INF FAKULTÄT FÜR INFORMATIK

Evolutionäre Algorithmen

Anwendungen von Evolutionären Algorithmen

Prof. Dr. Rudolf Kruse **Pascal Held**

{kruse,pheld}@iws.cs.uni-magdeburg.de

Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Institut für Wissens- und Sprachverarbeitung

Übersicht

Parallelisierung evolutionärer Algorithmen

EA: recht **teures Optimierungsverfahren**, da oft mit

- großer Population (tausende bis zehntausend Individuen)
- großer Zahl an Generationen (einige hundert)

gearbeitet werden muss für hinreichende Lösungsgüte

Vorteil: etwas höhere Lösungsgüte i.V.z. anderen Verfahren

Nachteil: unangenehm lange Laufzeit

Möglicher Lösungsansatz: **Parallelisierung**

Verteilung der notwendigen Operationen auf mehrere Prozessoren

Fragen:

- **Welche Schritte kann man parallelisieren?**
- **Welche vorteilhaften Techniken kann man bei der Parallelisierung zusätzlich anwenden?**

Was kann man parallelisieren?

Erzeugen der Anfangspopulation:

- meist problemlos, da i.A. Chromosomen der Anfangspopulation zufällig und unabhängig voneinander
- Versuch, Duplikate zu vermeiden, kann Parallelisierung behindern

  Parallelisierung dieses Schrittes eher wenig bedeutsam (Anfangspopulation wird nur einmal erzeugt)

Bewertung der Chromosomen:

- problemlos, da Chromosomen i.A. unabhängig voneinander bewertet (Fitness hängt nur vom Chromosom selbst ab)
- auch Gefangenendilemma: bearbeite Paarungen parallel

Berechnung der Ränge oder (relativen) Fitnesswerte:

- Bewertungen müssten hierfür zusammengeführt werden

Was kann man parallelisieren?

Selektion: ob Auswahlschritt parallelisierbar ist, hängt sehr stark vom verwendeten Selektionsverfahren ab

- **Erwartungswertmodell** und **Elitismus:**
  erfordern beide globale Betrachtung der Population und sind daher nur schwer parallelisierbar
- **Glücksrad-** und **Rangauswahl:**
  nachdem relativen Fitnesswerte bzw. Ränge bestimmt (was schwer parallelisierbar ist), ist Auswahl leicht parallelisierbar
- **Turnierauswahl:**
  ideal für Parallelisierung, besonders bei kleinen Turniergrößen, da keine globale Information notwendig (Vergleich der Fitnesswerte beschränkt sich auf Individuen des Turniers)

Was kann man parallelisieren?

Anwendung genetischer Operatoren
leicht zu parallelisieren, da jeweils nur ein (Mutation) oder zwei Chromosomen (Crossover) betroffen (zusammen mit Turnierauswahl: Steady-State-EA sehr gut parallelisierbar)

Abbruchbedingung:
einfacher Test, ob bestimmte Generationenzahl erreicht, bereitet bei Parallelisierung keine Probleme
Abbruchkriterien wie

- bestes Individuum der Population hat bestimmte Mindestgüte oder
- über bestimmte Anzahl von Generationen hat sich bestes Individuum nicht/kaum verbessert

dagegen für Parallelisierung weniger geeignet, da globale Betrachtung der Population erforderlich

Inselmodell

wenn z.B. schwer zu parallelisierendes Selektionsverfahren verwendet wird:

Parallelisierung erreichbar durch parallelle Berechnung mehrerer unabhängiger Populationen

jede Population bewohnt eine Insel, daher **Inselmodell**.

reines Inselmodell $\equiv$ mehrfache serielle Ausführung des gleichen EAs

liefert meist etwas schlechtere Ergebnisse als einzelner Lauf mit entsprechend größerer Population

Migration

zwischen Inselpopulationen können zu festgelegten Zeitpunkten (*nicht* in jeder Generation) Individuen ausgetauscht werden

Migration (Wanderung)

normalerweise keine direkte Rekombination von Chromosomen verschiedener Inseln

erst nach Migration: Rekombination genetischer Information einer Insel mit einer anderen Insel

Steuerung der Migration zwischen Inseln

Zufallsmodell:

zufälliges Bestimmen beider Inseln, zwischen denen Individuen ausgetauscht werden

beliebige Inseln können Individuen austauschen

Netzwerkmodell:

Inseln werden in Graphen angeordnet

Individuen wandern zwischen Inseln nur entlang der Kanten

Kanten werden zufällig bestimmt

Wettbewerb zwischen den Inseln

EAs, die auf Inseln angewandt werden, unterscheiden sich (in Verfahren und/oder Parametern)

Populationsgröße einer Insel wird entsprechend ihrer durchschnittlichen Fitness der Individuen erhöht oder erniedrigt

jedoch: Mindestpopulationsgröße, die nicht unterschritten wird

Zellulare evolutionäre Algorithmen

auch: “isolation by distance”

Prozessoren werden in (rechtwinkligen) Gitter angeordnet

Gitter bedeckt gewöhnlich Oberfläche eines Torus

Selektion und Crossover auf im Gitter benachbarte (durch Kanten verbundene), Mutation auf einzelne Prozessoren beschränkt

Beispiel: jeder Prozessor verwaltet ein Chromosom

- **Selektion:** Prozessor wählt bestes Chromosom seiner (vier) Nachbarprozessoren oder eines dieser Chromosomen zufällig nach ihrer Fitness
- **Crossover:** Prozessor führt Crossover mit gewähltem und eigenem Chromosom durch oder mutiert sein Chromosom (er behält besseren der beiden Nachkommen bzw. von Elter und Kind)

Bildung von Gruppen benachbarter Prozessoren, die ähnliche Chromosomen verwalten

mildert die oft zerstörende Wirkung des Crossover

Mühlenbeins Ansatz

Kombination von EAs **mit Zufallsaufstieg**

- jedes Individuum führt lokalen Zufallsaufstieg durch, d.h.
  - bei vorteilhafter Mutation wird Elter ersetzt
  - bei nachteiliger Mutation bleibt Elter erhalten
- Zufallsaufstieg leicht parallelisierbar
- Individuen suchen sich Crossover-Partner in ihrer Nachbarschaft (dazu: Definition des Abstandes zweier Individuen benötigt — vergleiche Nischentechniken, z.B. *power law sharing*)
- Nachkommen führen lokalen Zufallsaufstieg durch
- Individuen der nächsten Generation: nach **„lokalem“ Eliteprinzip** ausgewählt

$\Rightarrow$ übernehme beiden besten Individuen unter Eltern und optimierten Nachkommen

Übersicht

1. No-Free-Lunch-Theorem

2. **Parallelisierung evolutionärer Algorithmen**
   - Was kann man parallelisieren?
   - Inselmodell und Migration
   - Zellulare evolutionäre Algorithmen
   - Mühlenbeins Ansatz

3. Zufallszahlen

Zufallszahlen

EAs basieren i.A. auf zufälligen Veränderungen bestehender Lösungskandidaten und Selektion

alle erzeugten „Zufallszahlen“ $\neq$ richtig zufällig

von deterministischem Algorithmus erzeugt: **Pseudozufall**

- Versuche, Zufälligkeit physikalischer Prozesse auszunutzen Zahlen mit schlechteren Eigenschaften als deterministische Verfahren
- außerdem: Reproduzierbarkeit von Simulationen mit deterministischen Verfahren

Geschichtliches

aus [Knuth, 1997]

vor 1900: Forscher, die Zufallszahlen brauchten, zogen Kugeln aus „gut gemischter Urne“, rollten Würfel oder teilten Spielkarten aus

L. Tippet veröffentlicht 1927 Tabelle mit $\geq$ 40000 Zufallszahlen

Nachteile von Tabellen: aufwendig zu erstellen und zu speichern

Maschinen zur Generierung von Zufallszahlen

mechanische Methoden: fehleranfällig u. keine Reproduzierbarkeit

arithmet. Operationen auf Computern: John von Neumann (1946)

- bilde Quadrat der letzten Zufallszahl und extrahiere mittlere Ziffern
- z.B. 10-stellige Zufallszahlen und letzte Zahl sei 5772156649
- ihr Quadrat ist 33317792380594909201
- nächste Zahl also 7923805949

Einwand gegen arithmetische Methoden

Wie kann eine so generierte Sequenz zufällig sein, wenn jede Zahl vollständig durch ihren Vorgänger bestimmt ist?

Anyone who considers arithmetic methods of producing random digits is, of course, in a state of sin. — John von Neumann (1951)

- Antwort: Sequenz ist **nicht** zufällig, aber sie scheint es zu sein!
- deterministische Algorithmen erzeugen **pseudozufällige** Sequenzen
- von Neumanns Methode hat ihre Probleme:
  Sequenz tendiert zu kurzen Perioden wiederholender Zahlen
- z.B. bei 4-stelligen Zahlen: 6100, 2100, 4100, 8100, 6100, ...
- im Folgenden: Methoden, die von Neumann überlegen sind

Generierung gleichverteilter Zufallszahlen

- Generierung einer Sequenz reeller Zahlen gleichverteilt zwischen 0 und 1
- Endlichen Genauigkeit des Computers bewirkt, dass eigentlich ganze Zahlen X_n zwischen 0 und einer Zahl m bestimmt werden
- Bruchteil $U_n = X_n/m$ liegt zwischen 0 und 1
- normalerweise: m ist Wortgröße des Computers w

Die linear kongruente Methode

- bekanntesten Zufallszahlengeneratoren sind Spezialfälle des folgenden Schemas [Lehmer, 1951]
- wir wählen 4 magische Ganzzahlen

$$\begin{aligned} m, &\quad \text{Modulus;} && 0 < m, \\ a, &\quad \text{Faktor;} && 0 \leq a < m, \\ c, &\quad \text{Inkrement;} && 0 \leq c < m, \\ X_0, &\quad \text{Startwert;} && 0 \leq X_0 < m \end{aligned}$$

- gewünschte Sequenz von Zufallszahlen $\langle X_n \rangle$ durch

$$X_{n+1} = (a \cdot X_n + c) \bmod m, \qquad n \geq 0$$

- Rest mod m: Ortsbestimmung einer Kugel auf drehendem Roulettekessel

Die linear kongruente Methode

- für z.B. $m = 10$ und $X_0 = a = c = 7$ folgt

$$\langle X_n \rangle = 7, 6, 9, 0, 7, 6, 9, 0,$$

- Sequenz also nicht immer „zufällig“ für alle Werte von m, a, c, X_0
- solche Schleifen: bei allen Sequenzen der Form $X_{n+1} = f(X_n)$
- sich wiederholender Kreis: **Periode**
- Ziel: nützliche Sequenzen mit relativ langer Periode

Günstige Parameterwahl

- Startwert X_0 beliebig
- Modulo $m \geq 2^{30}$ oder größte Primzahl kleiner $w = 2^e$ wobei $e = \#$ darstellbare bits

$\Rightarrow$ *Sequenz* mit maximaler Länge

- falls m Potenz von 2: Faktor $a = 2^k + 1$ wobei $2 \leq k \leq w$

$\Rightarrow$ *Periode* mit maximaler Länge

- Inkrement c nebensächlich falls a gut: aber c darf keinen gemeinsamen Teiler mit m haben (z.B. $c = 1$ oder $c = a$)

$\Rightarrow$ vermeide Multiplikation durch Schieben und Addieren:

$$X_{n+1} = ((2^k + 1) \cdot X_n + 1) \bmod 2^e$$

- generiere höchstens $m/1000$ Zahlen

Andere Methoden

- quadratische Methode von R. R. Coveyou: sei

$$X_0 \bmod 4 = 2, \qquad X_{n+1} = X_n(X_n + 1) \bmod 2^e, \qquad n \geq 0$$

- berechenbar mit ähnlicher Effizienz wie linear kongruente Methode
- Mitchell and Moore (1958) schlugen folgende Methode vor

$$X_n = (X_{n-24} + X_{n-55}) \bmod m, \qquad n \geq 55$$

  wobei m gerade und $X_0, \ldots, X_{54}$ beliebig (nicht alle gerade)
- sehr effizient implementierbar mittels zyklischer Liste
- Periode von $2^{55} - 1 \Rightarrow$ vermutlich bester Algorithmus

Generierung von normalverteilten Zufallszahlen

Polar-Methode [Box and Muller, 1958]:
seien U_1, U_2 unabhängig zufällig gleichverteilt aus $[0, 1]$
folgende Zufallszahlen sind aus der selben $N(0, 1)$

$$X_1 = \sqrt{-2 \ln U_1} \cos 2\pi U_2, \qquad X_2 = \sqrt{-2 \ln U_1} \sin 2\pi U_2$$

- Beweis: inverse Beziehungen sind

$$U_1 = e^{\frac{-(X_1^2 + X_2^2)}{2}}, \qquad U_2 = -\frac{1}{2\pi} \arctan \frac{X_2}{X_1}$$

$\Rightarrow$ multivariate Dichte von X_1, X_2 ist

$$\begin{aligned} f(X_1, X_2) &= \frac{1}{2\pi} e^{\frac{-(X_1^2 + X_2^2)}{2}} \\ &= \frac{1}{\sqrt{2\pi}} e^{\frac{-X_1^2}{2}} \cdot \frac{1}{\sqrt{2\pi}} e^{\frac{-X_2^2}{2}} = f(X_1) \cdot f(X_2) \end{aligned}$$

Generierung von normalverteilten Zufallszahlen

Algorithm Polar-Methode

Output: zwei unabhängige, normalverteilte Zufallszahlen X_1, X_2

1: **do** $\{$
2: $\quad U_1, U_2 \leftarrow$ generiere 2 unabhängige Zufallszahlen aus $U([0,1])$
3: $\quad V_1 \leftarrow 2U_1 - 1$
4: $\quad V_2 \leftarrow 2U_2 - 1$
5: $\quad S \leftarrow V_1^2 + V_2^2$
6: $\}$ **while** $S < 1.0$ **and** $S \neq 0$
7: $X_1 \leftarrow V_1 \sqrt{\frac{-2 \ln S}{S}}$
8: $X_2 \leftarrow V_2 \sqrt{\frac{-2 \ln S}{S}}$
9: **return** X_1, X_2

Zufallszahlen für EAs

- sinnvoll für objektorientierter Entwurf:
  einen eigenen Zufallszahlengenerator pro Individuum
- Nachteil: Folgen aufgrund Zufälligkeit nicht abschätzbar

$\Rightarrow$ pro Optimierungsverfahren: nur einen Zufallszahlengenerator

- klar definierter Startwert sinnvoll $\Rightarrow$ Experimente reproduzierbar
- Systemzeit oder letzte erzeute Zufallszahl als Saat nicht sinnvoll

Literatur zur Lehrveranstaltung I

Box, G. E. P. and Muller, M. E. (1958).
A note on the generation of random normal deviates.
Annals of Mathematical Statistics, 29(2):610–611.

Knuth, D. E. (1997).
The art of computer programming, volume 2: Seminumerical algorithms.
pages 1–193. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 3rd edition.

Lehmer, D. H. (1951).
Mathematical methods in large-scale computing units.
In *Proc. 2nd Symp. on Large-Scale Digital Calculating Machinery*, pages 141–146, Cambridge, MA, USA. Harvard University Press.