

# Evolutionäre Algorithmen

## No-Free-Lunch-Theorem, Parallelisierung, Zufallszahlen

**Prof. Dr. Rudolf Kruse**      **Christian Moewes**

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

Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Institut für Wissens- und Sprachverarbeitung

# Übersicht

## 1. No-Free-Lunch-Theorem

Formale Definitionen

Das Theorem

Beispiel

Konsequenzen und Zusammenfassung

## 2. Parallelisierung evolutionärer Algorithmen

## 3. Zufallszahlen

# Der ultimative evolutionäre Algorithmus

## Überlegenheit des EA im Vergleich zu anderen Verfahren

- hängt stark vom Problem ab
- Idee eines universellen Optimierers
  - Erwartung eines solchen Optimierers?
  - bedenke: betrachtete Optimierungsprobleme seien unbekannt!

# Formale Definitionen

- Suchraum  $\Omega$
- Raum aller Optimierungsprobleme  $\mathcal{F}$  (Zielfunktionen)
- $\Omega$  und  $\mathcal{F}$  seien endlich
  - da Speicherplatz im Computer ebenfalls endlich
- Unwissenheit über Optimierungsproblem
  - Gleichverteilung aller dieser Probleme
  - jedes Problem  $F \in \mathcal{F}$  tritt mit Wahrscheinlichkeit  $\frac{1}{|\mathcal{F}|}$  auf
- weitere Vereinfachung
  - $\forall F \in \mathcal{F}$  gilt  $F : \Omega \mapsto \mathbb{R}$
  - $\forall F \in \mathcal{F}$  sind auf selbem Suchraum  $\Omega$  definiert
- $\mathcal{A}$  sei Menge aller Optimierungsalgorithmen, die auf  $\Omega$  arbeiten

# Formale Definitionen

Charakterisierung eines Algorithmus

- welche Ind. werden in welcher Reihenfolge auf  $F \in \mathcal{F}$  betrachtet
  - lediglich  $n$  Auswertungen zur Optimierung möglich:
    - Optimierung $_{F,n} : \mathcal{A} \mapsto \Omega^n$
  - bei jeder Optimierung bewertet Algorithmus ein Individuum 1x
- ⇒ Optimierung $_{F,n}(\text{Alg})$  enthält insgesamt  $n$  untersch. Individuen
- jeder Algorithmus Alg sei deterministisch
- ⇒ Optimierung $_{F,n}(\text{Alg})$  auch eindeutig

Für Problem  $F \in \mathcal{F}$ , Optimierungsproblem  $\text{Alg} \in \mathcal{A}$  und  $n \in \mathbb{N}$  ist

$$\text{Optimierung}_{F,n}(\text{Alg}) = (y_1, \dots, y_n) \in \Omega^n$$

mit  $y_i \neq y_j$  für  $i \neq j$  wobei  $y_k$  Individuum, das Alg mit  $F$  als  $k$ -tes Element untersucht.

# Vergleich zweier Algorithmen

Vergleich von  $\text{Alg}_1, \text{Alg}_2 \in \mathcal{A}$

- mittels Leistungsmaß  $\text{QuAlg}$  (Qualität eines Algorithmus)
  - definiert mit Optimierung  $f_{F,n}(\text{Alg}) = (y_1, \dots, y_n)$  durch  $q_n : \mathbb{R}^n \mapsto \mathbb{R}$  als

$$\text{QuAlg}_{f,n}(\text{Alg}) = q_n(f(y_1), \dots, f(y_n))$$

- z.B. durchschnittliche bzw. beste Güte oder
- Anzahl der benötigten Auswertungen bis Optimum erreicht
- zu erwartende Leistung  $E$  der  $n$  ersten Auswertungen von Alg auf beliebigem Problem

$$E \left[ \text{QuAlg}_{f,n}(\text{Alg}) \mid F \in \mathcal{F} \right] = \frac{1}{\#\mathcal{F}} \sum_{F \in \mathcal{F}} \text{QuAlg}_{f,n}(\text{Alg})$$

⇒ Mittel über alle möglichen Probleme

# No Free Lunch

Falls vorherigen Annahmen stimmen, gilt folgender Satz.

## Satz (No free lunch)

*Für je zwei Algorithmen  $Alg_1, Alg_2 \in \mathcal{A}$  und die Klasse aller Probleme  $\mathcal{F}$  gilt bezüglich eines Leistungsmaßes  $QuAlg$ :*

$$E \left[ QuAlg_{F,n}(Alg_1) \mid F \in \mathcal{F} \right] = E \left[ QuAlg_{F,n}(Alg_2) \mid F \in \mathcal{F} \right]$$

## Beweis I/II

- o.B.d.A. sei  $\Omega = \{x_1, \dots, x_m\}$
- o.B.d.A. seien  $r_i \in \mathbb{R}$  ( $1 \leq i \leq m$ ) vorkommenden Güterwerte
- jede mögliche  $F \in \mathcal{F}$  ist nun definiert über Permutation  $\pi \in \mathcal{S}_m$
- Zuweisung der Güterwerte zu Punkte in  $\Omega$  durch  $F(x_i) = r_{\pi(i)}$  für  $1 \leq i \leq m$

$\Rightarrow m!$  unterschiedliche Funktionen in  $\mathcal{F}$

- bei einer Optimierung
  - der Reihe nach Betrachtung der Punkte  $y_1, y_2, \dots$
  - Wahl des ersten Punkts  $y(= x_{j_1})$  völlig unabhängig von  $F$
  - jeder dem  $m$  Güterwerte  $r_i$  steht bei genau  $(m - 1)!$  Funktionen an Stelle  $x_{j_1}$



## Beweis II/II

$i$ -te Iteration ( $i > 1$ )

- $i - 1$  bisher gewählten Punkte können in  $m \cdot \dots \cdot (m - i + 2) = \frac{m!}{(m-i+1)!}$  resultieren
- jede dieser Gütefolgen kann beim Betrachten des  $i$ -ten Punktes mit  $m - i + 1$  verschiedenen Gütewerten als  $y_i = x_{ji}$  forgesetzt werden
- dies gilt auf wieder für  $(m - 1)!$  Funktionen
- also gilt für beliebiges  $i$ 
  - jede entdeckte Reihenfolge kommt bei genau gleich vielen Funktionen
  - völlig unabhängig vom Algorithmus

$\Rightarrow E \left[ \text{QuAlg}_{F,n}(\text{Alg}) \mid F \in \mathcal{F} \right]$  für jeden Alg identisch

## Beispiel zum Beweis

jede Zeile  $\hat{=}$  mögliche Bewertungsfunktion, 1. Punkt von Alg ist  $x_1$ :

$x_1$	$x_2$	$x_3$	$x_4$	$x_1$	$x_2$	$x_3$	$x_4$	$x_1$	$x_2$	$x_3$	$x_4$	$x_1$	$x_2$	$x_3$	$x_4$
1	2	3	4	2	1	3	4	3	1	2	4	4	1	2	3
1	2	4	3	2	1	4	3	3	1	4	2	4	1	3	2
1	3	2	4	2	3	1	4	3	2	1	4	4	2	1	1
1	3	4	2	2	3	4	1	3	2	4	1	4	2	3	1
1	4	2	3	2	4	1	3	3	4	1	2	4	3	1	2
1	4	3	2	2	4	3	1	3	4	2	1	4	3	2	1

zweiter betrachteter Punkt abhängig von Güte des Ersten:

$x_1$	$x_2$	$x_3$	$x_4$	$x_1$	$x_2$	$x_3$	$x_4$	$x_1$	$x_2$	$x_3$	$x_4$	$x_1$	$x_2$	$x_3$	$x_4$
1	2	3	4	2	1	3	4	3	1	2	4	4	1	2	3
1	2	4	3	2	1	4	3	3	1	4	2	4	1	3	2
1	3	2	4	2	3	1	4	3	2	1	4	4	2	1	1
1	3	4	2	2	3	4	1	3	2	4	1	4	2	3	1
1	4	2	3	2	4	1	3	3	4	1	2	4	3	1	2
1	4	3	2	2	4	3	1	3	4	2	1	4	3	2	1

## Beispiel zum Beweis

- 3. betrachteter Punkt hängt von Güte der ersten Beiden ab
- Vergleich zweier unterschiedlicher Algorithmen (oben und unten)

$x_1$	$x_2$	$x_3$	$x_4$
1	2	3	4
1	2	4	3
1	3	2	4
1	3	4	2
1	4	2	3
1	4	3	2

$x_1$	$x_2$	$x_3$	$x_4$
2	1	3	4
2	1	4	3
2	3	1	4
2	3	4	1
2	4	1	3
2	4	3	1

$x_1$	$x_2$	$x_3$	$x_4$
3	1	2	4
3	1	4	2
3	2	1	4
3	2	4	1
3	4	1	2
3	4	2	1

$x_1$	$x_2$	$x_3$	$x_4$
4	1	2	3
4	1	3	2
4	2	1	1
4	2	3	1
4	3	1	2
4	3	2	1

$x_1$	$x_2$	$x_3$	$x_4$
1	2	3	4
1	2	4	3
1	3	2	4
1	3	4	2
1	4	2	3
1	4	3	2

$x_1$	$x_2$	$x_3$	$x_4$
2	1	3	4
2	1	4	3
2	3	1	4
2	3	4	1
2	4	1	3
2	4	3	1

$x_1$	$x_2$	$x_3$	$x_4$
3	1	2	4
3	1	4	2
3	2	1	4
3	2	4	1
3	4	1	2
3	4	2	1

$x_1$	$x_2$	$x_3$	$x_4$
4	1	2	3
4	1	3	2
4	2	1	1
4	2	3	1
4	3	1	2
4	3	2	1

# Konsequenzen

- kein Alg. ist im Mittel über alle möglichen Probleme einem Anderen überlegen
- gäbe es Algorithmus, der auf  $\mathcal{F}' \subset \mathcal{F}$  überlegen, also

$$E \left[ \text{QuAlg}_{F,n}(\text{Alg}_1) \mid F \in \mathcal{F}' \right] < E \left[ \text{QuAlg}_{F,n}(\text{Alg}_2) \mid F \in \mathcal{F}' \right]$$

- dann folgt sofort

$$E \left[ \text{QuAlg}_{F,n}(\text{Alg}_1) \mid F \in \mathcal{F} \setminus \mathcal{F}' \right] > E \left[ \text{QuAlg}_{F,n}(\text{Alg}_2) \mid F \in \mathcal{F} \setminus \mathcal{F}' \right]$$

- folglich
  - für jeden Algorithmus:  $\exists$  Nische im Raum aller Probleme, für die Algorithmus besonders gut geeignet
  - Anwendung: welchen Algorithmus nutzen bei gegebenem Problem?
  - Wissenschaft: welche Klasse von Problemen optimal für bestimmten Algorithmus?

# Zusammenfassung

- falls keinerlei Problemwissen vorhanden
  - ⇒ Erwartung von EA gegenüber beliebigem Verfahren nicht höher
- falls Problemwissen vorhanden
  - z.B. Annahmen über gewisses Wohlverhalten der Gütelandschaft
  - ⇒ generelle Anwendbarkeit von bestimmten Algorithmen wird nahegelegt
- Wissen über Struktur des Problems
  - fließt in Auswahl oder in Entwurf des Optimierunsalgorithmus ein

# Übersicht

## 1. No-Free-Lunch-Theorem

## 2. Parallelisierung evolutionärer Algorithmen

Was kann man parallelisieren?

Inselmodell und Migration

Zellulare evolutionäre Algorithmen

Mühlenbeins Ansatz

## 3. Zufallszahlen

# 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 Auswahlsschritt 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 parallele 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 Inselepopulationen 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
- ⇒ übernehme beiden besten Individuen unter Eltern und optimierten Nachkommen



# Übersicht

1. No-Free-Lunch-Theorem
2. Parallelisierung evolutionärer Algorithmen
- 3. Zufallszahlen**
  - Generierung gleichverteilter Zufallszahlen
  - Linear kongruente Methode
  - Normalverteilte Zufallszahlen
  - Zufallszahlen für EAs

# 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. Tippett 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

- hier: Generierung einer Sequenz reeler Zahlen gleichverteiltet zwischen 0 und 1
- aufgrund der endlichen Genauigkeit des Computers: eigentlich Generierung ganzer Zahlen  $X_n$  zwischen 0 und einer Zahl  $m$
- 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
  - $m$ , Modulus;  $0 < m$ ,
  - $a$ , Faktor;  $0 \leq a < m$ ,
  - $c$ , Inkrement;  $0 \leq c < m$ ,
  - $X_0$ , Startwert;  $0 \leq X_0 < m$
- gewünschte Sequenz von Zufallszahlen  $\langle X_n \rangle$  durch

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

- 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

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

- 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

⇒ Sequenz mit maximaler Länge

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

⇒ Periode mit maximaler Länge

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

⇒ 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

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

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

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

wobei  $m$  gerade und  $X_0, \dots, X_{54}$  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  $N(0, 1)$

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

- Beweis: inverse Beziehungen sind

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

⇒ 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 1 Polar-Methode

---

**Output:** zwei unabhängige, normalverteilte Zufallszahlen  $X_1, X_2$




```
1: do {  
2:    $U_1, U_2 \leftarrow$  generiere 2 unabhängige Zufallszahlen aus  $U([0, 1])$   
3:    $V_1 \leftarrow 2U_1 - 1$   
4:    $V_2 \leftarrow 2U_2 - 1$   
5:    $S \leftarrow V_1^2 + V_2^2$   
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
- ⇒ pro Optimierungsverfahren: nur einen Zufallszahlengenerator
- 
- klar definierter Startwert sinnvoll ⇒ Experimente reproduzierbar
  - Systemzeit oder letzte erzeugte 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.