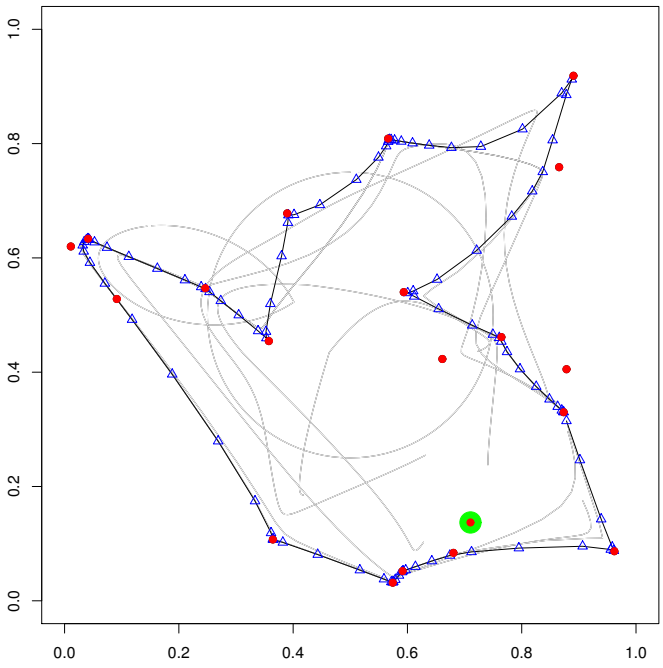
Das Handlungsreisenden-Problem per Kohonen-Karte

File: tsp-2018a.rev
in: /home/wiwi/pwolf/R

November 2009 / Februar 2017



1 Einleitung

Als Handlungsreisenden-Problem wird das Problem bezeichnet, den kürzesten Weg einer Rundreise durch vorgegebene Orte zu finden. Bei n Orten beträgt die Anzahl der möglichen Wege $(n-1)!/2$. Alle auszuprobieren, übersteigt selbst für moderate Werte von n die Möglichkeiten heutiger Rechner. Geschickte Algorithmen verschieben zwar die Machbarkeitsgrenze, doch auch diese helfen bei größeren Ortsanzahlen nicht weiter. Deshalb ist es naheliegend, sich mit ziemlich guten Lösungen zufrieden zu geben. Kohonen-Karten bilden den Ausgangspunkt für einen solchen Vorschlag. Die folgenden Ausführung gehen zurück auf: H. Ritter, T. Martinetz, K. Schulten (1990): *Neuronale Netze*, Addison-Wesley.

2 Kohonen-Karten

Ein Ansatz zur approximativen Lösung des Handlungsreisenden-Problems basiert auf der Idee selbstorganisierender sensorischer Karten, wie sie von Kohonen vorgeschlagen worden sind. Zum Verständnis ist es hilfreich, die Grundzüge von Kohonen-Karten vorzustellen. Kohonen-Karten bestehen aus vernetzten, virtuellen Neuronen. Während der Lernphase bilden sich in dem Netz der Neuronen Muster aus, die mit Mustern in der Menge der Sensoren korrespondieren.

Ausgangspunkt des Modellierung ist eine Menge von Sensoren, die über Nervenbahnen (Axone) ihren Reizzustand an die Synapsen der Neuronen übermitteln. Weiterhin sind die Neuronen untereinander verbunden. Die Synapsen bewerten ihre Inputs aufgrund interner Gewichte. Aus den verarbeiteten Inputs folgt der Aktivitätszustand der Neuronen. Ist $\mathbf{v}$ der Vektor der Input-Reize und $\mathbf{w}_r$ der Gewichtsvektor der Synapsen des Neurons r, dann sei der Erregungszustand des Neurons r ohne die Berücksichtigung anderer Neuronen modelliert durch:

$$f_r(\mathbf{v}) = \sigma^* \left(\mathbf{w}_r' \mathbf{v} - \theta\right)$$

Hierbei ist σ^* eine sigmoide Funktion, die Eigenschaften wie eine Verteilungsfunktion aufweist, und θ ein Shiftparameter. Fassen wir in $\mathbf{f}$ die Erregungszustände aller Neuronen zusammen. Dann modellieren wir für die Erregung von Neuron r mit den Koppelungsstärken $\mathbf{g}_r$ der Neuronen mit Neuron r:

$$f_r = \sigma^* \left(\mathbf{w}_r' \mathbf{v} + \mathbf{g}_r' \mathbf{f} - \theta\right) \quad (*)$$

Für die gewünschten Karten-Eigenschaften müssen Neuronen in der Nähe eines betrachteten Neurons einen positiven Einfluss haben ($g > 0$), wogegen ferne Neuronen negative Wirkungen ($g < 0$) aufweisen müssen. Damit stellen sich Korrelationen zwischen den Zuständen benachbarter Neuronen ein und benachbarte Sensoren führen zu maximalen Wirkungen bei nahe beieinander liegenden Neuronen. Näherungsweise kann man zu einem Input $\mathbf{v}$ das Zentrum der maximalen Erregung abschätzen durch

$$\max_r \mathbf{w}_r' \mathbf{v}$$

Falls einige akzeptable Bedingungen erfüllt sind, liefert die folgende Minimierungsaufgabe das gleiche Ergebnis: Dieses führt mit einigen akzeptablen Annahmen zu dem selben Ergebnis wie:

$$\min_r ||\mathbf{w}_r - \mathbf{v}|| \quad (**)$$

Die Gleichung $(*)$ beschreibt die Wirkungsweise eines stationären Systems in einer nur impliziten Form. Jedoch enthält es die Information, welches Neuron bei einem speziellen Reiz maximal reagiert. Mit $(**)$ erhält man für das Modell eine brauchbare Approximation. Durch die Beziehung zwischen einer Position im Bereich der Sensoren und dem jeweiligen Neuron mit höchster Erregung ist ein Blick in das System der Neuronen wie ein Blick auf eine Landkarte für die Landschaft der Sensoren. Umgekehrt können wir jedem Neuron denjenigen Punkt im Raum der Sensoren zuordnen, bei dem es maximal reagiert, und so jedes Neuron im Sensorenraum einzeichnen.

3 Lernphase

Im Rahmen der Lernphase soll das neuronale Netz in einer Weise sensibilisiert werden, dass zwischen den Neuronen ähnliche Beziehungen entstehen wie zwischen den Input-Sensoren vorhanden sind. Wie können wir uns das plastisch vorstellen? Stellen wir uns dazu beispielsweise die Signale von Sensoren einer Hand vor, dann liegen die Sensoren eines Fingers beieinander. In der Lernphase sollen sich die Zuständigkeiten der Neuronen für Sensoren in einer Weise ausbilden, dass benachbarte Neuronen für benachbarte Sensoren zuständig sind: Wenn das gelingt, entspricht die Struktur der Neuronen in etwa der der Sensoren, und die Neuronen können grob als Landkarte für Sensoren interpretiert werden. Grob deswegen, weil die Sensoren nicht auf einer Gitterstruktur liegen müssen und auch weil für sehr wichtige Bereiche (wie Zeigefinger, Daumen) viele Sensoren und damit auch viele Neuronen zuständig sind. Für weniger wichtige (wie Handrücken) bedarf es eben auch nur weniger viele Sensoren und Neuronen.

Die Lernphase hat die Aufgabe, die Gewichte der Synapsen für die jeweilige Problemstellung anzupassen. Hierzu wird das neuronale Netz wiederholt Reizen aus einer Testmenge ausgesetzt. Die Modifikation der Parameter geschieht Schritt für Schritt nach einer Lernregel. Bei Lernregeln für die Kohonen-Karten sind zwei Dinge zu berücksichtigen. Einerseits muss es zur Ausbildung von Korrespondenzen kommen, so dass irgendwie der Reiz-Input mit den Gewichten abzugleichen ist. Anderseits sind Nachbarschaftsbeziehungen der Neuronen zu integrieren. Für den Ablauf des Lernprozesses lässt sich überlegen, dass zur Ausbildung grober Strukturen zuerst eher große Veränderungen erfolgen sollten, während mit der Zeit ein Stabilisierungsprozess zweckmäßigerweise die Oberhand gewinnen muss.

Betrachten wir zunächst die Nachbarschaftfrage. Die Erregungsantwort auf einen Reiz setzt sich zusammen aus der direkten Wirkung des Reizes sowie des Einflusses der anderen Neuronen. Kohonen unterstellt vereinfachend, dass sich die *Erregungsantwort* für ein Neuron als Funktion h von den Orten der Neuronen und der Stelle des Erregungszentrums schreiben lässt. Beispielsweise

können wir für h eine Gauss-Kurve ins Auge fassen:

$$h_{\mathbf{rr'}} = \exp\left(-\frac{(\mathbf{r}-\mathbf{r'})^2}{2\sigma^2}\right)$$

Hierbei steht $\mathbf{r'}$ für die Koordinaten des Erregungszentrums und $\mathbf{r}$ für die des gerade betrachteten Neurons. Der Radius σ der Erregungsfunktion bestimmt die Umgebung um das Erregungszentrum, in der Neuronen in einem Lernschritt angepasst werden. Anfangs muss nach den Vorüberlegungen σ großzügig gewählt werden, zum Schuss dagegen immer kleiner, so dass es nur noch zum Fein-Tuning kommt.

Wie werden sich nun nach erfolgter Reizung die Gewichte $\mathbf{w_r}$ verändern? Nach der *Hebbschen Hypothese* ändern sich beim Lernen die Gewichte proportional zu der korrelierten Aktivität vor und hinter der Synapse, was im einfachsten Fall für den Input $\mathbf{x}$ und den Output y durch die Vorschrift $\Delta w_i = \varepsilon \cdot x_i \cdot y(\mathbf{x})$ abgebildet werden kann. ε ist hierbei eine positive Konstante, die die Größe des Lernschrittes angibt.

Für das Kohonen-Modell erhält man mit dieser Hypothese: $\varepsilon \cdot h_{\mathbf{rr'}} \cdot \mathbf{v}$. Zusätzlich wird im Kohonen-Modell ein Abklingterm integriert, der die Veränderung mit zunehmender Stimmigkeit reduziert. Anders formuliert soll in dem Modell die Veränderung der Gewichte proportional zu der Differenz aus Input und Gewichten wie auch zu $h_{\mathbf{rr'}}$ erfolgen, und es ergibt sich:

$$\Delta\mathbf{w_r} = \varepsilon \cdot h_{\mathbf{rr'}}(\mathbf{v} - \mathbf{w_r})$$

Damit ist ein Vorschlag gemacht, der sich letztlich aus zwei relativ einfachen Formeln zusammensetzt.

4 Übertragung auf das TSP

Wie können wir die Überlegungen auf unser Handlungsreisenden-Problem übertragen? Zunächst wählen wir ein eindimensionales Neuronennetz, wobei wir die Neuronen im Kreis anordnen. Das zweite Neuron ist mit dem ersten und dem dritten verbunden, das dritte mit dem zweiten und vierten usw. bis zum letzten, das mit dem vorletzten und dem ersten Neuron verknüpft ist. Die zu besuchenen Orte werden als Sensoren angesehen, die ihre zweidimensionalen Koordinaten übermitteln.

In der Lernphase werden die Ortsreize an das Netz übergeben, und es stellen sich bei den Neuronen Zuständigkeiten für geographische Bereiche ein. Da zu jedem Neuron der geometrische Ort maximaler Zuständigkeit bestimmt werden kann und die Neuronen auf einer Kette angeordnet sind, wird damit eine Rundreise durch die Landschaft der Orte definiert. Dadurch dass in jedem Lernschritt ein Ort die Rundreise in seine Richtung verbiegt, reagiert die Rundreise mit der Zeit auf alle vorgegebenen Orte. Wird im Rahmen der Lernfortschritte die Reichweite der Neuronen immer mehr eingeschrägt, bilden sich Rundreisen aus, die fast perfekt durch die Orte verlaufen.

Jetzt ist nur noch zu argumentieren, dass der gefundene Weg auch ein günstiger – sprich: kurzer Weg – ist. Diese Eigenschaft geht im wesentlichen

darauf zurück, dass die Neuronen-Kette begrenzt ist und von sich aus alle Neuronen so ähnlich sein wollen wie ihre Nachbarn. Unnötig lange Wege weisen aus Sicht des Netzes viel schlechtere Nachbarschaftseigenschaften auf als kürzere Wege. Jeder Umweg ist dem Netz unlieb und wird vermieden, die Abstecher zu den einzelnen Orten werden indes erzwungen. Im Verlauf der Lernphase werden zu lange Wege gegenüber kürzeren ausgetauscht und es verbleiben nur die notwendigen Wegstücke durch die vorgegebenen Orte.

Nach Abschluss der Lernphase können wir die Neuronen im Problemraum verorten, und sie lassen sich dort als Vorschlag einer Rundreise interpretieren. Wie lautet nun der Algorithmus konkret?

5 Der Algorithmus

Der Algorithmus besteht aus einer zentralen Schleife, in der immer wieder neue Reize ausgewählt und mit diesen die aktuellen Gewichte angepasst werden.

1. *Initialisierung:* Wähle anfängliche Synapsenstärken $\mathbf{w_r}$ — wir werden sie zufällig wählen.
2. *Stimuluswahl:* Wähle entsprechend der Dichte $P(\mathbf{v})$ einen Vektor von Reizen $\mathbf{v}$ aus — wir werden den jeweiligen Ort $\mathbf{v}$ gleichverteilt aus der Menge aller Orte ziehen.
3. *Response:* Bestimme Erregungszentrum, also das Neuron $\mathbf{r'}$, das maximal stark auf den Reiz reagiert.

$$||\mathbf{v} - \mathbf{w_{r'}}|| \leq ||\mathbf{v} - \mathbf{w_r}|| \quad \text{für alle } \mathbf{r}$$

4. *Adaptionsschritt:* Führe einen Lernschritt durch und verändere die Synapsenstärken nach der Formel:

$$\mathbf{w_r}^{neu} = \mathbf{w_r}^{alt} + \varepsilon \cdot h_{\mathbf{rr'}}(\mathbf{v} - \mathbf{w_r}^{alt})$$

5. *Schleifenende:* Wenn die Lernphase noch nicht beendet ist, gehe zu 2.

6 Umsetzung

Aus dem in Ritter et al. vorgeschlagenen Algorithmus ergibt sich das Programm `TSP`. Dem Header des Programmes ist zu entnehmen, was beim Aufruf festgelegt werden kann:

Variabel	Bedeutung	Default-Wert
`maxloop`	die Länge der Lernphase in Iterationen	50
`n.orte`	Anzahl der zu besuchenden Orte	10
`n.neuronen`	Anzahl der Neuronen	100
`wait`	Wartezeit zwischen Iterationen in Sek.	0.01
`eps`	Lernschrittgröße	0.8
`n.plots`	wie viele Zwischenschritte sollen gezeigt werden	10
`dump`	Bilder in Dateien ablegen?	`FALSE`

1 ⟨*start* 1⟩ ≡ ⊂ 2, 8

```
TSP <- function(maxloop = 50, n.orte = 10, n.neuronen = 100, seed = 13, seed2 = 1,
                wait = 0.01, eps = 0.8, n.plots = 10, dump = FALSE){
  ⟨wähle Zufallsorte 3⟩
  set.seed(seed2)
  ⟨bestimme Anfangszustände der Neuronen 4⟩
  ⟨bestimme Test-Set der Orte für die Lernphase 5⟩
  ⟨führe Lernphase durch 6⟩
  ⟨zeichne Orte und die den Neuronen zugeordneten Punkte 7⟩
}
```

Ein Testaufruf ist schnell geschrieben.

2 ⟨*Testaufruf von TSP* 2⟩ ≡

```
⟨start 1⟩
TSP(2500, 55, 1500, seed = 111, wait = 00, eps = 1, seed2 = 2) # #TSP(100, 20)
```

Wie siehe der Rumpf von TSP aus? Zunächst werden `n.orte` Zufallsorte gewählt.

3 ⟨*wähle Zufallsorte* 3⟩ ≡ ⊂ 1

```
set.seed(seed)
orte <- cbind( x = runif(n.orte), y = runif(n.orte))
```

Dann werden die Anfangszustände der Neuronen festgelegt. Dieses geschieht in einer Form, dass die zugeordneten Punkte im Raum der Orte auf einem Kreis liegen. Anschließend werden die Orte und die zu den Neuronen gefundenen Landschaftspunkten gezeichnet.

4 ⟨*bestimme Anfangszustände der Neuronen* 4⟩ ≡ ⊂ 1

```
# Kreislinie als Anfangszuordnung
r <- 1:n.neuronen; ek <- (2 * pi) * r / n.neuronen
gew <- gew.old <- 0.5 + 0.25 * cbind(sin(ek), cos(ek))
par(pty="s")
plot(gew, type = "l", xlim = 0:1, ylim = 0:1, ann = FALSE)
points(orte)
```

Die Reihenfolge der nacheinander für die Lernphase zu verwendenden Orte wird zufällig generiert.

5 ⟨*bestimme Test-Set der Orte für die Lernphase* 5⟩ ≡ ⊂ 1

```
# Reihenfolge der Orte fÃijr Lernphase festlegen
rortnr <- sample(1:n.orte, maxloop, replace = TRUE)
```

Der schwierigste Teil der Umsetzung besteht darin, die Lernphase umzusetzen. Dazu nehmen wir einen Ort aus der Testmenge und konstruieren mit seinen beiden Koordinaten den Vektor $\mathbf{v}$. Mit Hilfe von `v` bilden wir $\mathbf{w} - \mathbf{v}$, siehe: `gew-v.mat`, und die quadrierte Euklidische Entfernung. Mittels Minimierung finden wir das Reizzentrum seitens der Neuronen.

Für den Adaptionsschritt berechnen wir die Entfernung der Neuronen vom Reiz-Zentrum `reiz.zentrum` und berechnen die Funktion h. Mit Hilfe von h können wir dann die neuen Gewichte ermitteln. Damit ist fast alles gesagt, jedoch ist noch eine Bemerkung über σ^2 =`sigma.q` zu liefern. Die Reichweite beginnt zunächst mit einem Wert von etwa 50 und wird, wie bei Ritter et al. beschrieben, von Schritt zu Schritt verkleinert, bis in der letzten Iteration der Wert 1 erreicht wird.

Nach einigen Iteration wird ein Bild der Lage gezeichnet, so dass der Betrachter einen kleinen Film zu sehen bekommt.

6 ⟨*führe Lernphase durch* 6⟩ ≡ ⊂ 1

```
dump.set <- round(seq(1, maxloop, length = (n.plots + 1)))
for(i in 1:maxloop){
# Wahl des Test-Ortes und Bestimmung von v
  idx <- rortnr[i]; v <- orte[idx,]
  v.mat <- matrix(v, n.neuronen, 2, byrow = TRUE)
# Bestimmung des Reiz-Zentrums
  d <- gew - v.mat; abstand.q <- apply(d * d, 1, "sum")
  rstrich<-which.min(abstand.q)
# Adaptionsschritt
  sigma.q <- (50 * 0.02^(i / maxloop))^2
  reiz.zentrum <- pmin(abs(r - rstrich), abs(r - (rstrich - n.neuronen)))
  h <- exp(-0.5 * reiz.zentrum^2 / sigma.q ); hrr.str <-cbind(h, h)
  gew <- gew + eps * (v.mat - gew) * hrr.str
# Darstellung
  if(i %in% dump.set){
      par(col = "white"); lines(gew.old); print(i)
      par(col = 1); lines(gew[,1], gew[,2]); gew.old <- gew
      if(dump){
        name <- paste("k-p-", which(i == dump.set), ".ps", sep = "")
        dev.copy(postscript, name, horizontal = FALSE); dev.off()
      }
  }
  Sys.sleep(wait)
}
```

Abschließend können wir die Quasi-Lösung des TSP optisch hervorheben.

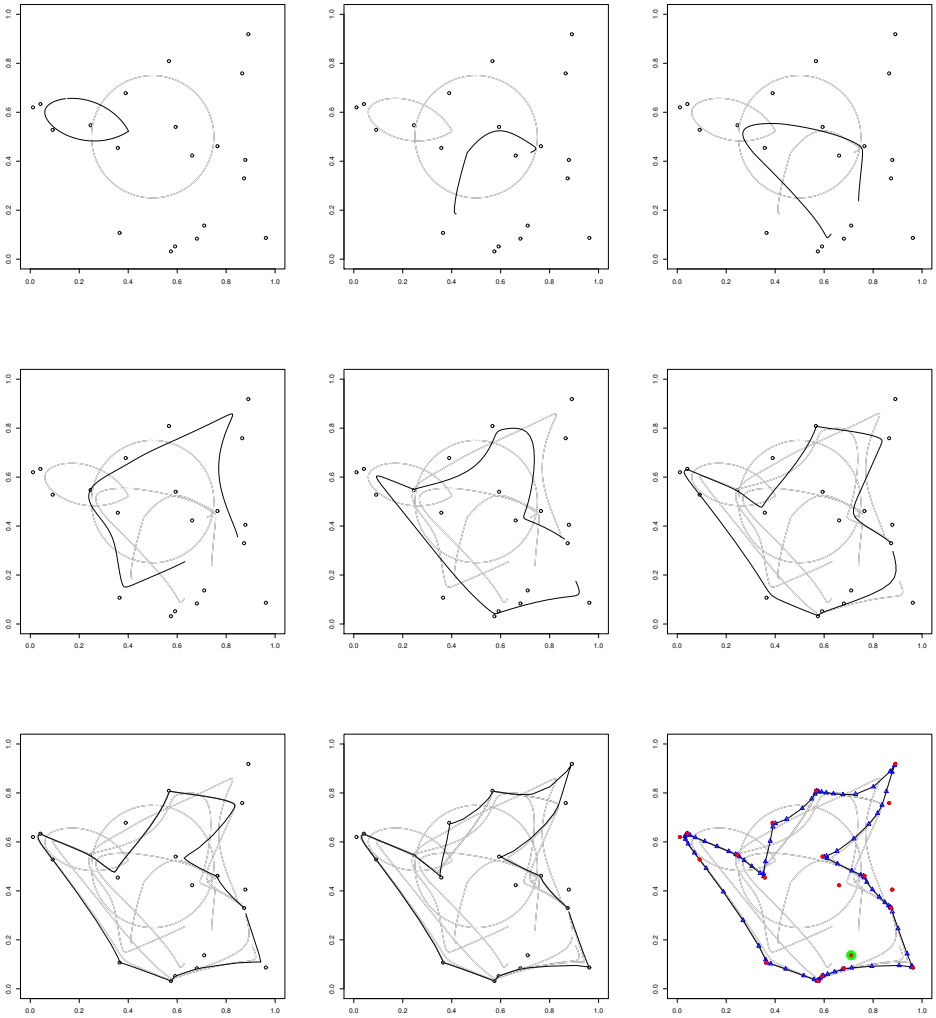
7 ⟨*zeichne Orte und die den Neuronen zugeordneten Punkte* 7⟩ ≡ ⊂ 1

```
points(orte[1,1], orte[1,2], pch = 19, cex = 3, col = "green")
points(gew[,1], gew[,2], pch = 2, col = "blue")
points(orte[,1], orte[,2], pch = 19, col = "red")
if(dump){
  name <- paste("k-p-", n.plots + 2, ".ps", sep = "")
  dev.copy(postscript, name, horizontal = FALSE); dev.off()
}
neuronen <- gew[,1] * Inf
for( i in 1:n.orte ){
  idx <- which.min((orte[i,1] - gew[,1])^2 + (orte[i,2] - gew[,2])^2)
  neuronen[idx] <- i
}
neuronen <- neuronen[ neuronen < Inf ]
path <- orte[ neuronen, ]
find.dist <- function(xy){
  # if( !is.matrix(xy) ) xy <- rbind(xy)
  xy <- rbind( xy, xy[1,])
  fitness <- sum((diff(xy[,1])^2 + diff(xy[,2])^2)^0.5)
  fitness
}
title(paste("fit:", round(d <- find.dist( path ), 3) ))
# find path
idx <- rep(0, n.orte)
for(i in 1:n.orte){
  idx[i] <- which.min(sqrt((gew[,1] - orte[i,1])^2 +  (gew[,2] - orte[i,2])^2))[1]
}
path <- seq(n.orte)[ order(idx) ]
lines( orte[ path,], col = "red")
list(d, path)
```

7 Demo

8 ⟨* 8⟩ ≡

```
⟨start 1⟩
set.seed(111)
TSP(maxloop = 100, n.orte = 20, n.plots = 7, dump = TRUE)
noquote(deparse(TSP))
```



8 Die Funktion TSP im Überblick

Wie sieht die Funktion im Zusammenhang aus? Diese Frage ist schnell geklärt.

9 ⟨*zeige Funktion* 9⟩ ≡

```
(function(name){
  dump(name); a <- deparse(parse(text = readLines("dumpdata.R")))
  names(a) <- rep("", length(a)); a <- sub("^structure.expression.", "", a)
  a <- sub("...srcfile.*$", "", a); noquote(cbind(" " = a))
})("TSP")
```

Hier ist das Ergebnis.

```
TSP <- function(maxloop = 50, n.orte = 10,
    n.neuronen = 100, wait = 0.01, eps = 0.8, n.plots = 10, dump = FALSE) {
    orte <- matrix(runif(2 * n.orte), n.orte, 2)
    r <- 1:n.neuronen
    ek <- (2 * pi) * r/n.neuronen
    gew <- gew.old <- 0.5 + 0.25 * cbind(sin(ek), cos(ek))
    par(pty = "s")
    plot(gew, type = "l", xlim = 0:1, ylim = 0:1, ann = FALSE)
    points(orte)
    rortnr <- sample(1:n.orte, maxloop, replace = TRUE)
    dump.set <- round(seq(1, maxloop, length = (n.plots + 1)))
    for (i in 1:maxloop) {
        idx <- rortnr[i]
        v <- orte[idx, ]
        v.mat <- matrix(v, n.neuronen, 2, byrow = TRUE)
        d <- gew - v.mat
        abstand.q <- apply(d * d, 1, "sum")
        rstrich <- which.min(abstand.q)
        sigma.q <- (50 * 0.02^(i/maxloop))^2
        reiz.zentrum <- pmin(abs(r - rstrich), abs(r - (rstrich -
            n.neuronen)))
        h <- exp(-0.5 * reiz.zentrum^2/sigma.q)
        hrr.str <- cbind(h, h)
        gew <- gew + eps * (v.mat - gew) * hrr.str
        if (i %in% dump.set) {
            par(col = "white")
            lines(gew.old)
            print(i)
            par(col = 1)
            lines(gew[, 1], gew[, 2])
            gew.old <- gew
            if (dump) {
                name <- paste("k-p-", which(i == dump.set), ".ps",
                  sep = "")
                dev.copy(postscript, name, horizontal = FALSE)
                dev.off()
            }
        }
        Sys.sleep(wait)
    }
    points(orte[1, 1], orte[1, 2], pch = 19, cex = 3, col = "green")
    points(gew[, 1], gew[, 2], pch = 2, col = "blue")
    points(orte[, 1], orte[, 2], pch = 19, col = "red")
    if (dump) {
        name <- paste("k-p-", n.plots + 2, ".ps", sep = "")
        dev.copy(postscript, name, horizontal = FALSE)
        dev.off()
    }
}
```

9 Indizes

Object Index

Code Chunk Index