Kombinatorische Optimierung & Sintflut-Algorithmus

Marcel Bronischewsky und Johannes Wollbold - 4. November 2004

„ALGORITHMEN DER MUSTERERKENNUNG UND KÜNSTLICHEN INTELLIGENZ“

Proseminar im Wintersemester 2004/05 – Institut für Informatik – Universität Jena

Zusammenfassung: Am Beispiel des Problems des Handlungsreisenden (TSP) werden Grundbegriffe der kombinatorischen Optimierung (Suchraum, Zielfunktion) eingeführt und Näherungs-Algorithmen wie Hill Climbing, Simulated Annealing und Sintflut-Algorithmus vorgestellt. Sie werden einerseits mit einem exakten mathematischen Ansatz in Form eines Linearen Programms verglichen, andererseits mit genetischen Algorithmen.

1. Optimierungsprobleme

1.1. Stetige Optimierung

Beispiel Regression: Aus n Wertepaaren $(x_1, y_1) \ldots (x_n, y_n)$ soll eine Ausgleichsgerade $y = ax + b$ bestimmt werden. Zwei Größen bestimmen das Problem:

Suchraum $S = \mathbb{R}^2$ (Die Parameter a und b sind gesucht.)

Zielfunktion $f(a,b) = \frac{1}{n}\sum_{i=1}^{n}(ax_i + b - y_i)^2$. Diese soll maximiert oder (im vorliegenden Fall) minimiert werden.

Eine klassische Lösungsmethode – für natürlich meist hochdimensionale Probleme - ist das **Gradientenverfahren** [GKK04, 16f.]. Von einem zufällig gewählten Startpunkt aus bewegt man sich sukzessive um eine festgelegte Schrittweite ρ in Richtung des Gradienten

$$\nabla f(x_1,\ldots,x_n) = \left(\frac{\partial f}{\partial x_1}(x_1,\ldots,x_n),\ldots,\frac{\partial f}{\partial x_n}(x_1,\ldots,x_n)\right)^T$$

Bei einem Minimierungsproblem betrachtet man $-f$. Das Verfahren erreicht im Allgemeinen nur ein lokales, jedoch kein globales Optimum.

1.2. Kombinatorische Optimierung

Ist der Suchraum S endlich oder abzählbar, spricht man von **diskreter Optimierung**. Ist S darüber hinaus eine Teilmenge der Potenzmenge 2^E eines Grundraums E oder eine Familie von Elementen aus E, so handelt es sich um ein Problem der **kombinatorischen Optimierung**. [Lee04, 1] Da in diesen Fällen Stetigkeit und Differenzierbarkeit nicht definiert sind, können Methoden wie das Gradientenverfahren nicht angewandt werden. Beispiele sind:

- Stunden-, Arbeits-, Fahr- oder Flugpläne.
- Auslastung von Maschinen: Stellt eine Maschine jeweils mehrere Produkte her, so sollen möglichst wenig Umrüstungen stattfinden; es müssen Termine eingehalten und der Lagerbestand soll gering gehalten werden. Solche Anforderungen lassen sich über eine zu minimierende Kostenfunktion modellieren, sowie über Nebenbedingungen. [DSW93, 43]
- Rucksackprobleme: Ein Laderaum soll optimal gefüllt oder auch ein Finanzbudget gewinnbringend in mehreren Sorten von Wertpapieren angelegt werden. [DSW93, 45]

Das Problem des Handlungsreisenden (Travelling Salesman Problem – TSP)

Ziel: In einem vollständigen Graphen soll ein Hamiltonkreis (= geschlossener Weg, bei dessen Durchlaufen jeder Knoten ausser dem Anfangs- und Endpunkt genau einmal besucht wird) gefunden werden, mit minimalen Kosten (Weglänge, Zeit). Somit müssen die Kanten des Graphen mit den Kosten „gewichtet“ sein.

Suchraum: Menge aller Touren = Permutationen der „Städte“ $s_1,\ldots,s_n$.

$$S = \{(c_1,\ldots,c_n) \mid c_1,\ldots,c_n \in \{s_1,\ldots,s_n\} \text{ und } c_i \neq c_j \text{ für } i \neq j\}$$

Anwendungen können vielfältige Transportprobleme sein, bei denen in der Praxis zusätzliche Restriktionen wie Zeitfenster, Gewicht, Volumen (bei kombiniertem Ein- und Ausladen) beachtet werden müsen. Mehrere Teiltouren beispielsweise eines Logistik-Unternehmens beschreibt das m-TSP.

2. Lösungsverfahren

2.1. Vollständige Suche

Um das Optimum zu bestimmen, wird der Wert der Zielfunktion auf allen Elementen des Suchraums S berechnet. Wegen $|S| = n!$ ist dieses Verfahren beim TSP meist aussichtslos.

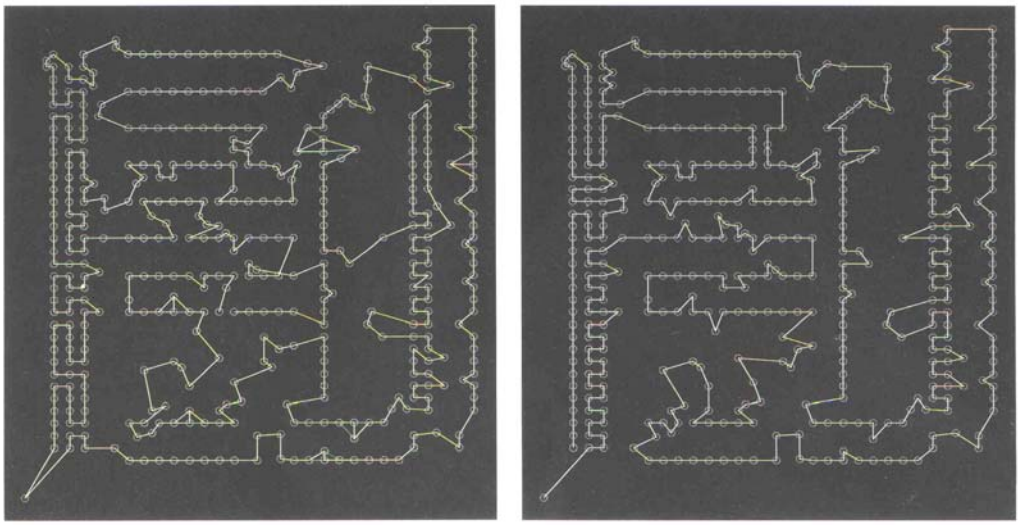
2.2. Mathematische Lösung

Der Ansatz ist ein **Lineares Programm** [Lee04, 1-5, 14], [KV00, 49, 280]:

- Lineare Zielfunktion $f(x) = c^T x, \quad c, x \in \mathbb{R}^n$
- Lineare Nebenbedingungen $Ax \leq b, \quad A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$

Für das TSP wird der Suchraum etwas anders gewählt als in Abschnitt 1.2. definiert (die o.a. Version genügt für die Näherungs-Algorithmen). S ist die Menge aller Kanten, und x ist der charakteristische Vektor einer Kantenmenge H aus S, d.h. jeder Kante, die zu einer „Tour“ H gehört, entspricht eine 1 an der entsprechenden Stelle von x, die anderen Komponenten sind 0.

Exakte Lösungen sind nur für **Spezialfälle des TSP** bekannt, bei denen man dessen (geometrische) Struktur ausnutzen kann. So wurde die kürzeste Tour durch 532 US-Großstädte 1987 berechnet, der Besuch aller 15.211 Dörfer der BRD ist erst seit 2001 optimal möglich [Due 04, 17f.]. Ein 1984 von M. Grötschel gestelltes Problem, 442 Löcher in eine Platine zu bohren, wurde einige Jahre später in einer Dissertation (O. Holland) gelöst. Er benötigte dafür eine Kombination analytischer und numerischer Methoden – rechts das so gefundene Optimum mit einem Gesamtweg von 50,69 inch, links eine mit dem Sintflut-Algorithmus gefundene Lösung von 51,20 inch. [DSW93, 45]



3. Näherungs-Algorithmen

Das allgemeine, rein kombinatorische TSP ist (wahrscheinlich) nicht in polynomialer Zeit algorithmisch zu lösen, erfordert also exponentiellen Rechenaufwand. Für viele praktische Probleme sind daher neuere Algorithmen eine gute Alternative, die zwar selten die optimale Lösung, mit Rechenzeiten im Sekunden- oder Minutenbereich jedoch gute **suboptimale Lösungen** finden. Sie setzen nur eine einfache – wenn auch oft mit aufwendigen Vorarbeiten verbundene - mathematische Modellierung des Problems voraus und sind daher universell einsetzbar. Andererseits ist ihr Erfolg schwer beweisbar, und zur Auswahl eines geeigneten Algorithmus und der Einstellung der Parameter gehört viel Erfahrung. Zunächst noch einige Voraussetzungen der im Folgenden beschriebenen Algorithmen:

Zielfunktion: $f : S \to \mathbb{R}$

Nachbarschaft: Menge aller Lösungen die durch einen „Zug“ erreicht werden können. Ein Zug stellt dabei z.B. beim TSP die Vertauschung von 2 Kanten dar.

Deterministische und **zufällige** Auswahl: Von einer deterministischen Auswahl spricht man, wenn der Nachbar nach einem bestimmten Verfahren ausgesucht wird (in einer bestimmten

Reihenfolge oder nach Beschaffenheit), und bei einer zufälligen Auswahl wird ein beliebiger Nachbar per Zufallsverfahren ausgewählt.

Optimierung: Folgende Algorithmen sind für die Suche nach Maxima formuliert.

3.1. Hill Climbing

Startpunkt ist ein beliebiges $s_0 \in S$. Im nächsten Schritt wird ein deterministisch oder zufällig bestimmtes s_1 in der Nachbarschaft mit s_0 verglichen. Es handet sich also um eine **lokale Suche**. Ist $f(s_1) \geq f(s_0)$, wird s_1 als Ausgangspunkt für eine erneute Suche verwendet $(s_0 := s_1)$. Dies wird so lange wiederholt bis keine Verbesserung mehr eintritt. Ausgabe des Algorithmus ist dann s_0.

Es werden nur Verbesserungen akzeptiert, wodurch gefundene Maxima nicht mehr verlassen werden können.

3.2. Threshold Accepting

1990 von Gunter Dueck und Tobias Scheuer entwickelt.

Startpunkt ist ein beliebiges $s_0 \in S$. Des Weiteren wird eine Schwelle T festgelegt. Im nächsten Schritt wird ein deterministisch oder zufällig bestimmtes s_1 in der Nachbarschaft mit s_0 verglichen. Ist $f(s_1) \geq f(s_0)$ **oder** $f(s_0) - f(s_1) \leq T$, wird s_1 als Ausgangspunkt für die weitere Suche verwendet $(s_0 := s_1)$. Der Schwellwert T wird im Laufe des Algorithmus verringert. Der Algorithmus endet, wenn $T = 0$ ist und keine Verbesserungen mehr auftreten. Da durch die Toleranzschwelle auch niedrigere Funktionswerte angenommen werden können, ist es diesem Algorithmus möglich lokale Maxima zu verlassen.

Richtet man die schrumpfende Toleranzschwelle am bisher höchsten Funktionswert aus, so werden nur noch Punkte akzeptiert, deren Funktionswerte $f(s_i) \geq f(s_j)$ **oder** $f(s_{MAX}) - f(s_j) \leq T$ erfüllen. Dabei ist $s_{MAX} \in S$ der Punkt der bis zu diesem Fortschritt des Algorithmus den höchsten Funktionswert besitzt. Den so entwickelten Algorithmus nennt man auch „**Record-to-Record Travel**“.

3.3. Sintflut-Algorithmus

1993 von Gunter Dueck entwickelt.

Startpunkt ist ein beliebiges $s_0 \in S$. Außerdem wird ein „Wasserpegel“ W vereinbart. Im nächsten Schritt wird ein deterministisch oder zufällig bestimmtes s_1 in der Nachbarschaft mit s_0 verglichen. Ist $f(s_1) > W$, wird s_1 als Ausgangspunkt für die weitere Suche verwendet. Im Laufe des Algorithmus wird der Wasserpegel W erhöht. Der Algorithmus endet, wenn keine Punkte mehr im Suchraum gefunden werden können deren Zielfunktionswerte über W liegen.

Diesem Algorithmus ist es durch die Akzeptanz niedrigerer Funktionswerte möglich, lokale Maxima zu verlassen.

3.4. Simuliertes Ausglühen

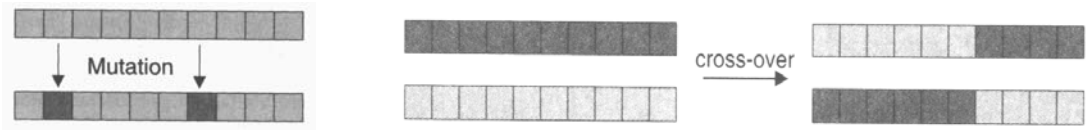
1983 entwickelt und an den physikalischen Prozess zur Erzeugung perfekter Kristalle angelehnt.

Startpunkt ist ein beliebiges $s_0 \in S$. Des Weiteren wird eine Temperatur T festgelegt. Im nächsten Schritt wird ein deterministisch oder zufällig bestimmtes s_1 in der Nachbarschaft mit s_0 verglichen. Ist $f(s_1) \geq f(s_0)$, so wird s_1 als Ausgangspunkt für die weitere Suche

verwendet. Ist $f(s_1) < f(s_0)$, so wird s_1 mit einer Wahrscheinlichkeit $p = e^{\frac{\Delta f}{T}}$ als Ausgangspunkt für die weitere Suche verwendet, wobei $\Delta f = f(s_1) - f(s_0)$ ist. Der Temperaturwert T wird während des Algorithmus gesenkt, womit die Akzeptanz-Wahrscheinlichkeit niedriger Funktionswerte eines $s_i \in S$ gegen 0 geht und nur noch gleich gute oder bessere Werte akzeptiert werden. Diesem Algorithmus ist es möglich lokale Maxima zu verlassen, da er mit einer von mehreren Variablen abhängigen Wahrscheinlichkeit auch schlechtere Funktionswerte akzeptiert.

3.5. Genetische Algorithmen

Prinzipien der biologischen Evolution machen sich genetische Algorithmen zu Nutze. So beginnen diese nicht mit einem einzelnen Startpunkt, sondern einer ganzen „**Population**“ von Lösungen. Ein Element des Suchraums wird also als Individuum aufgefasst, das durch sein (im Normalfall einziges) „**Chromosom**“ repräsentiert wird. Dies ist einfach eine Zeichenkette, beim TSP etwa eine Liste der Knoten eines Hamiltonkreises; eine einzelne „Stadt“ entspricht dann einem **Gen**. Ausgehend von einer Startpopulation werden schrittweise neue Generationen von Lösungen erzeugt, indem nach einem durch die Zielfunktion gesteuerten Zufallsprinzip Individuen ausgewählt werden. Diese werden dann – ebenfalls zufällig - durch **genetische Operatoren** wie Mutation und cross-over verändert. [GKK04, 36-41]



Die Hoffnung dabei ist, das sich das „Wissen“ um gute Lösungen in der Population ausbreitet. Auch hier besteht die Kunst darin, die Parameter so einzustellen, dass sich gute Lösungen durchsetzen und verbessert werden können, ohne frühzeitig neue Lösungen zu unterdrücken.

Als Beispiel für die vielfältigen Varianten genetischer Algorithmen sei zum Schluss die **Boltzmann Selektion** erwähnt [GKK04, 83]. Hier wird – ähnlich wie beim Simulated Annealing – der Selektionsdruck erhöht: Mit sinkender Temperatur T werden die Unterschiede zwischen Chromosomen mit hoher und niedriger Fitness $fit(i)$ (= Zielfunktion) schnell größer. Die Wahrscheinlichkeit $p(c_i)$ sinkt, dass ein Individuum ausgewählt wird, das schlecht an die „Umgebung“ des Problems angepasst ist.

$$p(c_i) = \frac{e^{fit(i)/T}}{\sum_{j=1}^{popsize} e^{fit(i)/T}}$$

Literatur

[Due04] Gunter Dueck. *Das Sintflut-Prinzip. Ein Mathematik-Roman.* Springer, Berlin / Heidelberg 2004.

[DSW93] G. Dueck, T. Scheuer und H.-M. Wallmeier. Toleranzschwelle und Sintflut: neue Iden zur Optimierung, in: *Spektrum der Wissenschaft* 3/1993.

[GKK04] I. Gerdes, F. Klawonn und R. Kruse. *Evolutionäre Algorithmen*. Vieweg, Wiesbaden 2004.
Zusatzinformationen: http://fuzzy.cs.uni-magdeburg.de/books/evoalg/

[KV00] B. Korte and J. Vygen. *Combinatorial Optimization. Theory and Algorithms.* Springer, Berlin / Heidelberg / New York 2000.

[Lee04] John Lee. *A First Course in Combinatorial Optimization*. Cambridge 2004.

Internetquellen zu den Näherungs-Algorithmen:

http://sirius.fh-friedberg.de/Sintflut/Html/Algorithmus.html

http://www.inf.hs-zigr.de/~wagenkn/TI/Komplexitaet/Referate98/diedrichhowitz/node3.html

Simulationsprogramme:
http://www.fh-augsburg.de/informatik/projekte/mebib/emiel/entw_inf/or_verf.html