OIP

JA.COM

Johannes Barth, Artur Front, Christina Wecker, Onur Göl & Miguel Moreno

Auswahl des Algorithmus

Ergebnis:

- Wahl des Genetischen Algorithmus
- Gründe:
  - schnelle Laufzeit
  - vermeiden von “Hängenbleiben” von lokalen Minima
  - relativ einfache Implementierung

Organisation Quellcode

Verfügbar unter:

- https://github.com/buegeleisen/oip
- Als Java Maven Projekt
- Eclipse Projekt

Aufbau des Projekts - God Klasse

- God Klasse
  - Erstellen von Random Startwerten (init())
  - Evaluieren der Populationen (immer in Vector<Individual> abgelegt)
- Festlegbare Größen bzw. Attribute
  - populationCount
    - Gibt Anzahl der Startwerte an
  - survivor
    - Anzahl der überlebenden besten Individuen
  - rangeMax
    - Maximal annehmbare Werte der Individuen
  - rangeMin
    - Minimal annehmbare Werte der Individuen

Aufbau des Projekts - God Klasse

- Weitere festlegbare Größen:
  - dimension
    - Dimensionen des Vektors (Im Fall der zu minimierenden Funktion → 17)
  - mutation
    - Anzahl der zu mutierenden Individuen

Aufbau des Projekts - God Klasse - Evolve

- Weiterentwicklung einer Population - Vorgehen:
  a. Auswahl der besten Individuen (Fitness ist hierbei anders als bei den herkömmlichen Implementierungen so, dass der niedrigste Wert die beste Fitness darstellt)
  b. Streu - Funktion (scatter):
     - Lokale Suche - Vergleichbar mit Streuung von Krebs
     - Jedes Individuum wird geklont und zufällig verändert. (Wertebereich zwischen 0-1)
     - Beschleunigung des Algorithmus um Faktor xy (Zuletzt 100)
  c. Kreuzung - Crossover:
     - Die [survivor] besten Individuen werden miteinander gekreuzt
     - Besseres Individuum erhält mit Faktor 0,8 seinen Wert + 0,2 * Wert des schlechteren Individuums

Aufbau des Projekts - God Klasse - Evolve

- Weiterentwicklung einer Population - Vorgehen Fortsetzung:
  d. mutieren - mutate:
    - Individuen werden geklont und zufällig mutiert
- Scatter Funktion ist prinzipiell Abwandlung einer Mutation

Aufbau des Projekts - Individual Klasse

- Individual Klasse
  - Repräsentiert den Übergabe Vektor an RabbitMQ

Ergebnisse aus Testfunktionen

- Testen mit lokalen Funktionen zeigt, dass das Minimum schnell gefunden wird
  - Abhängig von gewählter Testfunktion in nur wenigen Generationen
  - Auch abhängig von gewählter Größe der Startpopulation usw.
  - Scatter Funktion beschleunigt Algorithmus um hohen Faktor (Abhängig von Funktion ist Faktor > 100)
  - 17 Dimensionale Sphere (Minimum bei 0 für alle Werte 0) mit Genauigkeit 0.1:
    - Mit Scatter Funktion: ca. 150-200 Generationen
    - Ohne Scatter Funktion: Bei Generation 4-5000 ca einen Wert von 2-8

Abbruchkriterium

- Definition eines Fitnesswertes, der ein zufriedenstellendes Ergebnis darstellt
- Wenn Wert erreicht, aber Ergebnis noch zu ungenau -> Neustarten mit neuem Höchstfitnesswert

GodThread

- Überlegung mehrere Threads laufen zu lassen
- Theoretisch möglich aber nicht getestet weil nur eine Inbound Queue
- Gestartet wird deshalb nur ein Thread

Verwendete Testfunktionen

- Rastrigin Funktion
- Rosenbrock Funktion
- Spheren Funktion
- Himmelblau Funktion
- Easom Funktion
- Eggholder Funktion

https://en.wikipedia.org/wiki/Test_functions_for_optimization