Selbstorganisierende Karten und das Problem des Handlungsreisenden

Programmieraufgabe 3

Neuronale Netze
und ihre Anwendungen

Sommersemester 1996

Johannes Beck

Michael Ratz

17. Juli 1996

Zusammenfassung

Basierend auf der angegebenen Literatur wurde die Lösung des Travelling Salesman Problems mittels einer selbstorgansierenden Karte implementiert. Für eine vorgegebene Menge von Städten wird eine suboptimale Lösung gefunden. Der Algorithmus und die Implementation werden vorgestellt. Die Qualität der gefundenen Lösungen soll an Simulationsbeispielen gezeigt und diskutiert werden.

Einleitung

Das Travelling Salesman Problem

Das Travelling Salesman Problem (TSP) ist eines der klassisches Probleme aus der theoretischen Informatik. Gegeben ist dabei eine Menge von N Städten, die durch ihre Koordinaten (x, y) mit $x, y \in [0, 1]$ definiert sind. Gesucht ist nun ein ungerichteter Graph, der die Städte als Knoten besitzt. Für die Kanten des Graphen soll dabei gelten, daß jeder Knoten genau zwei Kanten besitzt und die gesamte Weglänge, d.h. die Summe der Länge der Kanten, minimal ist. Dabei wird zur Definition der Länge einer Kante zwischen zwei Knoten der euklidische Abstand der Koordinaten der Knoten benutzt.
Es ist leicht einzusehen, daß eine solche Kantenbelegung existiert, da es nur endlich viele Möglichkeiten, genauer $\frac{(N-1)!}{2}$, gibt, die N Städte miteinander zu verbinden. Genauso einfach läßt sich ein naiver Algorithmus zur Bestimmung des kürzesten Wegs angeben: Man durchläuft iterativ alle Möglichkeiten und bestimmt so die Lösung mit der minimalen Weglänge.
Bekanntlich steigt allerdings der Rechenaufwand bei diesem Algorithmus exponentiell mit der Anzahl der Städte was ihn in der Praxis nur bedingt einsetzbar macht.

Da dem Travelling Salesman Problem die Eigenschaft der $NP-Vollständigkeit$ nachgewiesen werden kann, ist für jede andere optimale Lösung ein im Prinzip ähnliches Laufzeitverhalten zu erwarten (es sei denn, es kann doch noch die Übereinstimmung der Mengen P und NP bewiesen werden).

Aus diesem Grund besteht ein großes Interesse an Algorithmen, die, auch wenn sie nur eine suboptimale Lösung für das TSP finden, in polynomieller Laufzeit zu einem Ergebnis kommen.
Der hier vorgestellte Algorithmus stammt aus dem Bereich der Neuronalen Netze.

Diskussion der Netzwerksstruktur

Kohonens Modell geht ursprünglich von einer zweidimensionalen Neuronenschicht aus, in welche durch eine endliche Zahl von Axiomen Reizimpulse eingeleitet werden. Kommt es nun zu solch einem Impuls, so bewirkt dieser eine lokale Erregung in der Neuronenschicht, dessen Gebiet durch die Lage derjenigen Neuronen bestimmt wird, die am stärksten auf ebendiesen Reiz ansprechen. Gleichzeitig haben auch die Neuronen untereinander Verbindungen und geben eingehende Impulse an ihre Nachbarn weiter, wobei der Reiz auf naheglegene (d.h. im Sinne der Nachbarschaftbeziehung) Neuronen am stärksten erregend wird, der Reiz sich aber mit zunehmender Entfernung vom ursprünglichen Erregungspunkt immer mehr abschwächt, oder sogar hemmend wirkt (lateral interaction, s. [Koh89]).
Durch die einlaufenden Reize werden wiederrum die Synapsen d.h. die Nachbarschaftverbindungen zwischen den Neuronen beinflußt, wodurch sich im Laufe der Zeit eine Selbstorganisation des Netzes ergibt in dem benachbarte Neuronen auf ähnlich Reize reagieren. (s. [RMS91])
Man kann eine Lösung des TSP, also eine Route durch die N Städte, auch als einen eindimensionalen Ring mit N Neuronen ansehen, wobei die Kante zwischen zwei Städten der direkten Nachbarschaft zwischen zwei Neuronen entspricht. Wenn eine Stadt einen Reiz an die Neuronen schickt reagiert derjenige Knoten am stärksten, der den geringsten (euklidischen) Abstand zu dieser Stadt hat. Dabei zieht er seine Nachbarn mit sich, wobei der Reiz, d.h. der Bewegungsimpuls mit zunehmender Entfernung auf dem Ring abnimmt (hiermit wird die lateral interaction simuliert).
Schnell bildet sich so ein durch die gegebene Städte bestimmte Grobstruktur des Neuronenrings heraus. Um nun auch noch die Feinstruktur des Netzes anzupassen, wird die lateral interaction im Laufe der Zyklen immer mehr abgschwächt, bis schließlich die von den Städten ausgehenden Impulse nur noch den jeweilig am besten passenden Neuron beeinflussen, d.h. seine Lage verändern. Dadurch wird dann die endgültige Konvergenz des Netzes erreicht.

Beschreibung von Algorithmus und Implementation

Algorithmus

Der Algorithmus zur Auffindung einer Tour geht im Gegensatz zu diskreten Suchverfahren nicht davon aus, in jedem Iterationsschritt nur korrekte Touren zu untersuchen. Alle Knoten in diesem Algorithmus können sich unabhängig von der Position der Städte in der Ebene bewegen. Es können auch mehr oder weniger Knoten als Städte in einem Iterationsschritt vorliegen. Erst am Ende, wenn der Algorithmus zu einer Lösung hin konvergiert ist, muß eine korrekte (wenn auch nicht optimale) Lösung des TSP vorliegen.

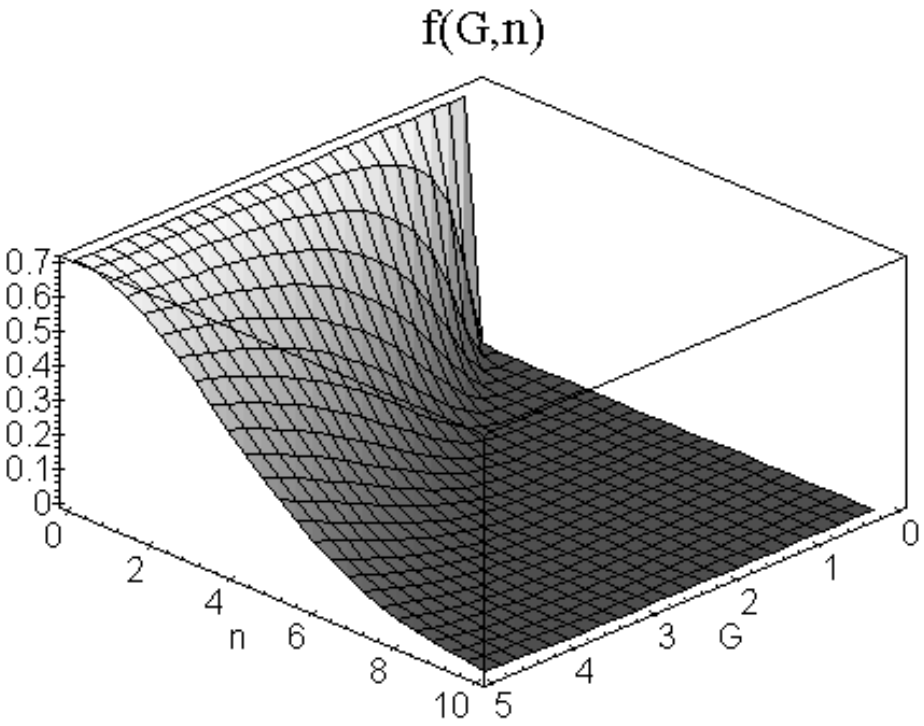
Der Algorithmus läßt sich wie folgt beschreiben (nach [AVT88]):

1. Zu Beginn besteht das Netz nur aus einem einzigen Knoten.
2. In einem Durchlauf werden nacheinander alle Städte präsentiert.
   (a) Für eine päsentierte Stadt i wird derjenige Knoten c_j berechnet, der dieser Stadt am nächsten liegt. Dies erfolgt über den euklidischen Abstand.
   (b) Der so gefundene Knoten c_j und seine Nachbarn auf dem Ring werden zur Stadt i hin verschoben. Der Betrag der Verschiebung ergibt sich dabei aus dem Abstand n eines Knotens j zum Knoten j_c auf dem Ring (nicht in der Ebene) und eines Einflußparameters G. Diese beiden Parameter werden über die Funktion

$$f(G,n) = \frac{1}{\sqrt{2}} \cdot \exp\left(\frac{-n^2}{G^2}\right)$$

   miteinander verknüpft, so daß sich für die Verschiebung der Koordinaten x und y des Knotens j folgende Gleichung ergibt:

$$c_j[x] = c_j[x] + f(G,n) \cdot (x_i[x] - c_j[x])$$
$$c_j[y] = c_j[y] + f(G,n) \cdot (x_i[y] - c_j[y])$$



3. Am Ende jeden Durchlaufs werden folgende Anderungen ausgeführt:

   (a) Der Parameter G wird über die Lernrate α vermindert:

   $$G = G \cdot (1 - \alpha)$$

   (b) Wenn in einem Durchlauf ein Knoten von mehr als einer Stadt als nahegelegenster gewählt wurde, wird dieser verdoppelt. Im Ring wird direkt neben diesem ein neuer Knoten eingehängt, welcher dieselben Koordinaten erhält. Weiterhin werden beide Knoten für den nächsten Durchlauf gesperrt, was bedeutet, daß sie, wenn sie als Gewinner gewählt werden, keine Bewegungen im Netzwerk verursachen. Dadurch wird sichergestellt, daß die beiden Knoten im Laufe des nächsten Durchlauf durch die Bewegungen ihrer Nachbarn auseinandergezogen werden. In der übernächsten Runde wird die Sperre dann wieder aufgehoben.

   (c) Knoten, die in 3 aufeinanderfolgenden Runden von keiner Stadt gewählt werden, werden gelöscht.

Es sollen folgende Bemerkungen zu Einzelheiten des Algorithmus festgehalten werden:

- Durch den Parameter G kann gesteuert werden, wie stark die Nachbarknoten eines gewählten Knotens mitverschoben werden. Für $G \to 0$ wird nur der gewählte Knoten beeinflußt, für große Werte von G werden viele der Nachbarn bewegt. Durch die Abnahme von G mit der Anzahl der Durchläufe wird daher eine Spezialisierung der einzelnen Knoten auf bestimmte Städte erreicht.

- Durch die Möglichkeit des Löschens und Einfügens von Knoten kann zum einen die Konvergenz des Netzes beschleunigt werden, da Knoten, die nicht zur Lösung beitragen, gelöscht werden können, und Knoten an Stellen hinzugefügt werden, an denen zu wenige Knoten vorhanden sind, um die dort vorhandenen Städte abzudecken. Zum anderen ist es dadurch möglich bessere Lösungen zu finden: Das Löschen und Einfügen erlaubt ein sprunghaftes Ändern der Netzstruktur und somit der Position im Fehlerraum. Damit ist es möglich, aus lokalen Minima wieder zu entkommen.

Implementation

Die Implementation des Algorithmus erfolgte in C++ und konnte ohne Änderungen nach dem vorgestellten Algorithmus durchgeführt werden. Bei den verwendeten Datenstrukturen wurden sowohl für die Städte als auch für die Knoten

ringförmig verkettete Listen verwendet. Bei den Knoten des Netzes konnte wegen der Erzeugung und Vernichtung von Knoten keine Arrays verwendet werden, die für eine effizientere Arbeitsweise benötigt würden. Bei den Städten wäre eine Verwendung von Arrays durchaus möglich gewesen, allerdings ergibt sich dadurch ein wesentlich geringerer Performancegewinn und die Verwendung von Listen bot sich dadurch an, daß auch für die Knoten Listen verwendet werden mußten. Die für die Simulation benötigten Parameter werden über eine Konfigurationsdatei zu Beginn eingelesen. Abweichend von dem in [AVT88] vorgestellten Algorithmus wurden folgende Änderungen vorgenommen:

- Die Position des Startknotens wurde auf $(0.5, 0.5)$ gesetzt. Eine Bevorzung einer Ecke der Karte ist nicht einzusehen.
- Bei der Erzeugung eines Knotens wurde die Position des neuen Knotens leicht in Richtung seines zweiten Nachbarns versetzt, um zwei Knoten mit identischen Positionen zu verhindern.

Beide Änderungen hatten aber keine beobachtbaren Folgen.
Folgende Änderungen am Algorithmus wurden in Erwägung gezogen, da sie möglicherweise Verbesserungen darstellen könnten:

- Anstatt mit einem Startknoten könnte mit N Startknoten, die z.B. kreisförmig angeordnet werden, begonnen werden. Dadurch könnten einige Durchläufe am Anfang gespart werden, in denen Knoten erzeugt werden.
- Da eine Lösung genau N Knoten hat, könnte der Algorithmus die Erzeugung von Knoten verhindern, falls schon N Knoten vorhanden sind. Durch die im Prinzip überflüssigen Knoten wird die Dauer der Simulation erhöht.

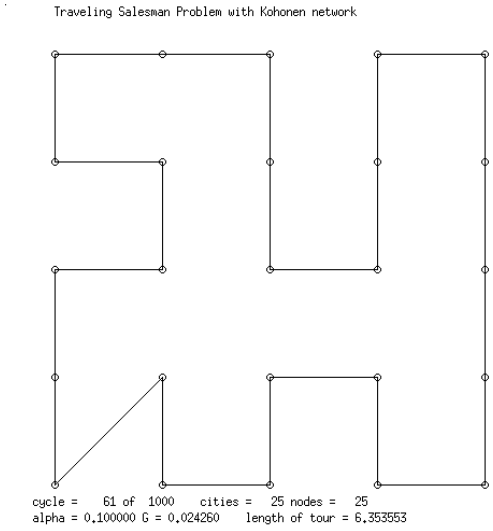
Diese Vorschläge wurden allerdings nicht ausgetestet.

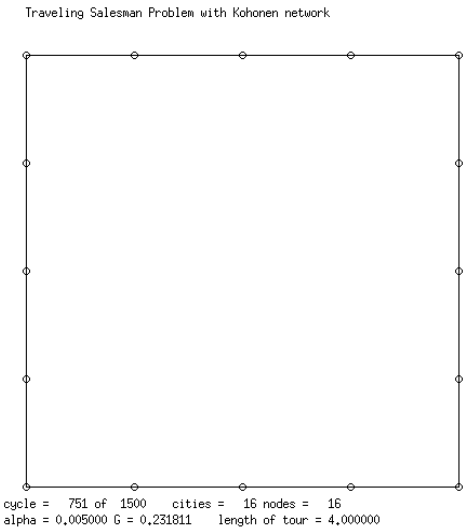
Laufzeitabschätzung

Im Folgenden soll eine grobe Abschätzung des Laufzeitverhaltens des Algorithmus gegeben werden. Für einen Durchlauf, d.h. die Präsentation von N Städten bei M vorhandenen Knoten hat der Algorithmus eine Laufzeit in der Größenordnung von $O(N \cdot M + M)$. Empirische Beobachtungen sowohl in [AVT88] als auch von unserer Seite ergaben $M < 2 \cdot N$. Für die Anzahl D der Durchläufe konnte folgende Abschätzung gemacht werden: $D = \exp(\alpha^{-1})$.
Die Anzahl der Städte hatte auf die Anzahl der Durchläufe bei festgehaltener Lernrate keinen Einfluß. Damit ergibt sich eine grobe Abschätzung der Gesamtlaufzeit von $O\left(N^2 \cdot \exp(\alpha^{-1})\right)$

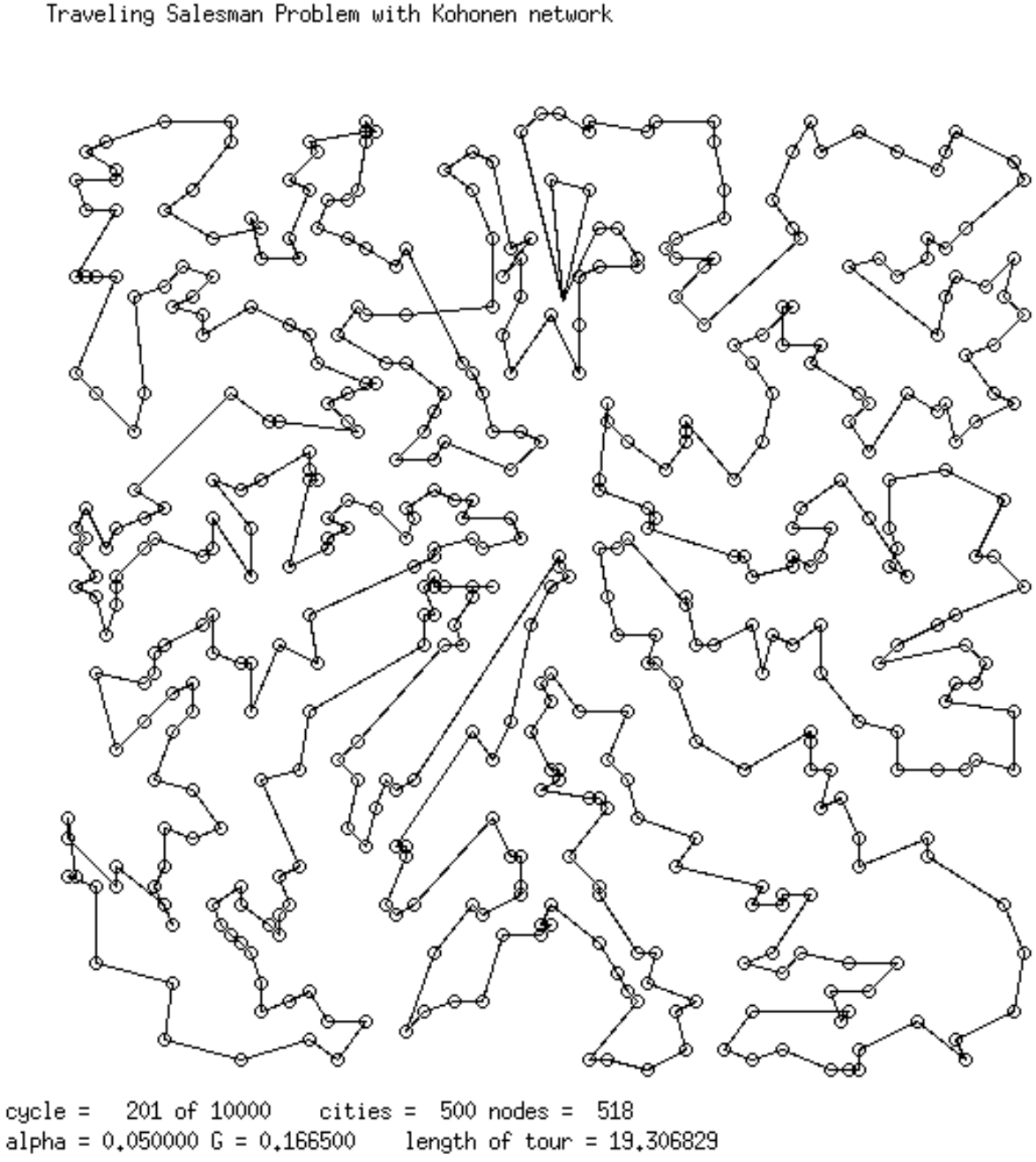
Simulationsergebnisse

Das erste Ziel der Simulationsläufe war es das korrekte Verhalten des Algorithmus und seine Grenzen auszuloten. Dazu wendeten wir das Programm zuerst auf einige Karten mit bekannten kürzeste Weg an. Nach einigem Experimentieren mit der Lernrate wurde in fast allen Fällen die optimale Lösung sehr rasch gefunden.

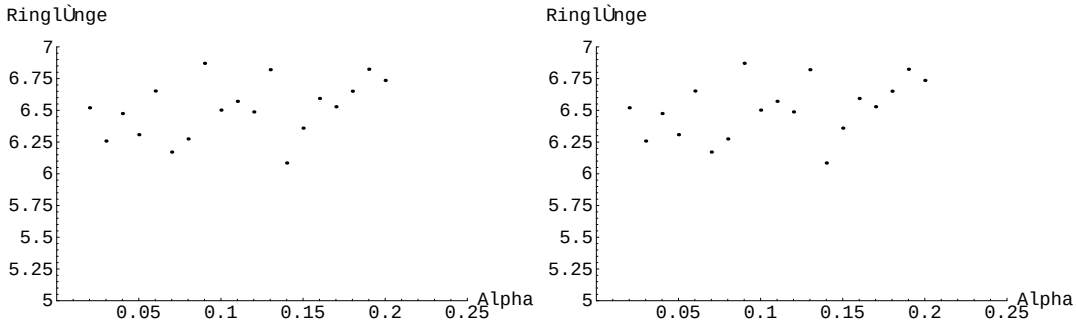




Daraufhin wurde der Algorithmus auf größere Karten angewendet. Dabei zeigte sich, daß auch bei einer großen Zahl von Städten ($N = 500$) innerhalb von wenigen hundert Zyklen ein Weg gefunden wurde, wobei sich die Rechenzeit selbst auf den älteren Rechnern des Cip-pools noch in einem erträglichen Rahmen bewegte.



Ein weiteres Ziel der Experimente war es, ein Zusammenhang zwischen der Lernrate α und der Weglänge in einer festgewählten Zufallskarte zu finden. Dies führte aber leider zu keinem befriedigenden Ergebnis.

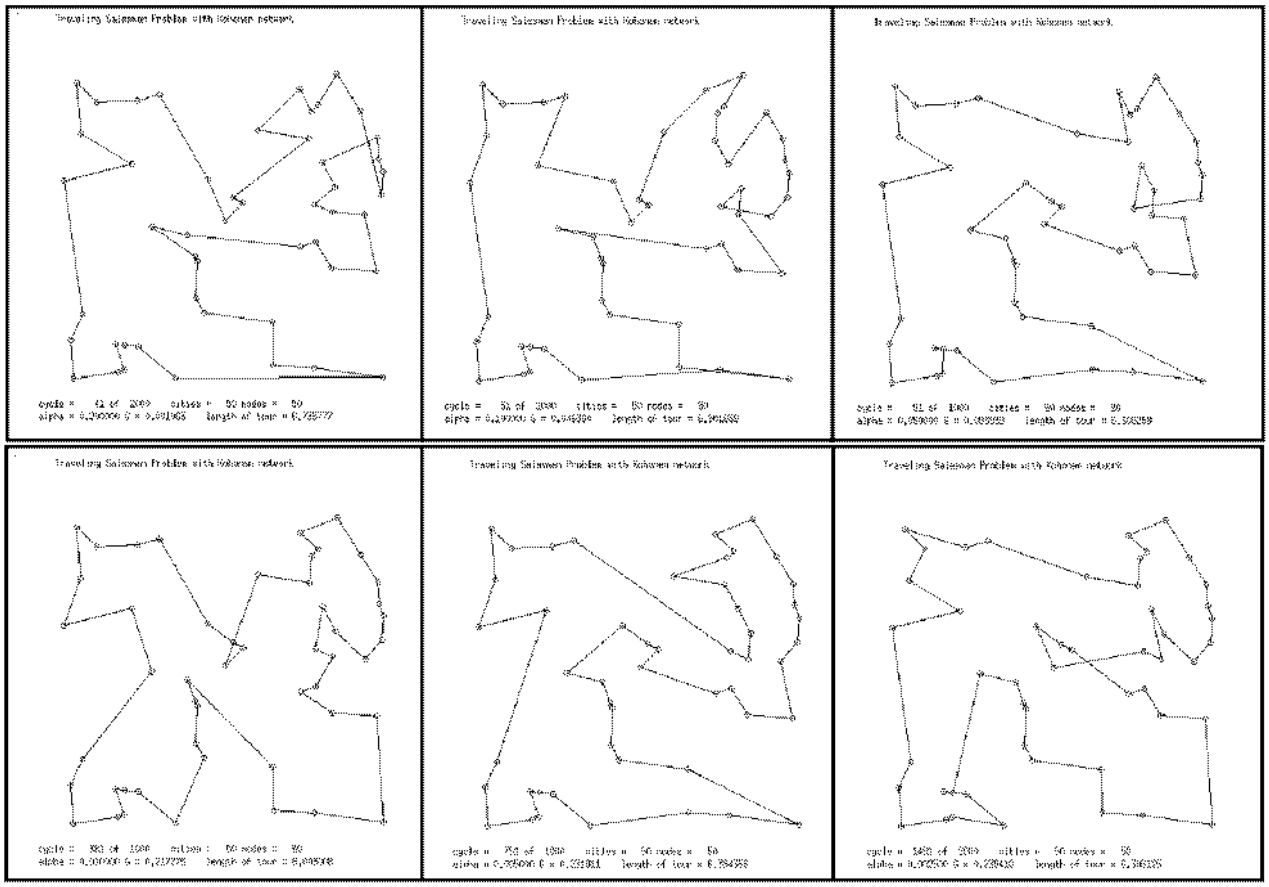


Ringlänge in Abhängigkeit von α für zwei Zufallskarten mit $N = 50$

Es ist anscheinend keinesfalls so, daß man durch eine kleinere Lernrate einen besseren Weg finden würde, wie man zuerst annehmen könnte. Auch ergeben sich auch bei kleineren Wertänderungen von α schon größere Unterschiede der Weglängen, in einigen Fällen mehr als 10%.

Folgende Tabelle und Grafik dokumentieren ein zweites Beispiel.

$N = 50, \alpha$	Weglänge	#Durchläufe
0.2	6.73577	40
0.1	6.50183	50
0.05	6.30825	90
0.01	6.00301	380
0.005	6.39436	750
0.0025	6.30613	1491



$N = 50$, α abnehmend von links oben nach rechts unten, s. Tabelle

Zusammenfassung und Diskussion

Das Travelling Salesman Problem läßt sich durch selbstorganisierende Karten vor allem bei einer kleinen Zahl von Städten gut und vor allem sehr schnell lösen. Gerade bei höheren Lernraten wird innerhalb weniger Schleifendurchläufen eine Lösung gefunden. Aber auch bei einer größeren Zahl von Städten wird selbst bei relativ niedrigen Werten von α schnell ein scheinbar guter Weg gefunden.

Da sich aber leider keinerlei direkter Zusammenhang zwischen dem Wert der Lernrate α und der Länge des Weges finden ließ und man dadurch nur wenig über die Güte des gefundenen Weges aussagen kann, ist die Anwendung des Algorithmus wohl auf solche Bereiche beschränkt in denen es weniger auf den wirklich küerzesten Weg sondern eher auf eine schnelle, wenn auch suboptimale Lösung ankommt.

Literatur

[AVT88] Bernhard Angeniol, Gael De La Croix Vaubois & Jean-Yves Le Texier
Self-Organizing Feature Maps and the Travelling Salesman Problem
Neural Networks, Vol. 1, pp. 289-293, 1988

[Koh89] T. Kohonen
Self-organization and associative memory
3rd Edition, pp. 119-139 Springer-Verlag, 1989

[RMS91] Helge Ritter, Thomas Martinetz & Klaus Schulten
Neuronale Netzwerke
2. Auflage, pp. 67-115, Addison-Wesley, 1991

[Fre92] James A. Freeman & David M. Skapura
Neuronal Networks: Algorithms, Applications and Programming Techniques
pp. 263-290, Addison-Wesley, 1992