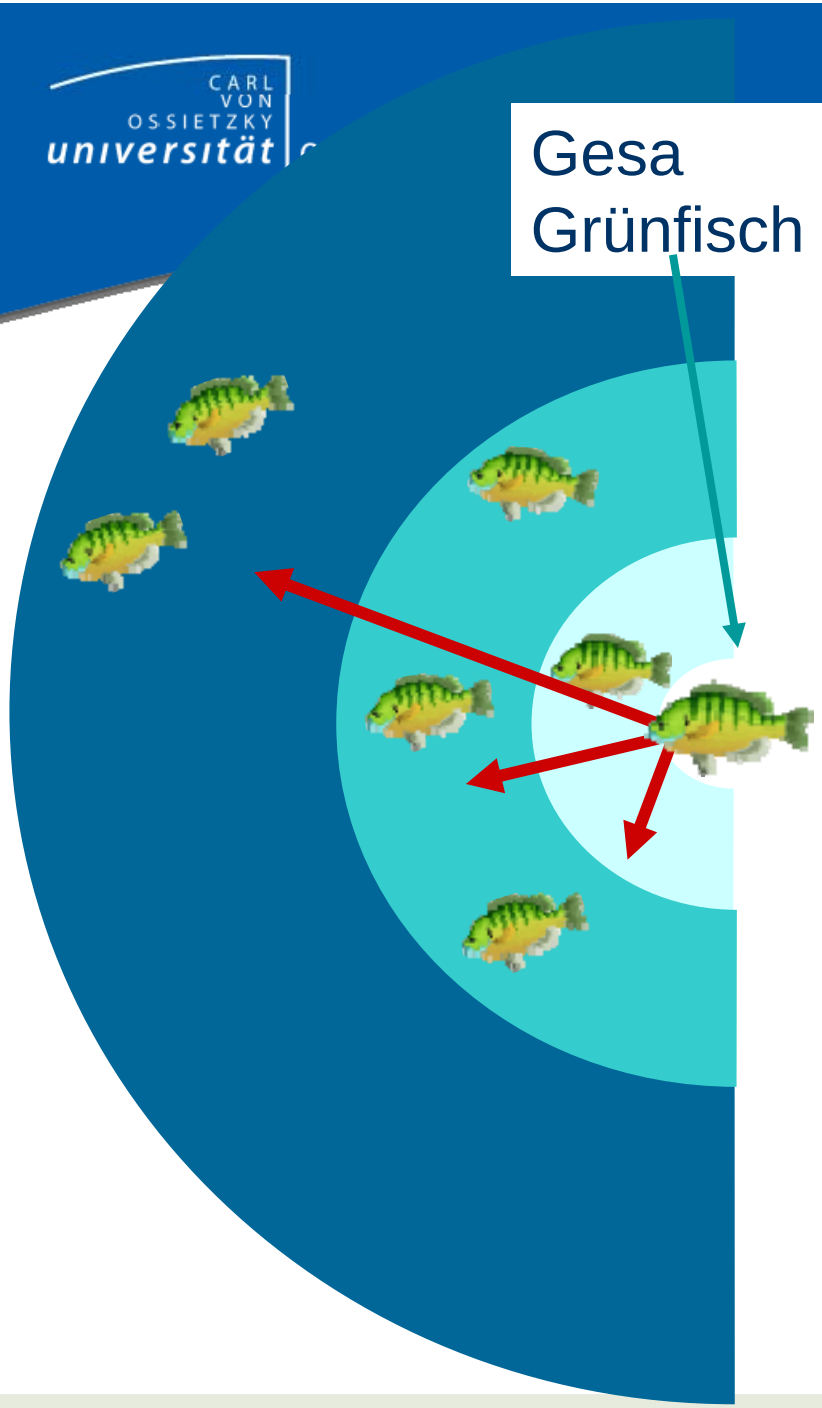
  → Keiner kann den anderen Befehle erteilen!
- Fische müssen sich selbst organisieren,
  indem sie sich an ihren Nachbarn orientieren.

CARL VON OSSIETZKY universität OLDENBURG

Was müssen Fische im Schwarm beachten?

- Fische dürfen nicht zusammenstoßen.
- Fische im Schwarm müssen gleich schnell und in die gleiche Richtung schwimmen.
- Fische wollen einen möglichst großen Schwarm bilden.

CARL VON OSSIETZKY universität



Organisation eines Fischschwarms

Regeln für jeden Fisch

1. Sehr nahen Fischen *weiche* ich möglichst gut *aus.*
2. Mit mittelnahen Fischen will ich *mithalten*: Ich schwimme in dieselbe Richtung.
3. Auf weiter entfernte Fische *schwimme* ich *zu.*

Stimmt das eigentlich?
Validierung des Modells

Wir überprüfen das Modellverhalten mit einem Computer:

- Programmiere Fische mit „Schwarmregeln“
- Beobachte, ob die programmierten Fische sich als Schwarm organisieren

Realistisches Computermodell

Computerprogramm

- Erzeuge Fisch“-Programme im Computer
- Für alle Fische:
  - Zeichne Fisch auf den Bildschirm
  - Lasse Fisch etwas „schwimmen“
  - Berechne neue Schwimmrichtung gemäß der drei Regeln
- bis Halt-Knopf gedrückt wird

Wie „baut“ man ein solches Modell?

- Es gibt viele Arten von Programmiersprachen
- Programmierparadigmen
  - Imperativ
  - Deklarativ
  - Objekt-orientiert
  - Agentenbasiert
  - Logik-orientiert
  - Funktional
  - …

Die Sprache NetLogo

- Agentenorientierte Programmiersprache
  - Einigermaßen leicht zu lernen,
  - Frei verfügbar: http://ccl.northwestern.edu/netlogo/
  - Aber natürlich: fast alles auf Englisch
  - Viele vordefinierte, natürlich-sprachlich wirkende Funktionen
  - Viele Beispiele
- Vorgängersprache Logo:
  - Kleine Schildkröte Turtle, die mit einfachen Befehlen programmiert werden kann.

Gibt es Schwärme auch beim Menschen?

Beispiel: Autos auf der Autobahn

Regeln

- Keine Kollision!
- Möglichst gleiches Tempo und gleicher Abstand
- Gleiche Richtung

CARL VON OSSIETZKY universität OLDENBURG

Autos als Schwarm

Ohne Abstandsregel

Mit Abstandsregel

Demo:

Netlogo-Modell *autobahn*
zu Fischschwärmen

Was ist ein „Agent“?

Ein Agent

- befindet sich in einer virtuellen Umgebung.
- nimmt seine Umgebung wahr.
- besitzt einen oder mehrere innere Zustände.
- verfügt über Verhaltensregeln.
- Kann abhängig von
  - der wahrgenommenen Umwelt,
  - den inneren Zuständen und
  - den Verhaltensregeln

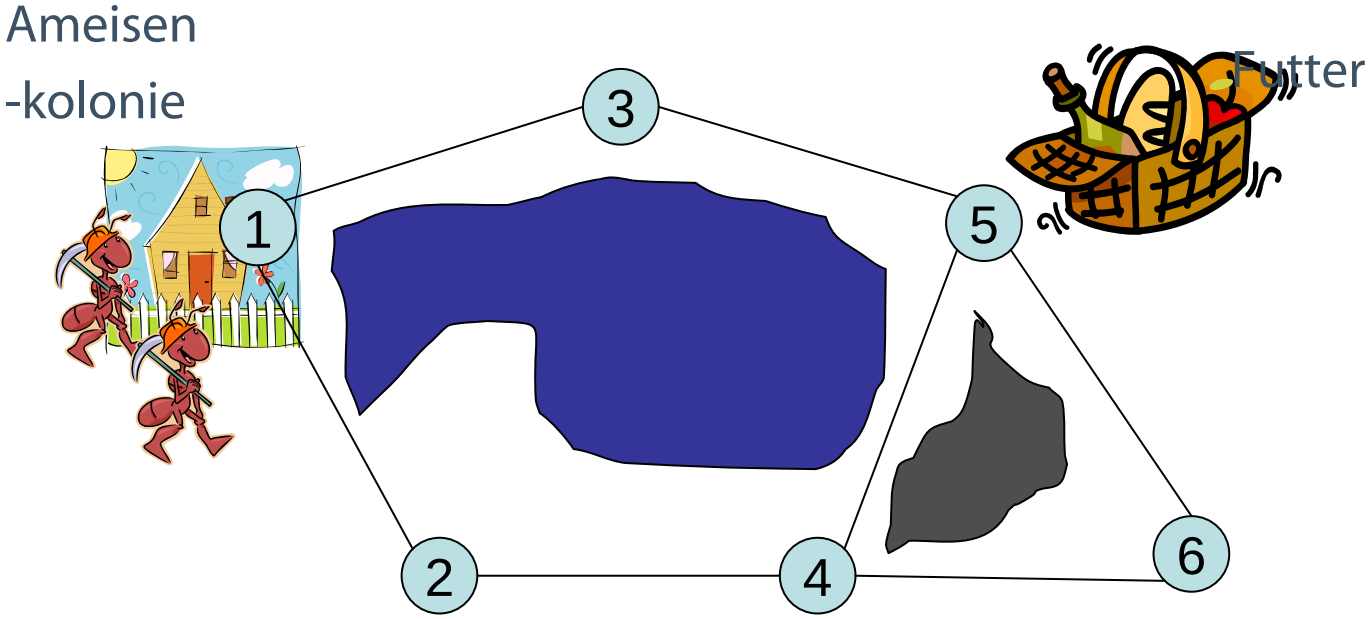
  sich selbst und seine Umgebung aktiv verändern.

CARL VON OSSIETZKY universität OLDENBURG

Lernen von Tieren: Ant Colony-Algorithmen

Wie finden Ameisen
einen möglichst kurzen Weg
vom Nest zu einer Futterquelle und
zurück?

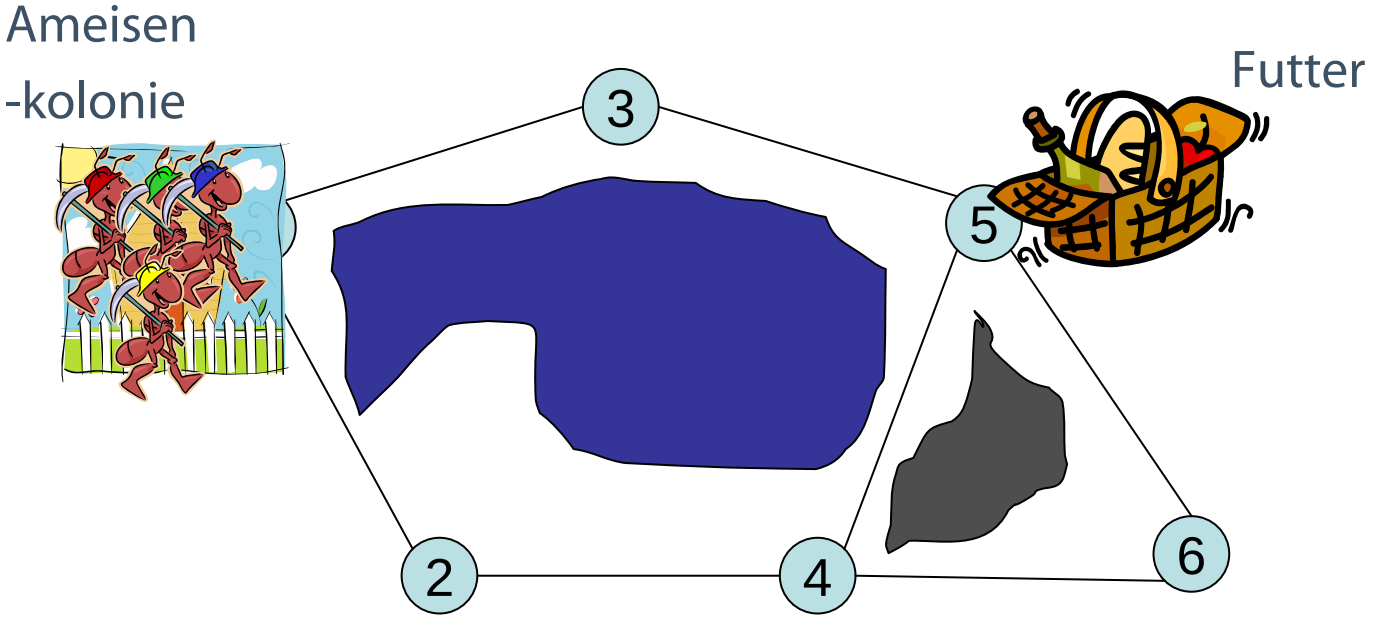
„Bau“ einer Ameisenstraße



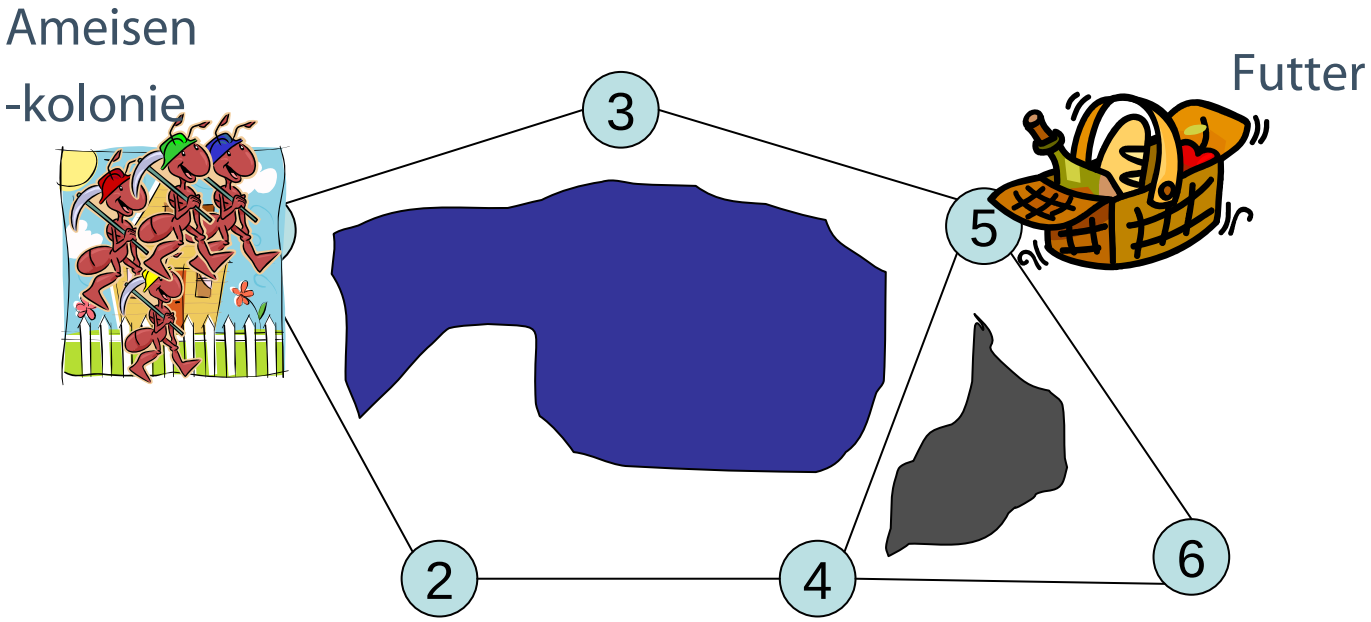
Regeln für jede Ameise

1. Hat eine Ameise eine Futterquelle gefunden, kehrt sie sofort zum Bau zurück.
2. Auf dem Rückweg legt die Ameise immer eine Duftspur, damit andere Ameisen den Weg auch finden.

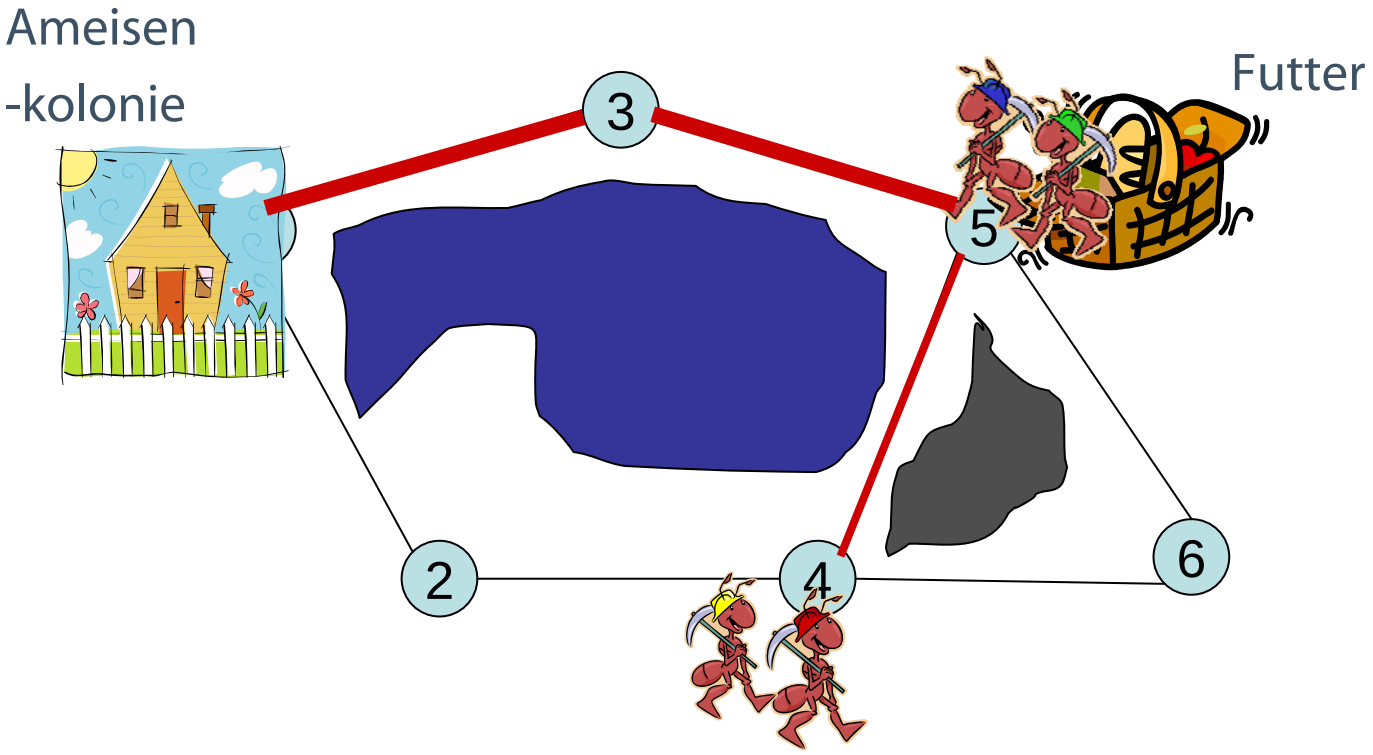
Bau einer Ameisenstraße
Auf der Suche nach Futter (1)



Bau einer Ameisenstraße
Auf der Suche nach Futter (1)



Bau einer Ameisenstraße
Auf der Suche nach Futter (1)

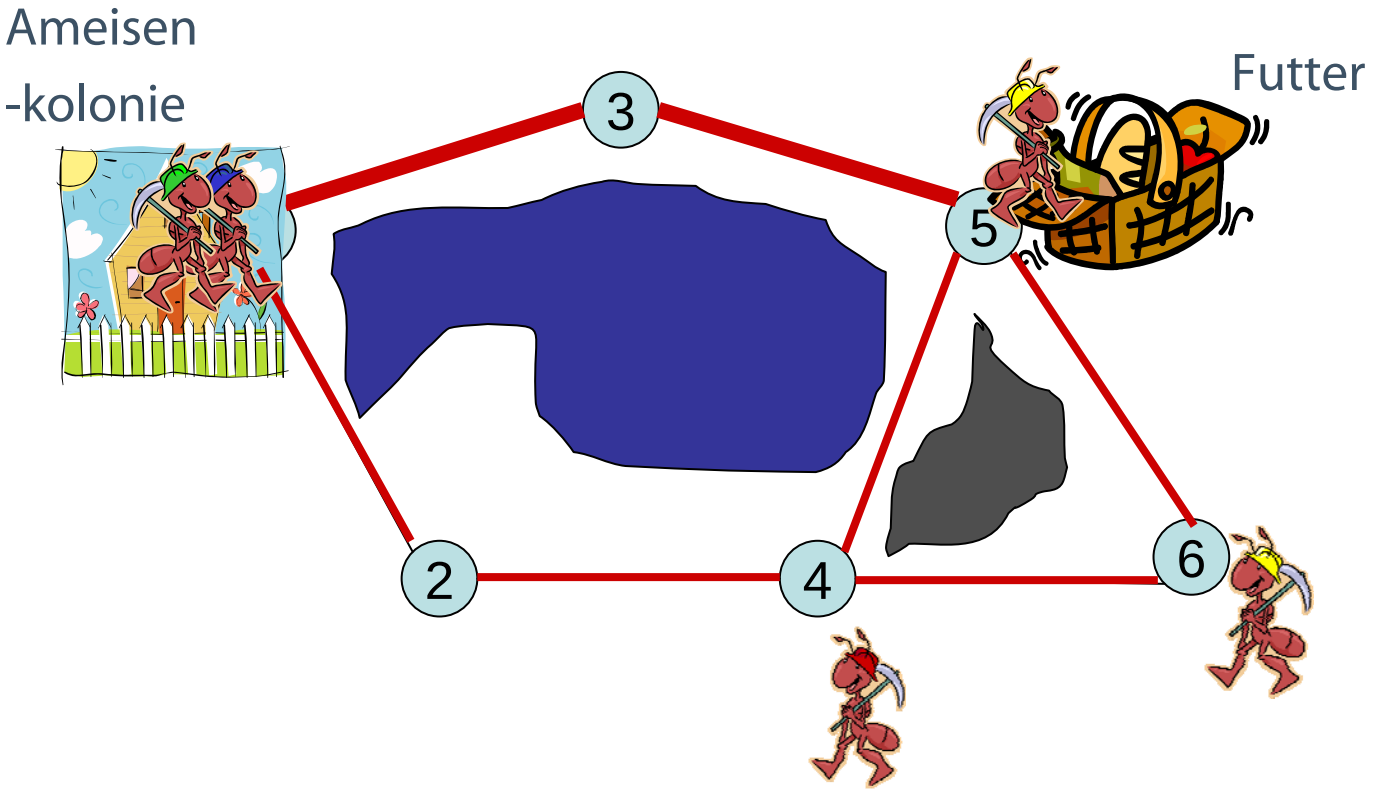


Weitere Regeln der Ameisen

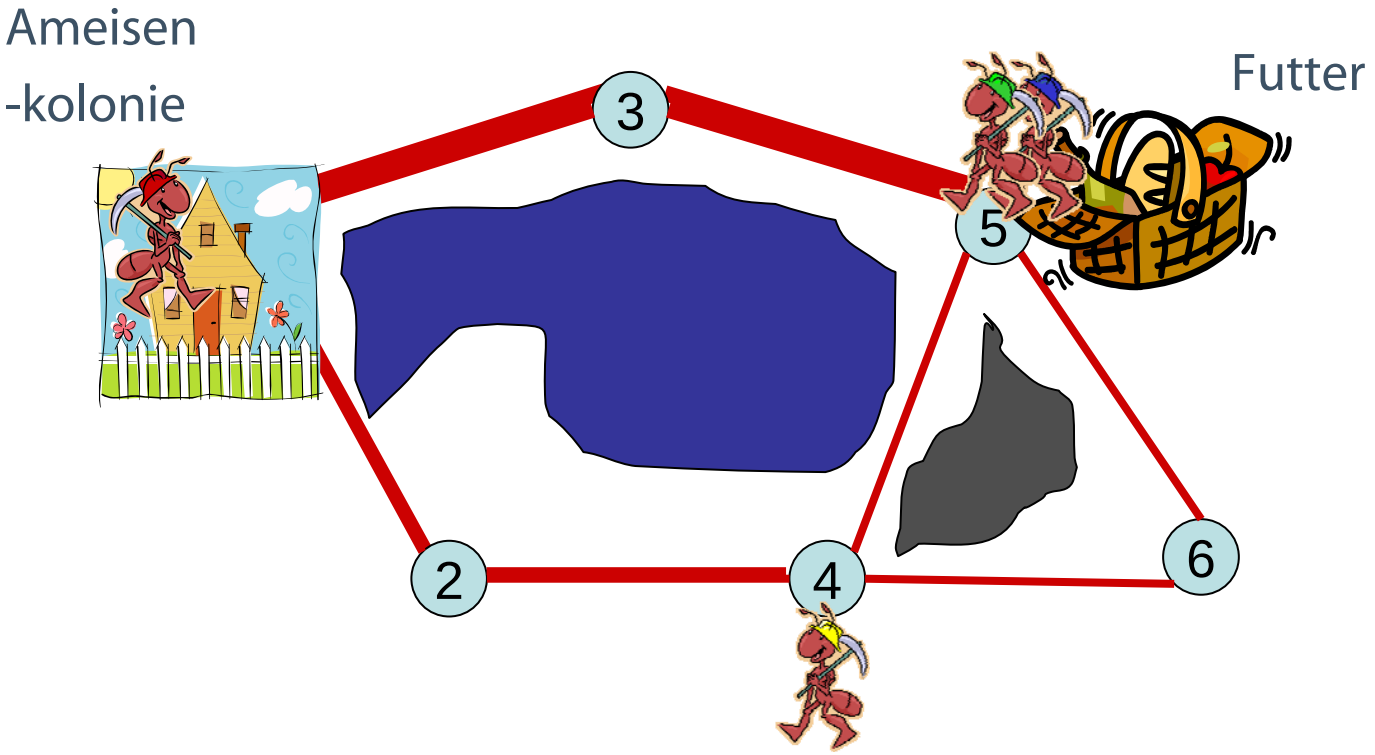
3. Eine Ameise an einer Weggabelung entscheidet sich lieber für den Weg mit stärkerer Duftspur – aber nicht immer
4. Duftspuren verblassen mit der Zeit wieder

Damit werden nach und nach kurze Wege zum Futter bevorzugt, da auf diesen eine stärkere Duftspur zu finden ist!

Bau einer Ameisenstraße
Auf der Suche nach Futter (2)



Bau einer Ameisenstraße
Auf der Suche nach Futter (2)



Computermodell

- Die Regeln scheinen recht gut zu sein, um einen kurzen Weg zu bestimmen.
- Überprüfung durch ein Computermodell mit programmierten Ameisen.
- Demo: NetLogo: Ant-Modell

Zusammenfassung des Ameisenverfahrens

0. Zu Beginn suchen die Ameisen zufällig auf allen erlaubten wegen nach Futter.
- Hat eine Ameise eine Futterquelle gefunden, kehrt sie sofort zum Bau zurück.
- Auf dem Rückweg legt die Ameise immer eine Duftspur, damit andere Ameisen den Weg auch finden.
- Eine Ameise an einer Weggabelung entscheidet sich lieber für den Weg mit stärkerer Duftspur – sie können sich auch für einen Weg mit schwacher Duftspur entscheiden.
- Duftspuren verblassen mit der Zeit wieder

ACO-Anwendung auf andere Probleme?

- Lassen sich solche „Ant Colony-Verfahren“ auf andere Optimierungsprobleme anwenden?
  - Ameisen suchen einen möglichst guten Weg.
  - Finden sie stets die beste Lösung?
    Wie lange sollen die Ameisen suchen?
    - Netlogo-Demo-Modell Biology→ Ants
    - Man kann nicht beliebig lange auf die beste Lösung warten.
  - Für welche Probleme ist es eine gute Lösung, die nicht unbedingt sie beste Lösungist, sinnvoll?
    - Für KOMPLEXE Optimierungsprobleme, in denen kein exaktes, schnelles verfahren bekannt ist.

Komplexität

- Jedes Verfahren hat eine bestimmte Komplexität, d.h. zur Lösung eines Problems benötigt es eine bestimmte Anzahl von Rechenschritten, bis die Lösung gefunden wird.
- Verschiedene Verfahren zur Lösung des gleichen Problems können unterschiedlich schnell sein.
- Beispiel:
  - Suchen einer bestimmten Information, z.B. einer bestimmten Spielkarte in einem Stapel von Spielkarten
  - → Ausprobieren

Problem: Suchen

- Aufgabe:
  Suche eine bestimmte Spielkarte in einem Stapel von n = 32 Spielkarten
- Angenommen die Spielkarten sind nicht sortiert:
  - Im schlechtesten Fall müssen alle Karten der reihe nach durchgesehen werden, d.h. es muss n mal verglichen werden, ob die aktuelle Karte die gesuchte Karte ist.
  - Im Mittel findet man die Karte nach n / 2 Vergleichen
- → Durchschnittlicher Aufwand eine Spielkarte in einem Stapel mit n Karten zu finden ist
  T(n) = n / 2.

Komplexitätsklasse

- Es ist nicht wichtig, den Aufwand „ganz genau“ abzuschätzen.
- Interessant ist:
  - Wie stark wächst die Rechenzeit, wenn man statt n = 32 Karten 2*n Karten durchsuchen muss.
  - → Dann müssen doppelt soviele, also maximal 2n Vergleiche durchgeführt werden.
  - Der Aufwand für das Suchen in einem nicht sortierten Kartenstapel wächst also linear mit der Anzahl der Karten.
  - → Linearer Aufwand

Geht es auch schneller?

- Annahme: Der Kartenstapel ist sortiert.
- Bessere Methode:
  1. Schaue die mittlere Karte an.
  2. Müsste die gesuchte Karte davor oder danach kommen?
  3. Wähle die Hälfte des Kartenstapels, in der die gesuchte Karte vorkommen muss und fange wieder bei Schritt1 an.

Wie oft werden zwei Karten miteinander verglichen?

Karten im Stapel	Geteilter Stapel	Vergleich Nr.
$N_0 = 32 = 2^n$	$N_1 = N_0 / 2 = 16$	1
$N_1 = 16$	$N_2 = N_1 / 2 = 8$	2
$N_2 = 8$	$N_3 = N_2 / 2 = 4$	3
$N_3 = 4$	$N_4 = N_3 / 2 = 2$	4
$N_4 = 2$	$N_5 = N_4 / 2 = 1$	5
$N_5 = 1$	$N_6 = N_5 / 2 = 0$	$6 = \log(n) + 1$: fertig!

Nach spätestens $\log_2(n)+1 =$ **6** Vergleichen hat man die Karte gefunden oder man weiß, dass sie nicht im Stapel enthalten ist.

Komplexitätsklassen

- Wir ordnen die Verfahren nach ihrem Aufwand in Klassen ein:
  - O (log(n)) ist die Klasse der Probleme, die (im Mittel) höchstens logarithmischen Aufwand haben,
  - O(n) ist die Klasse der Verfahren, die (im Mittel) höchstens linearen Aufwand haben,
  - O (n^2) ist die Klasse der Probleme, die (im Mittel) höchstens quadratischen Aufwand haben,
  - usw.

  wenn sie auf einen Kartenstapen (=„ein Problem“) der Größe n angewendet werden.

Beispiel: Sortieren

- Sortierverfahren
  - Entwickle eine Methode, mit der man eine Menge von Karten (schnell) sortieren kann
  - Idee:
    - Suche die kleinste Karte im Stapel der unsortierten Karten
    - Lege sie (umgedreht) auf den (zu Anfang leeren) Stapel der sortierten Karten

    Bis im unsortierten Stapel keine Karte mehr enthalten ist.
- Welchen Aufwand hat dies Sortierverfahren?

Aufwandsberechnung

- Beispiel: Seien n = 5 Karten gegeben
  - 4 Vergleiche um die erste (kleinste) Karte zu finden
  - 3 Vergleiche um die zweit-kleinste Karte zu finden
  - 2 Vergleiche um die dritt-kleinste Karte zu finden
  - 1 Vergleich um die viert-kleinste Karte zu finden
  - Letzte Karte ist die größte
  - Summe: 4+3+2+1= 10 Vergleiche
- Allgemein:
  - Gegeben n unsortierte Karten:
    - (n-1) + (n-2) + (n-3)+ ..+ 1 Vergleiche
    - = n * (n-1) /2 Vergleiche
    - = $(n^2 + n)/2$ aus O (n^2)
  - → quadratischer Aufwand

Sortieren

- Geht auch noch besser als mit quadratisch wachsendem Aufwand….

- Aber, welches sind nun die Probleme, bei denen sich ein Verfahren wie ACO lohnt?
  - Probleme, bei denen eine „beste“ Lösung
  - Aus einer großen Menge von erlaubten Lösungen herausgesucht werden muss.
  - Es ist kein Verfahren bekannt, dessen Aufwand zum Durchsuchen der Menge der erlaubten Lösungen sich durch ein Polynom abschätzen lässt.

Beispiel

- Gegeben seien n Karten und die Frage: „In welcher Reihenfolge sollten die Karten ausgespielt werden?“
- Zu lösen:
  - Wie viele mögliche Reihenfolgen gibt es?
  - Welche Reihenfolge ist die beste?+
- Anzahl der Reihenfolgen bei n = 4 Karten:
  - 4 Möglichkeiten für die erste Karte:
  - * 3 Möglichkeiten für die zweite Karte:
  - * 2 Möglichkeiten für die dritte Karte:
  - * 1 Möglichkeit für die letzte Karte:
  - Gesamt: 4*3*2*1 = 4! = 24 Möglichkeiten

Bei 4 Karten müssten also von 24 möglichen Reihenfolgen die beste gefunden werden.

Mit der Problemgröße wachsender Aufwand

Anzahl Karten	Suchen einer Karte		Sortieren		Mögliche Reihenfolgen
	Sortiert	Unsortiert	Quicksort	Auswählen	
2	1	1	2	1	**2**
3	2	2	5	4	**6**
4	2	3	8	9	**24**
5	3	4	12	16	**120**
6	3	5	16	25	**720**
7	3	6	20	36	**5040**
8	3	7	24	49	**40320**
9	4	8	29	64	**362880**
10	4	9	34	81	**3628800**
11	4	10	39	100	**39916800**
12	4	11	44	121	**479001600**
13	4	12	49	144	**6227020800**

Rechner sind doch schnell… - aber nicht schnell genug

- Angenommen, dein Rechner kann pro Sekunde 1000 Lösungen darauf überprüfen, wie gut sie sind (und sich die beste merken).
- Dann benötigt er für
  - 10 Karten nur 1 Stunde
  - 13 Karten immerhin schon 72 Tage
  - 20 Karten unvorstellbare 77 Millionen Jahre

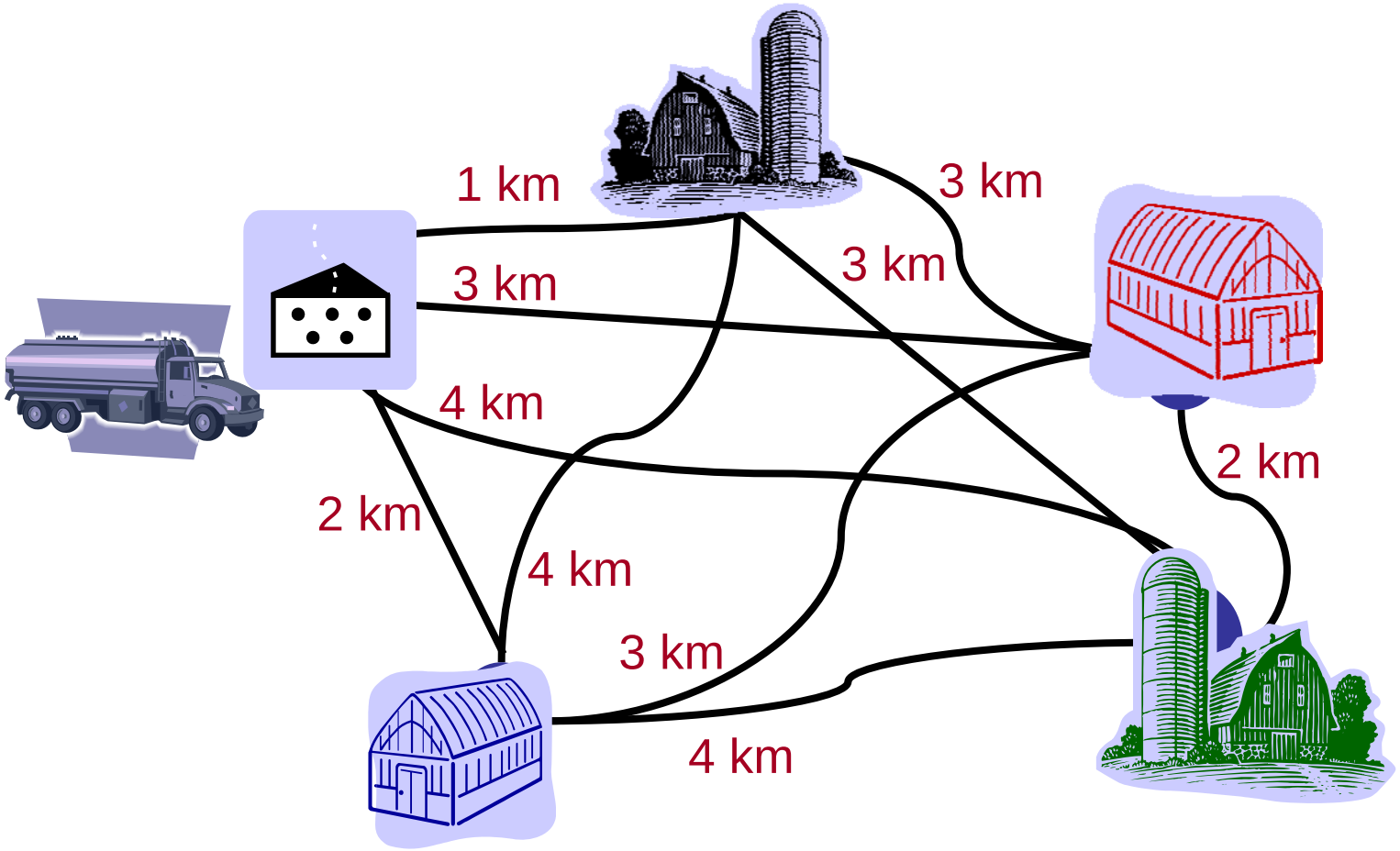
Komplexität

- Die Anzahl der möglichen Reihenfolgen wächst mit der Anzahl n (von Karten) exponentiell an.
- Bestimmung der besten Reihenfolge schwierig, da sehr viele Lösungen bestimmt und bewertet werden müssen.

Relevanteres Beispiel

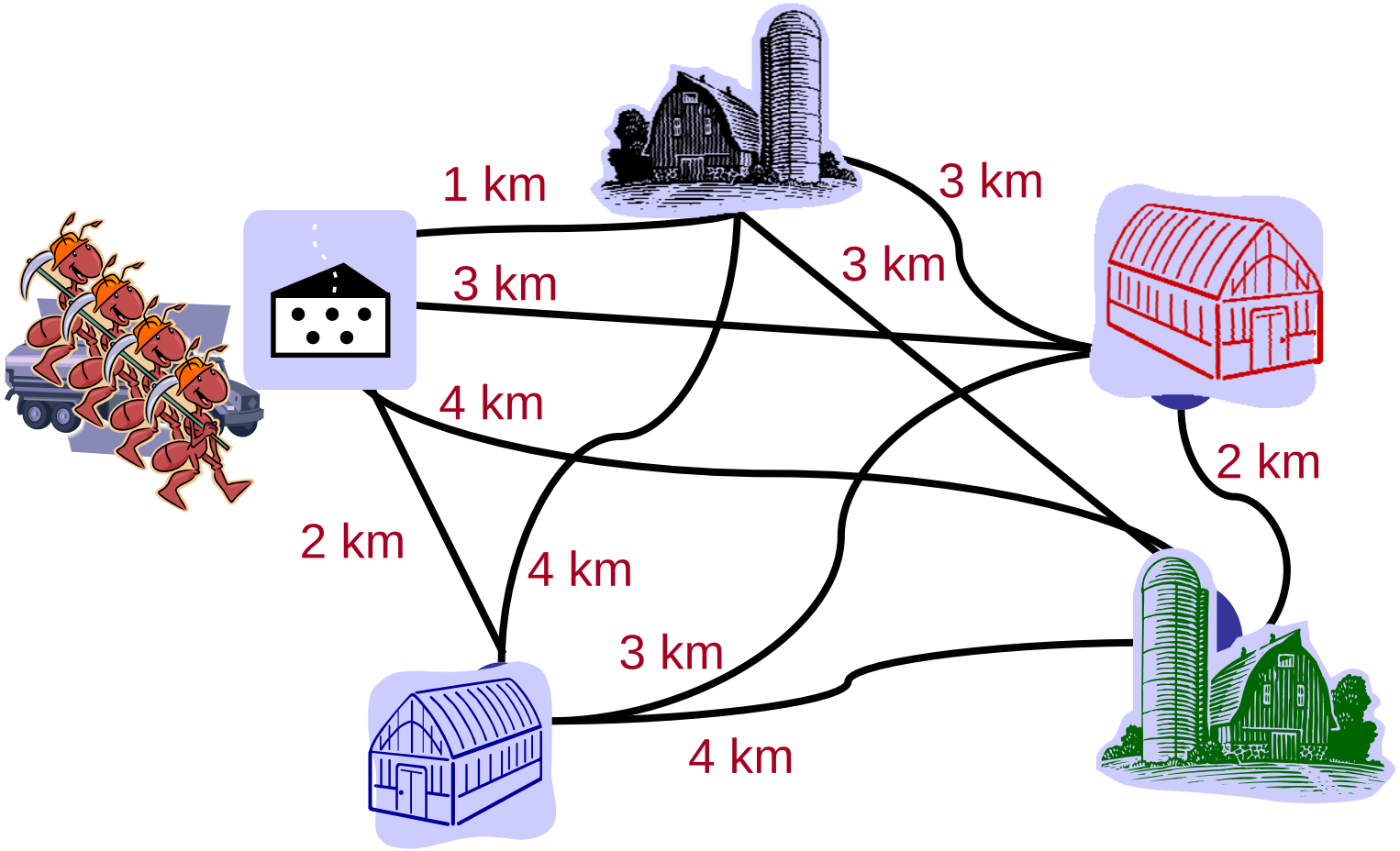
- Travelling Salesman Problem:
  - Ein sehr berühmtes Problem aus der Informatik
- Finde die kürzeste Rundreise, in der ein Handelsvertreter alle Städte besuchen kann.
  - Start und Ende an Ort X
  - Besuche alle anderen Städte, so dass am Ende möglichst wenig km gefahren werden

Beispiel
Rundfahrt eines Milch-LKW



Modellbildung

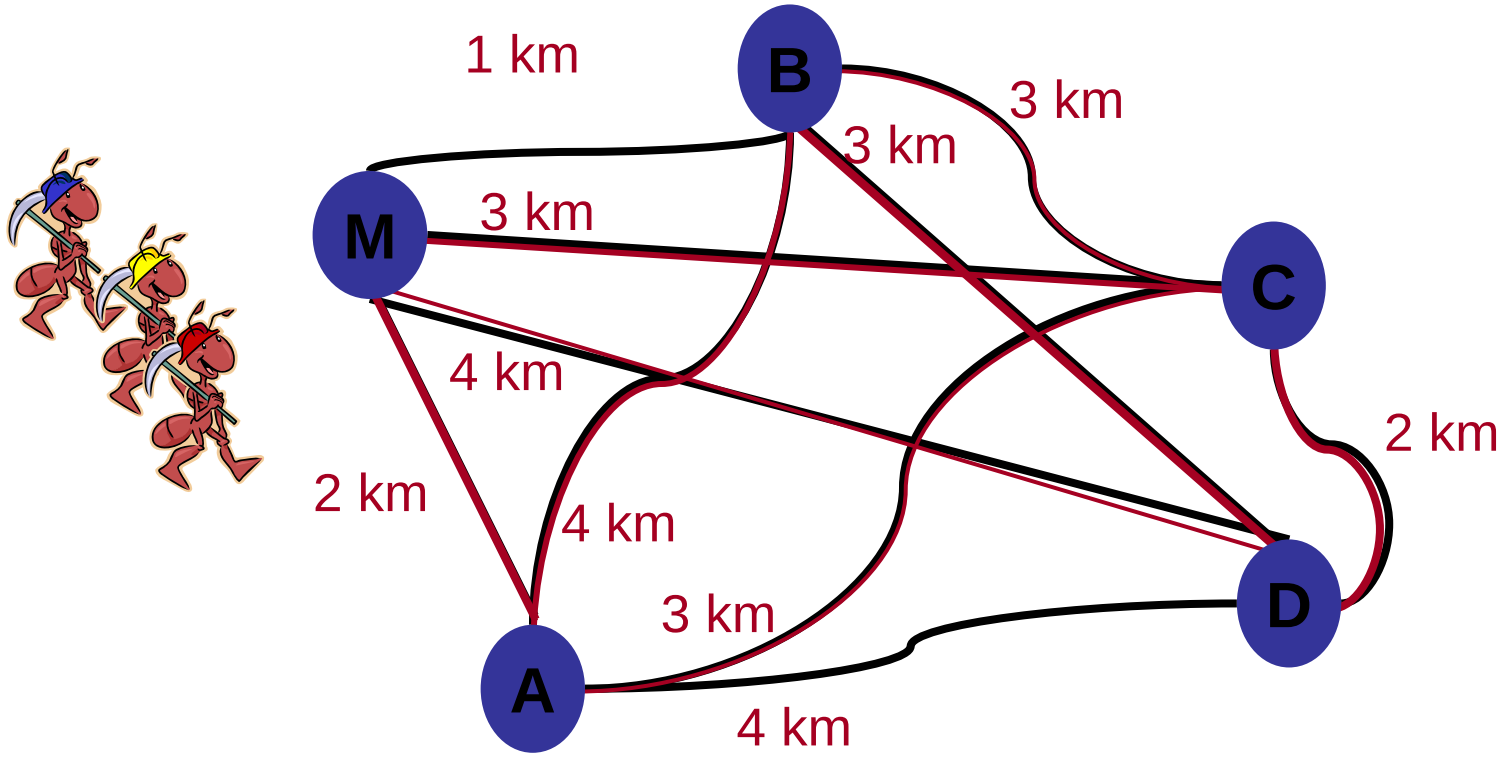
Vom Problem zum Modell



Zusätzliche Ameisenregeln

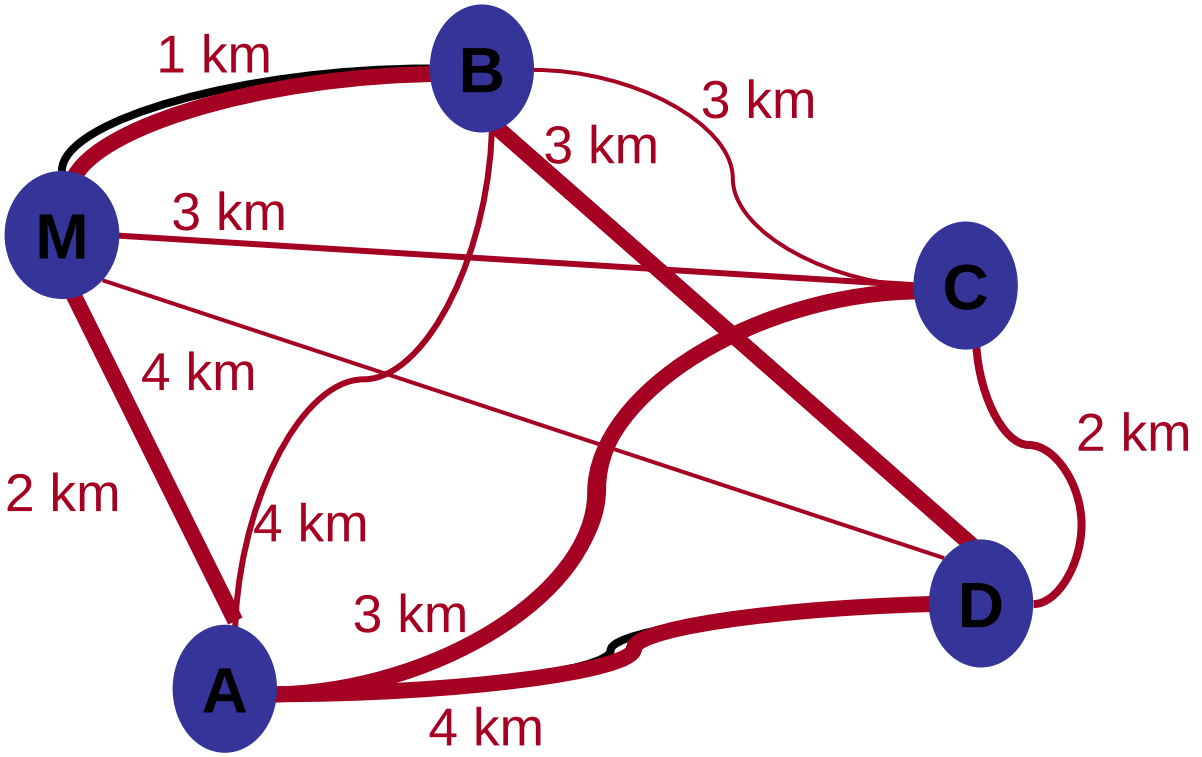
5. Jede Ameise besucht alle Höfe, ehe sie umkehrt.
6. Keine Ameise besucht einen Hof doppelt, ehe sie umkehrt.
7. Jede Ameise markiert den Rückweg um so stärker, je kürzer der Weg ist, den sie gelaufen ist.

Programm-Ameisen laufen los …



MABDC: 14 km MDBCA: 15 km

Programm-Ameisen laufen weiter …



MABDC: 14 km

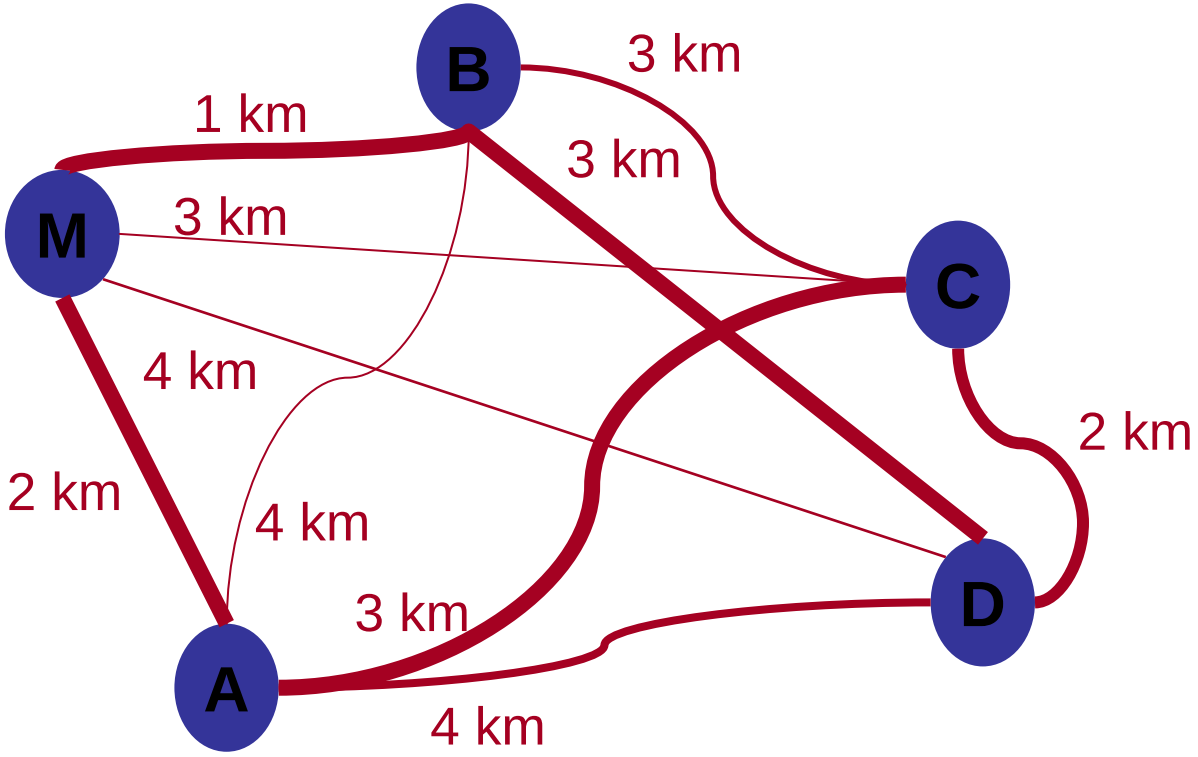
MDBCA: 15 km

MBDAC: 14 km

Nach dem Markieren …

Weg mit den stärksten Markierungen:

MACDBM: 11 km !



CARL VON OSSIETZKY universität OLDENBURG

Demo Netlogo Ant Modell

TSPAnts.nlogo

CARL VON OSSIETZKY universität OLDENBURG

Unsere Anwendung

- Elektrifizierung einer ländlichen Region
  - Optimale Auswahl von Kraftwerkstypen
    - Erneuerbare contra konventionelle
  - Optimale Wahl von Kraftwerksstandorten
  - Optimale Wahl des Netzes
- Minimiere die Gesamtkosten der Bereitstellung von Strom

Energieversorgungsnetze

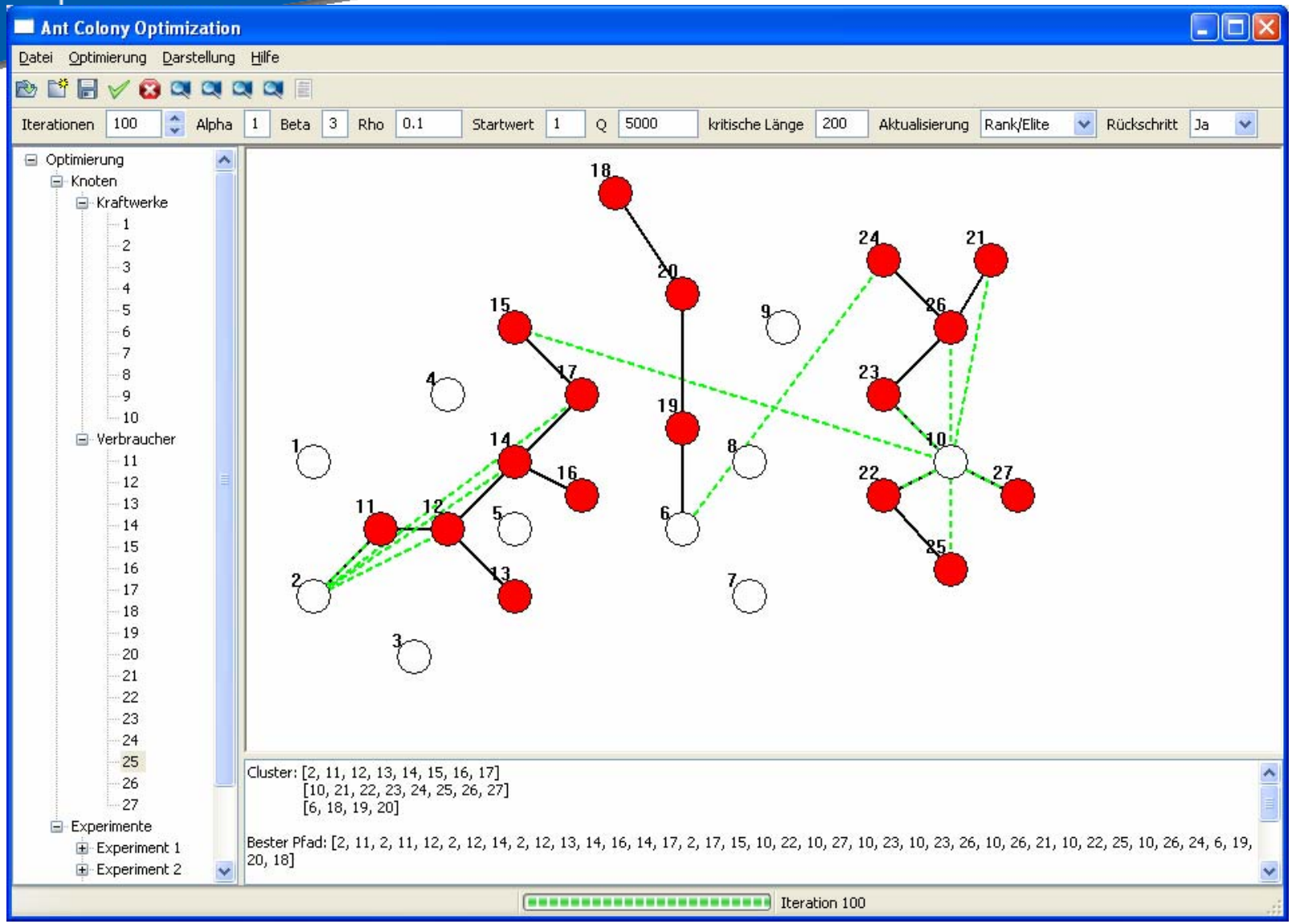
- Energieversorgungsnetze bestehen (sehr stark vereinfacht) aus
  - Konsumenten (“Dörfer”)
    - Vorhersehbarer Verbrauch muss bereitgestellt werden
  - Kraftwerken
    - Feste maximale Kapazität
  - Leitungen zwischen Kraftwerken und Verbrauchern
- Verbindungsnetz in der Ameisen-Optimierung
  - alle Konsumenten und
  - zunächst alle möglichen Kraftwerke und Leitungen.

CARL VON OSSIETZKY universität OLDENBURG

Ameisen müssen „schlauer“ werden …

- Ameisen
  - sammeln Energie bei besuchten Kraftwerken und
  - ordnen sie besuchten Konsumenten zu
  - müssen hierzu auch verzweigte Wege gehen können
- Zur Beurteilung der Qualität “ihres” Netzes summiert jede Ameise die Kosten
  - der besuchten Kraftwerke
  - der gebauten Verbindungen

→ Auswahl möglichst guter Kraftwerke und Verbindungen


Ant Colony Optimization
Datei Optimierung Darstellung Hilfe
Iterationen 100 Alpha 1 Beta 3 Rho 0.1 Startwert 1 Q 5000 kritische Länge 200 Aktualisierung Rank/Elite Rückschritt Ja
Optimierung
Knoten
Kraftwerke
1 2 3 4 5 6 7 8 9 10
Verbraucher
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
Experimente
Experiment 1
Experiment 2
Cluster: [2, 11, 12, 13, 14, 15, 16, 17]
[10, 21, 22, 23, 24, 25, 26, 27]
[6, 18, 19, 20]
Bester Pfad: [2, 11, 2, 11, 12, 2, 12, 14, 2, 12, 13, 14, 16, 14, 17, 2, 17, 15, 10, 22, 10, 27, 10, 23, 10, 23, 26, 10, 26, 21, 10, 22, 25, 10, 26, 24, 6, 19, 20, 18]
Iteration 100

Zusammenfassung

- In der Natur können sich Tiere durch einfache Regeln selbst organisieren.
  → Emergentes Verhalten
- Wir können die Regeln durch programmierte Tiere überprüfen und zur Optimierung anderer Probleme einsetzen.
- Informatiker können solche Regeln zum Beispiel in „intelligente“ Autos oder vielleicht auch in „intelligente“ Stromnetze einbauen.