Computational Intelligence

M. Sc. Andreas Buschermöhle, M. Sc. Jan H. Schoenke

Übungsblatt 4 - Schwarmalgorithmen: PSO, ACO & Stigmergie

Abgabe bis Mittwoch, 04.06.2014, 12:00 Uhr

In diesem Übungsblatt werden Algorithmen in Matlab geschrieben und zur Auswertung ein- und zweidimensionale Plots angefertigt. Erstellen Sie im Dateiordner Ihrer Gruppe in Stud.IP einen Ordner mit dem Namen `Blatt 4` und laden Sie zu zweit sowohl die m-Files, als auch die fertigen Matlab-Figures mit sinnvoller Achsenbeschriftung rechtzeitig in diesen Ordner hoch. Geben Sie in Vips je nach Aufgabenstellung die textuelle Interpretationen Ihrer Ergebnisse sowie ggf. die Namen der Dateien an, die sich auf die jeweilige Aufgabe beziehen.

Aufgabe 4.1: Partikelschwarmoptimierung (20 + 10 + 10 + 10 + 0 = 50 P)

Zunächst soll die Partikelschwarmoptimierung (PSO) an einem Beispiel untersucht werden. Die N-dimensionale Funktion

$$f(\vec{x}) = -20 \cdot \exp(-0.2 \cdot \sqrt{\frac{\sum_{i=1}^{N}(x_i - \pi)^2}{N}}) - \exp(\frac{\sum_{i=1}^{N}\cos(2\pi \cdot (x_i - \pi))}{N}) + 20 + e$$

soll auf dem Eingabebereich von $[-2\pi, 2\pi]^N$ minimiert werden. Zunächst betrachten wir die Funktion für $N = 2$, also in 2 Dimensionen. Hier lässt sich das Verhalten der Partikel anschaulich visualisieren und nachvollziehen. Das Grundgerüst für einen einfachen PSO in Matlab ist in Stud.IP vorgegeben. Ebenfalls liegt die zu optimierende Funktion als m-File vor.

a) Schreiben sie in einer Datei `PSO_a.m` den Algorithmus zur Partikelschwarmoptimierung für die gegebene Funktion nach dem Schema, das in der Vorlesung vorgestellt wurde. Als Abbruchbedingung für die Minimums-Suche sollte ein Optimum kleiner als 0.1 gefunden sein und sich das beste Ergebnis des Schwarms um weniger als 0.1 in einem Schritt ändern. Zur Parametrierung des Algorithmus müssen Sie die Anzahl der Partikel und die minimale, sowie maximale Geschwindigkeit wählen. Als Nachbarschaftsbedingung sollen die beiden Nachbarn in der Matrixrepräsentation (also die Zeile vor und nach einem Partikel) gewählt werden. Das erste Element ist dabei auch mit dem letzten Element der Matrix benachbart. Visualisieren Sie das Verhalten ihrer Partikel in jedem Schritt.

b) Ermitteln Sie nun die Anzahl der benötigten Iterationen bis der Algorithmus terminiert, gemittelt über 100 Durchläufe. Wie viele Funktionsauswertungen benötigen Sie im Mittel, um ein ausreichendes Minimum zu finden?

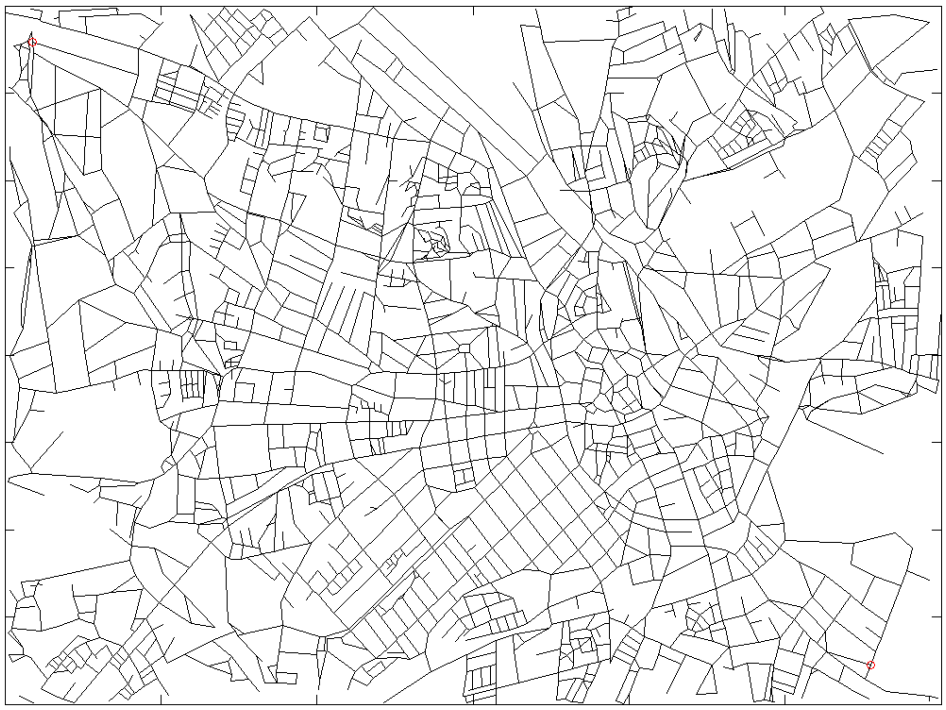
c) Zum Vergleich des PSO sollen zwei naive Strategien herangezogen werden. Eine Möglichkeit ist es, die Funktion an zufälligen Stellen auszuwerten. Lassen Sie die gleiche Anzahl an Auswertung wie in Aufgabenteil b) ermittelt zufällig durchführen und merken Sie sich den kleinsten gefundenen Wert. Führen Sie auch hier 100 Durchläufe durch. Welches ist der durchschnittliche Wert des gefundenen Minimums?

   Eine andere naive Methode ist es, den Eingangsraum mit einem regelmäßigen Gitter abzurastern. Wie fein wäre ein gleichmäßiges Gitter, das in etwa die gleiche Anzahl an Funktionsauswertungen zur Folge hätte wie in Aufgabenteil b) ermittelt? Mit welcher Genauigkeit würde sich dadurch ein Minimum finden lassen? Vergleichen Sie den PSO gegen die beiden naiven Ansätze auf Grund der ermittelten Werte.

d) Um zu sehen, wie sich der Vergleich der Minimierungsansätze mit größeren Eingangsräumen verhält, sollen nun die vergleichenden Untersuchungen aus Aufgabenteil b) und c) auch für $N = 5$, also in 5 Dimensionen, durchgeführt werden. Erweitern Sie hierzu ihren PSO so, dass die Position und der Geschwindigkeitsvektor fünfdimensional sind. Wie gut schlägt sich der PSO in diesem größeren Eingangsraum gegen die naiven Ansätze?

e) *Freiwillig:* Tragen Sie über die Dimensionen die Gütemaße der beiden naiven Ansätze im Vergleich zum PSO auf und interpretieren Sie das Ergebnis.

Aufgabe 4.2: Ameisenkolonie-Optimierung (30 P)



In dieser Aufgabe soll die einfache Ameisenkolonie-Optimierung (S-ACO) erweitert und zur Routenoptimierung in einem gewichteten Graphen verwendet werden. Die Topologie des Graphen ist in obiger Abbildung dargestellt und entspricht (in guter Näherung) einem Ausschnitt aus dem Stadtplan von Osnabrück. Die Knoten des Graphen repräsentieren Kreuzungen und Endpunkte von Straßen. Die Kantengewichte entsprechen den Entfernungen zwischen den Knoten (entlang der Straße). Ihre Aufgabe ist es den S-ACO Algorithmus in Matlab umzusetzen und um eine *Hall of Fame* zu erweitern, indem Sie in jedem Schritt den kürzesten jemals gefundenen Weg speichern. Ermitteln Sie nun mit Hilfe dieses ACO die kürzeste Route zwischen den in der Abbildung markierten Punkten. In Stud.IP finden Sie dazu eine .mat-Datei mit den Daten des Graphen und den Positionen der Knoten. Der Index des Startknotens in den vorgegebenen Daten ist 66, der Index des Zielknotens 2393.

Dokumentieren Sie Ihr Vorgehen bei der Lösung der Aufgabe und stellen Sie das Ergebnis Ihrer Optimierung graphisch dar! Welches Abbruchkriterium verwenden Sie? Wie viele Ameisen enthält Ihre Kolonie? Wie entfernen Sie Schleifen aus den Pfaden der Ameisen? Wie lange dauert die Berechnung eines Optimierungsschrittes im Verlauf der Optimierung? Wie verändert sich die minimale, mittlere und maximale Weglänge der einzelnen Ameisen im Verlauf der Optimierung? Interpretieren Sie Ihre Ergebnisse.

Aufgabe 4.3: Selbstorganisation zur Nahrungssuche (20 P)

In dieser Aufgabe soll die Selbstorganisation von einzelnen Agenten mit jeweils nur lokaler Sicht untersucht werden. In einer Rasterkarte sind dazu 5 Nahrungsquellen verteilt und 40 Partikel müssen möglichst lange überleben. Da diese jedoch in jeder Runde Nahrung verbrauchen, müssen sie von den Nahrungsquellen Nahrung aufnehmen. Das nächste Problem ergibt sich daraus, dass eine Quelle nur begrenzt Nahrung liefert und sich somit die Partikel auf die Quellen verteilen müssen, von denen Sie aber nicht die Position kennen. Zur Kommunikation steht ihnen die Stigmergie durch Pheromone zur Verfügung. Ausgebrachte Pheromone verfallen mit einer Halbwertszeit von 40 Runden.
Jede der fünf Nahrungsquellen ist an einer zufällig gewählten Position und kann maximal 100 Nahrungseinheiten vorhalten. In jeder Runde bekommt eine Quelle maximal 8 Nahrungseinheiten hinzu. Jedes Partikel kann 100 Nahrungseinheiten mit sich führen und verbraucht pro Runde eine Einheit. Sobald es keine Nahrungsmittel mehr hat, stirbt ein Partikel. An einer Nahrungsquelle kann ein Partikel maximal vier Nahrungseinheiten pro Runde aufnehmen, um seinen Vorrat zu füllen.
In jeder Runde kann jedes Partikel auf Grund lokaler Informationen agieren. Die zur Verfügung stehenden Informationen sind zum einen die Größe des eigenen Nahrungsmittelvorrats und zum anderen Informationen über die umgebenden Felder. Diese sind als `struct` mit den Feldern `food`, `pher` und `part` gegeben. Jedes Feld im `struct` enthält eine 3×3-Matrix mit den Informationen der Umgebung des Agenten (eigenes Feld und direkte Nachbarn) bezüglich der Anzahl von Nahrungseinheiten, den Pheromonen bzw. der Anzahl der Agenten auf dem jeweiligen Feld.
Die Aktion jedes Partikels wird durch drei Komponenten bestimmt. Es kann entscheiden[1], in welche Richtung es sich bewegen möchte (1 = Südwesten, 2 = Süden, 3 = Südosten, 4 = Westen, 5 = stehen bleiben, 6 = Osten, 7 = Nordwesten, 8 = Norden, 9 = Nordosten), wie viel Pheromone es ausstreuen möchte, wobei dieser Wert auf das Intervall $[0, 5]$ beschränkt ist, und zuletzt, ob es versuchen möchte, Nahrung aufzunehmen.
Ziel ist es nun, dass sich jedes Partikel nur durch diese lokalen Informationen so verhält, dass nach 300 Runden eine möglichst große Anzahl an Partikeln überlebt hat. Für die Simulation ist ein fertige Funktion in Stud.IP gegeben, die für die Aufgabe nicht verändert werden soll. Die Funktion erwartet als ersten Parameter ein Function Handle, nach dem das Verhalten der Partikel bestimmt wird. Optional können über weitere Parameter die Simulationsdauer eingestellt und die Anzeige der Simulation ausgeschaltet werden.
Überlegen Sie sich nun eine intelligente Strategie, um Ihre Partikel geschickt agieren zu lassen, sodass möglichst viele Partikel überleben. Die Funktion für das Verhalten sollte folgenden Prototypen implementieren:

```
[move, pher, eat] = myBestBehaviour(myFood, environment)
```

Dabei enthält `myFood` die Menge an Nahrungseinheiten, die das Partikel mit sich führt und `environment` das oben beschriebene Umgebungsstruct. Der Rückgabewert `move` bestimmt die Bewegungsrichtung des Partikels, `pher` bestimmt die Menge an Pheromonen, die freigesetzt werden soll und `eat` legt fest, ob gegessen werden soll (=`true`) oder nicht (=`false`).

[1] Die Entscheidung kann auch zufällig geschehen, z.B. wenn keine Nahrungsquelle und keine Pheromone in der aktuellen Nachbarschaft vorhanden sind.