Zufallsgesteuerte Algorithmen, der Natur abgeschaut

Roland Mittermeir
Institut für Informatik-Systeme
Universität Klagenfurt

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

Gliederung

- Algorithmische Komplexität und NP-vollständige Probleme
- *Optimal* oder *gut genug* – *Heuristiken* und *Metaheuristiken*
- *Swarm Intelligence* und *Stigmerie*
- Kooperationsstrategien von Ameisen
- Wie künstliche Ameisen das *Traveling Salesman* Problem lösen

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

Algorithmische Komplexität

Frage:
Wie lange dauert es im schlechtesten Fall, bis dieser Algorithmus das Ergebnis berechnet hat?

Antwort:

- Das hängt von der Anzahl ***n*** der zu verarbeitenden Daten ab!
- Das hängt davon ab, wie oft der Algorithmus ein einzelnes Datum „anschauen“ muss.

⇒ Dauer ist eine Funktion von ***n***.

Komplexitätsklassen

- Die (maximale) Bearbeitungsdauer ist durch ein Polynom in ***n*** beschreibbar.
  ⇒ Polynomielle Komplexität:

  $C = a*n^k + b*n^{k-1} + \ldots + c*n^2 + d*n^1 + e$

  ⇒ $O(f(n)) = \mathbf{O(n^k)}$

- Behandelbare Probleme (*tractable problems*) haben niedere polynomielle Komplexität
  z.B.: kürzeste Wege … $O(n^2)$, Bubblesort … $O(n^2)$, Quicksort … (O(n*ld n)

- Nicht-behandelbare Probleme (*intractable problems*) haben exponentielle Komplexität.

  . $C = \mathbf{O(const^n)}$

  z.B.: Traveling Salesman… O(n!), Teilmengensumme (Rucksackproblem)… $O(2^i)$,

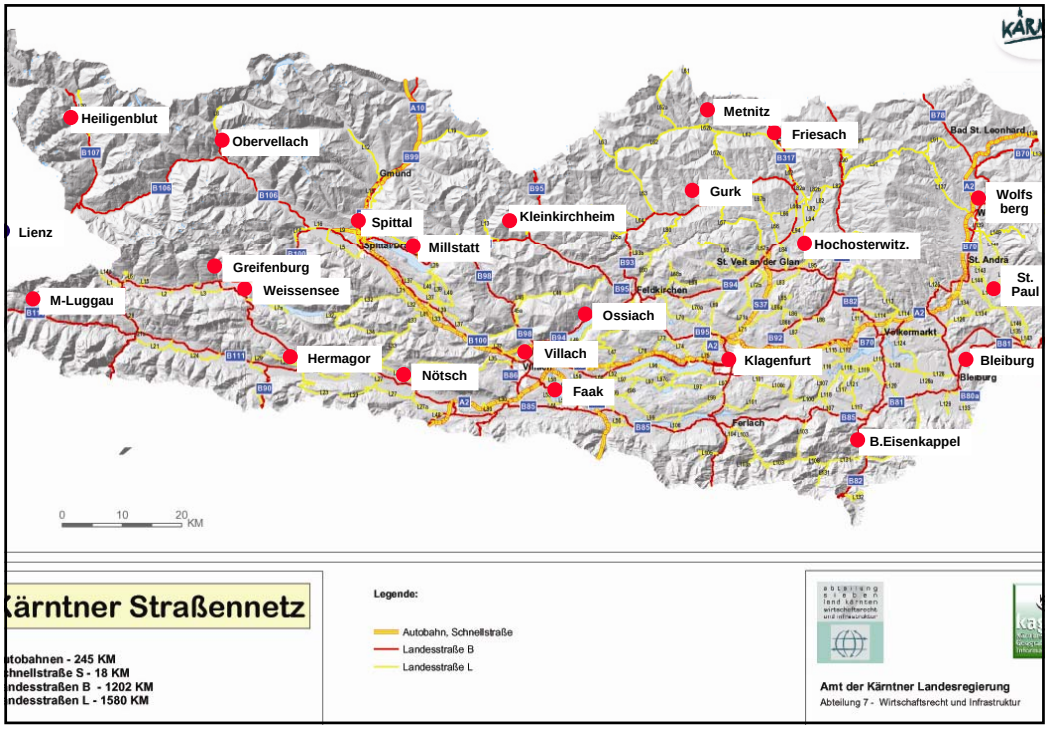
Besichtigungs-Rundreise

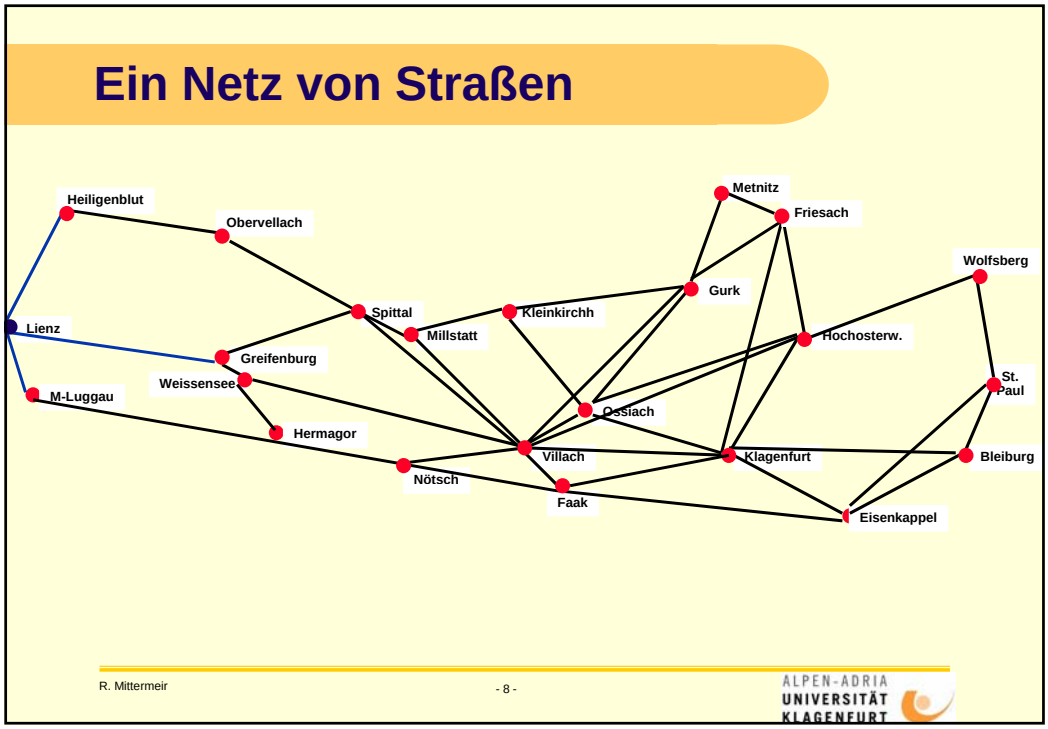
Sie wollen Ihrem Gast die Sehenswürdigkeiten Kärntens zeigen.

Dabei wollen Sie

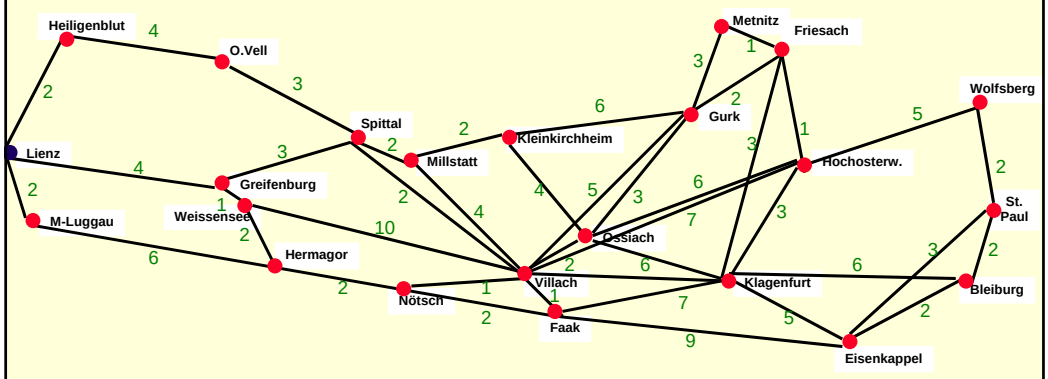
- nichts auslassen,
- nichts doppelt besuchen,
- insgesamt möglichst wenig Zeit für die Reise zwischen den einzelnen Sehenswürdigkeiten vergeuden.

In welcher Reihenfolge sollten Sie Kärntens Highlights besuchen?


Heiligenblut
Obervellach
Metnitz
Friesach
Gurk
Wolfs berg
Lienz
Spittal
Kleinkirchheim
Millstatt
Hochosterwitz.
Greifenburg
Weissensee
St. Paul
M-Luggau
Ossiach
Villach
Klagenfurt
Bleiburg
Hermagor
Nötsch
Faak
B.Eisenkappel
0 10 20 KM
Kärntner Straßennetz
Legende:
Autobahn, Schnellstraße
Landesstraße B
Landesstraße L
Amt der Kärntner Landesregierung
Abteilung 7 - Wirtschaftsrecht und Infrastruktur

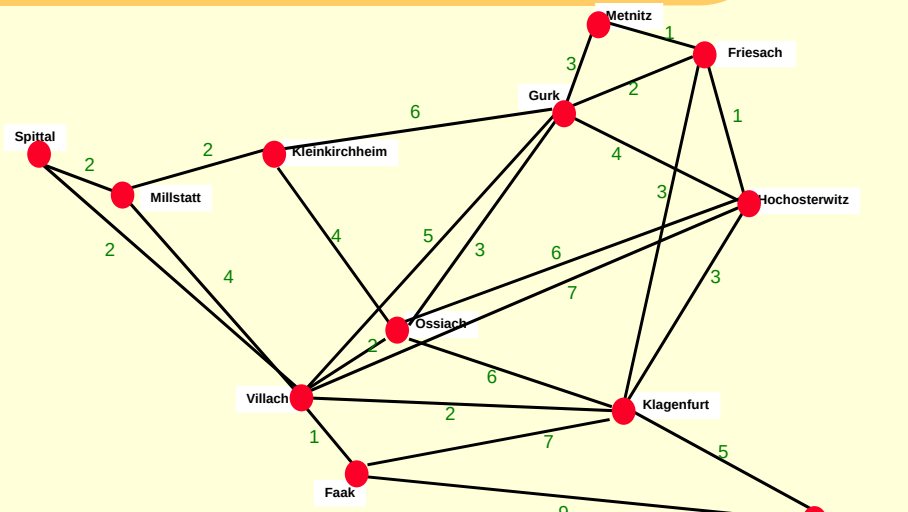

Ein Netz von Straßen
Heiligenblut
Obervellach
Metnitz
Friesach
Wolfsberg
Gurk
Spittal
Kleinkirchh
Lienz
Millstatt
Hochosterw.
Greifenburg
Weissensee
St. Paul
M-Luggau
Ossiach
Hermagor
Villach
Klagenfurt
Bleiburg
Nötsch
Faak
Eisenkappel
R. Mittermeir
- 8 -
ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

Reisedauer in Viertelstunden



Graph mit 22 Knoten (Städten) … 22! Wege wären zu berechnen!
~ 1,124 * 10^{21} Wege

Zentralkärnten



Graph mit 12 Knoten (Städten) … „nur mehr 12! Wege zu berechnen
= 479.001.600 Wege

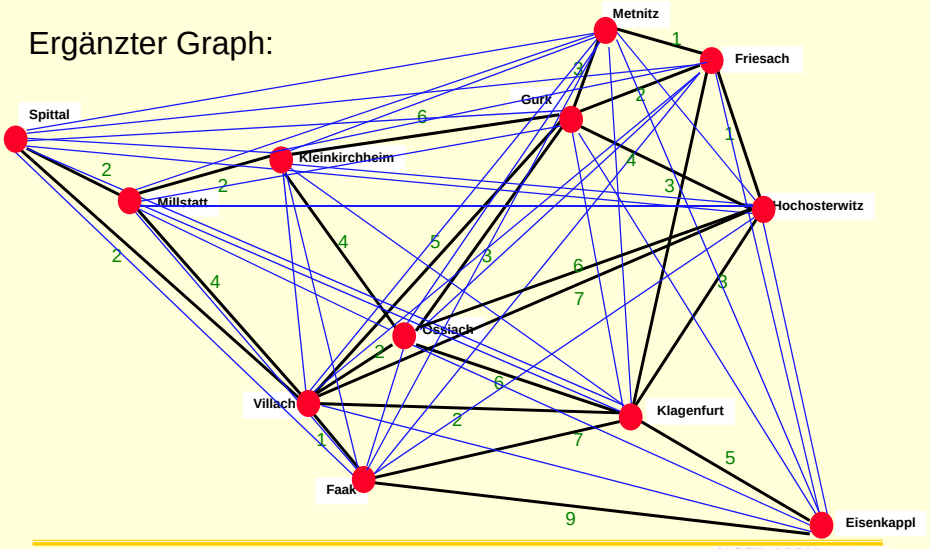
Mögliche Besichtigungsfolgen

Rundreise: (Kl, .. Liste aller anderen Städte .. , Kl)	Dauer:
(Kl, Vi, Fa, Ei, ---)	bleibe stecken
(Kl, Ho, Fr, Gu, Kk, …)	?? Metnitz verpasst !!
(Kl, Os, Ho, Fr, Me, Gu, Kk, Mi, Sp, Vi, Fa, Ei, Kl) 6 + 6 + 1 + 1 + 3 + 6 + 2 + 2 + 2 +1 + 9 + 5	Dauer: 44
(Kl, Ho, Fr, Me, Gu, Os, Kk, Mi, Sp, Vi, Fa, Ei, Kl) 3 + 1 + 1 + 3 + 3 + 4 + 2 + 2 + 2 + 1 + 9 + 5	Dauer: 36
(Kl, Ei, Fa, Vi, Sp, Mi, Kk, Os, Gu, Me, Fr, Ho, Kl) 5 + 9 + 1 + 2 + 2 + 2 + 4 + 3 + 3 + 1 + 1 + 3	Dauer: 36

verbotene Kanten/Wege

1000

Ergänzter Graph:



Kürzester Rundweg (Kreis)

- Initialisiere
  - bilde Liste aller Sehenswürdigkeiten
  - berechne die Reisedauer, nimm an sie sei die Kleinste
- Verbessere
  - berechne Permutation der Sehenswürdigkeitenliste
  - berechne zugehörige Reisedauer
  - ist diese kleiner als die bisher berechnete (kleinste) Reisedauer dann
    * merke Dir die Reihenfolge der eben berechneten Sehenswürdigkeitenliste
    * merke Dir die zugehörige Reisedauer

  solange bis alle Permutationen durchprobiert sind!

Liefert mit Sicherheit die optimale Lösung!
Aber das dauert noch immer zu lang! (12*11*10*…*3*2*1 Durchläufe).

Systematische Vorgangsweise

```
Permutiere (L, anf, end)
   von i := anf bis end mache:
        tausche (L[anf], L[i]);
        wenn anf < end
                    dann
                            Permutiere(L, anf+1, end)
                    sonst
                            gib aktuelle Lösung aus;
        tausche(L[i], L[anf]).
```

Aufruf mit Permutiere (Sehenswürdigkeitenliste, 1, 12)

Prüfe auf Zulässigkeit der Lösung

Permutiere ([A, B, C, D]; 1, 4)

```
anf  i      A  B  C  D
1    1      a  B  C  D
            permutiere([B, C, D]; 2, 4)
                    anf  i      B  C  D
                    2    2      b  C  D
                                permutiere [C, D]; 3, 4)
                                        anf  i      B  C  D
                                        3    3      c  D
                                                    permutiere [D]; 4, 4)      A  B  C  D
                                        3    4      d  c
                                                    permutiere [C]; 4, 4)      A  B  D  C
                    2    3      c  b  D
                                permutiere [b, D]; 3, 4)
                                        3    3      b  D
                                                    permutiere [D]; 4, 4)      A  C  B  D
                                        3    4      d  b
                                                    permutiere [C]; 4, 4)      A  C  D  B
                    2    4      d  C  b
                                permutiere [C, b]; 3, 4)
                                        3    3      C  b
                                                    permutiere [b]; 4, 4)      A  D  C  B
                                        3    4      b  c
                                                    permutiere [C]; 4, 4)      A  D  B  C
1    2      b  a  C  D
            permutiere([a, C, D]; 2, 4)
            ...
1    3      c  B  a  D
            permutiere([B, a, D]; 2, 4)
            ...
1    4      d  B  C  d
            permutiere([a, C, D]; 2, 4)
            ...
```

Aufwand: 2 => 2; 3 => 3x2 = 6; 4 => 4x(3x2) = 4x6 = 24; ...; 100 => 100x99x .. x 2 = 100 ! also n!

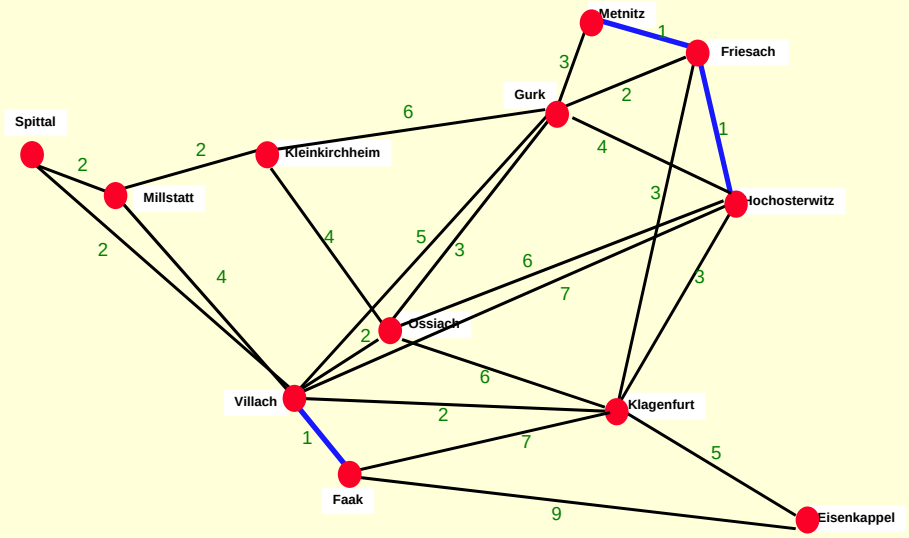
Heuristischer Ansatz

Heuristiken

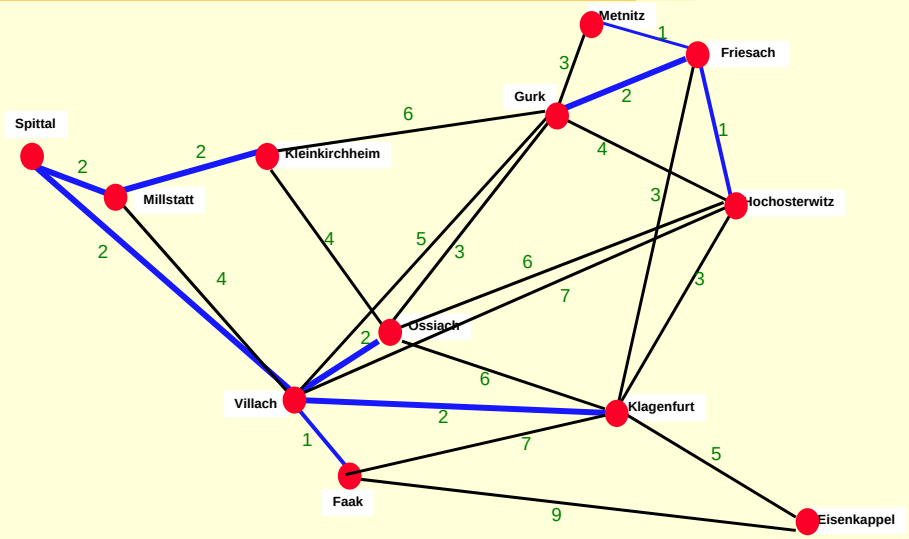
- Da *intractable problems* bei entsprechender Größenordnung in vernünftiger Zeit nicht exakt lösbar sind, begnügt man sich mit Näherungslösungen (**Approximationsalgorithmen, Heuristiken**).

- Diese liefern in der Regel Ergebnisse **in der Nähe** der optimalen Lösung.
  - Unklar, ob das berechnete Ergebnis tatsächlich optimal ist.
  - Unbekannt, wie weit das berechnete Ergebnis vom Optimum abweicht.

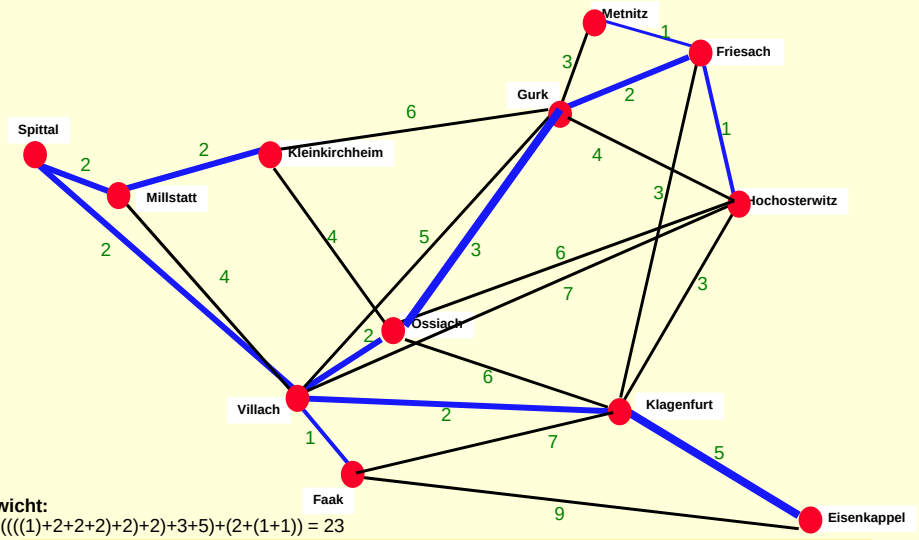
Minimaler spannender Baum 1



Minimaler spannender Baum 1,2

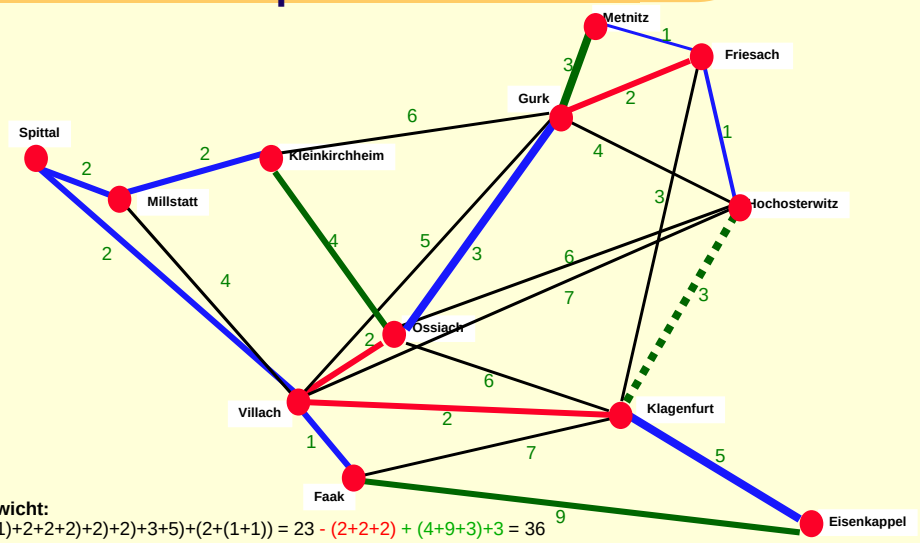


Minimaler spannender Baum 1,2,3,++



Gewicht:
((((1)+2+2+2)+2)+2)+3+5)+(2+(1+1)) = 23

Minimaler Weg über minimalem spannendem Baum



Gewicht:
((((1)+2+2+2)+2)+2)+3+5)+(2+(1+1)) = 23 - (2+2+2) + (4+9+3)+3 = 36

Metaheuristiken

Metaheuristiken

- Eine Metaheuristik ist ein Satz algorithmischer Konzepte, der verwendet werden kann, um Heuristiken, die auf ein breites Problemfeld anwendbar sind, zu definieren.
- Durch Parametrisierung wird das metaheuristische Konzept für konkrete Problemlösungen zugeschnitten.

Von Natur inspirierte Metaheuristiken

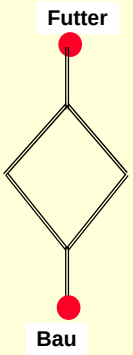
- Genetische Algorithmen
- Evolutionary Programming
- Simulated Annealing
- Bakteriologische Algorithmen
- Bienenstock-Algorithmen
- Ant Systems / Ant Colony Systems

Wie lösen Ameisen das Problem „kurze Reisen“?

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

Futtersuche von Ameisen

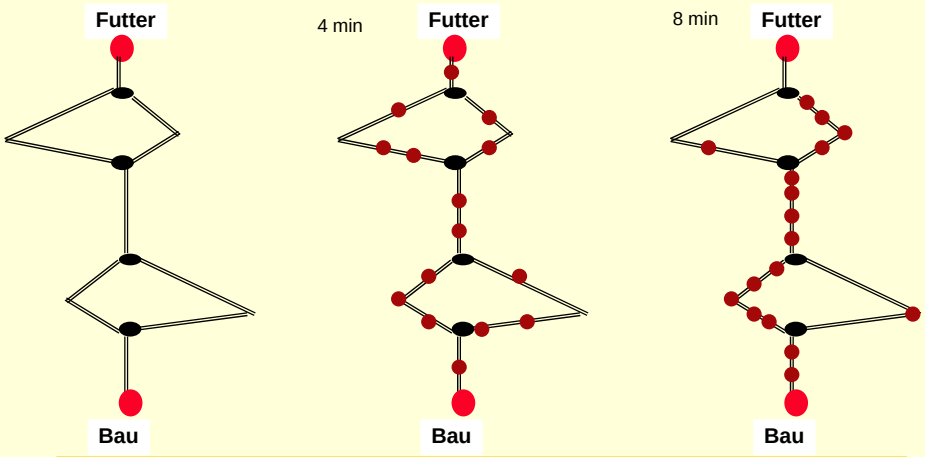
Deneubourg‘s 1. Brückenexperiment mit *Iridomyrex humilis: [1987 – 1990]*



- Auf welchem der beiden Wege wird sich eine Ameisenstraße bilden?
- Warum bildet sie sich?

Futtersuche von Ameisen

Deneubourg‘s Brückenexperiment mit *Lasius Niger:*



Der Ameisen-Trick: Kommunikation über *Stigmerie*

- Duftstoff ***Pheromon*** markiert die eigene Spur.
- Rasch gefundenes Futter ⇒ rasch am (eigenen !) Rückweg.
- Dies verstärkt die Pheromonspur.

- Dies lockt andere Ameisen an, der schon markierten Spur zu folgen.
- Auf verlassenen Spuren verdunstet das Pheromon mit der Zeit.

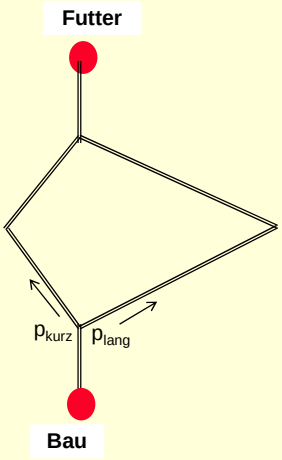
ABER

- Eine Futterstelle ist ja irgendwann erschöpft
  (oder vielleicht gibt es noch eine nähere/bessere):
- Daher wird nicht strikt der stärksten Pheromonspur gefolgt,
  sondern **auch zufällig andere Wege erprobt**.

Algorithmische Simulation des Verhaltens von Ameisen

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

Simulation des Deneubourg Experiments



Wahrscheinlichkeit p_{kurz} bzw. p_{lang}
dass Ameise *a*
zum Zeitpunkt *t*
den kurzen bzw. langen Weg wählt,
hängt von der aktuellen Pheromon-Intensität
$PhI_{a,w}(t)$ ab:

$$p_{a,kurz}(t) = PhI_{v,kurz}(t)^{\alpha} / (PhI_{a,kurz}(t)^{\alpha} + PhI_{a,lang}(t)^{\alpha})$$

$$p_{a,lang}(t) = PhI_{a,lang}(t)^{\alpha} / (PhI_{a,kurz}(t)^{\alpha} + PhI_{a,lang}(t)^{\alpha})$$

Pheromon-Ablagerung auf Hin- und Rückweg!

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

Problemlösungsprinzip

- Verstehe das Problemlösungsverhalten des biologischen Systems
- Baue ein entsprechendes mathematisches/algorithmisches Modell
- Prüfe, ob dieses das biologische System darstellen kann

- Wende das so erlernte Problemlösungsverhalten auf klassisch schwer lösbare Probleme an
- Verbessere diese
  - durch geeignete Parameter-Wahl
  - durch Variation von Details der Problemlösungsstrategie
  - durch algorithmische Optimierung

Dies kann zu einer Entfremdung von der biologischen Analogie führen.

Anwendung: Rundreiseproblem

- Modelliere Städte und Verbindungen dazwischen als Graph
- Stelle Städte-Graph als Distanzmatrix D(i,j) dar
- Platziere Ameisen in Ausgangsstädten
- Belege Graphen mit einer Ausgangsmenge von Pheromon τ_0
- Lass Ameisen laufen:

  Diese wählen die noch nicht besuchte nächste Stadt nach
  - Pheromonkonzentration auf der Wegstrecke dorthin $\tau_{i,j}$ und
  - „Erfolgserwartung“ eines kurzen Weges $\eta_{i,j}$
  - unter Berücksichtigung grundsätzlicher Randbedingung Ω aus.
- Sichere, dass
  - keine Kreise begangen werden,
  - Pheromon entsprechend der Nutzung und Güte der Wege aktualisiert wird.

Matrix d. Distanzen zw. Städten

	j	1	2	3	4	5	6	7	8	9	10	11	12
i	Städte	Fa	Fr	Gu	Ho	Kl	Kk	Me	Mi	Ob	Os	Sp	Vi
1	Faak	0	1000	1000	1000	7	1000	1000	1000	9	1000	1000	1
2	Friesach	1000	0	2	1	3	1000	1	1000	1000	1000	1000	1000
3	Gurk	1000	2	0	1000	1000	6	3	1000	1000	3	1000	5
4	Hochosterwitz	1000	1	1000	0	3	1000	1000	1000	1000	6	1000	7
5	Klagenfurt	7	3	1000	3	0	1000	1000	1000	5	6	1000	2
6	Kleinkirchheim	1000	1000	6	1000	1000	0	1000	2	1000	4	1000	1000
7	Metnitz	1000	1	3	1000	1000	1000	0	1000	1000	1000	1000	1000
8	Millstatt	1000	1000	1000	1000	1000	2	1000	0	1000	1000	2	4
9	Obirhöhle	9	1000	1000	1000	5	1000	1000	1000	0	1000	1000	1000
10	Ossiach	1000	1000	3	6	6	4	1000	1000	1000	0	1000	2
11	Spittal	1000	1000	1000	1000	1000	1000	1000	2	1000	1000	0	2
12	Villach	1	1000	5	7	2	1000	1000	4	1000	2	2	0

Erstes mathematisches Modell

Grundmodell: AS – Ant System [Dorigo 1992]

- Rundreiseproblem wird als Suchproblem aufgefaßt.
- Optimierungsziel (Suchziel) des Rundreiseproblems ist Weg mit geringsten Kosten C (kürzester Weg) .
- Randbedingung Ω:
  - Alle Sehenswürdigkeiten müssen besucht werden.
  - Letztlich muss man zum Ausgangspunkt zurückkehren.

- Ameisen machen sich auf den Weg, um den kürzesten zu finden.
- Sie wählen in Stadt *i* die jeweils nächste Stadt *j* zufällig
  - in Abhängigkeit von Pheromon τ_{ij} auf den Ausgangswegen
  - und heuristischer Attraktivität η_{ij}.

Umsetzung der Routenoptimierung

Konstruktionsgraph:

Problemgraph, erweitert um „teure“ verbotene Kanten ⇒ vollständiger Graph, Distanzmatrix zwischen Städten

Randbedingung:

Alle Städte müssen besucht werden ⇒ vollständiges Durchwandern, Liste noch zu besuchender Städte oder Tabuliste

Auswahlhilfe für Ameise:

$\tau_{i,j}$ .. Pheromonspur zwischen Knoten (Stadt) i und j

$\eta_{i,j}$ .. lokale Attraktivität von Knoten (Stadt) i nach j zu gehen
Meist: $\eta_{i,j} = 1/d_{i,j}$ … Kehrwert der Distanz zwischen zwei Städten.

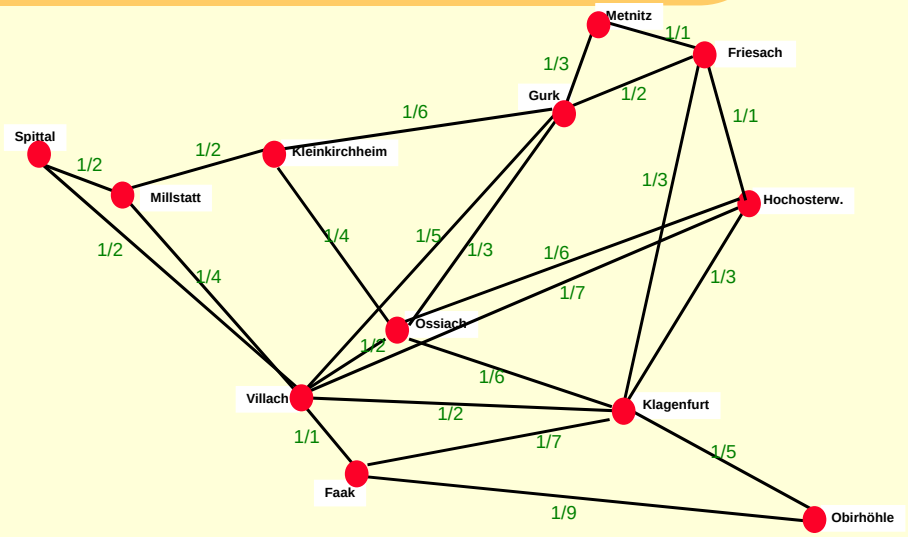
Lösungsansatz:

- Jede Ameise wird initial in eine zufällig gewählte Stadt gesetzt.
- In jedem Verfahrensschritt besucht sie eine von ihr noch nicht besuchte Stadt.
- Nachdem alle Städte besucht wurden, läuft sie ihren Weg zurück und aktualisiert die Pheromonwerte.

Abbruch des Verfahrens:

Wenn sich längere Zeit keine Verbesserungen der besten Lösung mehr ergeben.

Heuristische Attraktivität η_{ij}



Gerüst des AS Algorithmus

Prozedur AS {

Initialisiere das System;

über t_{max} Zyklen **do {**

berechne für jede Ameise zufällig deren Weg;
ist einer davon kürzer als bisher gefundener Kürzester,
dann merk in dir;

akutalisiere Pheromon-Matrix
}

gib beste Lösung aus

}

Auswahl und Pheromon-Update

- lokale Auswahlentscheidung für nächste Stadt

  $p_{ij}^k = [\tau_{i,j}]^a \times [\eta_{i,j}]^b / \Sigma <l \in J_i^k> ([\tau_{i,l}]^a \times [\eta_{i,l}]^b)$ wenn $j \in J_i^k$; sonst 0

  $J_i^k \subseteq N_i^k$ … Jene Städte in der Nachbarschaft von Stadt *i*, die von Ameise *k* noch nicht besucht wurden.

  a, b ... Parameter

- Pheromon-Initialisierung:

  τ_0 = # Ameisen / $C_{Tour\_nächsterNachbar}$ C .. Kosten der Rundreise (Tour)

- Pheromon-Aktualisierung:

  $\tau_{i,j}(t') \leftarrow (1 - \rho)\ \tau_{i,j}(t)$ für alle Kanten (i, j) im Graph (Verdunstung)

  $\tau_{i,j}(t+1) \leftarrow \tau_{i,j}(t') + \Sigma_k\ \Delta\tau_{i,j}^k(t')$ für alle Kanten (i, j) im Graph (Aktualisierung)

  $\Delta\tau_{i,j}^k(t') = 1/C^k$ wenn *(i,j)* auf der Tour T^k von Ameise *k*; liegt;
  sonst 0

AS Algorithmus

{ Initialisiere jede Kante (i,j) mit $\tau_{i,j}(0) = \tau_0$};
for k = **1 to m do** {platziere Ameise k in eine zufällig gewählte Stadt};
{initialisiere L^+ als Länge der bisher gefundenen kürzesten Tour T^+}

for t = **1 to** t_{max} **do {**
for k = **1 to m do**
while Stadt # Ausgangsstadt **do**
{konstruiere Tour $T^k(t)$ durch Wanderung von Ameise k in die nächste noch nicht besuchte Stadt gemäß p_{ij}^k};

for k = **1 to m do** {berechne Länge $L^k(t)$ der Tour $T^k(t)$ von Ameise k}
wenn Minimum der Längen $L^k(t) < L^+$ **dann** {$L^+ := L^k(t)_{min}$; $T^+ := T^k(t)_{min}$};

for jede Kante (i,j) **do** {aktualisiere Pheromon-Markierung mittels

$$\tau_{i,j}(t+1) \leftarrow (1-\rho)\tau_{i,j} + \Sigma_{1..m} \Delta\tau_{i,j}^k(t) + \Delta\tau_{i,j}^e(t) \}$$

}

drucke kürzeste Tour T^+ und ihre Länge L^+
}

Verbessertes Modell

ACS – Ant Colony System [Dorigo, Gambardella 1997]

- Ziel:
  Vermeiden von Konvergenz zu lokalen Optimum.

- Zur Lösung von Konvergenzproblemen bei großen Problemstellungen Änderungen bei
  - Tour-Konstruktion, Wahl der nächsten Stadt
  - geänderte (lokale) Pheromon-Aktualisierung
  - globale Pheromon-Aktualisierung durch globalen Dämonen

Tour-Konstruktion bei ACS

Lokale Auswahlentscheidung in *i* für nächste Stadt *j* hängt von neuem Parameter q_0 $[0 \le q_0 \le 1]$ ab.

An Stelle von:

$$p_{ij}^k = [\tau_{i,j}]^a \times [\eta_{i,j}]^b / \Sigma_{<l \in J_i^k>} ([\tau_{i,l}]^a \times [\eta_{i,l}]^b) \quad \text{wenn } j \in J_i^k \text{ ; sonst } 0$$

$J_i^k \subseteq N_i^k$ … Jene Städte in der Nachbarschaft von Stadt *i*, die von Ameise *k* noch nicht besucht wurden.

a, b ... Parameter

gilt:

$$j = \text{argmax}_{l \in} N_i^k \{\tau_{i,l} \times [\eta_{i,l}]^b\} \quad \text{wenn Zufallszahl } q \le q_0$$

sonst

$$j = J \text{ gemäß } p_{ij}^k = [\tau_{i,j}]^1 \times [\eta_{i,j}]^b / \Sigma_{<l \in J_i^k>} ([\tau_{i,l}]^1 \times [\eta_{i,l}]^b) \quad \text{wenn } j \in J_i^k \text{ ; sonst } 0$$

Pheromon-Update bei ACS

- Bereits nach jeder Verwendung einer Kante führen die Ameisen eine **lokale** Pheromon-Aktualisierung durch:

$$\tau_{i,j}' \leftarrow (1 - \xi)\, \tau_{i,j} + \xi\, \tau_0 \quad \text{mit Parameter } 0 < \xi < 1$$

Konsequenz: Verwendung einer Kante **reduziert** ihre Attraktivität für künftige Ameisen.

- Nachdem alle Ameisen die aktuelle Tour beendet haben, führt ein Dämon eine globale Pheromon-Aktualisierung durch:

$$\tau_{i,j}' \leftarrow \tau_{i,j} + \Delta\tau_{i,j}^{\text{beste}}$$

mit $\Delta\tau_{i,j}^{\text{beste}} = 1/C^{bs}$ oder $\Delta\tau_{i,j}^{\text{beste}} = 1/C^{bi}$
wobei C^{bs} bzw, C^{bi} die günstigsten Kosten seit Start der Analyse bzw. innerhalb der aktuellen Tour sind.

- **Gesamtkonseqenz**: bessere Streuung und geringere algorithm. Komplexität

Struktur der ACS Metaheuristik

ACS – Ant Colony System [Dorigo, Gambardella 1997]

Prozedur ACS_Metaheuristik

definiere Lösungsansatz

konstruiere Ameisen-Lösung;

aktualisiere Pheromonwerte;

setze Aktionen globaler Dämonen

beende Optimierungsverfahren.

Grundprinzipien von AS/ACS

- Finde Problemrepräsentation, die Ameisen den schrittweisen Aufbau (Modifikation) einer Lösung auf Grundlage einer wahrscheinlichkeitsgesteuerten Übergangsregel erlauben und der indirekten Kommunikation über Pheromon erhalten.
- Bestimme ein Maß für die heuristische Wünschbarkeit η von Übergängen.
- Bestimme eine wahrscheinlichkeitsgesteuerte Übergangsregel.
- Finde eine Constraint-Satisfaction-Methode, um die Zulässigkeit von Lösungen zu erzwingen.
- Definiere Pheromon-Aktualisierungs-Regel.

Things to Remember

- Generalisierung von Problemen führt oft erst zur effizienten Lösbarkeit.
- Wahrscheinlichkeitsbasierte Algorithmen sind nicht „exakt“, aber bei NP-vollständigen Problemen sehr effizient und effektiv.
- Kooperationsstrategie schlägt Einzelkämpfertum.
- Biologische Lösung diente der Inspiration, realer (Simulations-)Algorithmus weicht von der Initiallösung ab.
- Zu starker Fokus auf rasche Konvergenz führt zu suboptimalen Lösungen.

Danke für die Aufmerksamkeit !

Referenzen

[BoDoTh 99] Bonabeau E., Dorigo M., Theraulaz G.: *Swarm Intelligence – From Natural to Artificial Systems*; Oxford University Press, 1999.

[BlRo 03] Blum C., Roli A.: *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*. ACM Computing Surveys, 35(3), pp. 268 – 308.

[DeAGP 90] Deneubourg J.-L., Aron S., Goss S., Pasteels J.-M.: *The Self-Organizing Exploratory Pattern of the Argentine Ant.* Journal of Insect Behavior, 3, (1990), pp. 159 - 168.

[Dorigo 92] Dorigo M.: *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartimento di Elettronica, Politechnico di Milano, Milan.

[DoGa 97a] Dorigo M., Gambardella L.M.: *Ant colonies for the traveling salesman problem.* BioSystems, 43 (2)pp 73 – 81.

[DoGa 97b] Dorigo M., Gambardella L.M.: *Ant Colony System: A cooperative learning approach to the traveling salesman problem*. IEEE Trans on Evolutionary Computation, 6(4), pp. 31- 365.

[DoSt 04] Dorigo M, Stützle T.: *Ant Colony Optimization*, MIT Press, 2004.