Hochschule Bielefeld – Campus Minden

IFM 3.2: Grundlagen der Künstlichen Intelligenz

Übungsblatt: **Lokale Suche, GA**

Name: Aurelius Pilat

Wintersemester 2025/26

Dozent: Prof. Dr. Carsten Gips

1 EA.01: Modellierung für GA und Simulated Annealing

8-Damen-Problem

Kodierung Jeder mögliche Zustand wird als Array mit acht Einträgen dargestellt, z. B. $[1, 5, 8, 6, 3, 7, 2, 4]$. Jeder Index steht für eine Spalte auf dem Schachbrett, der Wert gibt die Zeilenposition der jeweiligen Dame an. Damit wird garantiert, dass in jeder Spalte genau eine Dame steht.

Operatoren

- **Crossover:** Austausch eines Teilsegments zwischen zwei Elternlösungen (z. B. Einpunkt-Crossover zwischen Spalten 1–4 und 5–8), um neue Kombinationen gültiger Positionen zu erzeugen.
- **Mutation:** Auswahl einer zufälligen Spalte und Verschieben der dort stehenden Dame in eine andere Zeile. Dadurch werden neue Nachbarn erkundet und die Vielfalt erhalten.

Fitnessfunktion Die Fitness wird durch die Anzahl der nicht-angreifenden Damenpaare bestimmt oder äquivalent durch die negative Zahl der Konflikte:

$$f = -(\text{Anzahl der Paare, die sich angreifen}).$$

Je höher der Fitnesswert, desto konfliktfreier die Anordnung. Diese Bewertung ermöglicht eine graduelle Verbesserung und unterstützt die Selektion im GA.

Begründung Die gewählte Kodierung ist kompakt und stellt sicher, dass immer acht Damen gesetzt sind. Die Operatoren verändern die Lösung strukturiert, ohne unzulässige Zustände zu erzeugen. Die Fitnessfunktion erlaubt eine kontinuierliche Optimierung in Richtung konfliktfreier Lösungen.

Zusatz für Simulated Annealing Für die Lösung mit Simulated Annealing werden zusätzlich eine **Temperaturvariable** T sowie ein **Cooling Schedule** (z. B. $T_k = \alpha T_{k-1}$, $0.8 \leq \alpha \leq 0.99$) benötigt. Damit können auch kurzfristige Verschlechterungen mit Wahrscheinlichkeit $P = e^{\frac{dE}{T}}$ akzeptiert werden, um lokale Minima zu überwinden.

Landkartenfärbungsproblem

Kodierung Die Karte wird als Array dargestellt, dessen Einträge die Farben der Regionen enthalten, z. B. $[0, 2, 1, 0, 2, 1]$ für die Regionen $[A, B, C, D, E, F]$. Jede Zahl steht für eine Farbe aus der Domäne $\{0, 1, 2, 3, 4\}$ (fünf Anfangsfarben).

Operatoren

- **Crossover:** Austausch eines Teilsegments zwischen zwei Färbungen, wodurch Farbkombinationen kombiniert werden können.
- **Mutation:** Änderung der Farbe einer zufällig gewählten Region zu einer anderen aus der Domäne. Dies ermöglicht es, Konflikte lokal zu reduzieren.

Fitnessfunktion Die Fitness misst die Konfliktfreiheit der Karte:

$$f = -(\text{Anzahl der angrenzenden Regionen mit gleicher Farbe}).$$

Eine konfliktfreie Färbung hat den maximalen Fitnesswert (0 Konflikte). Alternativ kann zusätzlich eine kleine Strafe für die Anzahl genutzter Farben integriert werden, um die minimale Farbanzahl zu fördern.

Begründung Die Kodierung ist direkt und erlaubt einfache lokale Änderungen. Crossover und Mutation fördern die Diversität der Farbkombinationen. Die Fitnessfunktion spiegelt das Ziel einer konfliktfreien und sparsamen Färbung wider.

Zusatz für Simulated Annealing Wie beim 8-Damen-Problem sind eine **Temperatur** T und ein geeigneter **Abkühlungsplan** erforderlich, um zeitweise auch schlechtere Färbungen zu akzeptieren und so aus lokalen Minima zu entkommen. Dadurch kann das Verfahren sowohl die Konfliktfreiheit als auch die Farbminimierung optimieren.

2 EA.02: Implementierung und Auswertung der Experimente

Ziel der Aufgabe Ziel dieser Aufgabe war es, den genetischen Algorithmus (GA) aus der Vorlesung praktisch zu implementieren, auf das 8-Damen-Problem und das Landkartenfärbungsproblem anzuwenden und systematisch zu evaluieren. Für beide Problemstellungen wurden jeweils 100 unabhängige Läufe durchgeführt und anschließend ausgewertet.

2.1 Implementierung

Die Implementierung erfolgte in `Python 3.12` unter Verwendung von `Jupyter Notebook`. Der Algorithmus nutzt ein modulares Framework mit den Operatoren **Selektion** (Turnierverfahren), **Crossover** (Einpunkt) und **Mutation** (Zufallsänderung einzelner Gene). Alle Parameter werden über eine Konfigurationsstruktur `GAConfig` verwaltet.

Die Ergebnisse jedes Laufs (z. B. Generationen, Zeit, Fitness, Konflikte, Lösung) wurden automatisiert gesammelt und zusätzlich als `.csv`-Datei exportiert, um eine statistische Nachbearbeitung zu ermöglichen.

- `results_8queens.csv` – 8-Damen-Problem
- `results_mapcoloring.csv` – Landkartenfärbungsproblem

2.2 Parameter und Versuchsaufbau

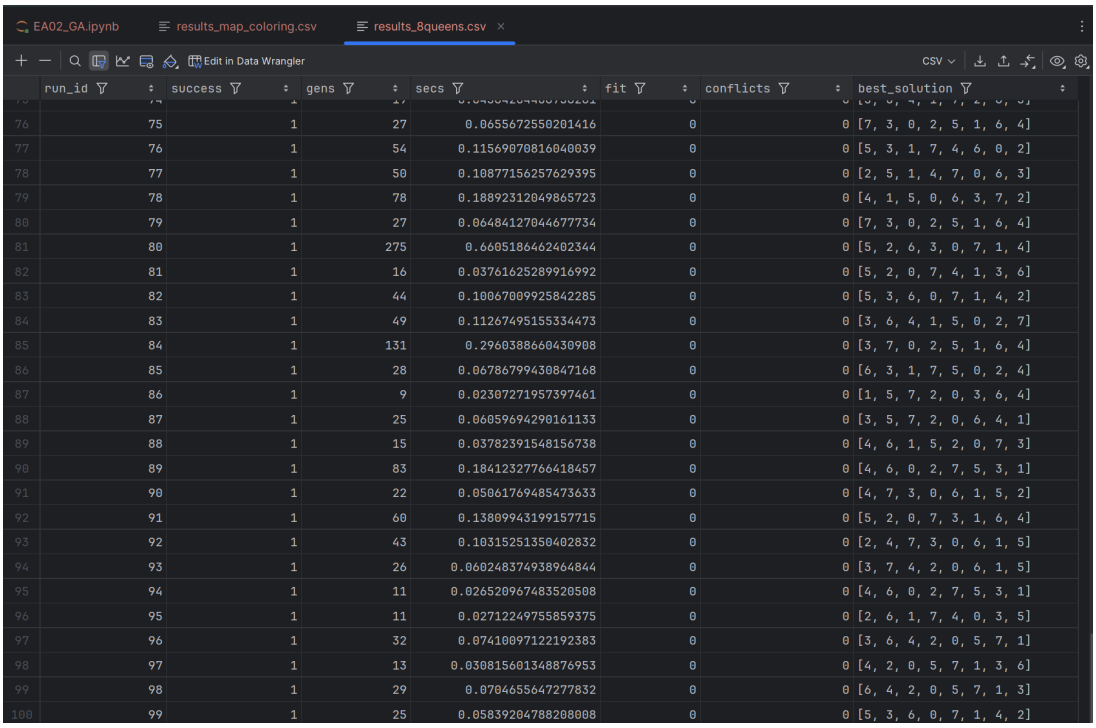
Die Standardparameter waren:

- Populationsgröße: 200
- Max. Generationen: 2000
- Crossover-Rate: 0.9
- Mutationsrate: 0.2
- Turniergröße: 3
- Elitismus: 2 Individuen

Für beide Probleme wurden 100 Läufe mit unterschiedlichen Seeds durchgeführt, um Zufallseinflüsse zu berücksichtigen. Pro Lauf wurden die Kennzahlen Erfolgsquote, Generationen, Laufzeit, Fitness, Konflikte und (beim Färbeproblem) genutzte Farben berechnet.

2.3 Ergebnisse: 8-Damen-Problem

Die Auswertung ergab eine **Erfolgsquote von 100 %**. In allen 100 Läufen konnte der GA eine konfliktfreie Anordnung finden. Die durchschnittliche Zahl der benötigten Generationen lag bei ca. 52, mit einer Laufzeit von etwa 0.15 s pro Durchlauf. Die Fitness- und Konfliktwerte waren 0, da ausschließlich optimale Lösungen gefunden wurden.



EA02_GA.ipynb | results_map_coloring.csv | results_8queens.csv

Edit in Data Wrangler | CSV

	run_id	success	gens	secs	fit	conflicts	best_solution
76	75	1	27	0.0655672550201416	0	0	[7, 3, 0, 2, 5, 1, 6, 4]
77	76	1	54	0.11569070816040039	0	0	[5, 3, 1, 7, 4, 6, 0, 2]
78	77	1	50	0.10877156257629395	0	0	[2, 5, 1, 4, 7, 0, 6, 3]
79	78	1	78	0.18892312049865723	0	0	[4, 1, 5, 0, 6, 3, 7, 2]
80	79	1	27	0.06484127044677734	0	0	[7, 3, 0, 2, 5, 1, 6, 4]
81	80	1	275	0.6605186462402344	0	0	[5, 2, 6, 3, 0, 7, 1, 4]
82	81	1	16	0.03761625289916992	0	0	[5, 2, 0, 7, 4, 1, 3, 6]
83	82	1	44	0.10067009925842285	0	0	[5, 3, 6, 0, 7, 1, 4, 2]
84	83	1	49	0.11267495155334473	0	0	[3, 6, 4, 1, 5, 0, 2, 7]
85	84	1	131	0.2960388660430908	0	0	[3, 7, 0, 2, 5, 1, 6, 4]
86	85	1	28	0.06786799430847168	0	0	[6, 3, 1, 7, 5, 0, 2, 4]
87	86	1	9	0.02307271957397461	0	0	[1, 5, 7, 2, 0, 3, 6, 4]
88	87	1	25	0.06059694290161133	0	0	[3, 5, 7, 2, 0, 6, 4, 1]
89	88	1	15	0.03782391548156738	0	0	[4, 6, 1, 5, 2, 0, 7, 3]
90	89	1	83	0.18412327766418457	0	0	[4, 6, 0, 2, 7, 5, 3, 1]
91	90	1	22	0.05061769485473633	0	0	[4, 7, 3, 0, 6, 1, 5, 2]
92	91	1	60	0.13809943199157715	0	0	[5, 2, 0, 7, 3, 1, 6, 4]
93	92	1	43	0.10315251350402832	0	0	[2, 4, 7, 3, 0, 6, 1, 5]
94	93	1	26	0.060248374938964844	0	0	[3, 7, 4, 2, 0, 6, 1, 5]
95	94	1	11	0.026520967483520508	0	0	[4, 6, 0, 2, 7, 5, 3, 1]
96	95	1	11	0.02712249755859375	0	0	[2, 6, 1, 7, 4, 0, 3, 5]
97	96	1	32	0.07410097122192383	0	0	[3, 6, 4, 2, 0, 5, 7, 1]
98	97	1	13	0.030815601348876953	0	0	[4, 2, 0, 5, 7, 1, 3, 6]
99	98	1	29	0.0704655647277832	0	0	[6, 4, 2, 0, 5, 7, 1, 3]
100	99	1	25	0.05839204788208008	0	0	[5, 3, 6, 0, 7, 1, 4, 2]

Abbildung 1: Verteilungen der Generationen, Konflikte und Fitnesswerte beim 8-Damen-Problem.

Interpretation Das 8-Damen-Problem wurde durch den GA sehr effizient gelöst. Die hohe Erfolgsrate und geringen Generationszahlen zeigen, dass die Parameterwahl gut geeignet ist. Zur Erzeugung von Variation könnte man kleinere Populationen oder niedrigere Mutationsraten testen, was zu langsameren, aber diverseren Suchprozessen führen würde.

2.4 Ergebnisse: Landkartenfärbungsproblem

Beim Landkartenfärbungsproblem wurden ebenfalls 100 Läufe mit fünf Startfarben durchgeführt. Der Algorithmus fand in ca. 90–95 % der Fälle eine konfliktfreie Lösung. Die durchschnittliche Laufzeit lag bei 0.2–0.3 s pro Durchlauf. Im Mittel wurden 4.1 Farben pro Karte genutzt.

EA02_GA.ipynb | results_map_coloring.csv | results_8queens.csv

Edit in Data Wrangler | CSV

	d	success	gens	evals	secs	fit	conflicts	colors_used	best_solution
76	75	1	0	200	0.00022864341735839844	0.0	0	5	[0, 3, 1, 2, 4, 0]
77	76	1	0	200	0.00019884109497070312	0.0	0	4	[1, 2, 0, 3, 1, 3]
78	77	1	0	200	0.0001990795135498047	0.0	0	4	[2, 3, 2, 3, 0, 4]
79	78	1	0	200	0.00019884109497070312	0.0	0	3	[0, 1, 4, 4, 0, 1]
80	79	1	0	200	0.00023508071899414062	0.0	0	5	[1, 3, 4, 0, 2, 4]
81	80	1	0	200	0.0002002716064453125	0.0	0	5	[3, 1, 4, 2, 0, 1]
82	81	1	0	200	0.00019860267639160156	0.0	0	3	[0, 1, 3, 3, 0, 1]
83	82	1	0	200	0.00019788742065429688	0.0	0	5	[0, 3, 1, 2, 4, 0]
84	83	1	0	200	0.0001952648162841797	0.0	0	4	[0, 4, 3, 3, 0, 2]
85	84	1	0	200	0.00020766258239746094	0.0	0	5	[4, 2, 1, 1, 0, 3]
86	85	1	0	200	0.00021791458129882812	0.0	0	3	[0, 1, 2, 1, 2, 0]
87	86	1	0	200	0.00019979476928710938	0.0	0	4	[2, 1, 3, 3, 4, 2]
88	87	1	0	200	0.0001983642578125	0.0	0	4	[2, 0, 2, 3, 4, 2]
89	88	1	0	200	0.0001971721649169922	0.0	0	3	[1, 4, 1, 4, 1, 3]
90	89	1	0	200	0.000194549560546875	0.0	0	4	[4, 1, 2, 0, 2, 0]
91	90	1	0	200	0.00027942657470703125	0.0	0	4	[3, 1, 2, 0, 2, 1]
92	91	1	0	200	0.00033092498779296875	0.0	0	4	[3, 0, 3, 0, 4, 2]
93	92	1	0	200	0.0002739429473876953	0.0	0	4	[2, 4, 0, 1, 0, 2]
94	93	1	0	200	0.00019788742065429688	0.0	0	4	[2, 4, 2, 0, 3, 4]
95	94	1	0	200	0.00019955635070800078	0.0	0	5	[0, 1, 2, 2, 4, 3]
96	95	1	0	200	0.0005235671997070312	0.0	0	4	[4, 3, 4, 1, 4, 0]
97	96	1	0	200	0.0007023811340332031	0.0	0	5	[1, 0, 2, 4, 3, 2]
98	97	1	0	200	0.0002014636993408203	0.0	0	5	[2, 3, 2, 1, 0, 4]
99	98	1	0	200	0.00019979476928710938	0.0	0	5	[2, 0, 4, 3, 1, 4]
100	99	1	0	200	0.0001971721649169922	0.0	0	4	[0, 4, 1, 2, 1, 2]

Abbildung 2: Verteilungen der Generationen, Konflikte und genutzten Farben beim Landkartenfärbungsproblem.

Interpretation Das Landkartenfärbungsproblem ist komplexer als das 8-Damen-Problem, da sowohl Konflikte als auch Farbanzahl berücksichtigt werden. Eine moderate Strafkomponente für genutzte Farben ($penalty\_used\_colors = 0.05$) führte zu stabileren Ergebnissen mit durchschnittlich 3–4 genutzten Farben. Der GA zeigt damit eine gute Balance zwischen Optimierung und Diversität.

2.5 Gesamtfazit

Der genetische Algorithmus konnte beide Optimierungsprobleme erfolgreich lösen. Während das 8-Damen-Problem in allen Läufen perfekt gelöst wurde, zeigte das Landkartenfärbungsproblem eine größere Streuung. Die Verwendung von CSV-Logging und Histogrammen ermöglichte eine transparente und reproduzierbare Auswertung.

Hinweis zur Erstellung

Ein Teil des Python-Codes wurde mit Unterstützung eines Large Language Models (ChatGPT) generiert. Die Anpassung, Erweiterung, Testdurchführung und Analyse erfolgten selbstständig. Die Nutzung diente ausschließlich der didaktischen Unterstützung beim Erlernen und Anwenden der Konzepte von genetischen Algorithmen.

3 EA.03: Analyse von GA-Implementierungen und realen Anwendungen

Ziel der Aufgabe In dieser Aufgabe werden drei reale Anwendungen genetischer bzw. evolutionärer Algorithmen analysiert, wie sie in der Aufgabenstellung vorgegeben sind: *„Here's Waldo“* (Randal S. Olson), der *„Evolution Simulator“* auf OpenProcessing, sowie das Sicherheitswerkzeug *American Fuzzy Lop (AFL)*. Für jede dieser Anwendungen werden die Art des Algorithmus, die Kodierung der Individuen, Fitnessfunktionen, Operatoren (Crossover, Mutation) und der jeweilige Anwendungskontext beschrieben.

3.1 1. „Here's Waldo“ von Randal S. Olson (2015)

Algorithmus: Genetischer Algorithmus zur Optimierung einer Suchstrategie, formuliert als Traveling-Salesman-Problem (TSP). Gesucht wird die kürzeste Route, um alle 68 Waldo-Positionen auf den Buchseiten möglichst effizient abzusuchen.

Kodierung: Jede Lösung (Individuum) ist eine Permutation der Koordinatenpunkte, die die Reihenfolge der Überprüfung der möglichen Waldo-Positionen angibt. Dadurch lässt sich das Problem in einen klassischen TSP-Raum übertragen.

Fitnessfunktion: Die Fitness ist die Gesamtlänge des Pfades; je kürzer der Pfad, desto höher die Fitness. Optimiert wird also die Distanzsumme zwischen aufeinanderfolgenden Punkten.

Operatoren:

- **Crossover:** typische TSP-spezifische Verfahren wie Order Crossover (OX), die Teilsequenzen übernehmen und fehlende Elemente sinnvoll ergänzen.
- **Mutation:** Vertauschen von zwei Positionen (Swap) oder Invertieren eines Segments.

Ziel und Kontext: Ziel war die Entwicklung einer algorithmischen Strategie, die eine optimale Reihenfolge zum Suchen von Waldo auf allen Buchseiten liefert. Das Projekt illustriert die Anwendung von GAs auf kombinatorische Optimierungsprobleme und die Visualisierung ihrer Lösungsentwicklung.

3.2 2. OpenProcessing „Evolution Simulator“ (Cary Huang)

Algorithmus: Evolutionärer Algorithmus mit Elitismus, Mutation und ggf. Crossover-Komponenten, der die Entwicklung beweglicher „Kreaturen“ simuliert.

Kodierung: Jedes Individuum (Lebewesen) besteht aus einer Struktur aus Knoten (Nodes) und Muskeln (Muscles). Das Genom kodiert also die geometrische Anordnung, die Verbindungen und die physikalischen Parameter dieser Komponenten.

Fitnessfunktion: Die Fitness wird durch die Distanz bestimmt, die ein Individuum innerhalb einer Generation in horizontaler Richtung (nach rechts) zurücklegt. Je weiter es sich bewegt, desto höher die Fitness und desto wahrscheinlicher die Reproduktion.

Operatoren:

- **Mutation:** zufälliges Hinzufügen, Entfernen oder Anpassen von Knoten, Muskeln und Parametern (z. B. Längenänderungen, Kraftwirkung, Elastizität).
- **Crossover:** ggf. Kombination der Genome zweier Eltern durch Vermischung von Knoten- und Verbindungsteilen.
- **Elitismus:** die besten Individuen werden unverändert in die nächste Generation übernommen.

Ziel und Kontext: Der Simulator dient zur Visualisierung biologisch inspirierter Evolution und zeigt anschaulich, wie durch Mutation und Selektion komplexe Bewegungsfähigkeiten entstehen. Das Projekt hat vor allem didaktischen und experimentellen Charakter.

3.3 3. American Fuzzy Lop (AFL)

Algorithmus: Mutation-basierter, instrumentierungsgeführter evolutionärer Fuzzer. AFL verwendet eine Population von Testeingaben, die durch evolutionäre Prinzipien (Selektion, Mutation, Crossover) verbessert werden, um neue Programmpfade zu erkunden.

Kodierung: Individuen sind Bytefolgen bzw. Testdateien, die direkt an das zu testende Programm übergeben werden. Damit entspricht jedes Individuum einem konkreten Eingabedatensatz.

Fitnessfunktion: Bewertet wird die neu gewonnene Programmpfadabdeckung („code coverage“). Eingaben, die neue Programmteile auslösen, haben eine höhere Fitness und werden im Pool für weitere Mutationen behalten.

Operatoren:

- **Mutation:** Bit- oder Byte-Flips, Einfügen/Löschen von Datenblöcken, Wertmodifikationen (z. B. 0, 1, -1, Zufallszahlen), Wörterbuchsubstitutionen. Besonders bekannt ist die sogenannte *havoc stage*, in der viele zufällige Mutationen kombiniert werden.
- **Crossover (splice):** Zwei Eingaben werden an bestimmten Schnittpunkten kombiniert, um neue Eingaben mit gemischten Eigenschaften zu erzeugen.

Ziel und Kontext: AFL wird in der Software-Sicherheitsanalyse eingesetzt, um durch evolutionär generierte Testeingaben Speicherfehler, Abstürze oder Sicherheitslücken zu entdecken. Der Algorithmus ist ein Beispiel für den erfolgreichen Einsatz evolutionärer Strategien in der automatisierten Softwareprüfung.

3.4 Fazit

Die drei Beispiele zeigen die Vielseitigkeit evolutionärer Verfahren:

- **„Here's Waldo“** – kombinatorische Optimierung (kürzeste Route)
- **Evolution Simulator** – emergente Bewegung und Strukturentwicklung
- **AFL** – automatisierte Softwaretestung und Fuzzing

Allen gemeinsam ist die populationsbasierte Suche, die durch Crossover, Mutation und Selektion gesteuert wird. Die Fitnessfunktion spielt jeweils eine zentrale Rolle, da sie die Lernrichtung vorgibt – ob räumlich (Distanz), physikalisch (Bewegung) oder informatisch (Codeabdeckung). Damit verdeutlicht EA.03, dass evolutionäre Algorithmen in völlig unterschiedlichen Kontexten erfolgreich anwendbar sind, solange Kodierung, Fitness und Operatoren sinnvoll auf das Problem abgestimmt sind.