Lokale Suche

Lokale Suche

Wir betrachten das allgemeine Minimierungsproblem

$$\min_y \quad f(x, y) \quad \text{so dass} \quad L(x, y).$$

Wir nehmen an, dass zu jeder Lösung y auch eine Nachbarschaft $\mathcal{N}(y)$ „benachbarter“ Lösungen gegeben ist.

Die strikte lokale Suche:

(1) Sei $y^{(0)}$ eine Lösung für die Eingabe x. Setze $i = 0$.

(2) Wiederhole solange, bis eine lokal optimale Lösung gefunden ist:

(2a) Bestimme einen Nachbarn $y \in \mathcal{N}(y^{(i)})$, so dass $f(x, y) < f(x, y^{(i)})$ und y eine Lösung ist.

(2b) Setze $y^{(i+1)} = y$ und $i = i + 1$.

Implementierung der lokalen Suche I

- Wie sind die Nachbarschaften $\mathcal{N}(y)$ zu definieren?
  - Wenn nur Elemente aus $\{0,1\}^n$ Lösungen sind, dann ist

$$\mathcal{N}_k(y) = \{y' \in \{0,1\}^n \quad | \quad y \text{ und } y' \text{ unterscheiden sich in in höchstens } k \text{ Positionen}\}$$

    die k**-Flip Nachbarschaft** der Lösung y.
  - Die k-Flip Nachbarschaft besteht aus $\binom{n}{k}$ Elementen und ist für große k zu groß. Deshalb wird häufig $k = 1$ oder $k = 2$ gewählt.
- Mit welcher Anfangslösung $y^{(0)}$ soll begonnen werden?
  - Neben der zufälligen Wahl einer Startlösung
  - werden auch Heuristiken eingesetzt, um mit einer guten Lösung $y^{(0)}$ zu beginnen.
    Achtung: $y^{(0)}$ darf nicht in der Nähe eines lokalen Minimums sein!

Implementierung der lokalen Suche II

- Mit welcher Nachbarlösung soll die Suche fortgesetzt werden?
  - Wenn die Nachbarschaft genügend klein ist, dann liegt die Wahl eines Nachbarn mit kleinstem Zielfunktionswert nahe.
  - Bei zu großen Nachbarschaften wählt man häufig benachbarte Lösungen mit Hilfe einer Heuristik, bzw. man wählt zufällig eine kleine Menge von Nachbarn.

- Die große Schwäche der lokalen Suche:

  Nur Abwärtsbewegungen sind erlaubt.

  Kurz über lang fällt eine Lösung in ein lokales Minimum mit großer Sogwirkung.
- **Auswege:**
  - Im Metropolis Algorithmus und in Simulated Annealing werden auch Aufwärtsbewegungen erlaubt.
  - Die lokale Suche in variabler Tiefe erlaubt ebenfalls Verschlechterungen.

Beispiel: Der Simplex Algorithmus

- Der Lösungsraum des linearen Programmierproblems

$$\min c^T \cdot y, \text{ so dass } A \cdot y \geqslant b \text{ und } y \geqslant 0$$

ist ein Durchschnitt von Halbräumen der Form $\{y \mid \alpha^T \cdot y \geqslant \beta\}$.

Deshalb wird das Optimum an einer „Ecke“ des Lösungsraums angenommen.

- Der Simplex-Algorithmus wandert solange von einer Ecke zu einer besseren, benachbarten Ecke, bis keine Verbesserung erreichbar ist.
- Wenn wir also nur Ecken als Lösungen zulassen, dann bildet die lokale Suche die Grundstruktur des sehr erfolgreichen Simplex-Algorithmus.

Beispiel: Minimierung reellwertiger Funktionen

- Die Zielfunktion $f : \mathbb{R}^n \to \mathbb{R}$ des Minimierungsproblems $P = (\min, f, L)$ sei differenzierbar.
- Der Lösungsraum sei eine kompakte Teilmenge des $\mathbb{R}^n$.
- Wenn wir uns gegenwärtig in der Lösung $a \in \mathbb{R}^n$ befinden:
  - In welcher Richtung sollten wir nach besseren Lösungen suchen?
  - In der Richtung des negativen Gradienten!

Die Methode des Gradientenabstiegs

Sei $f : \mathbb{R}^n \to \mathbb{R}$ eine zweimal im Punkt $a \in \mathbb{R}^n$ stetig differenzierbare Funktion mit $\nabla f(a) \neq 0$. Dann gibt es ein $\eta > 0$ mit

$$f\left(a - \eta \cdot \nabla f(a)\right) < f(a),$$

wobei $\nabla f(a)$ der Gradient von f an der Stelle a ist.

- Ersetze a durch $a - \eta \cdot \nabla f(a)$ für ein hinreichend kleines η.

Warum funktioniert der Gradientenabstieg?

- Approximiere die Funktion f durch ihr lineares Taylor-Polynom:

$$f(a+z) = f(a) + \nabla f(a)^T \cdot z + O(z^T \cdot z).$$

  gilt für hinreichend kleines z.

- Für $z = -\eta \cdot \nabla f(a)$ bei hinreichend kleinem $\eta > 0$ ist

$$f(a - \eta \cdot \nabla f(a)) = f(a) - \eta \cdot \ ||\nabla f(a)||^2 + \eta^2 \cdot O(||\nabla f(a)||^2).$$

- Für hinreichend kleines $\eta < 1$ wird somit $f(a - \eta \cdot \nabla f(a))$ kleiner als $f(a)$ sein.

Die Gefahr schlechter lokaler Optima

- Wir betrachten das Vertex Cover Problem VC für einen Graphen $G = (V, E)$.
- Die Nachbarschaft $\mathcal{N}(U)$ einer Teilmenge $U \subseteq V$ besteht aus allen Überdeckungen, die durch Hinzufügen oder Entfernen eines Elementes aus der Überdeckung U entstehen.
- Wir beginnen die lokale Suche mit der Lösung $U = V$ und betrachten einige Beispiele.
  - **Der Graph ohne Kanten:** Die lokale Suche entfernt in jedem Schritt einen Knoten und findet die leere Menge.
  - **Der Sterngraph:**
    - ⋆ Wird im ersten Schritt das Sternzentrum entfernt, dann wird das schlechte lokale Minimum der „Satelliten“ gefunden.
    - ⋆ Ansonsten führt lokale Suche zwangsläufig auf das Sternzentrum.
  - **Der Weg $0 \to 1 \to 2 \to \cdots \to n-2 \to n-1$**: Für gerades n ist $\{0, 2, 4, \ldots, n-2\}$ optimal. Wie sehen die lokalen Minima aus?

Lokale Suche in variabler Tiefe und Simulated Annealing

Lokale Suche in variabler Tiefe

Die Annahme: Der Lösungsraum ist eine Teilmenge von $\{0,1\}^n$.

(1) Setze EINGEFROREN $= \emptyset$, $\mathcal{L} = \{y\}$ und $z = y$.
// EINGEFROREN ist eine Menge von Positionen und
// $\mathcal{L}$ ist eine Menge von Lösungen.

(2) Wiederhole, solange EINGEFROREN $\neq \{1, \ldots, n\}$

(2a) Bestimme eine beste Lösung $z' \neq z$ in der k-Flip Nachbarschaft von z.
// Wechsel an bis zu k Positionen, die nicht in EINGEFROREN sind.

(2b) Friere alle im Wechsel von z nach z' geänderten Positionen ein, füge z' zu $\mathcal{L}$ hinzu und setze $z = z'$.
// Wir erlauben Aufwärtsbewegungen, da z' eine schlechtere
// Lösung als z sein darf. Durch das Einfrieren geänderter
// Positionen wird die Schleife höchstens n Mal durchlaufen.

(3) Gib die beste Lösung in $\mathcal{L}$ als neue Lösung y' aus und wiederhole das Verfahren gegebenenfalls für y'.

Minimum Balanced Cut

- Ein ungerichteter Graph $G = (V, E)$ gegeben.
- **Ziel:** Bestimme eine Zerlegung $V = V_1 \cup V_2$ mit $|V_1| = \lfloor \frac{|V|}{2} \rfloor, |V_2| = \lceil \frac{|V|}{2} \rceil$ so dass die Anzahl kreuzender Kanten, also die Anzahl aller Kanten mit einem Endpunkt in V_1 und einem Endpunkt in V_2,

  minimal ist.

- Eine Anwendung der lokalen Suche in variabler Tiefe wählt die 2-Flip Nachbarschaft.
  - Ein Nachbar einer Zerlegung $V = V_1 \cup V_2$ ist von der Form $V = U_1 \cup U_2$, wobei U_1 durch das Entfernen und das nachfolgende Einfügen eines Elementes aus V_1 entsteht.
  - Wir werden später eine Anwendung von Simulated Annealing kennenlernen.
- Für Minimum Balanced Cut werden gute experimentelle Ergebnisse berichtet. Gleiches trifft auf TSP zu.

Der Metropolis Algorithmus

(1) Sei y eine Anfangslösung und sei T die Temperatur.

(2) Wiederhole hinreichend oft:

- Wähle zufällig einen Nachbarn $y' \in \mathcal{N}(y)$.
- Wenn $f(y') \leqslant f(y)$, dann akzeptiere y' und setze $y = y'$.
  Ansonsten setze $y = y'$ mit Wahrscheinlichkeit $e^{-\frac{f(y')-f(y)}{T}}$.

  // Je schlechter der Wert des neuen Nachbarn y', umso geringer
  // die Wahrscheinlichkeit, dass y' akzeptiert wird.
  // Schlechte Nachbarn haben nur eine Chance bei
  // entsprechend hoher Temperatur.

Grenzverteilung und Konvergenzgeschwindigkeit

Mit welcher Wahrscheinlichkeit wird die Lösung y besucht, wenn der Metroplis-Algorithmus genügend häufig wiederholt wird?

$f_y(t)$ sei die relative Häufigkeit, mit der der Metropolis-Algorithmus die Lösung y in den ersten t Schritten besucht. Dann gilt

$$\lim_{t\to\infty} f_y(t) = \frac{e^{-f(y)/T}}{Z}, \text{ wobei } \quad Z = \sum_{y, L(y)} e^{-f(y)/T}.$$

- Also, wenn man lange genug sucht, dann besucht man gute Lösungen mit sehr viel höherer Wahrscheinlichkeit als schlechte.
- Wie lange ist lange? Leider häufig „sehr lange“.
  Die Konvergenzgeschwindigkeit ist leider i. A. sehr langsam.

Beispiel: Das Vertex Cover Problem I

Der leere Graph G $= (\mathbf{V}, \emptyset)$:

- Anfänglich wird die Knotenmenge nur reduziert und Metropolis verhält sich wie die lokale Suche.
- Umso kleiner die gegenwärtige Lösung ist, umso größer ist die Anzahl der schlechteren Nachbarn.
- Schlechtere Nachbarn werden dann höher wahrscheinlich als gute.
- Die Akzeptanzwahrscheinlichkeit einer schlechteren Lösung ist aber nur unwesentlich kleiner.
- Schlechtere Lösungen werden kurz über lang gewählt:

Der Metropolis-Algorithmus bekommt Angst vor der eigenen Courage, wenn gute, aber längst nicht optimale Lösungen erreicht werden.

Beispiel: Das Vertex Cover Problem II

Der Sterngraph:

- Der Metropolis-Algorithmus zeigt seine Stärke: Selbst wenn das Zentrum irgendwann entfernt wird, so ist die Wahrscheinlichkeit hoch, dass es nach nicht zu langer Zeit wieder betrachtet wird.
  - Mit beträchtlicher Wahrscheinlichkeit wird das Zentrum, wenn nicht im ersten, dann in folgenden Versuchen wieder aufgenommen.
  - Danach, wenn mindestens ein Satellit **nicht** in der jeweiligen Lösung liegt, bleibt das Zentrum erhalten, da sonst keine Überdeckung vorliegt.
- Allerdings hat Metropolis auch für den Sterngraphen Angst vor der eigenen Courage, wenn die Überdeckung genügend klein ist:

  Unbenötigte Satelliten werden aufgenommen!
- Wir brauchen ein Verfahren, das Aufwärtsbewegungen „mit der Zeit“ erschwert.

Simulated Annealing

Wir müssen Aufwärtsbewegungen nach entsprechend langer Suchzeit signifikant erschweren. Wie? Wir senken die Temperatur!

Eine Analogie aus der Physik:

- Erhitzt man einen festen Stoff so stark, dass er flüssig wird und läßt man ihn dann wieder langsam abkühlen, so ist der Stoff bestrebt, möglichst wenig der zugeführten Energie zu behalten.
- Der Stoff bildet eine Kristallgitterstruktur; Eiskristalle sind ein Beispiel.
- Je behutsamer nun das Ausglühen (*engl.: Annealing*) durchgeführt wird, umso reiner ist die Gitterstruktur.

  Unregelmäßigkeiten in der Gitterstruktur, die durch zu rasches Abkühlen entstehen, stellen lokale Energieminima dar.

Simulated Annealing: Der Algorithmus

(1) Sei y die Anfangslösung und sei T die Anfangstemperatur.

(2) Wiederhole hinreichend oft:

- Wähle zufällig einen Nachbarn $y' \in \mathcal{N}(y)$.
- Wenn $f(y') \leqslant f(y)$, dann akzeptiere y' und setze $y = y'$.
  Ansonsten setze $y = y'$ mit Wahrscheinlichkeit $e^{-\frac{f(y)-f(x)}{T}}$.

(3) Wenn die Temperatur noch nicht genügend tief ist, dann wähle eine neue, niedrigere Temperatur T und führe Schritt (2) aus.
Ansonsten gib die beste erhaltene Lösung aus.

Minimum Balanced Cut

Nach der Anwendung der lokalen Suche in variabler Tiefe wenden wir jetzt Simulated Annealing an. Wir beschreiben die Implementierung.

- Wir lassen beliebige Zerlegungen $(W, V \setminus W)$ zu und wählen

$$f(W) := |\{e \in E \mid |\{e\} \cap W| = 1\}| + \underbrace{\alpha \cdot (|W| - |V \setminus W|)^2}_{\text{Strafterm}}$$

  als zu minimierende Zielfunktion. Mit dem Strafterm versuchen wir eine perfekte Aufteilung zu erzwingen.
- Wir wählen die 1-Flip Nachbarschaft.
- Die Startlösung ist eine zufällig gewählte Teilmenge und ihr Komplement.
- Die Anfangstemperatur wird so gesetzt, dass 40% aller Nachbarn akzeptiert werden.
- Die Temperatur wird über den Zeitraum von $16 \cdot |V|$ konstant gehalten und dann um den Faktor $0,95$ gesenkt („geometrische Abkühlung“).

Minimum Balanced Cut: Die experimentellen Ergebnisse

- **Für zufällige Graphen:**
  - Simulated Annealing schneidet sehr gut ab und schlägt die lokale Suche in variabler Tiefe, selbst wenn beide Verfahren gleich lange laufen.
- **Strukturierte Graphen:** 500 (bzw. 1000) Punkte werden zufällig aus dem Quadrat $[0, 1]^2$ gewählt und zwei Punkte werden verbunden, wenn sie nahe beieinander liegen.
  - Simulated-Annealing bricht ein und die lokale Suche in variabler Tiefe ist deutlich überlegen.
- Die beiden Graphklassen unterscheiden sich in der Struktur ihrer lokalen Minima:
  - Die strukturierten Graphen haben Minima mit weitaus größeren Anziehungsbereichen als die zufällig generierten Graphen.
  - Die „Sogwirkung“ lokaler Minima ist für strukturierte Graphen schwieriger zu überwinden als für Zufallsgraphen.

Evolutionäre Algorithmen

Evolutionäre Algorithmen

- Die Zielfunktion f sei über einem Lösungsraum zu maximieren.
- Die Evolution als Vorbild: Wie können wir das Erfolgsprinzip der Evolution, „Survival of the Fittest“, algorithmisch umsetzen?
  - Die Fitness eines Individuums (oder einer Lösung) y stimmt mit dem Funktionswert $f(y)$ überein.
  - Eine gegebene Population von Lösungen ist zu verbessern, indem Lösungen „mutieren“ oder zwei oder mehrere Lösungen miteinander „gekreuzt“ werden.

    // Dazu werden Mutations- und Crossover-Operatoren angewandt.

Die Grobstruktur evolutionärer Algorithmen

(1) **Initialisierung:** Eine Anfangspopulation P von Lösungen ist zu bestimmen.

(2) Wiederhole, bis eine genügend gute Lösung gefunden wurde:

  (2a) **Selektion zur Reproduktion**: Eine Teilmenge $P' \subseteq P$ der gegenwärtigen Population P von Lösungen wird ausgewählt.

  (2b) **Variation:** Neue Lösungen werden aus P' mit Hilfe der Mutation- und Crossover-Operatoren gewonnen.

  (2c) **Selektion zur Ersetzung:** Die neue Population ist festzulegen.

Die Implementierung evolutionärer Algorithmen I

- **Initialisierung:** Die Anfangspopulation wird
  - durch eine zufällige Auswahl definiert, beziehungsweise
  - durch die sorgfältige Wahl von Anfangslösungen mit Hilfe problem-spezifischer Heuristiken.
  - Diversität ist zu erzwingen, damit der gesamte Lösungsraum „repräsentiert“ wird.
- **Selektion zur Reproduktion:** Eine Teilmenge $P' \subseteq P$ von Elternlösungen ist auszuwählen. Zu den häufig benutzten Auswahlverfahren gehören:
  - **die zufällige Auswahl nach der Gleichverteilung:** Jede Lösung in P wird mit gleicher Wahrscheinlichkeit gewählt,
  - **die zufällige Auswahl nach der Fitness:** Falls f positiv ist, wird $y \in P$ mit Wahrscheinlichkeit $\frac{f(y)}{N}$ für $N = \sum_{z \in P} f(z)$ oder mit Wahrscheinlichkeit $\frac{e^{f(x)/T}}{M}$ für $M = \sum_{z \in P} e^{f(z)/T}$ gewählt
  - oder die **Turnier-Selektion**: Wähle k Lösungen aus P nach der Gleichverteilung und lasse die k' fittesten der k Lösungen zu.

Die Implementierung evolutionärer Algorithmen II

- **Variation:** λ Nachkommen werden aus P' mit Hilfe der Mutations- und Crossover-Operatoren erzeugt.
- **Selektion zur Ersetzung:** Die neue Generation ist festzulegen. Die Populationsgröße sei stets μ.
  - In der Plus-Auswahl werden die μ Fittesten aus den μ Lösungen der alten Generation und ihren λ Nachkommen bestimmt. Man spricht von der $(\mu + \lambda)$-Selektion.
  - In der Komma-Auswahl wird $\lambda \geqslant \mu$ angenommen, und die μ fittesten Nachkommen bilden die neue Generation. Man spricht von der (μ, λ)-Selektion.

Mutationsoperatoren I

- **Bitmutationen**: Man nimmt an, dass der Lösungsraum der n-dimensionalen Würfel $\mathbb{B}^n = \{0,1\}^n$ ist.
  - Entweder flippt man jedes Bit einer Elternlösung $y \in \mathbb{B}^n$ mit einer kleinen Wahrscheinlichkeit p (mit $p \cdot n = \Theta(1)$)
  - oder man ersetzt y durch einen Nachbarn y' in der k-Flip Nachbarschaft von y für kleine Werte von k.
- **Mutationen für reellwertige Vektoren:** Im $\mathbb{R}^n$ ersetzt man eine Elternlösung y durch $\mathbf{y}' = \mathbf{y} + \mathbf{m}$, wobei die Komponenten von m zufällig und unabhängig voneinander mit Erwartungswert 0 gewählt werden.
  - Entweder man wählt die Komponenten m_i von m zufällig aus einem fixierten Intervall $[-a, a]$
  - oder man erlaubt unbeschränkte Komponenten, die zum Beispiel gemäß der Normalverteilung gezogen werden.

    Für schlechte Lösungen erlaubt man eine hohe Standardabweichung und reduziert die Standardabweichung je besser die Lösung ist.

Mutationsoperatoren II

- Für den Permutationsraum

$$\mathbb{S}^n = \{\pi \mid \pi \text{ ist eine Permutation von } \{1, \ldots, n\}\}$$

  betrachtet man **Austausch-** und **Sprungoperatoren**.
  - In einem Austauschoperator wird ein Paar (i, j) mit $1 \leqslant i \neq j \leqslant n$ zufällig und gleichverteilt aus allen möglichen Paaren gezogen. Die $i$te und $j$te Komponente der Elternlösung werden vertauscht.
  - Auch für einen Sprungoperator werden Paare (i, j) zufällig gezogen. Die $i$te Komponente y_i einer Elternlösung y wird an die Position j gesetzt und die restlichen Komponenten werden passend verschoben.
- Stärker problemabhängige Mutationsoperatoren sind sinnvoll, solange die Diversität nicht zu stark eingeschränkt wird.

  Wenn zum Beispiel eine Elternlösung y durch ein besseres lokales Maximum ersetzt wird, dann ist ein zukünftiges Verlassen des lokalen Maximums schwierig.

Crossover Operatoren

$y_1, \ldots, y_k$ seien k Eltern, wobei $k \geqslant 2$ gelte.

- Für den Würfel $\mathbb{B}^n$ betrachtet man nur den Fall $k = 2$. Im **r-Punkt Crossover** wählt man $r \geqslant 2$ Positionen $1 \leqslant i_1 < \cdots < i_r \leqslant n$. Der Sprößling z von y_1 und y_2 erbt die ersten i_1 Bits von y_1, die $i_2 - i_1$ Bits in den Positionen $[i_1 + 1, i_2]$ von y_2, die $i_3 - i_2$ Bits in den Positionen $[i_2 + 1, i_3]$ von y_1 und so weiter.
- Im $\mathbb{R}^n$
  - verwendet man den r-Punkt Crossover ebenfalls.
  - Sehr verbreitet ist das **arithmetische Crossover**: $\sum_{i=1}^{k} \alpha_i y_i$ wird zum Nachkommen von $y_1, \ldots, y_k$. Der wichtige Spezialfall $\alpha_1 = \cdots = \alpha_k = \frac{1}{k}$ wird **intermediäre Rekombination** genannt.
- Im Permutationsraum $\mathbb{S}^n$ wählt man meistens $k = 2$ Eltern y_1 und y_2. Zum Beispiel werden Positionen $1 \leqslant i_1 < i_2 \leqslant n$ zufällig ausgewürfelt. Die in den Positionen $i_1, \ldots, i_2$ liegenden Komponenten von y_1 werden dann gemäß y_2 umgeordnet.

Zusammenfassung

- Die strikte lokale Suche führt zu einer verbesserten Lösung, die aber nur einem lokalen Optimum entspricht.
- Die lokale Suche in variabler Tiefe erlaubt Aufwärtsbewegungen und führt für das TSP und das Minimum Balanced Cut Problem zu guten Lösungen.
- Der Metropolis Algorithmus akzeptiert Aufwärtsbewegungen mit einer durch einen Temperatur-Parameter T gesteuerten Wahrscheinlichkeit. Wenn die Temperatur T langsam gesenkt wird, erhält man Simulated Annealing.
- Evolutionäre Algorithmen verbessern eine Lösungspopulation mit Hilfe von Mutations- und Crossover-Operatoren. Die Auswahl der neuen Population wird durch die Fitness, also durch den Funktionswert der Lösungen gesteuert.