

# Software Projekt Anwendungen von Algorithmen

## Metaheuristiken II

Robert Gottwald, Lars Parmakerli, Phil Schmidt

Freie Universität Berlin  
Institut for Computer Science  
robert.gottwald@fu-berlin.de  
-Lars - fu-Adresse-  
philschmidt@inf.fu-berlin.de

<https://www.inf.fu-berlin.de/lehre/WS14/SWPA1g/index.html>

**Zusammenfassung** Der folgende Text befasst sich mit der Arbeit der Gruppe Metaheuristiken II des Softwareprojekts Anwendungen von Algorithmen im Wintersemester 2014/15. Dabei werden wir uns mit zwei spezifischen Metaheuristiken befassen, namentlich Simulated Annealing und ein genetisch-inspiriertes Verfahren, sowie mit dem globalen Lösungsverfahren Branch-and-Bound und seinen Schwierigkeiten für die Anwendung auf unsere Probleme und schließlich zuletzt mit nicht-linearen Optimierungsverfahren unter Zuhilfenahme numerischer Verfahren bezogen auf die gewählten Aufgabenstellungen der Stapeln und des Packens.

## 1 Metaheuristiken

Das Wort Metaheuristik setzt sich aus den beiden Wörtern 'Meta' und 'Heuristik' zusammen. Eine Heuristik ist ein Verfahren, welches für eine bestimmte Art von Problemen eine Lösung findet, die als ausreichend 'gut' erachtet wird. Das Wort 'Meta-' bedeutet in diesem Kontext 'auf einer Höheren Ebene'.

Eine Metaheuristik ist demnach möglichst effizientes Lösungsverfahren, welches sich auf einer höher abstrahierten Ebene abspielt und somit kein Wissen über das unterliegende Problem benötigt. Die einzige bestehende Voraussetzung für das Anwenden einer Metaheuristik auf ein System, ist eine Funktion existiert, welche die Güte einer Lösung berechnen kann.

Metaheuristiken stellen nützliche Werkzeuge dar, um Optimierungsprobleme in komplexeren Systemen zu lösen, ohne für diese eine spezifische, kompliziertere Heuristik zu entwerfen, sind also sozusagen wiederverwendbare Lösungswege. Der Nachteil von Metaheuristiken liegt in der Effizienz in Laufzeit, sowie der meist niedrigeren Güte der erzielten Lösung.

In unserem Softwareprojekt bestehen Metaheuristikmodule aus generischen Klassen mit einem Typenparameter für die Anwendungsdomäne, sowie mindestens einer Auswertungsfunktion. Einzelne Implementierungen werden in den Folgekapiteln beschrieben.

## 2 Simulated Annealing

### 2.1 Inspiration

Metaheuristiken liegen zumeist realen Prozessen zugrunde, welche ein bestimmtes Verhalten aufweisen. Simulated Annealing ist vom namensgebenden Annealing-Prozess aus der Schwerindustrie inspiriert.

Beim Annealing wird Metall stark erhitzt und über einen längeren Zeitraum langsam abgekühlt. Bei den hohen Temperaturen tritt eine starke Molekular-Bewegung im zu bearbeitenden Material auf. Dadurch, dass der Abkühlungsprozess über einen längeren Zeitraum gestreckt wird, reduziert sich die Teilchenbewegung auch nur allmählich. Der Abkühlungsprozess wird so gesteuert durch etwaige Behandlungsmethoden, dass die einzelnen Teilchen einen für einen bestimmten zu erzielenden Effekt, wie zum Beispiel eine stabile Statik, guten Zustand einnehmen.

Wie der Name uns sagt, ist die Herangehensweise beim Simulated Annealing nun so, dass wir einen derartigen physikalischen Prozess über einen abstrakten Wertebereich simulieren.

### 2.2 Simulation

Das zu behandelnde Metall wird in unserer Implementierung durch einen generischen Typ-Parameter namens 'State' realisiert. Bei der Anwendung auf die uns gegebene Problemstellung, wie zum Beispiel Stapeln, ist dies ein Feld aus Polygonen.

Die Molekularbewegung wird durch eine Umgebungsfunktion namentlich 'Mutator' umgesetzt. Eine Umgebungsfunktion bekommt einen 'State' als Input und gibt einen anderen 'State' zurück, der in der 'Nachbarschaft' des Eingabe-States befindet.

Die benötigte Gütefunktion, die jedem 'State' einen Double-Wert zuweist und für jede Metaheuristik notwendig ist, ist als Quantifizierung des durch das Annealing zu erzielenden Effekts zu interpretieren.

Die Unberechenbarkeit des Materials, beziehungsweise das chaotische Verhalten der Teilchen, lässt sich durch die Temperatur messen. In unserer Simulation wird eine CoolingSchedule-Klasse verwendet, die nach beliebig wählbaren Abkühlmustern, wie zum Beispiel logarithmisches oder lineares Abkühlen, nach jedem Abfragen einer T()-Funktion eine neue Temperatur zurückgibt.

### 2.3 Ablauf

Simulated Annealing ist ein iteratives Verfahren. Wir nutzen zwei Zeiger; einen, der auf das aktuell beste gefundene Ergebnis zeigt ('best'), und einen, der den aktuellen Ist-Zustand der Domäne speichert ('current'). In jeder Iteration wird ein neuer State in der Umgebung des aktuellen States mithilfe des Mutators generiert.

Anschließend wird die Objective Function dazu genutzt, den neuen State zu bewerten und mit den alten States zu vergleichen. Ist ein neues globales Optimum gefunden worden, wird dieses gespeichert (best).

Für die Bestimmung des neuen current-Zustandes wird eine zufällige Zahl zwischen 0 und der Maximaltemperatur generiert. Im Normalfall nimmt current den nach

Gütefunktion besseren State an, es sei denn es wird ein kleinerer Wert, als die aktuelle Temperatur gewürfelt. Dann nimmt current den schlechteren der beiden Werte an.

Dieser Prozess wird solange fortgeführt, bis entweder eine feste Iterationsanzahl erreicht ist, oder die Lösung 'gut'-genug ist. Unsere Implementierung terminiert einfach nach einer Festen Schrittzahl.

### 3 Genetische Heuristik

#### 3.1 Inspiration

Ähnlich wie beim Simulated Annealing gibt es für den folgenden genetischen Algorithmus Inspiration durch einen real-weltlichen Prozess. In diesem Fall ist es das biologische Konzept der Evolution.

Evolution wird ein biologischer Prozess genannt, der optimale Adaption von Lebensformen an einen bestimmten Lebensraum über mehrere Generationen erlaubt. Dieser Prozess lässt sich in verschiedene Iterationen (Generationen) aufteilen. Jede Iteration kennt verschiedene Phasen, die sie durchläuft, die den Evolutionsvorgang ausmachen:

##### Natürliche Selektion

Das Ökosystem um eine Population herum übt einen so genannten Selektionsdruck auf die darin lebenden Organismen aus. Lebensformen, die nicht bestimmten Ansprüchen genügen, werden aus dem Genpool eliminiert.

##### Rekombination

Durch sexuelle Fortpflanzung wird es zwei genetisch "fiten" Organismen ermöglicht, ihre Gene untereinander auszutauschen. In der nächsten Generation werden ihre Nachkommen eine zufällige Auswahl an Genen ihrer beiden Elternteile tragen.

##### Mutation

Im Vorgang der Gen-Rekombination kann es vorkommen, dass Reproduktionsfehler auftreten. Die Folge davon ist, dass die nachkommen Gen-Eigenschaften haben können, die nicht Teil des ursprünglichen Genpools der Elterngeneration waren. Diese Mutationen treten zufällig auf und haben ebenso willkürliche Auswirkungen.

Nachdem durch diesen Prozess eine neue Generation erschaffen wurde, durchläuft diese Generation erneut den selben Prozess. Dabei verbessert sich langfristig das Gen-Material in Bezug auf den Selektionsdruck des bestehenden Ökosystems. Der folgende Algorithmus versucht sich daran, ein derartiges Ökosystem mit verschiedenen Populationen zu simulieren, um einen ähnlichen Optimierungsprozess zu erzielen.

### 3.2 Simulation

## 4 Branch and Bound

## 5 Nicht-lineare Optimierung mit numerischen Verfahren des Stapelproblems

### 5.1 Formulierung als nicht-lineares Optimierungsproblem

Das Stapelproblem lässt sich als Optimierungsproblem einer nicht-linearen Zielfunktion ohne Nebenbedingungen formulieren. Die Zielfunktion  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  erhält die Translations-Vektoren und die Rotations-Winkel für jedes der gegebenen Polygone als Eingabe und ordnet diesen die Fläche der konvexen Hülle bzw. der Bounding-Box zu. Im zweidimensionalen sind das also drei Parameter für jedes Polygon.

### 5.2 Lösungs-Verfahren

**Newton-Verfahren** Für derartige Optimierungsprobleme gibt es verschiedene mathematische Verfahren zur numerischen Berechnung lokaler Optima. Viele dieser Verfahren basieren auf dem Newton-Verfahren zum Finden von Nullstellen einer Funktion. Um das Newton-Verfahren zur Optimierung einzusetzen wendet man es auf die erste Ableitung  $f'$  an. Wenn die Lösung dann zu einer Nullstelle konvergiert ist, ist die erste Ableitung null und damit ein Extrempunkt erreicht. Allerdings benötigt das Newton-Verfahren auch noch die zweite Ableitung der Funktion auf die es angewendet wird, deshalb wird auch noch die zweite Ableitung  $f''$  benötigt, wenn man das Newton-Verfahren zur Optimierung verwendet.

Das Verfahren berechnet die Lösung iterativ. Eine gegebene Lösung  $x_n$  wird in jedem Schritt mit der Formel

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

weiter der optimalen Lösung angenähert.

Bei der Minimierung der Funktion aus 5.1 ist die erste Ableitung eine vektorwertige Funktion, die einen Vektor mit partiellen Ableitungen, den Gradienten, zurückgibt  $f' : \mathbb{R}^n \rightarrow \mathbb{R}^n$  und die zweite Ableitung gibt dann sogar eine  $n \times n$ -Matrix, die Hesse-Matrix, zurück, nämlich die partielle Ableitung nach jedem der  $n$  Eingabeparameter für jede der  $n$  Komponenten des Ausgabevektors der ersten Ableitung  $f'' : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$ . Diese Matrix wird Hesse-Matrix genannt.

Um das Newton-Verfahren für vektorwertige Funktionen einzusetzen wird die Formel zu

$$x_{n+1} = x_n - H^{-1}g$$

Dabei ist  $H := f''(x_n)$  die Hesse-Matrix und  $g := f'(x_n)$  der Gradient.

**Ableitungen** Da wir in der Implementierung der Zielfunktion 5.1 die Ableitungen nicht gegeben haben, müssen wir sie numerisch Berechnen. Mit der Finite-Differenzen-Methode benötigt die Berechnung der ersten Ableitung zwei Funktionsauswertungen pro Komponente also insgesamt  $2n$  Funktionsauswertungen. Würde man nun die zweite Ableitung ebenfalls numerisch Berechnen bräuchte man  $2n$  Funktionsaufrufe der ersten Ableitung, also insgesamt  $4n^2$  Funktionsaufrufe. Das ist nicht nur sehr aufwendig, sondern auch Problematisch für die numerische Stabilität des Verfahrens. Denn die Finite-Differenzen-Methode führt leicht zu Auslöschungen, die bei der Subtraktion fast gleich großer Gleitkommazahlen auftreten.

Ausserdem ist die Ableitung der Zielfunktion für das Stapelproblem nicht glatt, denn wenn ein Polygon sich komplett innerhalb der konvexen Hülle befindet, ist die Ableitung bezüglich dessen Translations- und Rotationsparameter null. Sobald das Polygon jedoch so weit verschoben wird, dass es den Rand erreicht, wird die Ableitung positiv, hat also an dieser Stelle einen Knick und die zweite Ableitung ist somit nicht stetig. Beweise für die Konvergenzeigenschaften von Newton-artigen Verfahren setzen jedoch zweimal stetig differenzierbare Funktionen voraus.

In der Praxis hat sich jedoch gezeigt, dass es Newton-artige Verfahren zur Optimierung gibt, die gut für nicht glatte Funktionen funktionieren [1] und ohne zweite Ableitungen auskommen.

**Quasi-Newton Verfahren** Sogenannte Quasi-Newton Verfahren, sind Newton-artige Verfahren, die die direkte Berechnung der zweiten Ableitung vermeiden. Stattdessen wird eine Approximation der Hesse-Matrix verwaltet, die in jeder Iteration des Algorithmus mithilfe der Informationen der ersten Ableitung weiter der echten Hesse-Matrix angenähert wird. Dafür existieren verschiedene Update-Formeln. Wir verwenden die BFGS Update-Formel, benannt nach ihren Erfindern **B**royden, **F**letcher, **G**oldfarb und **S**hanno.

$$B_n + 1 = B_n + \frac{(y_n - B_n s_n)(y_n - B_n s_n)^T}{(y_n - B_n s_n)^T s_n}$$

, wobei  $s_n := x_{n+1} - x_n$  und  $y_k := f'(x_{n+1}) - f'(x_n)$ .

## Line search

## 6 Nicht-lineare Optimierung mit numerischen Verfahren des Packenproblems

### Literatur

1. A. S. Lewis and M. L. Overton, "Nonsmooth optimization via quasi-newton methods," *Math. Program.*, vol. 141, no. 1-2, pp. 135–163, 2013.