Selbstadaptive Fitness in evolutionären Prozessen[1]

Thomas Gabor[2]

Abstract: Evolutionäre Prozesse modellieren die Entwicklung von Objekten mit zunächst zufälligen Eigenschaften zu Objekten, deren Eigenschaften einer Ordnung oder einem bestimmten Ziel (genannt *Fitness*) folgen. Evolutionäre Prozesse treten in Software häufig als Optimierungsalgorithmen oder beim maschinellen Lernen auf, wobei ihr Ziel meist extrinsisch durch einen Designer oder Programmierer bestimmt ist. Oft ist es jedoch von Vorteil, wenn besagte Algorithmen ihre Fitnessberechnung während ihrer Ausführung intrinsisch selbst adaptieren können. Wir verfolgen dieses Phänomen zurück auf künstliche Chemiesysteme (*artificial chemistry systems*), wo Fitness ohne Designer entsteht. Wir untersuchen diversitätsbasierte Fitnessfunktionen in evolutionären Algorithmen und können erstmalig ihre Effektivität begründen, indem wir das theoretische Modell der produktiven Fitness definieren. Schließlich finden wir einen Effektivitätsgewinn auch beim Zusammenspiel von evolutionären Algorithmen und bestärkendem Lernen (*reinforcement learning*), wobei beide Methoden allein durch eine wechselseitig adaptive Fitness interagieren. Dieses Konzept lässt sich auch als Architekturmuster für Softwaresysteme verallgemeinern.

1 Einleitung

Evolutionäres Rechnen (*evolutionary computing*) hat sich von grundlegenden Konzepten der chemischen oder biologischen Evolution inspirieren lassen, um leistungsfähige und interessante Algorithmen zu entwerfen. Im praktischen Einsatz bleibt oft ein frappierender Unterschied zwischen der in Software umgesetzten Evolution und ihrem natürlichen Vorbild: der Ursprung der Fitness. In der biologischen Evolution geht der Begriff der *Fitness* auf Charles Darwin [Da09] selbst zurück und beschreibt ein abstraktes Konzept zur Erklärung *a posteriori*, bspw. warum sich bestimmte Merkmale eher durchsetzen als andere. Im Kontext des evolutionären Rechnens in Computern hat Fitness dagegen meist einen stark imperativen Charakter, denn über eine Definition von Fitness in Software teilt der Programmierer dem Algorithmus *a priori* die zu erreichenden Ziele mit [De17].

Selbst-adaptive Fitness vereint diese beiden Perspektiven: Eine Zielfunktion kann extrinsisch vorgegeben sein, doch sie bzw. ihre Auswirkung auf die Fitness wird aus dem Ablauf der Evolution heraus, also intrinsisch, angepasst. Formal definieren wir zunächst einen evolutionären Prozess, der an einer gegebenen (objektiven) Zielfunktion t gemessen wird. O.B.d.A. nehmen wir an, dass die Werte der Zielfunktion im Bereich $[0; 1]$ liegen und wir deren Maximierung verfolgen.

[1] Englischer Titel der Dissertation: "Self-Adaptive Fitness in Evolutionary Processes"

[2] Ludwig-Maximilians-Universität, Fakultät für Mathematik, Informatik und Statistik, Institut für Informatik, Oettingenstraße 67, 80538 München, Deutschland
thomas.gabor@ifi.lmu.de

Definition 1 (Evolutionärer Prozess) *Sei $\mathcal{X}$ ein beliebiger Suchraum (bspw. $\mathbb{R}^{42}$). Eine potenziell randomisierte Funktion*[3] *$e : \wp(\mathcal{X}) \to \wp(\mathcal{X})$ heiße evolutionäre Schrittfunktion. Eine Funktion $t : \mathcal{X} \to [0;1]$ heiße Zielfunktion. Sei $\langle X_i \rangle_{1 \leq i \leq n}$ eine Serie von sogenannten Populationen, d.h. $X_i \subset \mathcal{X}$ für alle i. Sei $X_0 \subset \mathcal{X}$ eine initiale Population. Ein Tupel $\mathcal{E} = (\mathcal{X}, E, t, \langle X_i \rangle_{i \leq g})$ heißt* evolutionärer Prozess *in g-ter Generation gdw. $X_i = e(X_{i-1})$ für alle $i \leq g$. Ein evolutionärer Prozess entwickelt sich gut, wenn $t(X_j) > t(X_i)$ für viele*[4] *i und jeweils ein $j > i$.*

Um sich gut zu entwickeln benötigt die evolutionäre Schrittfunktion e meist selbst Kenntnis über zumindest das ungefähre Ziel des evolutionären Prozesses. Dies erfolgt durch eine Fitnessfunktion $f : \mathcal{X} \times \wp(\mathcal{X}) \to [0;1]$, die ähnlich wie eine Zielfunktion zu verstehen ist, aber den aktuellen Zustand des evolutionären Prozesses (in unserem Fall üblicherweise die Population der aktuellen Generation) in eine Wertabschätzung miteinbeziehen kann. Natürlich können wir schlicht $f(x, \_) = t(x)$ wählen, doch wir zeigen im Laufe dieses Textes, dass eine Fitness auch ohne eine Zielfunktion entstehen kann (Abschnitt 2), die ideale Fitnessfunktion zum Maximieren von t praktisch und theoretisch nicht exakt t selbst ist (Abschnitt 3) und sich hilfreiche Fitnessfunktion automatisiert generieren lassen (Abschnitt 4). Damit orientieren wir uns an den drei Kernkapiteln 3 – 5 der diesem Text zugrundeliegenden Doktorarbeit [Ga21a].

2 Die Entstehung von Fitness

Wir betrachten zunächst künstliche Chemiesysteme (*artificial chemistry systems*). Diese bestehen aus mehreren Partikeln, die meist mit zufälligen Eigenschaften initialisiert sind, jedoch fixen Interaktionsregeln folgen, durch die für eine Menge an Partikeln ein bestimmtes Verhalten oder eine bestimmte Struktur entstehen kann. Derartige System sind gut untersucht, wenn man bspw. Skalare, Automaten oder λ-Ausdrücke als Partikel annimmt und entsprechende Interaktionen wie respektive mathematische Operationen, Automatenvereinigungen oder λ-Applikation als Interaktionen definiert [BY15; DZB01; FB96]. In den jeweiligen Experimenten ist oft zu beobachten, dass die Menge von Partikeln (in diesem Kontext auch Suppe genannt) bestimmte Zielzustände anzustreben scheint, auch wenn der Versuchsaufbau keine extrinsische Belohnung für bestimmte Zustände vorsieht. Dennoch scheinen bestimmte Interaktionsregeln Partikel dahingehend zu beeinflussen, dass sie stabilere oder sich selbst replizierende Belegungen ihrer Eigenschaften anstreben. Diesem Phänomen wird in der chemischen Ursuppe nicht zuletzt auch die Entstehung von Leben auf dem Planeten Erde zugeschrieben [Da96].

[3] Wir verwenden den Begriff *Funktion* in diesem Text eher im Sinne üblicher (imperativer) Programmiersprachen. Unsere Funktionen können dabei für dieselben Inputs unterschiedliche Outputs liefern. Für eine genauere mathematische Betrachtung verweisen wir auf den Ursprungstext [Ga21a].

[4] Die genaue Definition des Zielkriteriums hängt von der praktischen Umsetzung des Prozesses und seiner Anwendung ab. Für diese generische Definition bleiben wir bewusst unspezifisch und verweisen auf den Ursprungstext [Ga21a].

Diese „natürlichen Zielzustände“ des künstlichen Chemiesystems lassen sich als eine natürlich entstandene, emergente Fitnessfunktion auffassen. Da diese allein aus der Evolution einer Suppe entsteht und stark von den aktuell in der Suppe lebenden Partikeln abhängt, erfüllt eine derartige Fitness auch unsere Definition von Selbst-Adaption. Gleichzeitig können wir durch deren “natürliche” Entstehung zeigen, dass evolutionäre Prozesse eine intrinsische Fitness jenseits ihrer gegebenen Ziele entwickeln können. Wir werden diese Fitness später (siehe Kapitel 3, 4) auch in Ergänzung zu einer extrinischen Fitness beobachten.

Im Rahmen dieser Doktorarbeit wurden künstliche Chemiesysteme entwickelt, deren Partikel neuronale Netze sind [Ga19a]. Ähnlich wie oben erwähnte Automaten oder λ-Ausdrücke können neuronale Netze komplexe Funktionen repräsentieren. Ein neuronales Netz $\mathcal{N}$ führt eine Funktion $\mathcal{N} : \mathbb{R}^p \to \mathbb{R}^q$ aus und ist dabei definiert durch einen Gewichtsvektor $\overline{\mathcal{N}} \in \mathbb{R}^n$ aus $|\overline{\mathcal{N}}| = n$ reellen Zahlen [Kr11]. Da ein neuronales Netz stets mehr Gewichte als Inputs aufweist, d.h. $|\overline{\mathcal{N}}| > p$, lässt sich ein neuronales Netz nicht trivialerweise auf ein anderes neuronales Netz derselben Größe (bzgl. $p, q, n \in \mathbb{N}$) anwenden. Wir untersuchen daher mehrere *Reduktionen*, die es ermöglichen den Informationsgehalt des Netzes zu vermindern oder ein Netz iterativ in Teilen einem anderen Netz als Input weiterzugeben [CL18; Ga19a]. Wir schreiben hier kurz $\mathcal{M}(\overline{\mathcal{N}})$ für die Anwendung von $\mathcal{M}$ auf die Gewichte von $\mathcal{N}$, auch wenn wir dabei insbesondere für den Fall $|\overline{\mathcal{M}}| = |\overline{\mathcal{N}}|$ erwähnte Reduktionen anwenden müssen. Dies erlaubt uns die Einführung folgender zwei (Inter-)Aktionen:

Self-Train. Die Aktion *self-train*$(\mathcal{N})$ mit Hyperparameter $A \in \mathbb{N}$ für die Intensität des Trainings trainiert mittels Backpropagation [Kr11] ein einzelnes neuronales Netz $\mathcal{N}$ darauf, seine eigenen Gewichte wiederzugeben, d.h. $\mathcal{N}(\overline{\mathcal{N}}) = \overline{\mathcal{N}}$ anzunähern. Dabei wird jedoch nur die aktuelle Gewichtskonfiguration von $\mathcal{N}$ zum Training benutzt und es werden nicht bspw. zufällige Vektoren aus dem Inputraum gezogen.

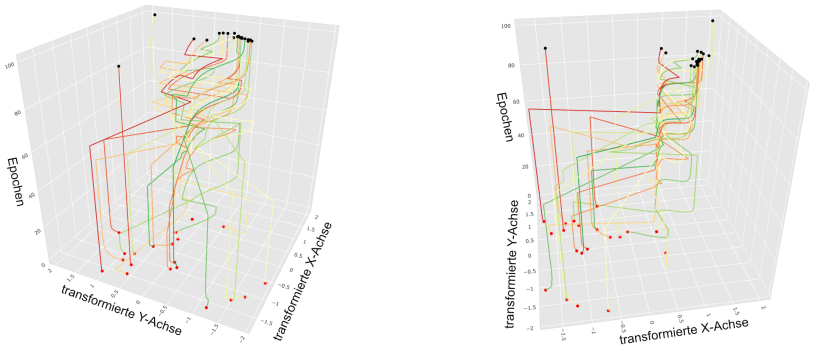


Abb. 1: Zwei Perspektiven auf die Evolution einer Suppe bestehend aus 20 zufällig initialisierten neuronalen Netzen mit jeweils zwei versteckten Schichten mit jeweils zwei Zellen. Die 20 Gewichte pro Netzwerk werden mittels *Principal Component Analysis (PCA)* in zwei Dimensionen X und Y dargestellt. Bild aus [Ga19a].

Attack. Die Interaktion *attack*$(\mathcal{M}, \mathcal{N})$ ersetzt die Gewichte des neuronalen Netzes $\mathcal{M}$ durch das Ergebnis der Anwendung von $\mathcal{N}$ auf diese Gewichte, setzt also $\overline{\mathcal{M}} := \mathcal{N}(\overline{\mathcal{M}})$.

Als ein Beispiel definieren wir nun eine Suppe, in der in jedem Evolutionsschritt (auch *Epoche* genannt) jeder Partikel der Aktion *self-train* mit Intensität $A = 30$ ausgesetzt wird und jeder Partikel mit einer Wahrscheinlichkeit von 0.1 einen zufälligen anderen Partikel für eine Interaktion *attack* zugewiesen bekommt. Abbildung 1 zeigt die entstehende Evolution. Hier können wir beobachten, dass auch ohne jede Zielvorgabe die Partikel zu einem bestimmten Bereich ihres Datenraums tendieren. Der ausgewählte Bereich ist je nach Initialisierung und anderen Zufallsfaktoren unterschiedlich. Meist bringt die Suppe jedoch vornehmlich Partikel hervor, die wir ε-Fixpunkte nennen.

Definition 2 (ε-Fixpoint) *Ein neuronales Netz $\mathcal{N}$ wird ε-Fixpunkt genannt, wenn die Anwendung des Netzes auf seine eigenen Gewichte die ursprünglichen Gewichte mit einem Fehler von höchstens $\varepsilon \in \mathbb{R}$ pro Element des Gewichtsvektors wiedergibt, d.h. $|\overline{\mathcal{N}(\overline{\mathcal{M}})}_i - \overline{\mathcal{N}}_i| < \varepsilon$ für jedes Gewicht mit Index i innerhalb der jeweiligen Netze.*

Ähnlich wie für Automaten oder λ-Ausdrücke bereits entdeckt, entsteht in dieser Art von Suppe eine Tendenz zu Stabilität und Selbst-Replikation, die man als emergente, intrinsische Fitnessfunktion auffassen kann. Mit neuronalen Suppen haben wir dabei ein spannendes Werkzeug geschaffen, um komplexe Algorithmen als Partikel mit (vordergründig) konstanter Platzkomplexität zu kodieren [Ga19a; Ga21b].

3 Die ideale Fitness

Wir folgen nun der Annahme, dass möglicherweise jeder evolutionäre Prozess eine intrinsische Fitness bzw. Fitnesskomponente mitbringt, auch wenn wir ihn extrinsisch in eine möglicherweise gänzlich andere Richtung steuern wollen. Zahlreiche Arbeiten beobachten ein Phänomen, dass diese Annahme unterstützt: Wenn wir eigentlich Individuen für ein bestimmtes Ziel gegeben durch eine Zielfunktion t optimieren wollen, können wir t dem evolutionären Prozess direkt als Fitnessfunktion f übergeben, also $f := t$; oft finden sich jedoch andere Fitnessfunktionen $f' \neq t$, die das Ziel t aber besser optimieren. Dieses Problem wird oft als *fitness design* oder (vor allem im Kontext von verstärkendem Lernen) als *reward engineering* bezeichnet. Wir untersuchen in diesem Abschnitt beispielhaft evolutionäre Algorithmen.

Definition 3 (Evolutionärer Algorithmus) *Sei mut : $\wp(X) \to \wp(X)$ eine Funktion, die aus einzelnen Individuen einer gegebenen Population neue (leicht veränderte) Varianten generiert, rec : $\wp(X) \to \wp(X)$ eine Funktion, die aus mehreren Individuen einer gegeben Population neue Individuen mit kombinierten Eigenschaften erzeugt, mig : $() \to \wp(X)$*

eine Funktion, die eine bestimmte Menge an Individuen zufällig generiert, und $sel : \mathbb{N} \times \wp(X) \to \wp(X)$ *eine Funktion, die eine gegebene Anzahl an Individuen aus einer gegebenen Population (probabilistisch und mit Bevorzugung nach einer Fitness* f*) auswählt. Ein evolutionärer Prozess* $\mathcal{E} = (X, e, t, \langle X_i \rangle_{i \leq g})$ *heißt* evolutionärer Algorithmus *gdw.* e *die Gestalt* $e(X) = sel(|X|, X \cup mut(X) \cup rec(X) \cup mig())$ *hat.*

Ein häufig implementiertes intrinsisches Ziel für evolutionäre Algorithmen ist Diversität. Im einfachsten Fall wird dabei eine Diversitätsfunktion $d : X \times \wp(X) \to \mathbb{R}$ vorausgesetzt, die die Diversität eines Individuums x in der Population X misst. Mit deren Hilfe kann eine neue diversitätsbewusste Fitnessfunktion $f'(x, X) = (1 - \zeta) \cdot t(x) + \zeta \cdot d(x, X)$ definiert werden. Wir sehen sofort, dass diese Fitnessfunktion selbst-adaptiv ist, da ihr Wert für ein gegebenes Individuum x von dem Zustand der Population X abhängt. Wineberg; Oppacher [WO03] zeigen, dass bspw. die durchschnittliche paarweise Manhattan-Distanz hier alle geläufigen Diversitätsfunktionen vertreten kann; wir zeigen, dass sich diese Distanz auch gut durch die paarweise Distanz zu nur einer kleinen zufälligen Untermenge der Gesamtpopulation gut approximieren lässt [Ga18; GB17; GBL18]. Wir entwickeln außerdem die Metrik der genealogischen Diversität, die die Diversität von Individuen anhand eines zufällig generierten Bitstrings ablesen lässt, der für zwei gegebene Individuen probabilistische Rückschlüsse über ihren Verwandtschaftsgrad zulässt [GB17]. Damit erzielen wir im Gegensatz zur Manhattan-Distanz zwar nicht notwendigerweise bessere Ergebnisse, doch wir benötigen keine Distanzfunktion zwischen den Daten der einzelnen Individuen, was praktisch wird, wenn diese keine reellwertigen Vektoren sondern bspw. Programmbäume oder neuronale Netze sind.

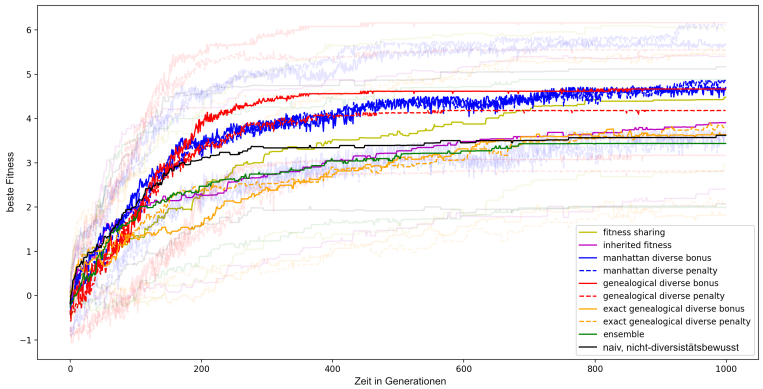


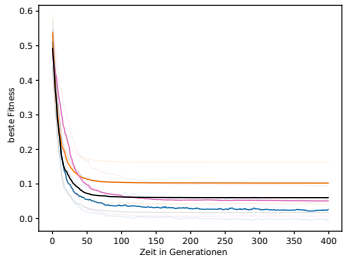
Abb. 2: Evaluation von evolutionären Algorithmen mit verschiedenen (meist diversitätsbasierten) Fitnessfunktionen für ein beispielhaftes Optimierungsproblem („Pathfinding“ [GBL18]). Es wurden jeweils 20 Durchläufe ausgeführt; ausgefüllte Linien zeigen den Durchschnitt, transparente Linien das Band einer Standardabweichung. Bild aus [GBL18].

Abbildung 2 zeigt ein Beispiel für die Optimierung derselben Zielfunktion mit Hilfe verschiedener Fitnessfunktionen. Wir sehen einen deutlichen Vorteil für (einige) diversitätsbasierte Fitnessfunktionen gegenüber der naiven Variante $f(x, \_) = t(x)$ und das eben obwohl diese Fitnessfunktionen das eigentliche Ziel zunächst zu verfälschen scheinen. Um dieses Phänomen erklären zu können, entwickeln wir das Modell der *final produktiven Fitness*, mit dem wir u.A. für bereits abgelaufene Evolutionen sagen können, welche Fitness bestimmte Individuen am besten hätten bekommen sollen.

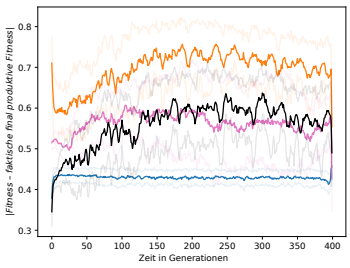
Definition 4 (Faktische final produktive Fitness) *Sei x ein gegebenes Individuum und X_i für $i = 1, ..., n$ eine Population der Generation i, so dass X_0 die zufällige Initialisierung eines evolutionären Algorithmus ist und X_i durch Evolution aus X_{i-1} hervorgeht. X_n ist die finale Population vor Abbruch der Evolution. D_x sei die Menge aller evolutionären Nachkommen des Individuums x. ω sei ein Schlimmstwert für die Zielfunktion t. Dann ist die final produktive Fitness $\phi^\dagger$ gegeben durch*

$$\phi^\dagger(x) = \begin{cases} \text{avg}_{x' \in D_x \cap X_n} \quad t(x') & \text{wenn } D_x \cap X_n \neq \emptyset \\ \omega & \text{sonst.} \end{cases}$$

Die final produktive Fitness misst also den Fitnesswert, den ein Individuum über seine Nachkommen in die letzte Generation des Optimierungsprozesses einbringen kann. Wir argumentieren formal [GL20] und zeigen empirisch [GPL21], dass die final produktive Fitness die ideale Fitnessfunktion für einen evolutionären Algorithmus darstellen könnte. Freilich ist sie ohne hellseherische Fähigkeiten nicht praktisch zur Optimierung einsetzbar, da die Mengen D_x und X_n beide erst nach Abschluss der Evolution zur Verfügung stehen.



(a) Beste Werte der Zielfunktion t.



(b) Durschnittlicher Unterschied zwischen verwendeter Fitness f und faktischer final produktiver Fitness $\phi^\dagger$ pro Individuum.

Abb. 3: Evolutionsläufe für das klassische Problem von Schwefel. Eine naive Evolution mit $f(x, \_) = t(x)$ in Schwarz; diversitätsbasierte Evolution mit $f(x, X) = 0.5 \cdot t(x) + 0.5 \cdot d(x, X)$ in Blau; die Verfahren *inherited fitness* und *fitness sharing* in respektive lila und orange. Es wurden jeweils 20 Durchläufe ausgeführt; ausgefüllte Linien zeigen den Durchschnitt, transparente Linien das Band einer Standardabweichung. Bild aus [GPL21].

Doch wenigstens können wir diese Größe *a posteriori* abschätzen[5]. Wie Abbildung 3 zeigt, ist diejenige Fitnessfunktion für das beste Ergebnis verantwortlich, die über die Evolution hinweg die final produktive Fitness am *stabilsten* approximiert. Wir können deswegen vermuten, dass die final produktive Fitness am besten *a priori* abzuschätzen der ideale Nutzen jeder adaptierten Fitnessfunktion ist [GL20; GPL21].

4 Koevolutionäre Adaption von Fitness

Im vorangehenden Abschnitt konnten wir ein besseres Wissen um intrinsische Ziele zwar für eine bessere Performanz bei der Optimierung nutzen, mussten als Designer der Algorithmen jedoch auch komplexere Fitnessfunktionen definieren. In diesem Abschnitt zeigen wir, dass wir die exakte Fitnessfunktion auch durch eine eigene parallele Evolution steuern können. Wir nennen diesen Aufbau einen *koevolutionären Prozess*. Abbildung 4 zeigt ein Beispiel für so einen Prozess, das wir *Szenarien-Koevolution* (*scenario co-evolution*) nennen. Die grundlegende Optimierung findet in einem Prozess des verstärkenden Lernens statt: Ein virtueller Roboter soll eine virtuelle Fabrik durchlaufen, um bestimmte Ziele aufzusuchen. Auch dieser Prozess kann als Spezialfall eines evolutionären Prozesses betrachtet werden. In der virtuellen Fabrik können an jedem Ort zufällig Hindernisse erscheinen, die den Roboter beim Erreichen seiner Ziele behindern. Eine naive Simulation würde also zufällige Hindernisse mitsimulieren und den Roboter so im Laufe der Zeit auf deren Auftreten und seine richtige Reaktion trainieren. Wir haben gezeigt, dass die Lernergebnisse des Roboters jedoch besser ausfallen, wenn er nicht gegen zufällige sondern gegen möglichst schwierige Hindernisse trainiert [Ga19b]; leichte Konfigurationen von Hindernissen löst er dann ohnehin. Um möglichst schwere Konfigurationen für Hindernisse zu finden, nutzen wir einen evolutionären Algorithmus, der parallel zu dem verstärkenden Lernen läuft. Wir sprechen von einer *kompetitiven Koevolution*, weil Hinderniskonfigurationen danach bewertet werden, wie schlecht sie den aktuell besten Roboter werden lassen, und der Roboter danach bewertet wird, wie gut er mit den aktuell schwierigsten Hindernissen zurecht kommt. Damit sind die Fitness für Roboter und die Fitness für Hinderniskonfigurationen jeweils voneinander abhängig und das System aus beiden weist eine selbst-adaptive Fitness auf.

Obwohl der Mehraufwand an Rechenzeit durch den evolutionären Algorithmus durchaus erheblich ist, ist der bessere Lernfortschritt durch Szenarienkoevolution doch den Aufwand wert. Abbildung 5 zeigt, dass Szenarienkoevolution tendenziell bessere Ergebnisse pro Lernzeit erreicht.

Während wir das Schema eines koevolutionären Prozesses hier auf einen konkreten verstärkenden Lerner und einen evolutionären Algorithmus anwenden, könnte es sich auch

[5] Wir klammern hier eine kleine Diskussion aus: Final produktive Fitness schätzt den Wert eines Individuums über all seine möglichen Nachkommen in allen möglichen Folgeevolutionen ab, was natürlich noch viel mehr Berechnungskomplexität mit sich bringt. Mit der *faktischen* final produktiven Fitness approximieren wir diesen Wert nur noch, in dem wir ihn für nur eine faktisch beobachtete Folgeevolution berechnen und schlicht annehmen, dass diese ausreichend repräsentativ war.

als generelles Architekturmuster zur Entwicklung und dem Test von adaptiven System eignen [Ga20]. Insbesondere ist auch eine Diskussion zu verschiedenen Formen von adaptivem Verhalten und dessen Absicherung (idealerweise koevolutionär durch weitere adaptive Komponenten) Teil der hier beschriebenen Doktorarbeit [Ga16; Ga18; Ga20].

5 Schluss

Wir haben gezeigt, dass selbstadaptive Fitness natürlicherweise entsteht, dass sie hilft eine optimale Fitnessfunktion (für eine gegebene Zielfunktion) anzunähern, und dass wir angepasste Fitnessfunktionen effizient dynamisch adaptieren können, sogar für grundsätzlich verschiedene Formen von Lernen und Evolution. Die zugrundeliegende Doktorarbeit geht noch auf weitere Einsatzmöglichkeiten für (selbst-)adaptive Zielfunktionen ein und bietet ein deutlich solideres formales Fundament für unsere Definitionen.

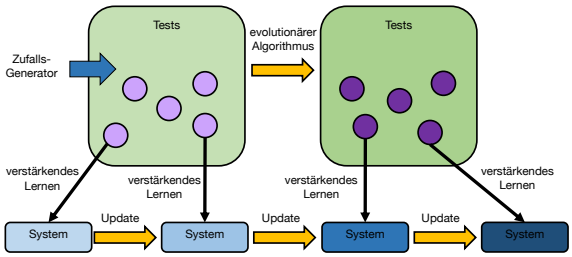


Abb. 4: Schematische Darstellung von Szenarien-Koevolution. Eine Population aus Testszenarien wird zunächst zufällig erzeugt und dann durch einen evolutionären Algorithmus stetig verbessert, während sich ein Agent mit verstärkendem Lernen auf zunehmend schwierigere Testszenarien einstellen kann. Bild aus [Ga19b].

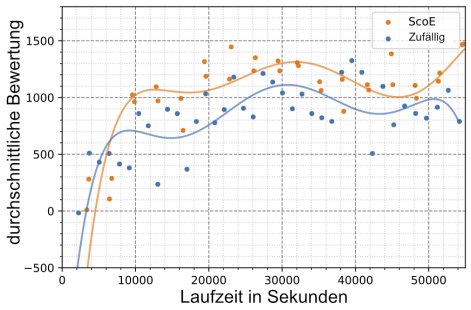


Abb. 5: Bewertungen, die von Szenarienkoevolution und einem naiven verstärkendem Lerner (der gegen zufällige Testzenarien trainiert) in der Fabrik-Hindernis-Testdomäne erreicht wurden, wobei beide gegen zufällige Testzenarien evaluiert wurden. Die Grafik zeigt einzelne Durchläufe und eine Trendkurve. Bild aus [Ga19b].

Aktuelle Arbeiten zielen nun darauf ab, selbst-replizierende Netze für nicht-triviale Lernaufgaben einzusetzen und ihre besonderen Eigenschaften und möglichen Vorteile genau zu analysieren. Eine systematische Betrachtung des Zusammenhangs zwischen extrinischer Zielfunktion und (teilweise) intrinsischer Fitness scheint für viele Anwendungsgebiete lange überfällig und eine Übertragbarkeit der final produktiven Fitness auf andere Techniken des maschinellen Lernens jenseits evolutionärer Algorithmen bleibt zu untersuchen. Koevolutionäre Ansätze finden sich mittlerweile in vielen Algorithmen im Bereich der künstlichen Intelligenz. Ein generelles Framework, das in der Lage ist hierüber viele verschiedene Lernalgorithmen sinnvoll zu kombinieren, bleibt eine Aufgabe für zukünftige Arbeiten.

Literatur

[BY15] Banzhaf, W.; Yamamoto, L.: Artificial Chemistries. MIT Press, 2015.

[CL18] Chang, O.; Lipson, H.: Neural Network Quine. In: ALIFE 2018: The 2018 Conference on Artificial Life. MIT Press, S. 234–241, 2018.

[Da09] Darwin, C.: The Origin of Species. PF Collier & son New York, 1909.

[Da96] Dawkins, R.: The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design. WW Norton & Company, 1996.

[De17] Dennett, D. C.: From Bacteria to Bach and Back: The Evolution of Minds. WW Norton & Company, 2017.

[DZB01] Dittrich, P.; Ziegler, J.; Banzhaf, W.: Artificial Chemistries—A Review. Artificial life 7/3, S. 225–275, 2001.

[FB96] Fontana, W.; Buss, L. W.: The Barrier of Objects: From Dynamical Systems to Bounded Organizations, 1996.

[Ga16] Gabor, T.; Belzner, L.; Kiermeier, M.; Beck, M. T.; Neitz, A.: A Simulation-Based Architecture for Smart Cyber-Physical Systems. In: The International Workshop on Models@run.time for Self-Aware Computing Systems. 2016.

[Ga18] Gabor, T.; Belzner, L.; Phan, T.; Schmid, K.: Preparing for the Unexpected: Diversity Improves Planning Resilience in Evolutionary Algorithms. In: 15th IEEE International Conference on Autonomic Computing (ICAC). 2018.

[Ga19a] Gabor, T.; Illium, S.; Mattausch, A.; Belzner, L.; Linnhoff-Popien, C.: Self-Replication in Neural Networks. In: Artificial Life Conference Proceedings. MIT Press, S. 424–431, 2019.

[Ga19b] Gabor, T.; Sedlmeier, A.; Kiermeier, M.; Phan, T.; Henrich, M.; Pichlmair, M.; Kempter, B.; Klein, C.; Sauer, H.; Schmid, R.; Wieghardt, J.: Scenario Co-Evolution for Reinforcement Learning on a Grid World Smart Factory Domain. In: Proceedings of the Genetic and Evolutionary Computation Conference. S. 898–906, 2019.

[Ga20] Gabor, T.; Sedlmeier, A.; Phan, T.; Ritz, F.; Kiermeier, M.; Belzner, L.; Kempter, B.; Klein, C.; Sauer, H.; Schmid, R.; Zeller, M.; Linnhoff-Popien, C.: The Scenario Coevolution Paradigm: Adaptive Quality Assurance for Adaptive Systems. International Journal on Software Tools for Technology Transfer/, S. 1–20, 2020.

[Ga21a] Gabor, T.: Self-Adaptive Fitness in Evolutionary Processes, Dissertation, Ludwig-Maximilians-Universität München, 2021.

[Ga21b] Gabor, T.; Illium, S.; Zorn, M.; Linnhoff-Popien, C.: Goals for Self-Replicating Neural Networks. In: ALIFE 2021: The 2021 Conference on Artificial Life. MIT Press, 2021.

[GB17] Gabor, T.; Belzner, L.: Genealogical Distance as a Diversity Estimate in Evolutionary Algorithms. In: Measuring and Promoting Diversity in Evolutionary Algorithms (MPDEA@GECCO). ACM, 2017.

[GBL18] Gabor, T.; Belzner, L.; Linnhoff-Popien, C.: Inheritance-Based Diversity Measures for Explicit Convergence Control in Evolutionary Algorithms. In: The Genetic and Evolutionary Computation Conference (GECCO). 2018.

[GL20] Gabor, T.; Linnhoff-Popien, C.: A Formal Model for Reasoning about the Ideal Fitness in Evolutionary Processes. In: International Symposium on Leveraging Applications of Formal Methods (ISoLA). 2020.

[GPL21] Gabor, T.; Phan, T.; Linnhoff-Popien, C.: Productive Fitness in Diversity-Aware Evolutionary Algorithms, 2021.

[Kr11] Kruse, R.; Borgelt, C.; Klawonn, F.; Moewes, C.; Ruß, G.; Steinbrecher, M.; Held, P.: Computational Intelligence. Springer, 2011.

[WO03] Wineberg, M.; Oppacher, F.: The Underlying Similarity of Diversity Measures Used in Evolutionary Computation. In: Genetic and Evolutionary Computation Conference. Springer, S. 1493–1504, 2003.

Thomas Gabor studierte von 2009 bis 2015 Informatik an der Ludwig-Maximilians-Universität München. Von 2015 bis 2021 promovierte er ebenda bei Claudia Linnhoff-Popien und unterstützte den Aufbau der Themenfelder *Artificial Intelligence* und *Quantum Computing* am Lehrstuhl für mobile und verteilte Systeme. Dabei betreute er auch zahlreiche Industrie- und Förderprojekte. Er ist Mitgründer der 2021 am Lehrstuhl entstandenen Ausgründung Aqarios, die Softwarewerkzeuge (u.A. Optimierungsalgorithmen) für das Zeitalter der Quantencomputer entwickelt. Für seine Disseration wurde er 2021 mit dem Heinz-Schwärtzel-Preis ausgezeichnet. Als Postdoc bereitet er für 2022 eine Vorlesung zu *Natural Computing* vor.