Evolutionäre Optimierungsalgorithmen

Federico Ramírez Villagrana

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

Zusammenfassung

In dieser Arbeit wird darüber diskutiert, sowohl was Optimierung ist und warum es schwierig ist, als auch was evolutionäre Algorithmen (EA) sind und wie, warum, und wann sie hilfreich im Optimierungsbereich sind. Es wird auch detailliert, wie EA funktionieren und warum sie als teil der künstlichen Intelligenz betrachtet werden. Es werden auch einige bekannte EA erwähnt und kurz erläutert erklärt. Am Ende wird darüber diskutiert, welche Vor- und Nachteile EA haben können.

I. Einleitung

Das Problem des Handlungsreisenden (engl. traveling salesman problem oder TSP) ist ein sehr bekanntes und studiertes Thema in den Informatik- und Optimierungsbereich. Es sieht folgendermaßen aus:

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 ist, und die erste Station gleich wie die letzte Station ist [wikc].

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

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 seit Anfang des Universums eine Billion Lösungen pro Sekunde berechnen würde, so wären bisher nur $4,7 \times 10^{41}$ Lösungen berechnet worden.

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

Natürlich kann man nicht sicher sein, dass man die beste Lösung gefunden hat, außer würde man jede mögliche Lösung berechnen. Aber wie mit dem TSP-Beispiel gezeigt worden, kann man nicht immer alle mögliche Lösungen (in angemessener Zeit) berechnen.

II. Stand der Forschung

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

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

stet.

Heutzutage finden sich immer mehr Anwendungen der EA in nahezu allen Bereichen. Von der Medizin [PRS00] und Biologie [Won16] über das Motor- [KD09] und Flugzeugdesign [FGSwn] [CA14] bis hin zur Kunst [DeL12] [Mü12] und Wirtschaft [TC07].

III. Optimierung

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

Bei Optimierungsproblemen 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 genannt.

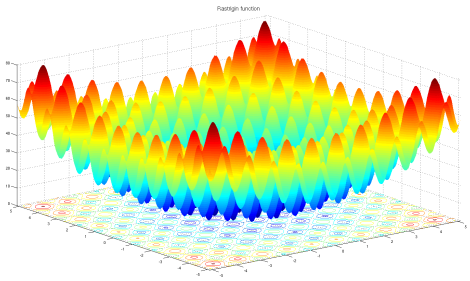
Sei $f(x)$ eine Zielfunktion, und x ein Vektor, der „Entscheidungsvariable“ genannt wird. Die Anzahl an Elementen in x wird die „Dimension“ des Problems genannt. Die Domäne der Zielfunktion repräsentiert die möglichen Lösungen des Problems.

Das Ziel der Optimierung ist es, 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 irgendwelche Werte aus der Domäne der Zielfunktion nehmen. Es gibt Einschränkungen dafür.
- **Multi-objective Optimierung** - Das Problem besteht aus vielfältigen Zielfunktionen.
- **Multi-modale Optimierung** - Die Zielfunktion hat 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 haben eine große Anzahl an 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.

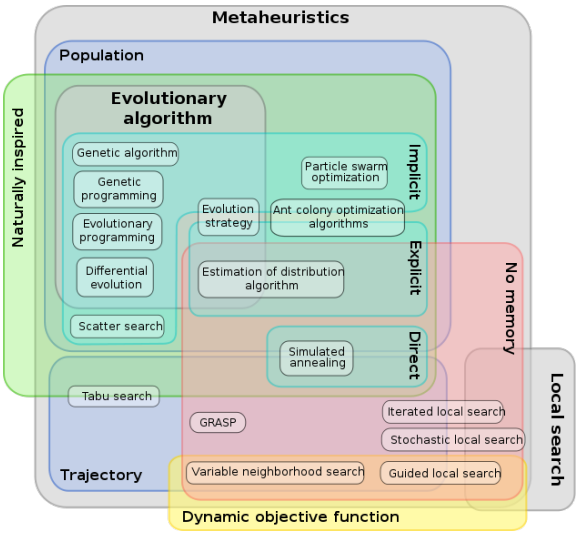
IV. Was ist ein evolutionärer Algorithmus?

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

EA werden zwar als Teil der künstliche Intelligenz (KI) betrachtet, aber genau wo sie in Verbindung mit anderen KI-Methoden stehen, und was der EA-Bereich beinhaltet, ist vom Autor abhängig. Abbildung 2 zeigt eine von mehreren möglichen Klassifikationen von KI-Methoden in Bezug auf Metaheuristics (welche über den Rahmen dieser Arbeit hinausgehen).

In Abbildung 2 kann auch erkannt werden, dass der Autor Particle Swarm Optimization (PSO) und Ant Colony Optimization (ACO)

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



nicht als EA betrachtet. Trotzdem gibt es andere Autoren (wie z. B. [Sim13]), die diese Algorithmen genau für EA halten.

In dieser Arbeit wird folgende Definition für evolutionärer Algorithmus herangezogen: *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, damit es als „evolutionär" betrachten werden kann:

- **Eine Zielfunktion** - Der Wert, den ein Element der Bevölkerung darstellt, wird die Eingabe für die Zielfunktion. Der Wert, den die Zielfunktion ergibt, wird als „Fitness" oder „Kosten" des Elements bezeichnet
- **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 verbessert werden.
- **Vorgänge, die die Elementen der Bevölkerung betreffen und verändern.**

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

ii. Eigenschaften der Intelligenz

Wie bereits erwähnt, EA werden als Teil der KI zu betrachten, weil sie Eigenschaften intelligenter natürlicher Systeme emulieren.

Laut [Sim13, Kapitel 2.7] 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 als intelligent gesehen, aber eine Ameisenkolonie wird (dank Kommunikation durch Pheromone) als eine superintelligente Einheit betrachtet.
- **Rückmeldung** - Ein System kann sich nicht anpassen und verbessern, wenn es seine Umgebung nicht erkennen und darauf reagieren kann. Auch um zu erlernen, 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 verbunden.
- **Ausbeutung** - Das Gegenteil von Erkundung. Das heißt, schon bekannte Lösun-

gen und Wege (oder Kenntnisse) ausnutzen.

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 kurz 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 ersten vorgestellten, bekanntesten und am meisten verwendeten EA. Ursprünglich waren sie dafür entwickelt um adaptierbare Systeme zu studieren. Sie sind Simulationen der natürlichen Selektion. Überdies versuchen sie die Evolution zu simulieren.

Folgende sind die Hauptteile der GA:

1. **Codierung der Lösungen** - Als erstes muss man eine geeignete Codierung für die möglichen Lösungen finden. Das heißt, eine einfach zu verarbeitende Darstellung für die Lösungen zu definieren. Die am häufigsten verwendeten Codierungen sind eine Bitfolge und ein Vektor reeller Zahlen. In diesen Codierungen wird jedes Bit beziehungsweise reelle Zahl „Gen" genannt.

2. **Ursprüngliche Bevölkerung** - Eine ursprüngliche Bevölkerung muss initialisiert werden. Diese Initialisierung kann entweder zufällig oder auf Vorkenntnis basiert werden. Wie viele Elemente die Bevölkerung hat, ist ein wichtiger Parameter. Je mehr Elemente es gibt desto, höher ist die Wahrscheinlichkeit um geeignete Lösungen zu finden, und desto geringer ist der Zeitaufwand.

3. **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 der Fitness der Elemente basieren. Die ausgewählten Elemente werden „Eltern" genannt.

4. **Crossover** - Dies ist der wichtigste Teil der GA. Es definiert wie ein neues Element aus den Eltern erstellt wird. Das heißt, welche Gene die neuen Elemente von jedem Elternteil erben werden.

5. **Mutation** - Einige per Zufall ausgewählte neu erstellte 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 entschieden werden.

Abbildung 3: *Wesentlich genetic Algorithmen pseudo-Code. Quelle: [Sim13, p. 50]*

```
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 sind insgesamt Problem-unabhängig, das bedeutet, 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 allen Teile der GA außer die Erste kann **Zufälligkeit** verwendet werden, je nachdem welche Verfahren jeweils ausgewählt werden. In dem Mutationsteil, eine hohe Wahrscheinlichkeit, um Elemente zu mutieren, heißt viel **Erkundung** und weniger **Ausbeutung**, während eine geringe Wahrscheinlichkeit bedeutet viel **Ausbeutung** und weniger **Erkundung**.

Abbildung 4: *Wesentlich PSO Algorithmus pseudo-Code. Quelle: [Sim13, p. 268]*

```
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
```

ii. Partikelschwarmoptimierung

Partikelschwarmoptimierung (engl. particle swarm optimization oder PSO) wird ursprünglich in [KE95] und in [SE98] präsentiert.

PSO basiert auf menschlichen sozialen Verhalten. [KE01] Genauer gesagt, 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 verfeinern.

Die Grundidee hinter PSO ist die folgende:

- Es gibt eine Menge von Partikeln, die sich mit einer beliebigen Geschwindigkeit durch den Suchraum bewegen.
- Der Suchraum wird in „Nachbarschaften" mit einer beliebigen Größe unterteilt.
- Jede Partikel speichert sowohl die beste Position, die sie im Suchraum besucht hat, als auch die beste Position, die ihre Nachbarn erreichen haben.
- Sowohl die beste Position des Partikels, als auch die der Nachbarn beeinflussen die Geschwindigkeit des Partikels.

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 müssen, gibt es auch **Kommunikation**. **Rückmeldung** wird verwendet, indem die besten bisher erreichten Positionen die Geschwindigkeit der Partikel beeinflussen. In welchem Ausmaß die Partikel beeinflusst wird, 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 [PS97].

Im Gegensatz zu fast allen anderen EA, ist DE nicht nach der Natur inspiriert. Eine der wichtigsten Merkmale der DE ist, dass es einfach ist. Dieses Merkmal ist wichtig, da es Benutzer aus anderen Bereiche (d. h. nicht InformatikernInen oder MathematikerInen) erlaubt, es umzusetzen.

Die wesentliche Schritte eines DE Algorithmus sind die folgende:

1. Die Bevölkerung eines DE Algorithmus besteht aus Vektoren mit n Elemente. Wobei n die Dimension des Problems ist. Weiterhin stellt jeder Vektor selbst 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. Daraus wird sich eine neue mögliche Lösung v_i ergeben.
5. v_i wird mit einem Vektor $x_i \mid i \notin \{r1, r2, r3\}$ kombiniert.
6. Die Fitness von x_i und u_i werden vergleichen und nur den Vektor mit die besten Fitness wird behalten.

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 der DE. Quelle: [Sim13, p. 294]*

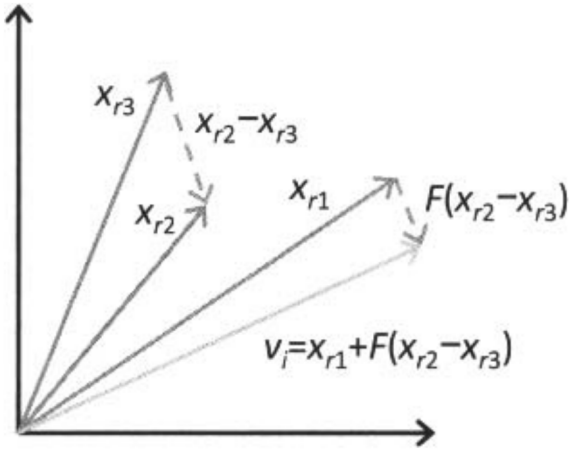


Abbildung 6: *Wesentlich DE Algorithmus pseudo-Code. Quelle: [Sim13, p. 295]*

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 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 die 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 per Zufall ausgewählt werden, während „best“ heißt, dass die beste Vektoren eine höhere Wahrscheinlichkeit haben, um ausgewählt zu werden.

In [MMVRC06] werden die Leistungen mehrerer Varianten von DE Algorithmen verglichen.

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 der beste Vektor im 6. Schritt behalten wird, zeigt die Verwendung von **Rückmeldung**. Die Kombination, die im 5. Schritt stattfindet, könnte auch entweder **Erkundung** oder **Ausbeutung** fördern.

VI. Diskussion

i. Vor- und Nachteile

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

Im Vergleich mit anderen Lösungsmethoden sind EA vorteilhafter, denn sie annehmbare Lösungen in weniger Zeit und mit geringen Gesamtaufwand finden können.

Ein Nachteil der EA ist, dass die Definition einer geeigneten Zielfunktion ein Problem gleicher Größe wie das Ursprüngliche darstellen

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

könnte. Dies ist vermeidbar, indem das Problem geteilt wird. In der Praxis werden oft EA verwendet, um mehrere separate Teile großer komplexer Systeme zu optimieren.

Ein weiteres Nachteil der EA ist, dass die Einstellung der Parametern der Algorithmen auch ein großes Problem darstellen könnte. Die Leistung der Algorithmen ist in großen Anteil davon abhängig, wie die ursprüngliche Bevölkerung initialisiert wird, und welche Werte für die Parametern der Algorithmen ausgewählt werden. Dies ist zu einem so großen Problem geworden, dass dazu Forschung betrieben wird. Um einige Beispiele zu nennen, [THH09] und [PTA09] stellen Verfahren zur Parametereinstellung für PSO vor.

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

ii. Vergleich mit anderen Methoden

Optimierungsprobleme wurden seit langem umfassend untersucht. Aus diesem Grund gibt es viele verschiedene Ansätze und Techniken, um sie zu lösen.

Es gibt zwei Hauptkategorien, in welche die Lösungsverfahren für Optimierungsprobleme eingeordnet werden können.

- **Exakte Algorithmen** - Algorithmen, die genau die beste Lösung eines Problems finden. Diese Algorithmen haben einen große Rechenaufwand, denn sie müssen (implizit) alle mögliche Lösungen berücksichtigten, um die Beste zu identifizieren.
- **Heuristische Algorithmen** - Diese Algorithmen finden annehmbare Lösungen dank der Nutzung von Heuristiken. Normalerweise sind solche Algorithmen weder zeit- noch rechenaufwendig. EA werden in dieser Kategorie eingeordnet.

Die optimale Lösung einer Instanz des Problems des Handlungsreisenden mit 2392 Knoten haben Padberg und Rinaldi [PR90] [PR91] nach zwei Stunden und vierzig Minuten Verarbeitung mit einem exakten Algorithmus in einem leistungsfähigen Vektor-Rechner (IBM 3090/600) gefunden. Interessanterweise, der gleiche Rechner brauchte fünf Stunden, um ein Problem mit nur 532 Knoten zu lösen. Dies zeigt, dass die Größe des Problems nicht der einzige entscheidende Faktor in Bezug auf Zeitaufwand ist.

Die Leistung eines heuristischen Algorithmus kann durch die statistische Verteilung von produzierten Lösungen über ein Spektrum von Problemen definiert werden. Ein Maß dafür ist, zum Beispiel die Frage: Wie oft ergibt der Algorithmus aus einer zufälligen Anfangskonfiguration eine optimale Lösung.

In Bezug auf algorithmische Analyse haben exakte Algorithmen normalerweise eine Komplexität von $O(n^2)$, $O(n^2 \log n)$, $\Theta(n^2)$, oder $\Theta(n^2 \log n)$. Für heuristische Algorithmen ist schwieriger und abhängig von Umsetzung die algorithmische Komplexität zu berechnen.

Aus wirtschaftlicher Sicht ist das Ausführen eines exakten Algorithmus über Stunden auf einem leistungsfähigen Computer möglicherweise nicht kosteneffektiv, wenn eine annehmbare Lösung schnell mit einem kleinen Rechner und einem heuristischen Algorithmus gefunden werden kann. In der Praxis werden heuristische Algorithmen daher oft bevorzugt, um Travelling Salesman Probleme zu lösen.

VII. Fazit

Dank des Wachstums des Informatikbereichs und der heutigen Rechnerleistung ist es immer besser möglich, größere und komplexere Systeme zu entwickeln. Dies bringt auch große und komplexe Herausforderungen mit. EA (und KI insgesamt) sind aber gute Werkzeuge, um diese neuen Herausforderungen zu überwinden.

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

Quellenverzeichnis

[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.

[CA14] Imran Ali Chaudhry and Ali Ahmed, *Preliminary aircraft design optimization using genetic algorithms*, Journal of Scientific and Industrial Research **73** (2014), 302 – 307.

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

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

[Gol98] David Goldberg, *Genetic algorithms in search, optimization and machine learning*, MA: Addison-Wesley Professional, 1998.

[Hol75] John Henry Holland, *Adaptation in natural and artificial systems*, MIT Press, 1975.

[KD09] A. Krishnamoorthy and K. Dharmalingam, *Application of genetic algorithms in the design optimization of three phase induction motor*, Journal of Computer Applications **II** (2009), no. 4, –.

[KE95] James Kennedy and Russell C. Eberhart, *Particle swarm optimization*, Proceedings of IEEE International Conference on Neural Networks **IV** (1995), 1942 – 1948.

[KE01] _____, *Swarm intelligence*, Morgan Kaufmann Publishers Inc., 2001.

[MMVRC06] Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A. Coello Coello, *A comparative study of differential evolution variants for global optimization*, GECCO '06 Proceedings of the 8th annual conference on Genetic and evolutionary computation (2006), 485 – 492.

[Mü12] John Müller, *Evolutionary art and what it means to art and science*, VU University Press, 2012.

[PR90] Manfred Padberg and Giovanni Rinaldi, *Facet identification for the symmetric traveling salesman polytope*, Mathematical Programming **47** (1990), 219 – 257.

[PR91] _____, *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems*, SIAM Review **33** (1991), 60 – 100.

[PRS00] Carlos Andrés Peña-Reyes and Moshe Sipper, *Evolutionary computation in medicine: an overview*, Artificial Intelligence in Medicine **19** (2000), no. 1, 1 – 23.

[PS97] Kenneth Price and Rainer Storn, *Differential evolution: Numerical optimization made easy*, Dr. Dobb's Journal (1997), 18 – 24.

[PTA09] Millie Pant, Radha Thangaraj, and Ajith Abraham, *Particle swarm optimization: Performance tuning and empirical analysis*, Foundations of Comput. Intel. (2009), 101 – 128.

[SE98] Yuhui Shi and Russell C. Eberhart, *A modified particle swarm optimizer*, Proceedings of IEEE International Conference on Evolutionary Computation (1998), 69 – 73.

[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 (1996), 268 – 273, Nagoya, Japan.

[Sto96b] _____, *On the usage of differential evolution for function optimization*, Conference of the North American Fuzzy Information Processing Society (1996), 519 – 523.

[TC07] Maria Guadalupe Castillo Tapia and Carlos A. Coello Coello, *Applications of multi-objective evolutionary algorithms in economics and finance: A survey*, Conference paper - 2007 IEEE Congress on Evolutionary Computation (2007), –.

[THH09] Girma S. Tewolde, Darrin M. Hanna, and Richard E. Haskell, *Enhancing performance of pso with automatic parameter tuning technique*, Symposium paper - 2009 IEEE Swarm Intelligence Symposium (2009), –.

[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.

[Won16] Ka-Chun Wong, *Evolutionary algorithms: Concepts, designs, and applications in bioinformatics*, ch. 6, pp. 111 – 137, Information Resources Management Association, 2016.