1 Genetischer Algorithmus

Der genetische Algorithmus dient zur iterativen Optimierung der Testfälle. Er erzeugt eine Anzahl Chromosome, die über mehrere Stufen durch diverse Operationen verbessert werden.

1.1 Architektur

1.2 Chromosome

Chromosome dienen der Persistierung der Testfall-Konfigurationen. Sie enthalten die Bestandteile eines realen Testfalls – so wie er für die Simulation benutzt wird – in einer für den Genetischen Algorithmus geeigneten Kodierung. Abbildung 1 bietet einen Überblick über die Bestandteile. Die Information in den Chromosomen sind als prozentualer Anteil

Chromosome
carx
cary
carangle
slotlength
slotdepth

Abbildung 1: Bestandteile eines Chromosoms

eines Wertebereichs kodiert. Die Unterteilung der Wertebereiche erfolgt durch eine 8-Bit Diskretisierung, somit entsprechen 0% dem Wert 0 und 100% dem Wert 255. Tabelle 2 stellt die tatsächliche Verteilung der Werte dar. Die Chromosomen werden in Matlab in

Wert	Minimum	Maximum
Fahrzeug X-Koordinate	-7.5	+7.5
Fahrzeug Y-Koordinate	-1	+4
Fahrzeug Orientierung	0	$2\,\pi$
Slot-Länge	2.25	5
Slot-Breite	1	2

Abbildung 2: Wertebereiche der Chromosom-Bestandteile

einer eigenen Klasse abgelegt. Sie speichert die zugewiesenen prozentualen Anteile an der Gesamt-Range sowie die zugeordnete Fitness. Zudem wird eine Funktion angeboten, die die Berechnung des durch das Chromosom repräsentierten Szenarios – und somit das Mapping auf die spezifizierten Wertebereiche – vornimmt.

1.3 Austauschbare Operatoren

Um eine skalierende Lösung vorzuhalten – insbesondere im Hinblick auf Erweiterungen und Optimierungen – wurden die Teil-Implementierungen des Evolutionsprozesses

austauschbar realisiert. Sie werden als *Funktion-Handle* im Evolutionsframework verankert und können zu Beginn oder nach einer gewissen Epoche durch eine Funktion mit äquivalenter Signatur getauscht werden. So ist es Beispielsweise möglich, Anfangs nach vielversprechenden Bereichen zu suchen, um im späteren Verlauf eine lokale Optimierung durchzuführen.

Operatoren Erzeugung Die Operatoren werden durch das *Factory-Pattern* generiert. Dabei wird eine Fabrik-Funktion verwendet um die Operation gegebenenfalls zu konditionieren. Sie liefert als Resultat die fertige Operation, die in den genetischen Algorithmus eingebunden werden kann.

2 Operatoren

Die Operatoren, die im genetischen Algorithmus angewendet werden, sind modular tauschbar. Sie müssen lediglich die Schnittstellendefinition einhalten.

2.1 Initialisierung

2.2 Mutation

Die Mutation genügt der Signatur

$$\text{mutate: chromosome} \rightarrow \text{chromosome}$$

Durch die generische Auslegung der Mutation im Rahmen des genetischen Algorithmus ist ein Austausch jederzeit möglich. Für den aktuellen Stand wurde ein Algorithmus entwickelt, der das Genom als Integer interpretiert und in dessen Binärdarstellung – als String – umwandelt. Dieser String wird iterativ verarbeitet. Dabei wird jedes Bit mit einer zuvor definierten Wahrscheinlichkeit umgekehrt.

Probleme Die Mutation erfolgt mit einer identischen Wahrscheinlichkeit für hoch- und niederwertige Bits. Dies kann von Vorteil sein, um randomisiert auch große Sprünge zu erlauben. Allerdings können somit auch vielversprechende Kandidaten stark verändert werden. Es kann also eventuell Notwendig sein, Spezialformen der Bit-Mutation zu integrieren. Das modulare System bietet hierfür hervorragende Skalierungsmöglichkeiten.

2.3 Rekombination

2.4 Fitness

Die Fitness dient als Grundlage für den Selektionsprozess. In erster Instanz wurde die Fitnessfunktion als umgekehrt Proportional zum minimalen Abstand realisiert:

$$Fitness = \frac{1}{minDist + maxValue^{-1}}$$

Die Skalierung mit $maxValue^{-1}$ legt den Maximalwert im Kollisionsfall fest und definiert damit die obere Schranke für die Fitnessfunktion. Der minimale Abstand wird aus der Simulation ermittelt.

2.5 Selektion