2.6 Evolutionäre (Genetische) Algorithmen

Das Ziel bzw. die Aufgabe von evolutionären Algorithmen ist eine Optimierung von Objekten mit komplexer Beschreibung, wobei es variable Parameter gibt. Die Objekte kodiert man so als Bitstrings, dass die Aufgabe die Optimierung einer reellwertigen Funktion auf Bitfolgen fester Länge ist.

Gegeben:	Menge von Zuständen
	Bewertungsfunktion
Gesucht:	ein optimaler Zustand

Die Anzahl der möglichen Zustände ist i.a. astronomisch, so dass eine Durchmusterung aussichtslos ist. Da die Evolution Lebewesen „optimiert“, versucht man die Methoden der Evolution und zwar Mutation, Crossover (Rekombination) und Selektion analog auf diese Optimierungen zu übertragen, und mittels Simulationsläufen ein Optimum zu finden.

Idee:

- Zustände werden als Bitfolgen (Chromosomen) codiert
- Die Bewertung entspricht der Fitness der Bitfolge (dieses Chromosomensatzes bzw. Individuums),
- Höhere Fitness bedeutet mehr Nachkommen
- Man beobachtet die Entwicklung einer Menge von Bitfolgen (Population)
- Nachkommen werden erzeugt durch zufällige Prozesse unter Verwendung von Operationen analog zu Mutation und Crossover.

Zu adjustierende Parameter sind:

- Mutationswahrscheinlichkeit: wieviel Bits werden geflippt?, welche Bits?
- Crossover-Wahrscheinlichkeit: an welcher Stelle wird zerschnitten? Welche Individuen dürfen Nachkommen zeugen?
- Abhängigkeit der Anzahl der Nachkommen von der Fitness
- Größe der Population

Wie im Kapitel zur Suche sind auch folgende Aspekte hier wichtig:

- Finden einer geeigneten Repräsentation der Zustände
- Finden geeigneter genetischer Operatoren

Problem: auch bei genetischen Algorithmen kann sich die Population um ein lokales Maximum versammeln. Z.B. durch Inzucht oder durch einen lokalen Bergsteigereffekt, bei dem die Mutation das lokale Optimum nicht verlassen kann.

2.6.1 Genetische Operatoren

Um von einer Population der Generation n zur Population $n+1$ zu kommen, werden verschiedene Operatoren verwendet, die auf der Basis der Population zu n die Population $n+1$ ausrechnen.

Selektion aus der Population zur Generation n wird ein Individuum (Chromosom) ausgewählt. Diese Auswahl kann zum Zwecke der Fortpflanzung oder der Mutation, des Weiterlebens usw. passieren. Diese Auswahl erfolgt für ein Individuum mit hoher Fitness mit höherer Wahrscheinlichkeit, wobei die Wahrscheinlichkeit auch vom Zweck der Selektion abhängen kann.

Rekombination Zwei Chromosomen werden ausgewählt mittels Selektion. Danach wird innerhalb der Chromosomen (zufällig) eine Stelle ermittelt, ab dieser werden die Gene ausgetauscht. Es entstehen zwei neue Chromosomensätze.
Alternativ kann man auch mehrere Abschnitte der Chromosomen austauschen.

Mutation. mit einer gewissen Wahrscheinlichkeit werden zufällig ausgewählte Bits in den Chromosomen verändert. ($0 \rightarrow 1$ bzw. $1 \rightarrow 0$)

Aussterben Wenn die Populationsgröße überschritten wird, stirbt das schlechteste Individuum. Alternativ ein zufällig gewähltes, wobei das schlechteste mit größerer Wahrscheinlichkeit als des beste ausstirbt.

Die Codierung der Chromosomen ist im allgemeinen eine Bitfolge, in diesem Fall spricht man von *genetischen Algorithmen.*
Die Kodierung kann aber auch dem Problem angepasst sein, so dass nicht mehr alle Kodierungen einem Problem entsprechen (ungültige Individuen). In dem Fall ist es meist auch sinnvoll, Operatoren zu verwenden, die nur gültige Nachkommen erzeugen. In diesem Fall spricht man von *evolutionären Algorithmen*

2.6.2 Ermitteln der Selektionswahrscheinlichkeit

Alternativen:

1. Proportional zur Fitness.
   Dies macht die Optimierung sehr sensibel für den genauen Wert und der Steigung der Fitness. Flacht die Fitness ab – z.B. in der Nähe zum Optimum oder auf einem Plateau– dann gibt es keine bevorzugte Auswahl mehr.

2. Proportional zur Reihenfolge innerhalb der Population.
   z.B bestes Individuum 2-3 fach wahrscheinlicher als schlechtestes Individuum. In diesem Fall ist man flexibler in der Auswahl der Fitness-Funktion. Die Auswahl hängt nicht vom genauen Wert ab.

3. Proportional zu einer normierten Fitness (z.B. Fitness $-c$), wobei man die c als das Minimum wählen kann.

Beispiel 2.6.1 *n-Dame mit evolutionären Algorithmen. Die Fitnessfunktion sei bezeichnet als* $\varphi()$.

Codierung: *Die Position der Damen wird pro Zeile als String kodiert. z.B. 12345 entspricht der Platzierung der Damen auf A1, B2, C3, D4, E5. Bei dieser Kodierung kann man einige nutzlose sofort ausschließen: z.B. 11234. Wir wählen die Kodierung so speziell, dass die Kodierung eine Permutation der Zahlen* $1, 2, 3, 4, 5$ *sein muss.*

Fitness: *Anzahl der Bedrohungen (negativ)* $\varphi(12345) = -20$

Operatoren:

Mutation: *Vertauschen von 2 Positionen z.B* $12345 \rightarrow 32145$. *Es gilt dann* $\varphi(32145) = -12$.

Crossover: *Erscheint für diese Aufgabe nicht sinnvoll. Wenn man es verwendet, sollte man verhindern, dass verbotene Lösungen entstehen: Alternativen zur Vermeidung ungültiger Individuen:*

1. *großer negativer Wert der Fitnessfunktion*
2. *sofortiges Aussterben*
3. *deterministische oder zufällige Abänderung des Individuums in ein erlaubtes*

Eine beispielhafte Veränderung einer Population, wobei die Population nur aus 1-2 Elementen besteht:

Mutationen	*32145* $\rightarrow$ *52143; 32145* $\rightarrow$ *34125*
Population	{*52143,34125*}
Bewertung	$\varphi(52143) = -8$, $\varphi(34125) = -8$
Mutationen	$52143 \rightarrow 25143$, $52143 \rightarrow 52413$
Population	{*52143,34125,25143,52413*}
Bewertung	$\varphi(25143) = -4$, $\varphi(52413) = 0$
Selektion	{*25143,52413*}

Die Kodierung 52413 ist eine Lösung.

Reine Mutationsveränderungen sind analog zu Bergsteigen und Best-First.

Bemerkungen zu evolutionären Algorithmen:

Wesentliche Unterschiede zu Standardsuchverfahren

- GAs benutzen Codierungen der Lösungen
- GA-Suche basiert auf einer Menge von Lösungen
- GAs benutzen nur die Zielfunktion zum Optimieren

- GAs benutzen probabilistische Übergangsregeln. Der Suchraum wird durchkämmt mit stochastischen Methoden.

Eine gute Kodierung erfordert:
annähernde Stetigkeit
D.h. es gibt einen annähernd stetigen Zusammenhang zwischen Bitfolge und Fitnessfunktion. D.h. optimale Lösungen sind nicht singuläre Punkte, sondern man kann sie durch Herantasten über gute, suboptimale Lösungen erreichen.
Damit Crossover zur Optimierung wesentlich beiträgt, muss es einen (vorher meist unbekannten) Zusammenhang geben zwischen Teilfolgen in der Kodierung (Gene) und deren Zusammenwirken bei der Fitnessfunktion. Es muss so etwas wie „gute Gene "geben, deren Auswirkung auf die Fitness eines Individuums sich ebenfalls als gut erweist.

Parameter einer Implementierung

Es gibt viele Varianten und Parameter, die man bei genetischen (evolutionären) Algorithmen verändern kann.

- Welche Operatoren werden verwendet? (Mutation, Crossover, ...)
- Welche Wahrscheinlichkeiten werden in den Operatoren verwendet? Wahrscheinlichkeit einer Mutation, usw.
- Welcher Zusammenhang soll zwischen Fitnessfunktion und Selektionswahrscheinlichkeit bestehen?
- Wie groß ist die Population? Je größer, desto breiter ist die Suche angelegt, allerdings dauert es dann auch länger.
- Werden Individuen ersetzt oder dürfen sie weiterleben?
- Gibt es Kopien von Individuen in der Population?
- ...

2.6.3 Statistische Analyse von Genetischen Algorithmen

Vereinfachende Annahme: Fitness = Wahrscheinlichkeit, dass ein Individuum 1 Nachkommen in der nächsten Generation hat.
Es soll die Frage untersucht werden: „wie wächst die Häufigkeit einzelner Gene?"von Generation zu Generation?
Sei S die Menge der Individuen, die ein bestimmtes Gen G enthält.
Die „Schema-theorie" stellt diese mittels eines Schemas dar:
z.B. durch [$*\ *\ *\ *\ 1\ 0\ 1\ 0\ 1\ *\ *\ *\ *$]. Die Folge 1 0 1 0 1 in der Mitte kann man als gemeinsames Gen der Unterpopulation S ansehen. Die ebenfalls vereinfachende Annahme ist, dass das Gen vererbt wird: die Wahrscheinlichkeit der Zerstörung muss gering sein.

Sei $p()$ die Fitnessfunktion (und Wahrscheinlichkeit), wobei man diese auch auf Mengen anwenden darf, und in diesem Fall die Fitness aller Individuen summiert wird. Sei V die Gesamtpopulation.

Der Erwartungswert der Anzahl der Nachkommen aus der Menge S ist, wenn man annimmt, dass $|V|$ mal unabhängig ein Nachkomme ermittelt wird.
$p(s_1) * |V| + \ldots + p(s_{|S|}) * |V| = p(S) * |V|$
Verbreitung tritt ein, wenn $p(S) * |V| > |S|$, d.h. wenn $\frac{p(S)}{|S|} > 1/|V|$ ist. D.h. wenn die mittlere Wahrscheinlichkeit in der menge $|S|$ erhöht ist:

$$\frac{p(S)}{|S|} > 1/|V|$$

Damit kann man $\frac{p(S)}{|S|}$ als Wachstumsfaktor der Menge S ansehen. Wenn dieser Vorteil in der nächsten Generation erhalten bleibt, dann verbreitet sich dieses Gen exponentiell. Es tritt aber eine Sättigung ein, wenn sich das Gen schon ausgebreitet hat. Analog ist das Aussterben von schlechten Genen exponentiell. Eine etwas andere Analyse ergibt sich, wenn man annimmt, dass über die Generationen die Fitness a der S-Individuen konstant ist, ebenso wie die Fitness b der Individuen in $V \setminus S$. Dann erhält man die Nachkommenswahrscheinlichkeit durch Normierung der Fitness auf 1. Wir nehmen an, dass S_t die Menge der Individuen zum Gen S ist.
Das ergibt

$$p_S := \frac{a}{a * |S_t| + b * (|V| - |S_t|)}$$

Damit ist der Erwartungswert für $|S_{t+1}|$:

$$|S_{t+1}| = \frac{a * |S_t| * |V|}{a * |S_t| + b * (|V| - |S_t|)}$$

Wenn wir $r(S,t)$ für den relativen Anteil von S schreiben, dann erhalten wir:

$$|r(S,t+1)| = \frac{a}{a * r(S,t) + b * (1 - r(S,t))} * r(S,t).$$

2.6.4 Anwendungen

Man kann evolutionäre Algorithmen verwenden für Optimierungen von komplexen Problemstellungen, die

- keinen einfachen Lösungs- oder Optimierungsalgorithmus haben,
- bei denen man relativ leicht sagen kann, wie gut diese Lösungen sind
- die nicht in Echt-Zeit lösbar sein müssen, d.h. man hat ausreichend (Rechen-)Zeit um zu optimieren.

- bei denen man aus einer (bzw. zwei) (Fast-)Lösungen neue Lösungen generieren kann.
- wenn man die Problemstellung leicht mit weiteren Parametern versehen will, und trotzdem noch optimieren können will.

Beispiele:

Handlungsreisenden-Problem.
Es gibt verschiedene Kodierung, und viele Operatoren, die untersucht wurden. Die einfachste ist eine Kodierung als Folge der besuchten Städte. Die Fitnessfunktion ist die Weglänge (evtl. negativ).
Mutationen, die gültige Individuen (Wege) erzeugen soll, braucht Reparaturmechanismen. Man kann einen algorithmischen Schritt dazwischen schalten, der **lokale Optimierungen** erlaubt, z.B. kann man alle benachbarten Touren prüfen, die sich von der zu optimierenden Tour nur um 2 Kanten unterscheiden.
Ein vorgeschlagener Operator ist z.B. das partielle Invertieren: Man wählt zwei Schnittpunkte in einer gegebenen Tour aus, und invertiert eine der Teiltouren.

Ähnliches Bemerkungen wie für Mutation gelten für **Crossover**, das eine neue Tour aus Teiltouren anderer Touren der Population zusammensetzt.

Operations Research-Probleme: Schedulingprobleme, Tourenplanung.

SAT (Erfüllbarkeit von aussagenlogischen Formeln)
Bei SAT kann man die Interpretation als Bitstring kodieren: Man nimmt eine feste Reihenfolge der Aussagenvariablen und schreibt im Bitstring die Belegung mit 0/1 hin. Mutation und Crossover erzeugen immer gültige Individuen. Als Fitnessfunktion kann man die Anzahl der wahren Klauseln nehmen.

2.6.5 Transportproblem als Beispiel

Wir betrachten als Beispiel ein einfaches Transportproblem mit 3 Lagern L_1, L_2, L_3 von Waren und 4 Zielpunkten $Z_1, \ldots, Z_4$. [1]. Man kennt die Kosten für den Transport von Waren von L_i nach Z_i pro Einheit.
Folgende Daten sind gegeben:

Lagerbestand			Warenwünsche			
L_1	L_2	L_3	Z_1	Z_2	Z_3	Z_4
15	25	5	5	15	15	10

Die Wünsche sind erfüllbar.
Die Kosten pro Einheit sind:

[1] siehe Michalewicz (1992)

10	0	20	11
12	7	9	20
0	14	16	18

Eine optimale Lösung ist:

	5	15	15	10
15	0	5	0	10
25	0	10	15	0
5	5	0	0	0

Die Kosten sind:
$5 * 0 + 10 * 11 + 10 * 7 + 15 * 9 + 5 * 0 = 315$
Will man das Problem mittels evolutionären Algorithmen lösen, so kann man verschiedene Kodierungen wählen.

1. Nehme die Lösungsmatrix als Kodierung.
2. Nehme eine Matrix der gleichen Dimension wie die Lösungsmatrix, Prioritäten 1,2,3,... als Einträge.

Als Fitnessfunktion wählt man die Kosten (bzw. negativen Kosten: Ersparnisse).

1. hat den Vorteil, dass man eine exakte Kodierung hat. Allerdings weiss man bei diesem Problem, dass man die billigste Lösung zuerst nehmen kann, so dass man damit evtl. Lösungen variiert, die man eigentlich als äquivalent ansehen kann. Als Bedingung hat man noch, dass es keine negativen Werte geben soll und dass die Spaltensummen den Wünschen entsprechen sollen, die Zeilensummen sollen nicht größer sein als der jeweilige Lagerbestand.
2. hat den Vorteil, dass man das Wissen über das Problem nutzt. Aus den Prioritäten wird nach folgendem festgelegten Schema eine Lösung berechnet: zunächst wird die Fuhre mit der Priorität 1 so voll wie möglich gemacht. Danach unter Beachtung, dass schon geliefert wurde, die Fuhre mit Priorität 2. usw. Hierbei verliert man keine optimalen Lösungen. Zudem kann man leichter neue Kodierungen erzeugen.

   Eine Kodierung entspricht einer Permutation der 12 Matrixeinträge. Man kann sie als FOlge von 12 Zahlen auffassen.

Mutation in 1): verschiebe eine Wareneinheit auf eine andere Fuhre. Hierbei kann aber eine ungültige Lösung entstehen. Crossover: unklar.

Mutation in 2): permutiere die Folge durch Austausch von 2 Prioritäten. Hierbei entsteht eine gültige Kodierung.
Inversion: invertiere die Reihenfolge der Prioritäten:
Crossover: Wähle zwei Permutation p_1, p_2 aus der Population aus, und wähle aus p_1 ein Unterfolge aus. Entferne aus p_2 die Prioritäten, die in p_2 vorkommen, ergibt Folge p_2'. Dann setze die Prioritäten an einer neu gewählten Stelle in p_2' wieder ein.

2.6.6 Schlußbemerkungen

Bemerkung 2.6.2 *Es gibt mittlerweile einige generische Programme, die die Berechnung der Generationen usw. bereits ausführen, und denen man nur noch die Kodierung der Problemstellung geben muss, die Kodierung der Operatoren und die Parameter.*

Bemerkung 2.6.3 *Für komplexe Probleme und deren Optimierung gilt die Heuristik: mit einem Evolutionären Algorithmus kann man immer ganz brauchbar verbessern, wenn man wenig über die Struktur und Eigenschaft der Problemklasse sagen kann.*
Aber sobald man sich mit de Problemklasse genauer beschäftigt und experimentiert, wird ein Optimierungsalgorithmus informierter sein, und sich mehr und mehr zu einem eigenständigen Verfahren entwickeln, das bessere Ergebnisse produziert als allgemeine Ansätze mit evolutionären Algorithmen