Differential Evolution (DE)

Einführung

Über mich

- Martin Ankerl
  - martin.ankerl@gmail.com
  - http://martin.ankerl.com
  - http://j.mp/MartinAnkerlLinkedin
- 2000-2004 FH Hagenberg
  - Software Engineering
- 2004-heute PROFACTOR, Researcher
  - Vision Algorithmen (3D Objekterkennung, …)
  - Robotik (Kollisionsfreie Pfadplanung, …)

Inhalt

- Grundlagen - Was ist DE?
- Motivation - Wozu DE?
- Komzept Evolutionäre Algorithmen
- Beispiel: DE für Antennendesign
  - Parametrierung
  - Initialisierung
  - Mutation & Rekombination
  - Selektion
- Sourcecode
- Zusammenfassung

Grundlagen - Was ist DE?

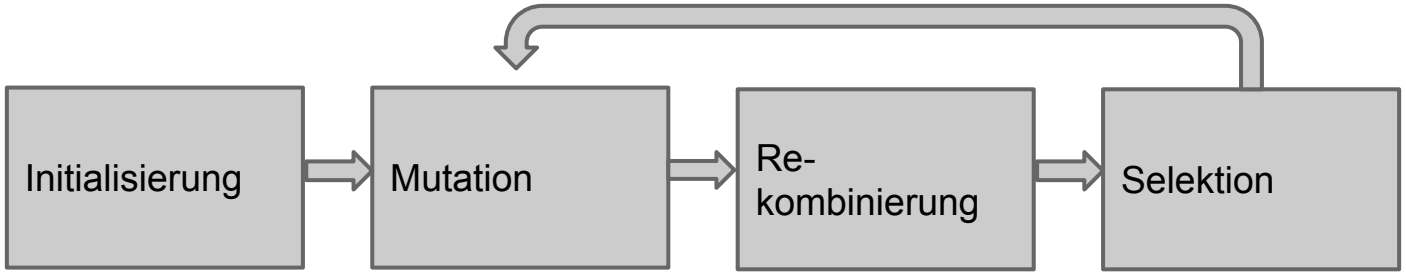
- Zufallsbasierter, populationsbasierter Optimierungsalgorithmus
- Vergleichbar mit Genetischen Algorithmen
- Entwickelt 1996 zur Optimierung von Funktionsparameter (floating point Werte)

Motivation- Wozu DE?

- Optimierung von Parametern
- Mathematisch schwer lösbare Probleme
- Oft reicht gute Lösung, nicht die beste
- DE ist einfach
  - Implementierung
  - Parametrisierung
- Sehr gute Ergebnisse

Evolutionäre Algorithmen

- DE ist Evolutionärer Algorithmus
- NP Partikel, jeder Partikel repräsentiert einen Parametersatz (Lösung).



Beispiel: DE für Antennendesign

- Gegeben
  - Simulation die Parametersatz bewertet
- Gesucht
  - optimale Parameter für Antennenform (Länge, Breite, Distanzen) zur maximale Reflexion

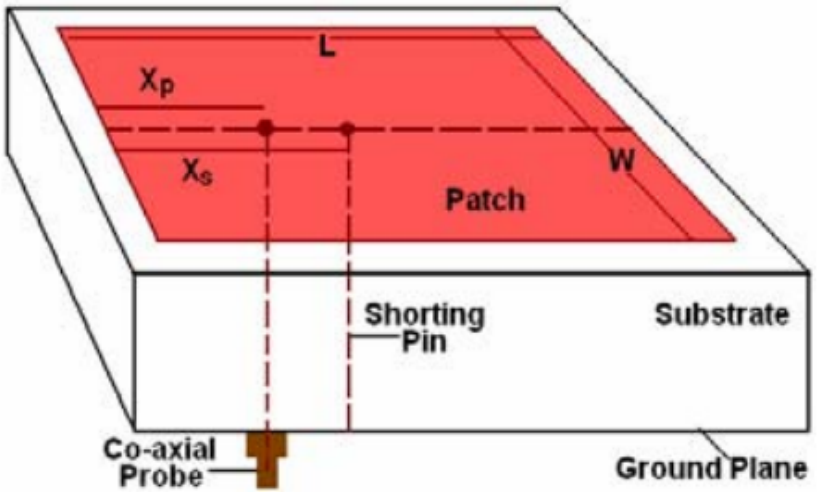
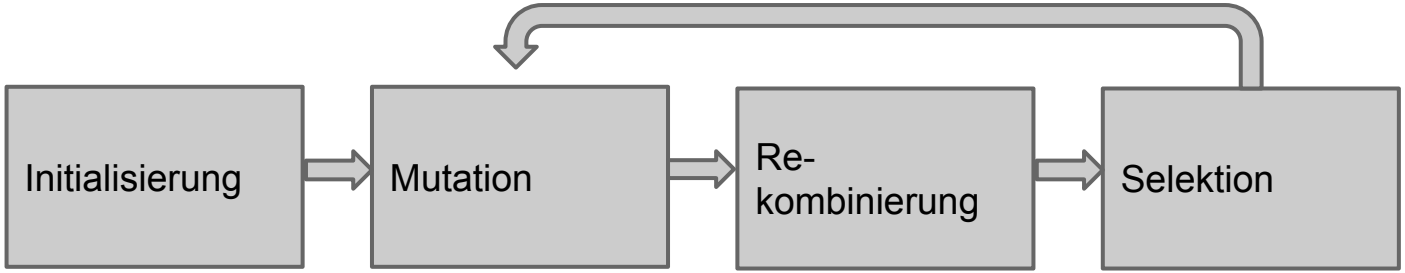


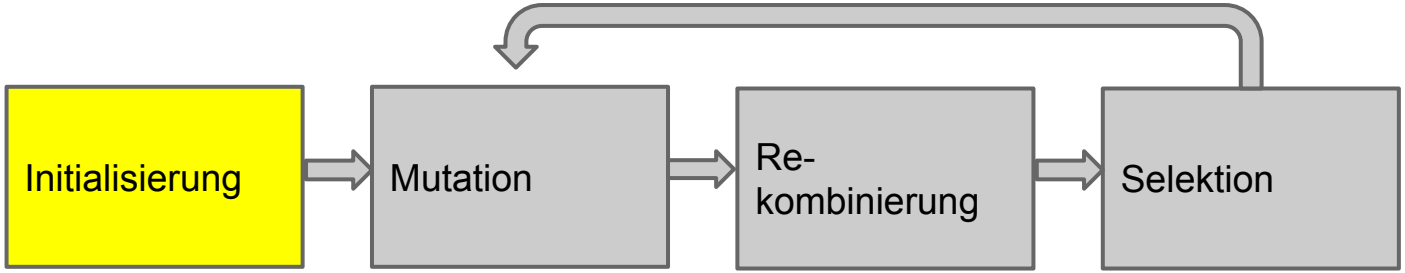
Fig. 1. Microstrip antenna with shorted pin

Parametrierung



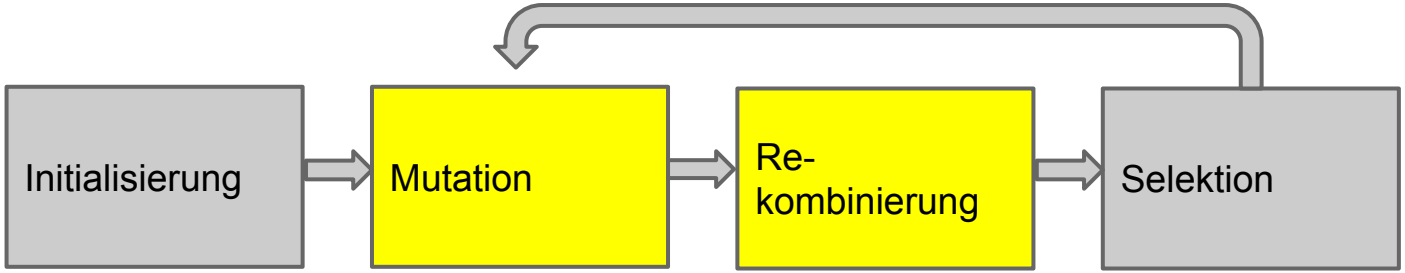
1. Partikel Repräsentation: (L, W, xp, xn)
2. Unteres und oberes Limit für L, W, xp, xn.
3. DE Parameter: NP=40, CR=0.35, F=0.1

Initialisierung



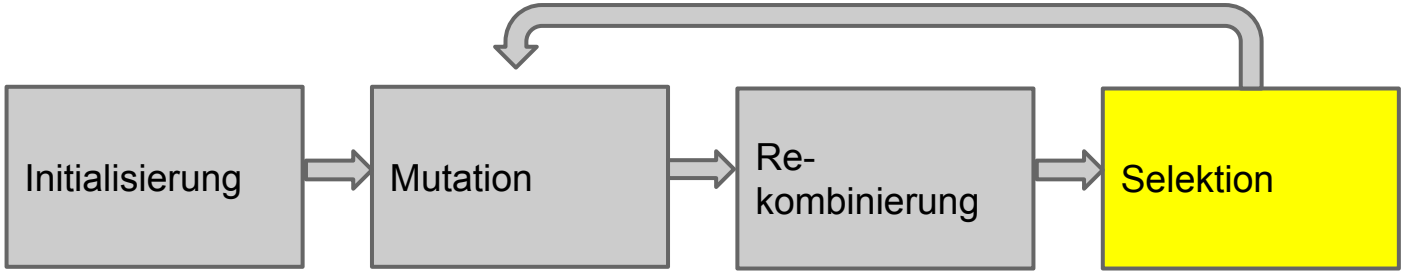
1. Alle NP Partikel mit Zufallswerten zwischen oberen & unteren Limit initialisieren.

Mutation & Rekombinierung



1. Mutation ändert Parameter (erweitert Suchraum)
2. Rekombinierung fügt gute Lösungen zusammen

Selektion



1. Neue Lösung wird evaluiert und ersetzt vorherige Lösung falls besser

Sourcecode

```
/* Example adapted from http://www.drdobbs.com/database/differential-evolution/184410166
 * This implements the DE/rand/1/bin optimization algorithm. */

/* Initialize individuals */
for (i=0; i<NP; i++) {
    /* randomly initialize all individuals */
    for (j=0; j<D; j++) {
        currentPos[i][j] = rnd_uni()*(maxPos[j] - minPos[j]) + minPos[j];
    }
    cost[i] = evaluate(currentPos[i]);
}
```

```
/* Halt after gen_max generations. */
while (count < gen_max) {

    /* Start loop through population. */
    for (i=0; i<NP; i++) {
        /********** Mutate/recombine **********/
        /* Randomly pick 3 vectors, all different from i */
        do a = rnd_uni()*NP; while (a==i);
        do b = rnd_uni()*NP; while (b==i || b==a);
        do c = rnd_uni()*NP; while (c==i || c==a || c==b);

        /* Randomly pick an index for forced evolution change */
        k = rnd_uni()*D;
        /* Load D parameters into trialPos[]. */
        for (j=0; j<D; j++) {
            /* Perform D-1 binomial trials. */
            if (rnd_uni() < CR || j==k) {
                /* Source for trialPos[j] is a random vector plus weighted differential */
                trialPos[j] = currentPos[c][j] + F * (currentPos[a][j] - currentPos[b][j]);
            } else {
                /* or trialPos parameter comes from currentPos[i][j] itself. */
                trialPos[j] = currentPos[i][j];
            }
        }
```

```
        /********** Evaluate/select **********/
        /* Evaluate trialPos with your function. */
        score = evaluate(trialPos);
        /* If trialPos[] improves on currentPos[i][], store it */
        if (score <= cost[i]) {
            for (j=0; j<D; j++) {
                personalBestPos[i][j] = trialPos[j];
            }
            cost[i] = score;
        } else {
            /* otherwise, move currentPos[i][] to secondary array. */
            for (j=0; j<D; j++) {
                personalBestPos[i][j] = currentPos[i][j];
            }
        }
    }
    /********** End of population loop; swap arrays **********/
    for (i=0; i<NP; i++) {
        /* After each generation, move secondary array into primary array. */
        for (j=0; j<D; j++) {
            currentPos[i][j] = personalBestPos[i][j];
        }
    }
    count++;
}
```

Zusammenfassung

- DE ist einfach & effizient
- Breites Anwendungsgebiet
  - Bei PROFACTOR: Optimierung Roboterpositionen zur Datenaufnahme, Bestimmung Roboterkinematik, Parameteroptimierung für exponential smoothing forecast, Tuning Kamerakalibrierungsparameter, …

Links

- Präsentation
  - https://docs.google.com/presentation/d/16WqRc6N5CEwDNsuVp2rmtR7bSNkWx10UoKWIeuopOic/edit?usp=sharing
- Sourcecode
  - https://gist.github.com/martinus/7434625df79d820cd4d9
- Google “differential evolution” :-)