Evolutionäre Optimierungsalgorithmen

Federico Ramírez Villagrana

Universität Hamburg
Methodenkompetenz - Einführung in das wissenschaftliche Arbeiten
Dozent: Dr. Andreas Günther

Zusammenfassung

In diesem Paper wird darüber diskutiert, sowohl was Optimierung ist und warum ist es schwer, als auch was evolutionäre Algorithmen (EA) sind und wie, warum, und wann sind sie hilfreich im Optimierung-Bereich. Es wird auch im Detail erklärt, wie EA funktionieren und warum sind sie als teil der künstlichen Intelligenz betrachtet. Es werden auch einige bekannte EA erwähnt und oberflächlich erklärt. Am ende wird darüber diskutiert, welche Vor- und Nachteile die EA haben.

I. Einleitung

Das Problem des Handlungsreisenden (engl. traveling salesman problem oder TSP) ist ein sehr bekanntes und studiertes Problem im Informatik-Bereich. Es geht wie folgendes:

Eine Reihenfolge für den Besuch mehrerer Orte muss so gewählt sein, dass keine Station außer der ersten mehr als einmal besucht wird, die gesamte Reisestrecke des Handlungsreisenden möglichst kurz, und die erste Station gleich der letzten Station ist. [wikc]

Sei n die Anzahl der Stationen, dann es $(n-1)!$ mögliche Lösungen für das TSP gibt. Daher wenn $n = 4$, dann es 6 mögliche Lösungen gibt. Es ist nicht schwer, einen brute-force [1] Ansatz zu verfolgen, für ein Problem, das nur 6 Lösungen hat. Aber wenn $n = 50$, dann es circa $6,1 \times 10^{62}$ Lösungen gibt.

Folgendes ist hilfreich, um diese Anzahl in einer Perspektive zu setzen: Das Universum ist circa 15 Milliarde Jahre alt, das ist $4,7 \times 10^{17}$ Sekunden. Wenn es eine Billion Rechner gäbe, die jede einzeln seit dem Anfang des Universums eine Billion Lösungen pro Sekunde berechnete, bisher wären nur $4,7 \times 10^{41}$ Lösungen berechnet worden.

Das TSP ist nur ein Beispiel von vielfältigen Problemen, die zu den kombinatorischen Problemen gehören, das heißt, Probleme, für die kein brute-force Ansatz möglich ist. In diesem Fall sind evolutionärer Algorithmen (EA) ein gutes Werkzeug, um gute Lösungen zu finden.

Natürlich können wir nicht sicher sein, dass wir die beste Lösung gefunden haben, außer wenn wir jede mögliche Lösung berechnet haben. Aber wie mit dem TSP-Beispiel gezeigt worden, ist es jedoch nicht immer möglich, alle mögliche Lösungen (in angemessener Zeit) zu berechnen.

II. Stand der Forschung

In den 70ern Jahren wurde erstmals über EA diskutiert. Die erste Form von EA waren die genetische Algorithmen, die in [Hol75] eingeführt würden. Seit damals wurden jedes Jahr mehrere EA präsentiert. Es ist auch üblich, dass nicht nur Varianten von schon bekannten EA eingeführt werden, sondern auch neue Anwendungen. In [alg] allein werden circa 48 EA

[1] Auch Exhaustionsmethode, ist eine Lösungsmethode für Probleme, die auf dem Ausprobieren aller möglichen (oder zumindest vieler möglicher) Fälle beruht. [wika]

gelistet.

Heutzutage finden sich immer mehr Anwendungen der EA in nahezu allen Bereichen. Von der Medizin [CAPR00] und Biologie [Wonwn] über das Motor- [AK09] und Flugzeugdesign [AFwn] [IAC14] bis hin zur Kunst [DeL12] [Mü12] und Wirtschaft [MGCT07].

III. Optimierung

Im Mathematik-Bereich bedeutet Optimierung, die Findung von Parametern eines Systems, die ein bestmögliches Ergebnis erzielen. [wikb] Wir können diese Findung von Parametern auch wie folgendes definieren: Die Auswahl der bestmöglichen Lösung eines Problems von einer Menge möglicher Lösungen.

In Optimierungsprobleme wird eine Zielfunktion (engl. objective function) definiert, die entweder maximiert oder minimiert werden soll. Die Zielfunktion wird „Kosten-Funktion“ oder „Fitness-Funktion“ in Minimierung- und Maximierungsprobleme beziehungsweise genannt.

Sei $f(x)$ eine Zielfunktion, dann x ein Vektor ist, der „Entscheidungsvariable“ genannt wird. Die Anzahl von Elementen in x wird die „Dimension“ des Problems genannt. Die Domäne der Zielfunktion repräsentiert die mögliche Lösungen des Problems.

Das Ziel der Optimierung ist es denn, die bestmögliche Lösung des Problems zu finden, das heißt, einen Wert für die Entscheidungsvariable zu finden, sodass die Zielfunktion einen Maximal- beziehungsweise Minimalwert ergibt.

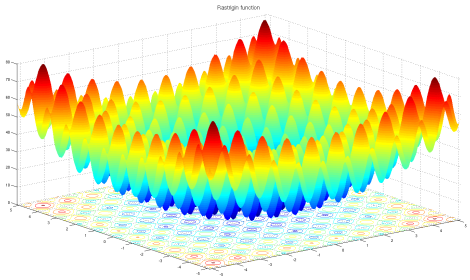
Es gibt verschiedene Klassifizierungen oder Arten von Optimierung:

- Eingeschränkte Optimierung.
  Die Entscheidungsvariable kann nicht irgendwelchen Wert aus der Domäne der Zielfunktion nehmen. Es gibt Einschränkungen dafür.

- Multi-objective Optimierung.
  Das Problem besteht aus vielfältigen, voneinander unabhängigen Zielfunktionen.

- Multi-modale Optimierung.
  Die Zielfunktion(en) hat/haben mehrere Minima bzw. Maxima. Abbildung 1 zeigt eine multi-modale Funktion.

Abbildung 1: *Rastrigin function. Quelle:* `https://commons.wikimedia.org/wiki/File:Rastrigin_function.png`



Eine der wichtigsten und schwierigsten Teile der Optimierung ist es, eine geeignete Zielfunktion zu definieren, die die wichtige zu optimierenden Faktoren berücksichtigt. Probleme im wirklichen Leben sind normalerweise eingeschränkt, multi-Objectiv und multi-modal, oder mit einer großen Anzahl von Dimensionen. Wegen dieser Eigenschaften der Zielfunktionen und der Optimierungsprobleme ist es schwierig, eine gute Lösung mittels traditioneller Vorgänge zu finden. Im Laufe der Zeit haben EA sich als gutes Werkzeug zur Lösung dieser Art von Probleme in angemessener Zeit erwiesen.

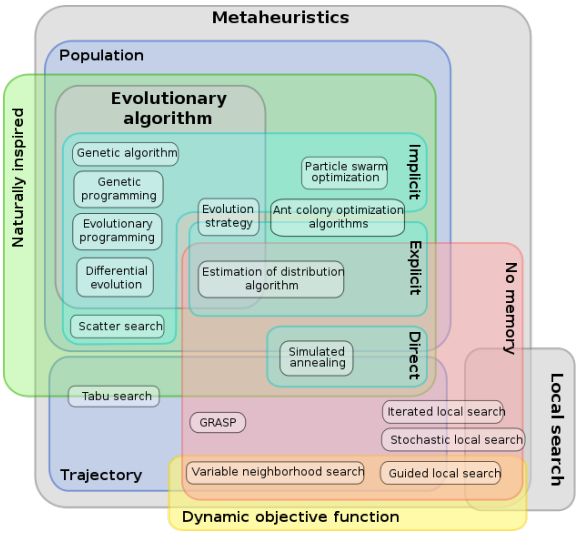
IV. Was ist ein evolutionärer Algorithmus?

Der EA-Bereich ist ziemlich neu, deswegen gibt es bisher keine allgemein akzeptierte Definition von evolutionären Algorithmus.

EA sind zwar als Teil der künstliche Intelligenzes (KI) betrachtet, aber genau wo sie in Verbindung mit anderen KI-Methoden stehen, und was der EA-Bereich beinhaltet, ist von dem Autor abhängig. Abbildung 2 zeigt eine von mehreren möglichen Klassifikationen von KI-

Methoden in Bezug auf Metaheuristics (welche über den Rahmen dieses Papiers hinausgehen).

Abbildung 2: *Klassifikation von Metaheuristics. Quelle:* `https://en.wikipedia.org/wiki/File:Metaheuristics_classification.svg`



In Abbildung 2 kann auch erkannt werden, dass der Autor Particle Swarm Optimization (PSO) und Ant Colony Optimization (ACO) nicht als EA betrachtet. Trotzdem gibt es andere Autoren (wie z. B. [Sim13]), die diese Algorithmen genau für EA halten.

In diesem Paper wird folgende Definition für evolutionäre Algorithmus genommen: Ein Algorithmus, der die Lösung eines Problems durch viele Iterationen entwickelt .

i. Eigenschaften der evolutionären Algorithmen

Folgende sind die Hauptelemente, die ein Algorithmus haben soll, um er als „evolutionär" zu betrachten:

- Eine Zielfunktion.

- Eine - sogenannte - Bevölkerung von mögliche Lösungen.
  Eine Menge von Darstellungen der Lösungen (Elemente der Domäne der Zielfunktion) des Problems, die mittels des Algorithmus und mehreren Iterationen (auch „Generationen" genannt) verarbeitet und hoffentlich verbessert werden.

- Vorgänge, die die Elementen der Bevölkerung betreffen und verändern.

Die meisten EA sind aus der Natur inspiriert, das heißt, dass Wissenschaftler Ereignisse, Systeme, und Mechanismen der Natur beobachten und danach versuchen einigermaßen, sie mit Algorithmen zu reproduzieren. EA werden „evolutionär" genannt, denn sie sind ursprünglich auf die Evolution basiert. [Hol75]

ii. Eigenschaften der Intelligenz

Wie vorher gesagt, EA sind als Teil der KI betrachtet, weil sie Eigenschaften intelligenter natürlicher Systeme emulieren.

Laut [Sim13] sind die folgenden Eigenschaften notwendig, um ein System als intelligent zu betrachten:

- Adaptation.
  Ein intelligentes System muss in hohem Maße an unvorhersehbare Änderungen anpassbar sein. Lernen ist daher unerlässlich.

- Zufälligkeit.
  Obwohl Zufälligkeit meistens als eine schlechte Eigenschaft betrachtet wird, ist es in gewissem Maße notwendig, damit neue Lösungen eines Problems gefunden werden können.

- Kommunikation.
  Zwar wird eine einzige Ameise nicht wirklich als intelligent beurteilt, aber eine Ameisenkolonie wird (dank Kommunikation durch Pheromone) als eine super-intelligente Einheit überlegt.

- Rückmeldung.
  Ein System kann sich nicht anpassen und verbessern, wenn es seine Umgebung nicht erkennen und darauf reagieren kann.

Auch um zu lernen, muss ein System seine Fehler (mittels Rückmeldung) erkennen.

- Erkundung.
  Das heißt, neue Lösungen finden, neue Wege erkunden, neue Ideen erzeugen. Normalerweise ist Erkundung eng mit Zufälligkeit verbindet.

- Ausbeutung.
  Der Gegenteil von Erkundung. Das heißt, schon bekannte Lösungen und Wege (oder Kenntnisse) ausnutzen.

In nächster Sektion werden verschiedene Beispiele von EA gezeigt und in jedem Beispiel wird erwähnt, welches Teil des Algorithmus welche Eigenschaft der Intelligenz versucht zu emulieren. Es ist auch wichtig zu nennen, dass nicht jede Eigenschaft des Intelligenzes von jeden EA repliziert ist.

V. EA Beispiele

Heutzutage gibt es viele EA und jedes Jahr werden neue entwickelt. In dieser Sektion werden nur ein paar der bekanntesten und wichtigsten erwähnt und oberflächlich erklärt.

i. Genetische Algorithmen

Genetische Algorithmen (engl. genetic algorithms oder GA) sind ursprünglich in [Hol75] präsentiert und danach in [Gol98] weiterentwickelt. Der Name „Genetic Algorithms“ bezieht sich mehr auf eine Familie von Algorithmen als auf einen einzigen Algorithmus.

GA sind die erste vorgestellten, bekanntesten, und am meisten verwendeten EA. Ursprünglich waren sie dafür entwickelt, um adaptierbare Systeme zu studieren. Sie sind Simulationen der natürlichen Selektion. Daher sind sie genau auf die Evolution basiert.

Es gibt sechs Hauptteile der GA:

1. Codierung der Lösungen.

   Als erstes muss man eine geeignete Codierung für die möglichen Lösungen finden. Das heißt, eine Darstellung für die Lösungen finden, die einfach zu verarbeiten ist. Eine die am meisten verwendeten Codierungen ist eine Bit-Folge. In dieser Codierung jedes Bit wird „Gen“ genannt.

2. Ursprüngliche Bevölkerung.

   Eine ursprüngliche Bevölkerung muss initialisiert werden. Diese Initialisierung kann entweder zufällig oder auf Vorkenntnis basiert sein. Wie viele Elemente die Bevölkerung hat, ist ein wichtiger Parameter. Je mehr Elemente es gibt desto höher die Wahrscheinlichkeit, um gute Lösungen zu finden.

3. Fitnessfunktion.

   Die Zielfunktion des Problems.

   Der Wert, den ein Element der Bevölkerung darstellt, wird die Eingabe für die Zielfunktion. Der Wert, den die Zielfunktion ergibt, wird die „Fitness“ des Elements genannt.

4. Auswahl von Einzelnen.

   Zwei Elemente der Bevölkerung werden ausgewählt, um ein neues Element herzustellen. Das verfolgte Verfahren, um die Elemente auszuwählen, kann entweder zufällig sein, oder auf die Fitness der Elemente basiert. Die ausgewählten Elemente werden „Eltern“ genannt.

5. Crossover.

   Dies ist der wichtigste Teil der GA. Es definiert wie das neue Element aus die Eltern erstellt wird. Das heißt, welche Gene wird das neue Element von jeden Elternteil erben.

6. Mutation.

   Einige zufälligerweise-ausgewählte neuerstellte Elemente werden mutiert. Das heißt, dass einige Bits gekippt werden. Welche Elemente mutiert werden und wie viele und welche Bits gekippt werden, kann mittels verschiedener Verfahren beziehungsweise entscheidet werden.

Abbildung 3: *Wesentlich genetic Algorithmen pseudo-Code. Quelle: [Sim13]*

```
Parents ← {randomly generated population}
While not(termination criterion)
    Calculate the fitness of each parent in the population
    Children ← ∅
    While |Children| < |Parents|
        Use fitnesses to probabilistically select a pair of parents for mating
        Mate the parents to create children c1 and c2
        Children ← Children ∪ {c1, c2}
    Loop
    Randomly mutate some of the children
    Parents ← Children
Next generation
```

EA insgesamt sind Problem-unabhängig, das heißt, dass wenn es eine geeignete Darstellung der Lösungen und eine geeignete Zielfunktion gibt, kann theoretisch jedes Problem mittels EA optimiert werden. Aus diesem Grund gibt es bei jeden EA ein Element der **Adaptation**.

In 2. Teil sowie in 4. bis 6. Teilen kann **Zufälligkeit** verwendet werden, je nachdem welche Verfahren jeweils ausgewählt werden.

In dem Mutation-Teil, eine hoch Wahrscheinlichkeit, um Elemente zu mutieren, heißt viel **Erkundung** und weniger **Ausbeutung**, und umgekehrt.

ii. Partikelschwarmoptimierung

Partikelschwarmoptimierung (engl particle swarm optimization oder PSO) wird ursprünglich in [JK95] und in [YS98] präsentiert.

PSO ist auf menschlichen sozialen Verhalten basiert. [JK01] Genauer, PSO basiert auf der Beobachtung von Gruppen von Individuen, die zusammenarbeiten, um nicht nur ihr kollektives Verhalten in einer bestimmten Aufgabe zu verbessern, sondern auch ihr individuelles Verhalten zu verbessern.

Die Grundideen hinter PSO sind die folgende:

- Es gibt eine Menge Partikeln, die sich mit einer beliebigen Geschwindigkeit durch den Suchraum bewegen.
- Der Suchraum wird in „Nachbarschaften“ mit einer beliebigen gleichen Größe unterteilt.
- Jeder Partikel speichert sowohl die beste Position, die sie im Suchraum besucht hat, als auch die beste Position, die ihre Nachbarn erreicht haben.
- Beide die Partikels beste Position und ihre Nachbarns beste Position beeinflussen die Partikels Geschwindigkeit.

Abbildung 4: *Wesentlich PSO Algorithmus pseudo-Code. Quelle: [Sim13]*

```
Initialize a random population of individuals {x_i}, i ∈ [1, N]
Initialize each individual's n-element velocity vector v_i, i ∈ [1, N]
Initialize the best-so-far position of each individual: b_i ← x_i, i ∈ [1, N]
Define the neighborhood size σ < N
Define the maximum influence values φ_1,max and φ_2,max
Define the maximum velocity v_max
While not(termination criterion)
    For each individual x_i, i ∈ [1, N]
        H_i ← {σ nearest neighbors of x_i}
        h_i ← arg min_x{f(x) : x ∈ H_i}
        Generate a random vector φ_1 with φ_1(k) ~ U[0, φ_1,max] for k ∈ [1, n]
        Generate a random vector φ_2 with φ_2(k) ~ U[0, φ_2,max] for k ∈ [1, n]
        v_i ← v_i + φ_1 ∘ (b_i − x_i) + φ_2 ∘ (h_i − x_i)
        If |v_i| > v_max then
            v_i ← v_i v_max / |v_i|
        End if
        x_i ← x_i + v_i
        b_i ← arg min{f(x_i), f(b_i)}
    Next individual
Next generation
```

In PSO können die Geschwindigkeiten der ersten Bevölkerung zufällig initialisiert werden. Daher gibt es ein Element der **Zufälligkeit**.

Da die Partikeln die besten bisher erreichten Positionen ihrer Nachbarn kennen muss, gibt es auch **Kommunikation**.

Rückmeldung wird verwendet, indem die besten bisher erreichten Positionen die Geschwindigkeit der Partikel beeinflussen. In welchen Ausmaß wird die Partikel beeinflusst, kann entweder **Erkundung** oder **Ausbeutung** fördern.

iii. Differenziale Evolution

Differenziale Evolution (engl. differential evolution oder DE) wurde erstmals in [Sto96a] und [Sto96b] erwähnt. Aber die erste vielfach gelesene DE-Veröffentlichung war [KP97].

Im Gegensatz zu fast alle anderen EA, DE ist nicht auf der Natur inspiriert. Eine der wichtigsten Merkmale der DE ist, dass es einfach ist. Dies Merkmal ist wichtig, da es Benutzer

aus anderen Bereiche (d. h. nicht Informatikern/inen oder Mathematiker/inen) erlaubt, es umzusetzen.

Die wesentliche Schritte eines DE Algorithmus sind die folgende:

1. Die Bevölkerung eines DE Algorithmus besteht auf Vektoren mit n Elemente. Wo n die Dimension des Problems ist. Weiterhin, jeder Vektor selbst stellt eine mögliche Lösung des Problems dar.
2. Zwei Vektoren $x_{r2}, x_{r3} \mid r2 \neq r3$ werden ausgewählt.
3. Die Differenz zwischen x_{r2} und x_{r3} wird berechnet.
4. Die skalierte Differenz wird zu einem dritten Vektor $x_{r1} \mid r1 \notin \{r2, r3\}$ addiert. Dies wird eine neue mögliche Lösung v_i ergeben.
5. v_i wird mit einem Vektor $x_i \mid i \notin \{r1, r2, r3\}$ kombiniert.
6. x_i wird gegen u_i vergleicht und nur der beste wird behaltet.

Abbildung 5 zeigt 2. bis 4. Schritte für ein Problem mit 2 Dimensionen. Abbildung 6 zeigt einen kompletten DE Algorithmus pseudo-Code für n Dimensionen.

Abbildung 5: *Grundlegende Idee von DE. Quelle: [Sim13]*

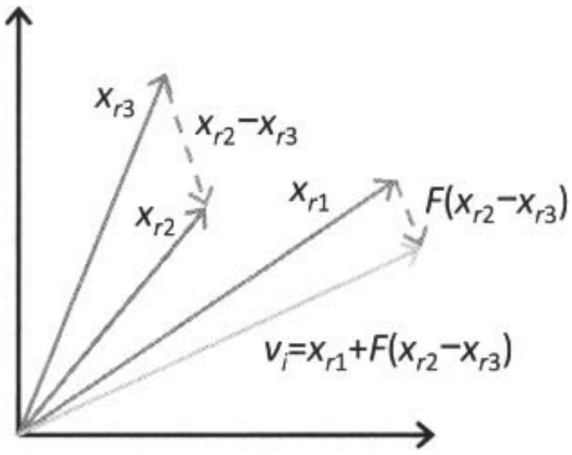


Abbildung 6: *Wesentlich DE Algorithmus pseudo-Code. Quelle: [Sim13]*

F = stepsize parameter $\in [0.4, 0.9]$
c = crossover rate $\in [0.1, 1]$
Initialize a population of candidate solutions $\{x_i\}$ for $i \in [1, N]$
While not(termination criterion)
  For each individual x_i, $i \in [1, N]$
    $r_1 \leftarrow$ random integer $\in [1, N] : r_1 \neq i$
    $r_2 \leftarrow$ random integer $\in [1, N] : r_2 \notin \{i, r_1\}$
    $r_3 \leftarrow$ random integer $\in [1, N] : r_3 \notin \{i, r_1, r_2\}$
    $v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$ (mutant vector)
    $\mathcal{J}_r \leftarrow$ random integer $\in [1, n]$
    For each dimension $j \in [1, n]$
      $r_{cj} \leftarrow$ random number $\in [0, 1]$
      If $(r_{cj} < c)$ or $(j = \mathcal{J}_r)$ then
        $u_{ij} \leftarrow v_{ij}$
      else
        $u_{ij} \leftarrow x_{ij}$
      End if
    Next dimension
  Next individual
  For each population index $i \in [1, N]$
    If $f(u_i) < f(x_i)$ then $x_i \leftarrow u_i$
  Next population index
Next generation

Der Algorithmus, den die Abbildung 6 beschreibt, wird „DE/rand/1/bin“ genannt und es stellt die einfachste Variante der DE Algorithmen dar. Andere Varianten sind zum Beispiel „DE/best/1/L“, „DE/best/2/bin“, „DE/rand/1/exp“.

Jeder Teil dieser Namen bezieht sich auf der Verfahren, die für bestimmte Schritte der DE Algorithmen verwendet werden.

Zum Beispiel der „rand/best“-Teil[2] beschreibt wie Vektoren x_{r1} und x_{r2} im 2. Schritt ausgewählt werden. „rand“ bedeutet, dass die Vektoren zufälligerweise ausgewählt werden, während „best“ heißt, dass die beste Vektoren eine höher Wahrscheinlichkeit haben, um ausgewählt zu werden.

In [EMM06] werden die Leistungen mehrerer Varianten von DE Algorithmen vergleicht.

Wenn Vektoren x_{r1} und x_{r2} zufällig ausgewählt werden, gibt es ein Element der **Zufälligkeit**. Wenn nur die besten Vektoren ausgewählt werden, wird **Ausbeutung** gefördert. Die Tatsache, dass nur die beste Vektor im 6. Schritt behaltet wird, zeigt die Verwendung von **Rückmeldung**. Die Kombination, die im 5. Schritt stattfindet, könnte auch entweder **Erkundung** oder **Ausbeutung** fördern.

[2] „best“ und „rand“ sind nicht die einzige Varianten.

VI. Diskussion

Der größte Vorteil der evolutionären Algorithmen ist, dass es praktisch keine Grenze für ihren Verwendungsbereich gibt. Wie vorher gesagt, wenn es eine geeignete Darstellung der Lösungen eines Problems gibt, und eine geeignete Zielfunktion sich definieren lässt, kann jedes Problem mit EA optimiert werden.

Ein Nachteil der EA ist, dass die Definition einer geeigneten Zielfunktion ein Problem gleich groß als das ursprüngliche erstellen könnte. Dies ist vermeidbar, indem das Problem geteilt wird. In der Praxis werden oft EA verwendet, um mehrere separate Teile großer komplexer Systemen zu optimieren.

Noch ein Nachteil der EA ist, dass die Einstellung des Parametern der Algorithmen auch ein großes Problem darstellen könnte. Die Leistung der Algorithmen ist in großer Maßnahme davon abhängig, wie die ursprüngliche Bevölkerung initialisiert wird, und welche Werte für die Parameter der Algorithmen ausgewählt werden. Dies ist so ein großes Problem geworden, dass dazu Forschung betrieben wird. Um einige Beispiele zu nennen, [GST09], [Ped10], und [MP09] stellen Verfahren zur Parametereinstellung für PSO vor.

EA sollten berücksichtigt werden, immer wenn ein schwieriges Optimierungsproblem gelöst werden soll. Das bedeutet aber nicht, dass EA immer die beste Wahl für den Job sind; Wenn eine Gebäude oder ein komplexes Transportsystem projektieren werden soll, dann sind EA eine geeignete Werkzeug; Wenn eine komplexe elektrische Schaltung oder ein Computerprogramm entwerfen werden soll, dann können die EA die Aufgabe erfüllen; Aber wenn das Problem jedoch mit klassischen Methoden, entweder analytisch oder numerisch, eine gute genug Lösung bietet, ist es besser, diese Methoden zu verwenden.

VII. Fazit

Dank dem Wachstum des Informatikbereichs und der heutigen Rechnerleistung ist es immer mehr möglich, größer und komplexer Systeme zu entwickeln. Dies bringt auch große und komplexe Herausforderungen mit. EA (und KI insgesamt) sind aber gute Werkzeuge, um diese neue Herausforderungen zu überwinden.

Künftig sollen EA in möglichst immer mehr Bereiche eingeführt worden. Ferner sollen wir weiterhin die Natur als Inspiration betrachten. Sowohl in der Wissenschaft, als auch in der Kunst, hat sich die Natur als die beste Muse erwiesen.

Quellenverzeichnis

[AFwn] A. Suleman Artur Fouto, M. A. Gomes. Multidisciplinary optimization strategies using evolutionary algorithms with application to aircraft design. Technical report, American Institute of Aeronautics and Astronautics, unkown.

[AK09] Dharmalingam K. A. Krishnamoorthy. Application of genetic algorithms in the design optimization of three phase induction motor. *Journal of Computer Applications*, II(4):unkown, 2009.

[alg] What are the recently invented evolutionary algorithms (eas)/ optimization method? which method you found en effective method? `https://www.researchgate.net/post/What_are_the_recently_invented_Evolutionary_algorithms_EAs_Optimization_method_Which_method_you_found_en_effective_method`. Accessed: 06.01.2019.

[CAPR00] Moshe Sipper Carlos Andrés Peña-Reyes. Evolutionary computation in medicine: an overview. *Artificial Intelligence in Medicine*, 19(1):1 – 23, 2000. Lausanne, Switzerland.

[DeL12] Manuel DeLanda. The use of genetic algorithms in art. *Conference paper*, pages –, 2012.

[EMM06] Carlos A. Coello Coello Efrén Mezura-Montes, Jesús Velázquez-Reyes. A comparative study of differential evolution variants for global optimization. *GECCO '06 Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 485 – 492, 2006. Seattle, Washington, USA.

[Gol98] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning. Reading*. MA: Addison-Wesley Professional, 1998.

[GST09] Richard E. Haskell Girma S. Tewolde, Darrin M. Hanna. Enhancing performance of pso with automatic parameter tuning technique. *Symposium paper - 2009 IEEE Swarm Intelligence Symposium*, pages –, 2009.

[Hol75] John Henry Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[IAC14] Ali Ahmed Imran Ali Chaudhry. Preliminary aircraft design optimization using genetic algorithms. *Journal of Scientific and Industrial Research*, 73:302 – 307, 2014.

[JK95] Russell C. Eberhart James Kennedy. *Particle Swarm Optimization*, volume IV. Proceedings of IEEE International Conference on Neural Networks, 1995. pp. 1942 - 1948.

[JK01] Russell C. Eberhart James Kennedy. *Swarm intelligence*. Morgan Kaufmann Publishers Inc., 2001.

[KP97] Rainer Storn Kenneth Price. Differential evolution: Numerical optimization made easy. *Dr. Dobb's Journal*, pages 18 – 24, 1997.

[MGCT07] Carlos A. Coello Coello Ma. Guadalupe Castillo Tapia. Applications of multi-objective evolutionary algorithms in economics and finance: A survey. *Conference paper - 2007 IEEE Congress on Evolutionary Computation*, pages –, 2007.

[MP09] Ajith Abraham Millie Pant, Radha Thangaraj. Particle swarm optimization: Performance tuning and empirical analysis. *Foundations of Comput. Intel.*, pages 101 – 128, 2009.

[Mü12] John Müller. *Evolutionary Art And what it means to Art and Science*. VU University Press, 2012.

[Ped10] Magnus Erik Hvass Pedersen. Good parameters for particle swarm optimization. Technical report, Hvass Laboratories, 2010.

[Sim13] Dan Simon. *EVOLUTIONARY OPTIMIZATION ALGORITHMS - Biologically-inspired and Population-based Approaches to Computer Intelligence*. John Wiley and Sons, Inc., 2013.

[Sto96a] Rainer Storn. Differential evolution design of an iir-filter. *IEEE Conference on Evolutionary Computation*, pages 268 – 273, 1996. Nagoya, Japan.

[Sto96b] Rainer Storn. On the usage of differential evolution for function optimization. *Conference of the North American Fuzzy Information Processing Society*, pages 519 – 523, 1996. Berkeley, California.

[wika] Brute-force-methode. `https://de.wikipedia.org/wiki/Brute-Force-Methode`. Accessed: 05.01.2019.

[wikb] Optimierung. `https://de.wiktionary.org/wiki/`

Optimierung. Accessed: 03.01.2019.

[wikc] Problem des handlungsreisenden. https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden. Accessed: 03.01.2019.

[Wonwn] Ka-Chun Wong. *Evolutionary Algorithms: Concepts, Designs, and Applications in Bioinformatics*. Department of Computer Science, University of Toronto, Ontario, Canada, unkown.

[YS98] Russell C. Eberhart Yuhui Shi. *A modified particle swarm optimizer*. Proceedings of IEEE International Conference on Evolutionary Computation, 1998. pp. 69 - 73.