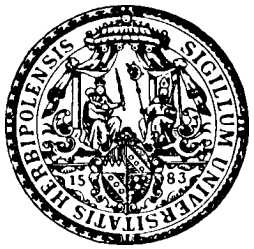


Julius–Maximilians–Universität Würzburg
Institut für Informatik
Lehrstuhl für Verteilte Systeme (Informatik III)

Neuronale Netze

und ihre Anwendungen

Aufgabe 4.3

Selbstorganisierende Karten und das Problem des Handlungsreisenden

Marius Heuler
Jörg Linhart

Angefertigt am
Lehrstuhl für verteilte Systeme (Informatik III)
Bayerische Julius–Maximilians–Universität Würzburg
Betreuer: Prof. Dr.–Ing. P. Tran–Gia
und Dipl.–Inform. Kurt Tutschku

Abstrakt

Unsere Programmieraufgabe bestand darin, das bekannte Problem des Handlungsreisenden („Traveling Salesman Problem“) mit Hilfe von selbstorganisierenden Karten nach Kohonen zu lösen. Hierbei soll eine möglichst kurze Tour zum Besuch aller Städte gefunden werden, wobei die Position der Städte vorgegeben ist. Der hier verwendetete Lösungsansatz legt eine Art Gummiband durch das Gebiet der Städte, das von den Städten angezogen wird und so die endgültige Reiseroute festlegt. Das Band wird dabei von einzelnen Punkten (Nodes) gebildet, wobei auch Nodes hinzugefügt oder gelöscht werden können

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Aufgabenstellung des Netzes 3

2 Diskussion der Netzwerkstruktur **4**

3 Beschreibung des Algorithmus und Implementation **5**

4 Simulationsergebnisse **7**
4.1 Ergebnisse des Berichts 7
4.2 Eigene Untersuchungen 7

5 Zusammenfassung und Diskussion **9**

1 Einleitung

In der Informatik treten häufig Probleme auf, die als sogenannte „NP-vollständige Probleme“ bezeichnet werden. Hierbei bedeutet NP-vollständig, daß die Laufzeit zum Auffinden einer optimalen Lösung exponentiell mit der Größe des Problems anwächst. Diese Probleme stellen eine große Herausforderung an die Algorithmen dar, da für große Problemgrößen Verfahren mit exponentieller Laufzeit in der Regel nicht mehr anwendbar sind. Ein Beispiel für ein NP-vollständiges Problem ist das Problem des Handlungsreisenden. Hierbei soll eine möglichst kurze Rundreise zum Besuch von vorgegebenen Städten gefunden werden.

1.1 Aufgabenstellung des Netzes

Das Netz, welches nach dem in [AVT88] und [Koh88] beschriebenen Verfahren der selbstorganisierenden Karten von Kohonen arbeitet, soll eine möglichst kurze Tour durch eine vorher gegebene Anzahl von Städten finden. Hierbei kann der Ausgangspunkt der Reise frei gewählt werden. Es müssen aber alle Städte besucht und am Ende zum Ausgangspunkt zurückgekehrt werden. Genauer formuliert liegen von M Städte jeweils eine $x-$ und $y-$Koordinate vor und das Netz soll eine Liste der Städte erzeugen, die die Reihenfolge angibt, in der die Städte besucht werden. Dabei gilt folgendens:

$$\begin{aligned} d(i,j) &= \text{Entfernung der Städte } i \text{ und } j \\ s_{l=1\ldots M} &= \text{Nummer der Stadt auf Position } l \text{ der Reise} \\ length &= d(s_1, s_M) + \sum_{k=1}^{M-1} d(s_k, s_{k+1}) \end{aligned}$$

Das Ziel ist es, nun eine Reiseroute zu finden, bei der $length$ minimal wird. Ob man eine optimale Lösung finden kann, bzw. wie gut die gefundenen Lösungen sind, soll im folgenden untersucht werden.
Anschaulich erklärt beruht das hier verwendete Verfahren von Kohonen für das „Traveling Salesman Problem“ (TSM) darauf, eine Art „Gummiband“ durch das Gebiet der Städte zu legen, welches von diesen angezogen wird. Das Band wird dabei durch eine Folge von Einzelpunkten, sog. Nodes, definiert. Wichtig bei diesem Verfahren ist, daß jeder Node, der einer Stadt am nächsten ist, zu dieser hinwandert und dort verbleibt, wenn er die Position der Stadt erreicht hat. Das Gummiband besteht dabei am Anfang aus einem einzigen Punkt (Node), wobei nach einem bestimmen Verfahren Punkte hinzugefügt oder gelöscht werden, je nach dem, ob sie günstig für die Lösung liegen oder nicht.

2 Diskussion der Netzwerkstruktur

In dieser Untersuchung wird ein Netz nach Kohonen verwendet, um das Problem des Handlungsreisenden zu lösen. Das Netz ist ein vorwärtsgekoppeltes Modell, das einfache adaptive Neuronen verwendet. Das Grundsystem besteht aus einem ein- oder zweidimensionalen Feld aus Neuronen mit Rückkoppelung zwischen benachbarten Einheiten. Das essentielle an Kohonens Algorithmus liegt darin, das Netz sich selbst so verändern zu lassen, daß nahe benachbarte Einheiten ähnlich reagieren. Ziel dabei ist es, daß sich jedes Neuron bzw. ein Gebiet von benachbarten Neuronen auf einen Unterbereich des Definitionsbereiches spezialisiert, so daß seine Erregung bei Eingaben aus diesem Bereich maximal wird. Die Neuronen konkurrieren dabei in einem modifizierten „Der-Gewinner-bekommt-alles" („winner-take-all") Vefahren. Das Neuron, das am stärksten reagiert, ist der „Gewinner" und darf dementsprechend seine Ausgabeaktivität weiterleiten. Das Besondere in diesem Modell ist auch, daß nicht nur die Gewichte des Gewinners, sondern auch die seiner Nachbarn ensprechend justiert werden. Die Nachbarschaft wird hierbei je nach Problemstellung definiert.
Zur Lösung des Problems des Handlungsreisenden wurde hier ein eindimensionales Netz in Form eines Rings verwendet. Jedes Neuron k hat also als direkte Nachbarn die Neuronen $k-1$ und $k+1$. Als Nachbarschaftsfunktion wird folgende Funktion verwendet:

$$f(gain, n) \quad = \quad 1/\sqrt{2} * e^{-n^2/gain^2}$$

Die Nachbarschaftsfunktion gibt an, wie stark Neuron i während des Lernens an das Nachbarschaftszentrum k gekoppelt ist. Der Parameter n ist hier die Entfernung der Neuronen i und k auf dem Ring. Die Verwendung dieser Funktion $f(gain, n)$ bewirkt immer eine Adaption aller Neuronen auf dem Ring, wobei aber der Grad der Justierung negativ exponentiell mit dem Abstand n abnimmt. Der Parameter $gain$ wird im Laufe der Lernphase verringert, bis am Ende der Lernphase quasi nur noch einzelne Neuronen berücksichtigt werden.

3 Beschreibung des Algorithmus und Implementation

Am Anfang des Programms werden beliebig die Koordinaten der Städte ausgewählt. Danach wird auch zufällig eine Reihenfolge der Städte definiert, in der sie nachher in jedem der Iterationsschritte besucht werden. Die zufällige Reihenfolge der Städte ist wichtig, um bei gleicher Lage der Städte unterschiedliche Routen erzielen zu können. Das Ergebnis des Algorithmus hängt nämlich auch von der Reihenfolge der Untersuchung ab. Die Position der Städte und Nodes wird als $x-$ und $y-$Koordinate in den Arrays ($c\_x[]$, $c\_y[]$ bzw. $v\_x[]$, $v\_y[]$) gespeichert, wobei die Variable M die Anzahl der Städte, und N die Anzahl der Nodes angibt. Die Nodes bilden dabei logisch gesehen einen Ring, in dem jeder Node mit seinen Nachbarn korrelliert ist.
Zu Beginn wird nur ein Node an der Position $(0,0)$ erzeugt und die Iteration gestartet. Bei jedem Iterationsschritt wird jede Stadt genau einmal besucht, also werden insgesamt M Schritte pro Iteration durchgeführt. Wie schon oben ausgeführt wurde, wird die Reihenfolge in der die Städte besucht werden am Anfang per Zufall bestimmt.

Iterationsschritt

Ein Iterationsschritt des Algorithmus besteht im Besuchen jeder einzelnen Stadt. Für jede besuchte Stadt i werden dann folgende Schritte durchgeführt:

- Zuerst wird der Node gesucht, der der Stadt am nächsten liegt:

$$dist_j = (c_x[i] - v_x[j])^2 + (c_y[i] - v_y[j])^2 \quad (1)$$
$$dist_{j_c} = \min dist_j \quad (2)$$

  j_c ist nun die Nummer des Nodes, der am nächsten bei der Stadt i liegt.

- Nun wird der Node j_c und seine Nachbarn auf dem Ring zu der Stadt i hinbewegt. Die Distanz, die jeder Node dabei zurücklegt, wird durch die Funktion $f(gain, n)$ (Gl. (6)) festgelegt, wobei n die Entfernung auf dem Ring zwischen Node j und j_c angibt:

$$n = \inf((j - j_c) \,\%\, N,\ (j_c - j) \,\%\, N) \quad (3)$$

  Jeder Node wird nun an seine neue Position bewegt:

$$v_x[j] = v_x[j] + f(gain, n) * (c_x[i] - v_x[j]) \quad (4)$$
$$v_y[j] = v_y[j] + f(gain, n) * (c_y[i] - v_y[j]) \quad (5)$$

  Wobei hier die Funktion $f(gain, N)$ folgendermaßen definiert wird:

$$f(gain, n) = 1/\sqrt{2} * e^{-n^2/gain^2} \quad (6)$$

Dies bedeutet:

- Falls $gain \to \infty$; alle Nodes wandern zur Stadt i mit Stärke $(1/\sqrt{2})$
- Falls $gain \to 0$; nur Node j_c wandert zur Stadt i.

Der Parameter $gain$ bestimmt dabei die Größe der Bewegung der Nodepositionen. Nach jeder kompletten Iteration wird der Wert von $gain$ vermindert:

$$gain = (1 - \alpha) * gain \tag{7}$$

Die Auswahl von α, sowie die Start- und Endwerten für den Parameter $gain$ werden in Kapitel 4.1 beschrieben. Da am Anfang des Algorithmus mit nur einem Node begonnen wird, aber am Ende die Nodes auf den Positionen der Städte liegen sollen, müssen noch Funktionen zum Erzeugen bzw. Entfernen von Nodes vorhanden sein.

Erzeugen von Nodes

Ein Node wird verdoppelt, falls er in einem Iterationsschritt, von mehr als einer Stadt als Gewinner gewählt wurde, d.h. er wurde von mehreren Städten als am nächsten liegender Node ausgewählt. Ein Iterationsschritt ist dabei als ein kompletter Durchlauf durch alle Städte definiert. Der neue Node bekommt die gleichen Koordinaten wie der alte. Beide Nodes werden aber für einen Durchgang gesperrt, d.h. sie werden beim nächsten Durchgang, falls sie als Gewinner gewählt werden, nicht bewegt. Nach dem nächsten Durchgang werden sie wieder aktiviert und normal bewegt. Dies garantiert, daß die beiden Nodes durch die Bewegungen ihrer Nachbarn getrennt werden, bevor sie von den Städten eingefangen werden. Die Trennung der beiden Nodes mit den gleichen Koordinaten wird durch Gl. (6) sichergestellt. Da sie auf dem Ring nicht die gleiche Position haben, und die Bewegung der Nodes von der Ringposition abhängt, werden sie unterschiedlich bewegt.

Entfernen von Nodes

Ein Node wird entfernt, falls er während dreier aufeinanderfolgender Durchgänge nicht als Gewinner gewählt wurde, er also nicht der nächste Nachbar irgendeiner Stadt während der drei Durchläufe war.

Simulationslauf

Das Erzeugen von Nodes ist wichtig, damit zu jeder Stadt ein Node wandern kann, wenn Nodes zwischen den Städte liegen. Zu Beginn tritt dies immer auf, da weniger Nodes als Städte vorhanden sind. Das Entfernen von Nodes verbessert die Qualität der Lösung, da Nodes, die nicht optimal liegen, entfernt werden und so anderen Nodes, die besser liegen, den Weg frei machen. Im Verlauf des Algorithmus steigt die Anzahl der Nodes von einem Node, ganz am Anfang, bis zu etwa zweimal Anzahl der Städte an und sinkt dann wieder auf die Anzahl der Städte ab, wobei dann jeder Nodes zu „seiner“ Stadt wandert.

4 Simulationsergebnisse

4.1 Ergebnisse des Berichts

Der Algorithmus, der oben beschrieben wurde, ist das Ergebnis von verschiedenen Variatonen der Grundidee von Kohonens selbstorganisierenden Karten. Er ist auch sehr stabil in Bezug auf Parameterveränderung. Der Parameter α, der Startwert für *gain* und das Abbruchkriterium sind die einzigen Parameter, die angepaßt werden müssen. Hierbei ist der Startwert von *gain* in weiten Bereichen wählbar, ohne das Endergebnis zu beeinflussen. Die wenigen Nodes verweilen am Anfang gemäß dem Algorithmus um den „Schwerpunkt" des Städtesystem, da sie von allen Städten gleichermaßen angezogen werden. Deshalb ist das Gewicht der Anziehung, also der Startwert für *gain*, zu Beginn unkritisch. Das Abbruchkriterium kann vielfältig gewählt werden. Am Besten ist wohl dann abzubrechen, wenn jede Stadt einen Node eingefangen hat und dieser an der Positon der Stadt angekommen ist. Diese Prüfung ist aber gar nicht nötig. In der Praxis hat es sich gezeigt, daß bei einem kleinen Wert von *gain*, etwa $gain \leq 0.1$, die Nodes an den Städten angekommen sind. Nach Gl. (4) und Gl. (6) kann dann kaum noch eine Bewegung stattfinden. Der einzige Parameter, der wirklich angepaßt werden muß, ist der Parameter α in der Gl. (6). Denn dieser bestimmt direkt die Anzahl voller Iterationen bis zum Endergebnis und damit natürlich die Qualität dieses Ergebnisses. In dem Bericht von [AVT88] werden für α Werte im Bereich von etwa 0.2 bis 0.01 verwendet. Es zeigt sich, daß die Wahl des Parameters die Qualität nur wenig beeinflußt. Dort wird ein Vergleich durchgeführt und es ergibt sich folgendes Ergebnis: Für $\alpha = 0.02$ ist die mittlere Weglänge 4.36, und bei $\alpha = 0.2$ ist sie 4.38. Man sieht, daß sich das Ergebnis kaum verschlechtert, wenn man α verzehnfacht, wobei natürlich auch die Rechenzeit auf ein zehntel absinkt. Die optimale Weglänge in diesem Beispiel ist übrigens 4.267 und wird mit einer Wahrscheinlichkeit von nur 0.15 % erreicht. Dies unterstreicht die in der Einleitung schon beschriebenen Möglichkeiten von neuronalen Netzen. Man bekommt nur sehr selten die optimale Lösung heraus. Im Normalfall wird eine sehr gute, aber nur suboptimale Lösung erreicht. Dafür ist das Verfahren aber sehr schnell, was vor allem bei einer sehr großen Anzahl Städte wichtig ist. Verfahren, um die optimale Lösung zu finden scheiden dann wegen der extremen Rechenzeit aus. Im allgemeinen reicht eine sehr gute Lösung für die Praxis aus. Die optimale Lösung ist in den meisten Bereichen nicht nötig.

4.2 Eigene Untersuchungen

Programmlauf

Für unsere Untersuchungen wurde für α ein Wert von 0.02 genommen. Wie in Kapitel 4.1 beschrieben, reicht normalerweise ein größerer Wert von α aus, um sehr gute Ergebnisse zu erzielen. Da die Rechenzeit aber selbst bei 1000 Städten auf einer normalen Workstation erträglich war, wurde α so klein gewählt. Für eine erste Darstellung des Programmlaufs wurden 50 Städte gewählt. In Abb. 1

ist ein kompletter Durchlauf des Programms dargestellt. Man sieht deutlich, wie sich am Anfang die Nodes um den „Schwerpunkt“ des Systems versammeln und das Band der Nodes dann langsam auf das gesamte Gebiet der Städte verteilt und von den einzelnen Städten angezogen wird, bis jeder Node auf einer Stadt zur Ruhe gekommen ist. Außerdem kann man gut erkennen, wie die Anzahl der Nodes erst zunimmt und sich dann, hier nach etwa 170 Iterationen, auf die Anzahl der Städte einstellt. Wenn die Anzahl der Nodes wieder auf die Anzahl Städte gesunken ist, wandern nur noch die Nodes auf die Position der Städte und es werden dann keine Nodes mehr entfernt oder hinzugefügt.

Verschiedene Routen

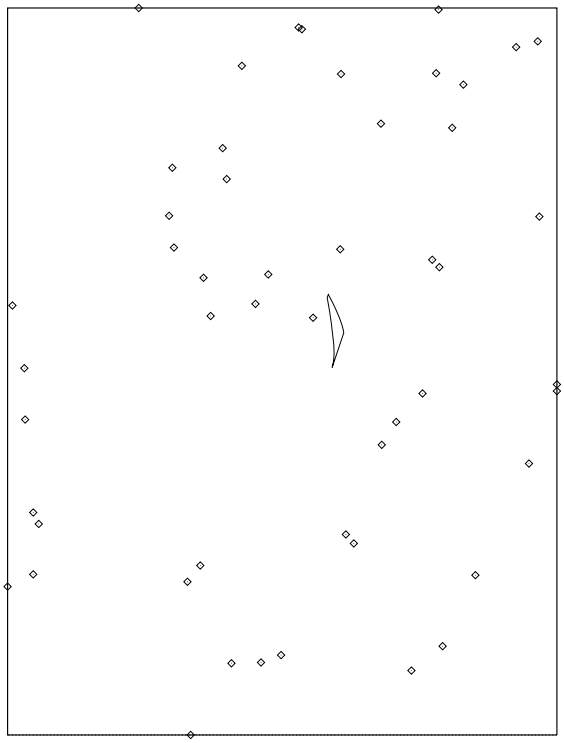
In einem zweiten Beispiel wurden wieder 50 Städte gewählt. Es soll nun dargestellt werden, wie sich die vom neuronalen Netz gefundenen Wege unterscheiden. Bei dem hier verwendeten Algorithmus hängt der gefundene Weg bei gleicher Lage der Städte nur von dem Parameter α und der Reihenfolge der Städte bei der Iteration ab. Die Auswirkungen bei einer Veränderung von α wurden schon in Kapitel 4.1 beschrieben. Bei 50 Städten wirkt sich eine Veränderung kaum aus, nur die durchschnittliche Länge des Weges erhöht sich geringfügig. Um die Auswirkung, die im Bericht von [AVT88] beschrieben sind, darzustellen müßte man statistische Untersuchungen anstellen. Um verläßliche Ergebnisse zu bekommen, sind daher eine große Anzahl von Testläufen nötig (im Bericht von [AVT88] z.B. 20000). Dies war aus zeitlichen Gründen nicht möglich. Daher wurde die Untersuchung auf die Auswirkung unterschiedlicher Reihenfolgen beim Besuchen der Städte beschränkt. In Abb. 5 sind vier verschiedene Ergebnisse aufgetragen, wobei sich die Eingabe für das Netz nur in der Reihenfolge der Städte unterscheidet. Man kann die unterschiedlichen Wege erkennen, die das Netz gewählt hat. Dabei wird deutlich, daß bestimmte Teilrouten schon optimal sind, d.h. sie sind in jeder Route gleich und stellen den optimalen Weg für dieses Teilgebiet dar. In anderen Bereichen der Karte scheinen mehrere vollkommen verschiedene Wege kurze Routen zu ergeben. Das Problem für das neuronale Netz ist hier, daß viele teilweise sehr verschiedene Wege sehr gute Lösungen ergeben, wobei sich die Länge kaum unterscheidet. Die optimale Lösung ist dabei nur unwesentlich kürzer als die vielen suboptimalen. Energetisch bedeutet dies für das Netz, daß sehr viele lokale Minima vorhanden sind, die sich in der Tiefe kaum vom absolutem Minimum unterscheiden. Deswegen stabilisiert sich das Netz meist in einem lokalen Minimum, was zwar i.a. sehr gute Wege ergibt, aber selten den optimalen. Dies gilt vor allem bei einer großen Anzahl von Städten, bei der es natürlich viel mehr Lösungen gibt.
Ein Problem, das auch bei diesem Algorithmus auftreten kann, sind Überschneidungen der Wege. In Abb. 2 ist hierfür ein Beispiel mit 150 Städten. Mit weniger Städten, z.B. 100, sind bei einigen Versuchen keine Überschneidungen aufgetreten. Im rechten Weg sind zwei Überschneidungen enthalten. Diese Wege sind natürlich trotzdem korrekt, aber eine Überschneidung bedeutet immer einen Verlust im Vergleich zu einer anderen Streckenführung. Meist kann durch eine loka-

le Anpassung die Überschneidung aufgehoben werden, z.B. durch Vertauschung zweier Städte auf der Route, und die Strecke dadurch verkürzt werden. Es gibt einige Algorithmen auf heuristischer Basis, die auch nach diesem Prinzip verfahren.

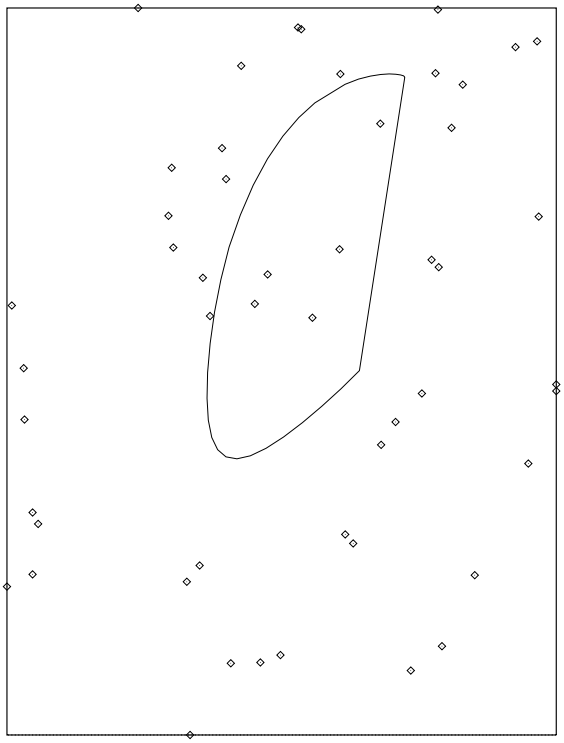
5 Zusammenfassung und Diskussion

Wie in Kapitel 4.2 schon beschrieben, sind die Ergebnisse des verwendeteten Algorithmus meist nicht optimal. Dies gilt eigentlich für alle neuronalen Ansätze, bei einer Problemstellung dieser Art. Nichtsdestoweniger sind die gefundenen Weglängen von sehr guter Qualität. In der Praxis wird i.a. auch nicht die optimale Lösung sondern eine sehr gute Lösung gebraucht. Diese muß aber in einer vernünftigen Zeit vorliegen, was viele Verfahren bei großer Anzahl Städte schon ausschließt. Dieser Algorithmus ist dagegen relativ schnell. Bei 150 Städten ist er in unter einer Minute fertig. Bei einem Durchlauf mit 1000 Städten dauert es etwa 15 Minuten. Hierbei ist anzumerken, daß alle Durchläufe mit $\alpha = 0.02$ durchgeführt wurden. Wie in Kapitel 4.1 beschrieben, kann α ohne große Qualitätseinbußen auf 0.2 erhöht werden, was die ganze Berechnung um den Faktor 10 beschleunigt. Alle Angaben beruhen auf Meßungen auf einer Sun Sparc 20 (sisyphus). Im Vergleich zu anderen neuronalen Ansätzen, wie dem von Hopfield (vgl. [HT86]) sind die Ergebnisse wirklich hervorragend. Vor allem ist zu betonen, daß dieser Algorithmus immer zulässige Wege findet, wobei jede Stadt wirklich nur einmal besucht wird. Dies ist z.B. bei dem Verfahren nach Hopfield ein großes Problem. Abschließend kann man sagen, daß dieser Algorithmus eine sehr brauchbare Lösung für das Problem des Handlungsreisenden liefert. Natürlich gibt es noch bessere bzw. schnellere Verfahren, auch aus dem Neuronalen Bereich. Dafür ist der hier gewählte Ansatz relativ einfach, sowohl in der Implementation, als auch im Verständnis.

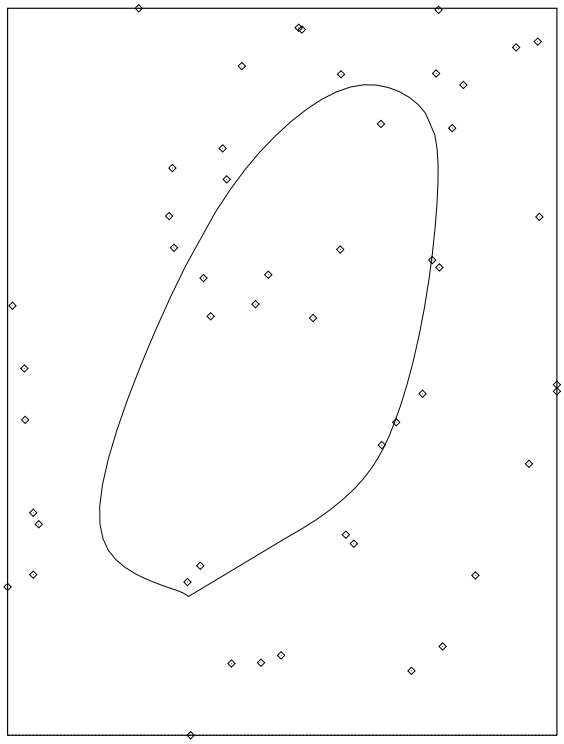
10 Iterationen, 17 Nodes



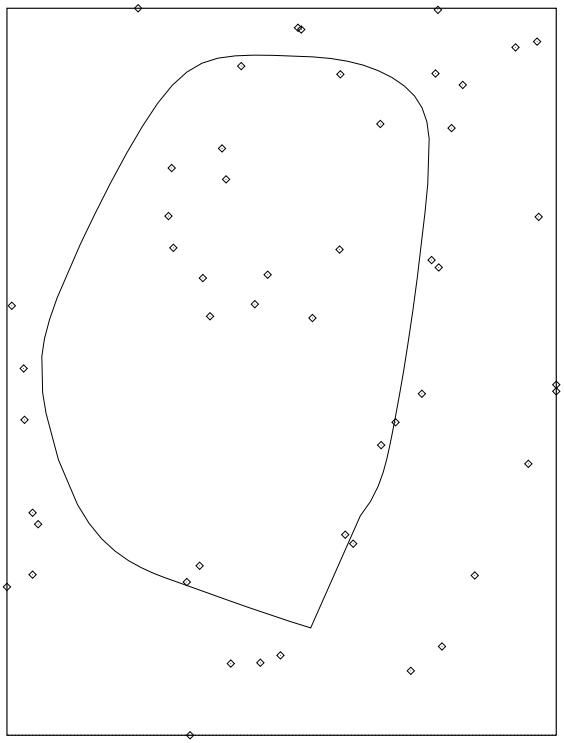
20 Iterationen, 40 Nodes



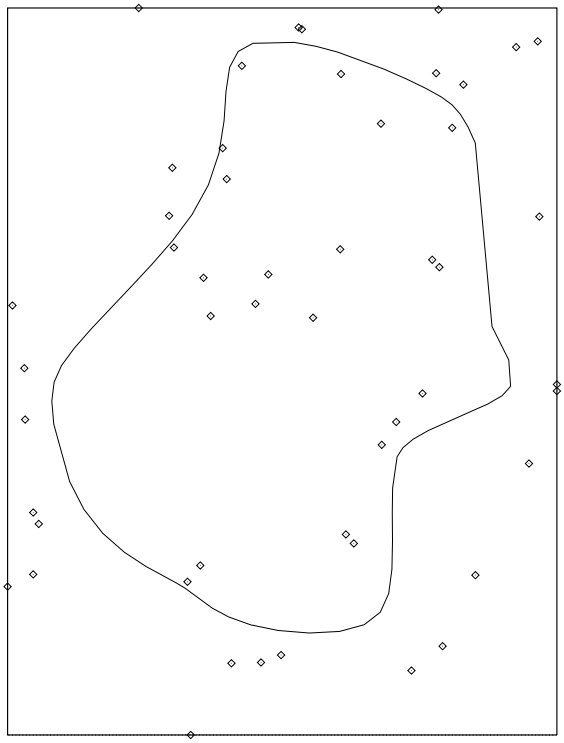
30 Iterationen, 71 Nodes



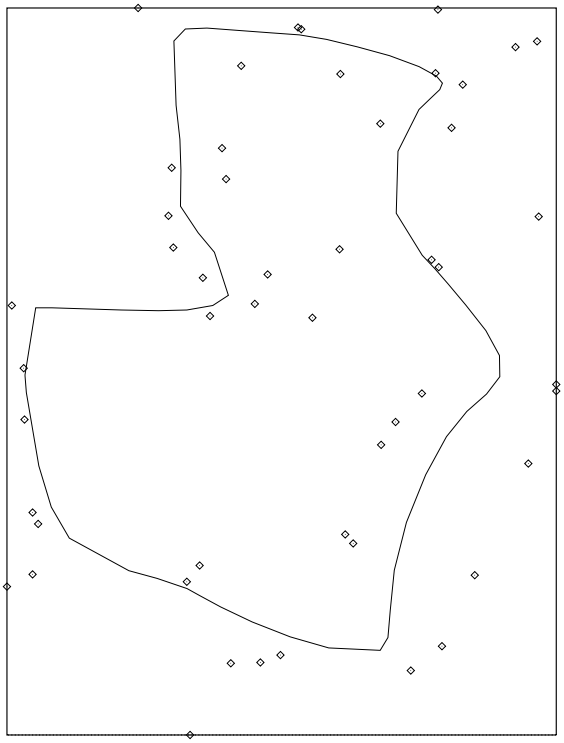
40 Iterationen, 70 Nodes



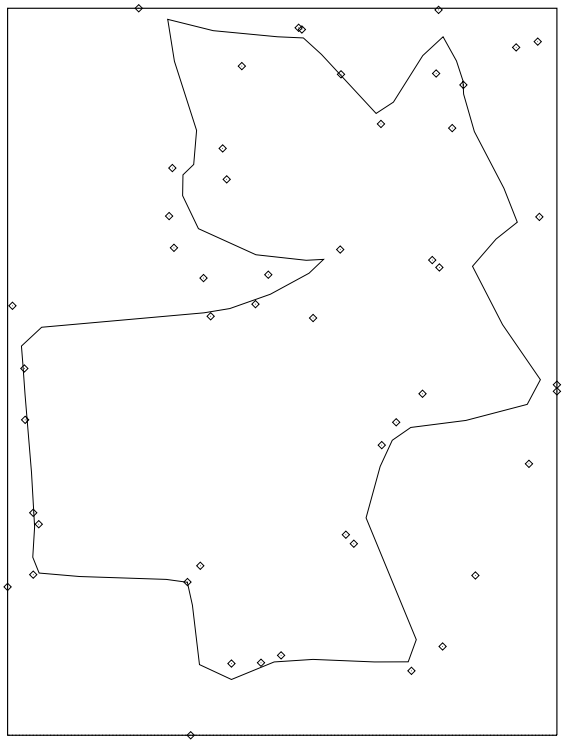
60 Iterationen, 69 Nodes



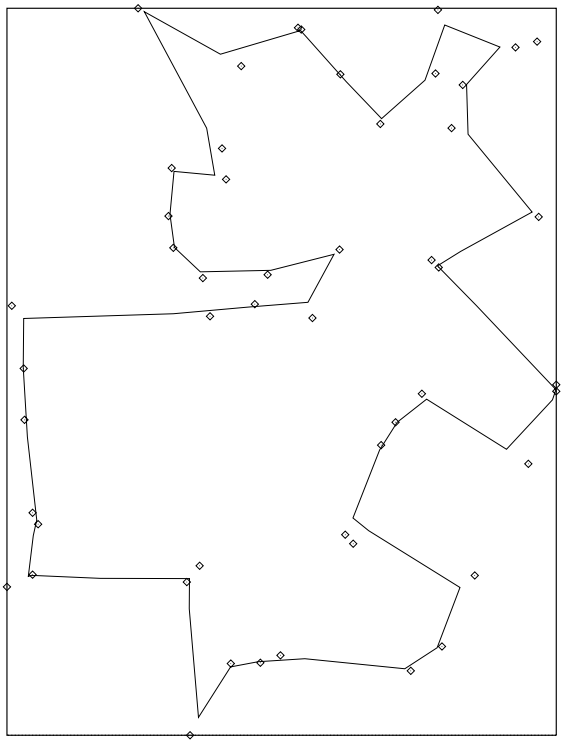
80 Iterationen, 62 Nodes



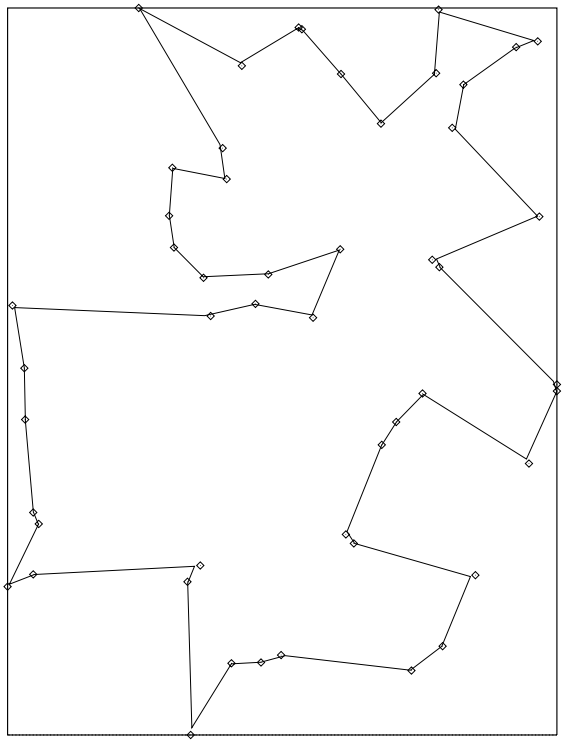
120 Iterationen, 64 Nodes



160 Iterationen, 56 Nodes



180 Iterationen, 50 Nodes



Endergebnis, 50 Nodes

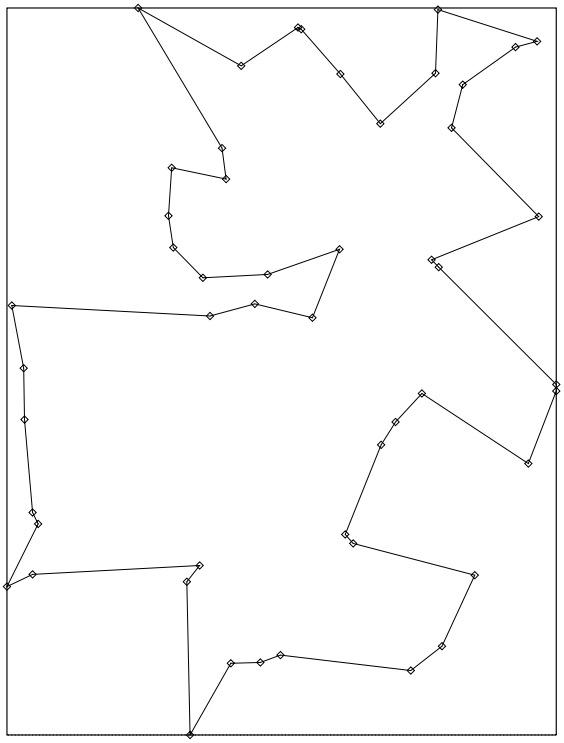


Abbildung 1: Beispiel für einen Durchlauf mit 50 Städten

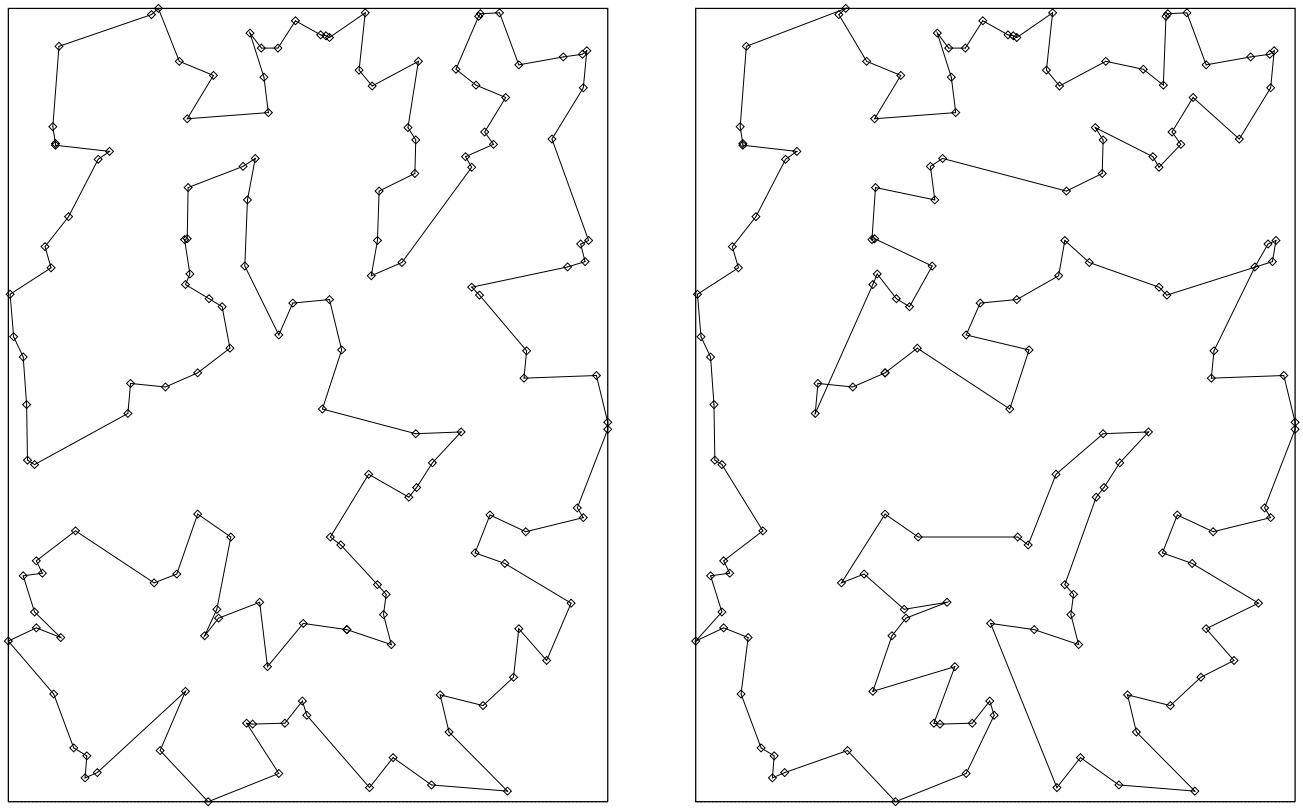
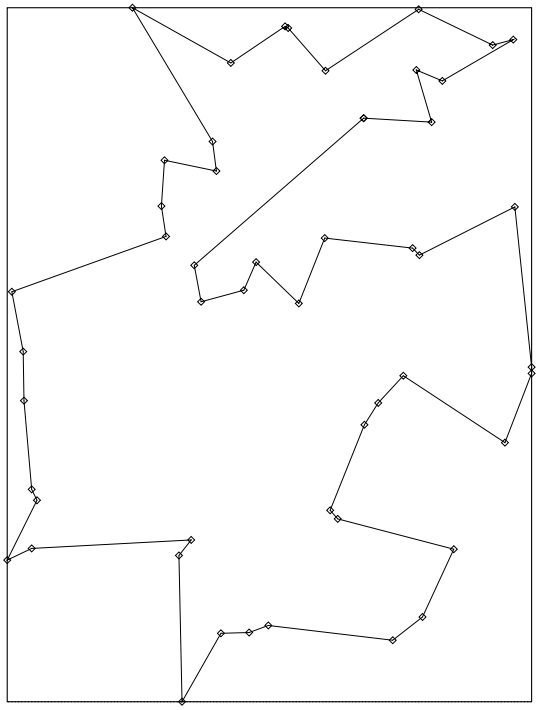
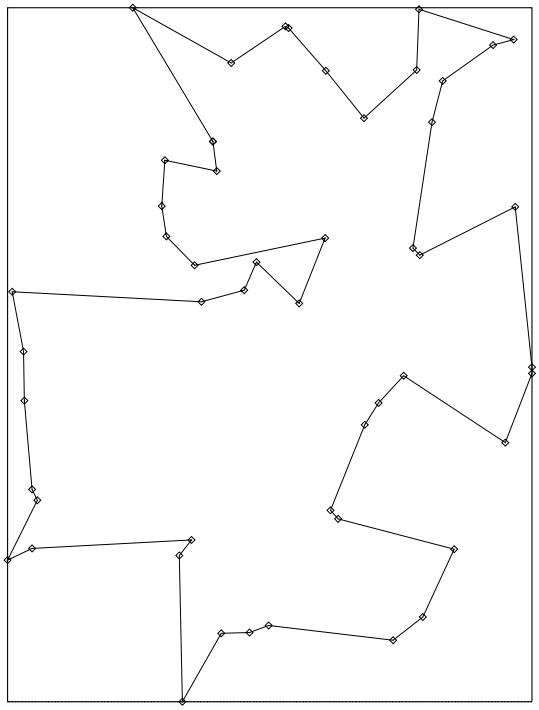


Abbildung 2: Beispiel für Schleifen bei 150 Städten

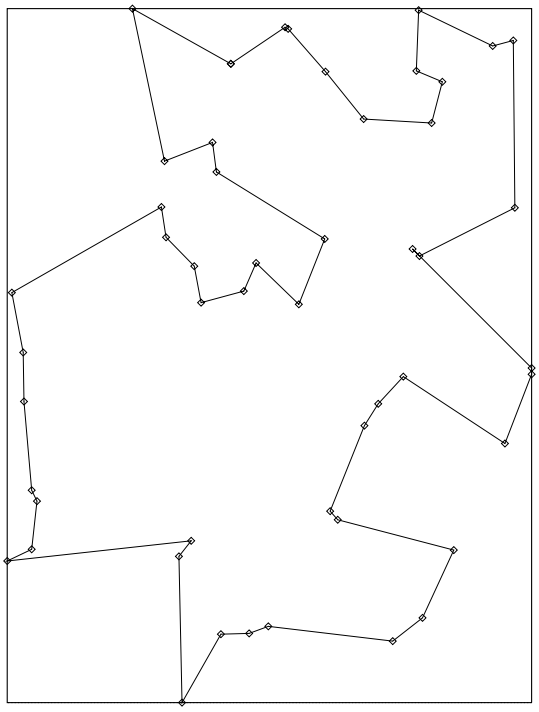
Route 1, Länge 4521



Route 2, Länge 4445



Route 3, Länge 4425



Route 4, Länge 4388

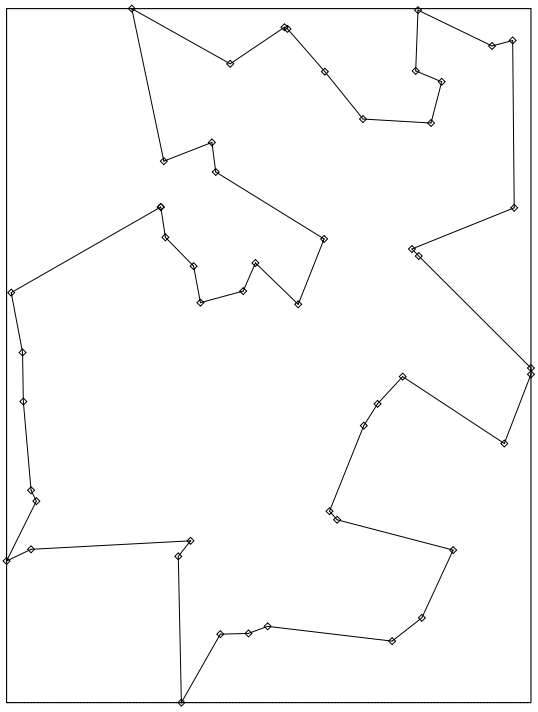


Abbildung 3: Beispiel für verschiedene Routen bei gleichem Stadtplan, 50 Städte

Literatur

[AVT88] B. Angèniol, G. D. L. C. Vaubois, and J.Y.L. Texier. Self-organizing feature maps and the Travelling Salesman Problem. *Neural Networks*, 1(4):289–293, 1988.

[HT86] J.J. Hopfield and D.W. Tank. Computing with neural circuits. a model. *Science*, 233(625), 1986.

[Koh88] T. Kohonen. *Self-Organization and Associative Memory.* Springer Series in Information Sciences. Springer-Verlag, Berlin Heidelberg New York, 2 edition, 1988.