Evolutionäre Algorithmen – Endabgabe

Alexander Reiprich, 84818

Kurzbeschreibung

In diesem Projekt geht es um ein Quadrat, welches mit unterschiedlich eingefärbten kleineren Quadraten gefüllt ist. Die einzige Aktion des „Spiels“ ist das Tauschen der Position zweier Quadrate. Ziel ist es, die Farben so anzuordnen, dass ähnliche Farben nebeneinander liegen, und so ein Farbverlauf entsteht. Das Projekt ist inspiriert von der App „I Love Hue“.

Installationsanleitung

Umgesetzt wurde das Projekt in HTML, CSS und TypeScript/JavaScript, weshalb keine Installation nötig ist, um das Projekt zu testen.
Link zum Projekt: https://alexanderreiprich.github.io/EvoAlgo/src/pages/

Möchte man die Anwendung lokal hosten, muss man folgende Schritte befolgen:

1. Das Repository https://github.com/alexanderreiprich/EvoAlgo klonen.
2. Sicherstellen, dass npm installiert ist.
3. Im Ordner des Repositories **npm i** und **npm run build** ausführen.
4. Die **index.html**-Datei aus /src/pages in einem Browser öffnen (Funktionalität getestet mit Chrome v126 und Firefox v128).

Genotyp & Phänotyp und Mutation

Für diese Aufgabe wurden zwei Algorithmen implementiert. Dabei blieben Genotyp und Phänotyp für beide Algorithmen gleich. Der Genotyp besteht aus einem zweidimensionalen String-Array, in welchem die Farbwerte der einzelnen Quadrate gespeichert sind. Position [0][0] im Array entspricht dabei dem Farbwert, welcher im Quadrat oben links zu sehen ist, [0][1] dem Quadrat rechts davon usw. Der Genotyp ist daher nicht sehr abstrahiert vom Phänotyp, was die Arbeit an den Algorithmen vereinfachte.
Ein Individuum wurde mutiert, indem eine zufällige Auswahl an Quadraten miteinander getauscht wurde. Die Anzahl der getauschten Quadrate wurde dabei zufällig zwischen 0 und 5 beim genetischen Algorithmus, bzw. zwischen 0 und 20 beim 1+1-Algorithmus bestimmt.

Fitnessfunktion/Gütebewertung

Die Bewertung der Fitness der einzelnen Individuen wird von der Fitness-Klasse übernommen. Hier werden bei der Kalkulation jedes Quadrat und seine direkt umliegenden Nachbarn betrachtet. Von jedem dieser Quadrate wird der RGB-Wert in R, G und B aufgeteilt, um anschließend jeden der drei Werte mit denen des ursprünglichen Quadrats zu subtrahieren, um so die Farbdistanz zu berechnen. Je kleiner der darauf resultierende Wert ist, desto ähnlicher sind sich die Farben in diesem Wert. Das Ziel ist also eine Minimierung der Fitness.
Man sollte anmerken, dass diese Methode jedoch nicht optimal ist, da der RGB-Farbraum für Bildschirme ausgelegt ist, und die Farbwahrnehmung des Menschen diesen nicht identisch widerspiegelt, weshalb ein (laut dem Computer) akkurates Ergebnis für den Menschen nicht richtig aussehen könnte.

One-Plus-One-Algorithmus

Der erste implementierte Algorithmus ist der 1+1/One-Plus-One-Algorithmus. Dieser beinhaltet ein ähnliches Vorgehen wie der „Hill Climber“. Zuerst wird eine Anfangspopulation durch zufälliges Vertauschen der Originalanordnung der Quadrate erzeugt, welche dann bewertet wird. Darauf wurde das beste Individuum dieser Population vervielfältigt und anschließend mutiert. Ist nun eines dieser Individuen besser als das ursprüngliche Individuum, ersetzt es dieses und wird für die Erzeugung der nächsten Generation genutzt. Ist dies nicht der Fall, wird das vorher genutzte Individuum weiterverwendet.

Dieser Algorithmus ist auf kurze Hinsicht der erfolgreichere der beiden, da er schnell Verbesserungen erreicht, und der Fitnesswert sich bis zu einem gewissen Bereich stetig positiv entwickelt. Der größte Nachteil dieses Algorithmus sind Plateaus, aus denen nur sehr schwer wieder herausgefunden werden kann. Ebenfalls sind grobe Verbesserungen für diesen Algorithmus kein Problem, präzise Feinabstimmungen jedoch umso mehr. Dies führt dazu, dass auch bei sehr langer Laufzeit nur mit enormem Glück Verbesserungen erzielt werden. Ein fertiges, für Menschen vollständiges und fehlerfreies Ergebnis kann nicht erreicht werden.

Jedoch kann man die Verbesserung aktiv beobachten – ähnliche Farben werden nebeneinandergesetzt und es bilden sich kleinere Verläufe einzelner Farben.

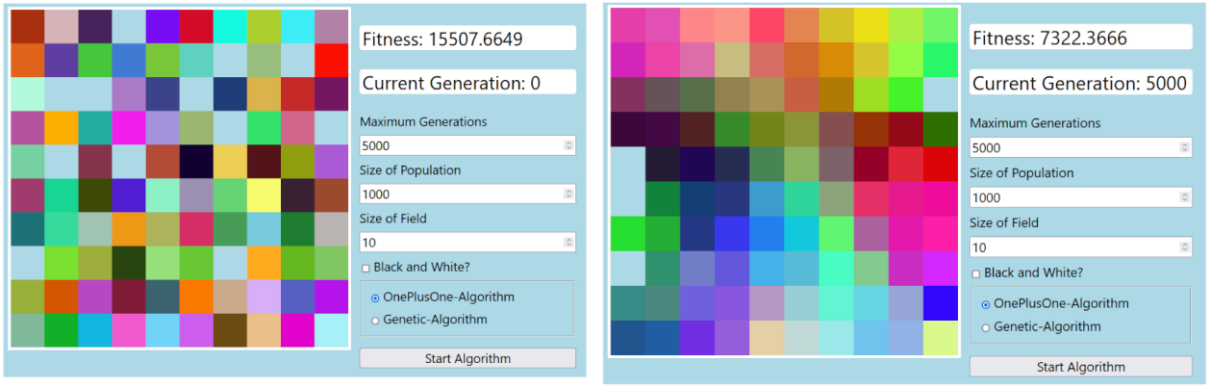

Abbildung 1 - Vorher-Nachher-Vergleich mit dem 1+1-Algorithmus

Aktiviert man die Schwarz-Weiß-Option, bei welcher nur 4 verschiedene Grautöne, anstatt ein volles Farbspektrum genutzt werden, kann man gut erkennen, dass der Algorithmus funktioniert. Hier reichen bereits wenige Generationen, um eine „vollständige“ Lösung zu erhalten.

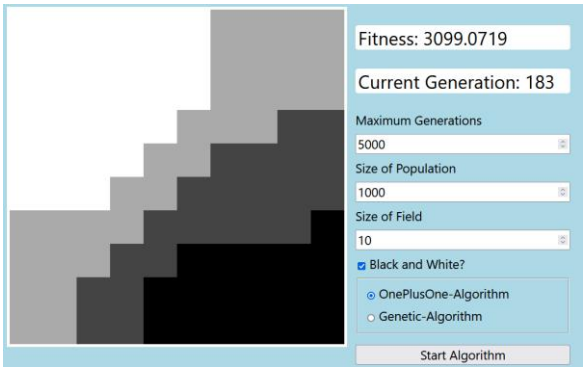

Abbildung 2 - Ergebnis des 1+1-Algorithmus mit Aktivierter Schwarz-Weiß-Option

Genetischer Algorithmus

Der zweite implementierte Algorithmus ist der genetische Algorithmus. Dieser beginnt ähnlich wie der 1+1-Algorithmus, in dem eine Anfangspopulation erzeugt wird, bestehend aus zufälligen Vertauschungen der anfänglichen Anordnung. Anschließend werden alle Individuen bewertet und nach Fitnesswert sortiert. Die zwei besten Individuen werden nun per Crossover kombiniert und das daraus entstandene Individuum anschließend mutiert, um die neuen Individuen der darauffolgenden Generation zu erzeugen. Die zwei besten Individuen dieser so entstandenen Kindgeneration stellen nun die Eltern der nächsten Generation, und der Ablauf wiederholt sich.

Die Art des Crossovers, welche hier gewählt wurde, ist der Order Crossover. Dies schien für diesen Anwendungsfall die passendste Methodik zu sein, da das Risiko, invalide Individuen zu erzeugen, minimiert wird.
Praktisch wurde dieses Crossover wie folgt umgesetzt. Aus dem ersten Elternteil wurde eine zufällige Zeile, und in dieser Zeile eine zufällig lange Abfolge von Quadraten ausgewählt. Diese Quadrate werden nun direkt in das Kindindividuum an die gleiche Position geschrieben. Anschließend werden alle „leeren“ Positionen des Kindindividuums mit den Quadraten des zweiten Elternteils aufgefüllt. Dabei wird festgehalten, welche Quadrate bereits im Kindindividuum vorhanden sind, sodass die Quadrate, welche aus dem ersten Elternteil genommen wurden, nicht durch das zweite Elternteil erneut eingefügt werden. Dies verhindert invalide Individuen, wodurch man auf eine Reparaturfunktion verzichten kann.

Die Nutzung der Schwarz-Weiß-Option in Kombination mit dem genetischen Algorithmus ist nicht möglich. Dies liegt daran, dass die Crossover-Funktion darauf basiert, dass jedes Quadrat einen einzigartigen Wert besitzt, sodass dieser zwischengespeichert werden kann, um doppelte Einträge zu verhindern. Da jede Farbe eine 1 zu 16.777.215 Chance besitzt, gewählt zu werden, ist eine Dopplung bei 100 Quadraten sehr unwahrscheinlich, weshalb dieser Ansatz gewählt wurde, um das Einführen eines speziellen Identifikators zu vermeiden.
Da bei der Schwarz-Weiß-Option jedoch nur 4 verschiedene Farbwerte existieren und es daher viele Dopplungen gibt, funktioniert dieses Tracking nicht, weshalb das Crossover in seiner jetzigen Implementation nicht genutzt werden kann.

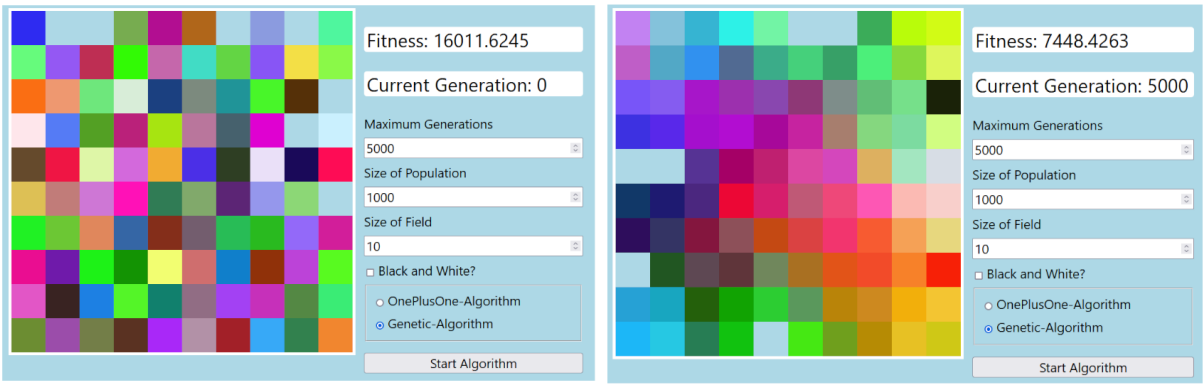

Abbildung 3 – Vorher-Nachher-Vergleich mit dem genetischen Algorithmus

Direkter Vergleich

Während der 1+1-Algorithmus schnelle Verbesserungen erzielt, und teilweise innerhalb der ersten 100 Generationen den Fitnesswert halbieren konnte, ist der genetische Algorithmus nicht nur in der Performance, sondern auch bei den Verbesserungen langsamer. Beide erreichen ähnliche Ergebnisse nach 5000 Generationen, jedoch änderte sich bei dem 1+1-Algorithmus nach knapp 600 Generationen nur wenig, während der genetische Algorithmus sich auch kurz vor dem Ende immer noch anpasste und besser wurde.

Auf lange Laufzeit betrachtet scheint der genetische Algorithmus etwas besser zu sein, wie man an Abbildung 5 erkennen kann, da dort das Ergebnis zusammenhängender scheint. Ebenfalls sollte man erwähnen, dass der 1+1-Algorithmus sich über 10.000 Generationen nicht weiterentwickelt hatte, während der genetische Algorithmus sich über die gesamte Laufzeit verbessert, auch wenn dieser Verbesserungen nur gering waren. Auch der Fitnesswert spricht für den genetischen Algorithmus.

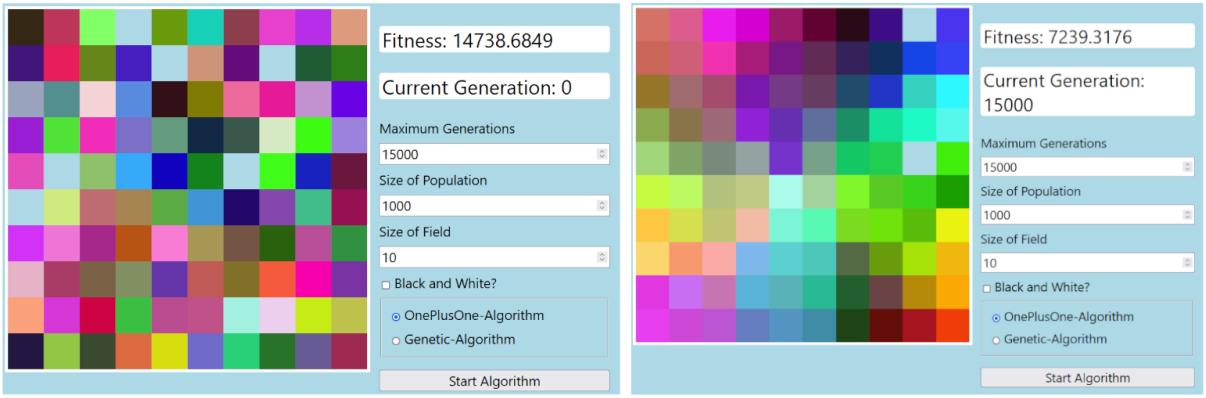

Abbildung 4 - 1+1-Algorithmus nach 15.000 Generationen

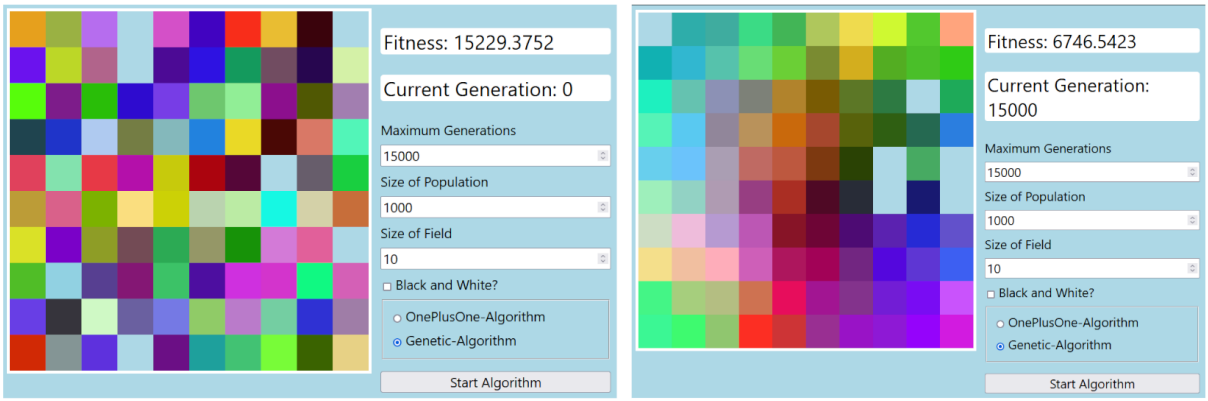

Abbildung 5 - Genetischer Algorithmus nach 15.000 Generationen

Probleme/Herausforderungen

Auch nach längerer Laufzeit ist ersichtlich, dass keiner der ausprobierten Algorithmen das Problem „vollständig“ lösen kann; dafür ist die Aufgabe zu komplex. Ebenso ist man durch den gegebenen Genotypen etwas eingeschränkt in der Auswahl der Verfahren zur Optimierung. Die Implementierung des genetischen Algorithmus ist nicht optimal, da man in der Auswahl der Crossover recht begrenzt ist, wenn man auf eine Reparaturfunktion verzichten möchte. Eine Implementation so einer Funktion wurde im Rahmen des Projektes ausprobiert, hat jedoch zu enormen Performanceproblemen geführt, ohne sichtliche Verbesserungen zu erzielen.
Durch die momentane Umsetzung des Crossover ist es ebenfalls nicht möglich, die Schwarz-Weiß Option in Kombination mit dem genetischen Algorithmus zu nutzen. Dieses Problem wäre vermutlich vermeidbar gewesen, wurde jedoch erst sehr spät im Entwicklungsprozess bemerkt, weshalb eine Umstrukturierung des Projektes für diese Funktion zeitlich nicht sinnvoll gewesen wäre.

Mit einer anderen Herangehensweise, beispielsweise einer anderen Codierung des Genotyps, welche binärer ist, als die hier gewählt, oder durch eine Limitierung der möglichen Farben, ähnlich der Schwarz-Weiß-Option würden sich die Ergebnisse wahrscheinlich weiter verbessern. Trotz dieser Erkenntnisse und der Tatsache, dass keine „vollständige“ Lösung gefunden werden kann, sehe ich das Projekt nicht als Misserfolg, da dennoch Fortschritt verzeichnet werden konnte, sowohl bei der Sortierung der Quadrate als auch bei dem persönlichen Lernerfolg im Themenbereich der evolutionären Algorithmen.