INF FAKULTÄT FÜR INFORMATIK

Evolutionäre Algorithmen

Einführung

Prof. Dr. Rudolf Kruse **Christian Moewes**

{kruse,cmoewes}@iws.cs.uni-magdeburg.de

Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Institut für Wissens- und Sprachverarbeitung

Übersicht

Zu meiner Person: Rudolf Kruse

- 1979 Dipl. Mathematik (Nebenfach Informatik) von TU Braunschweig
- dort 1980 promoviert, 1984 habilitiert
- 2 Jahre hauptamtlicher Mitarbeiter bei Fraunhofer
- 1986 Ruf als Professor für Informatik der TU Braunschweig
- seit 1996 Professor an der Universität Magdeburg
- **Forschung:** Data Mining, Explorative Datenanalyse, Fuzzy-Systeme, Neuronale Netze, EA, Bayes'sche Netze
- `mailto:kruse@iws.cs.uni-magdeburg.de`
- Büro: G29-008, Telefon: 0391 67-18706
- Sprechstunde: Mi., 11:00–12:00 Uhr

Zur Arbeitsgruppe: Computational Intelligence

Lehre:

- Intelligente Systeme — Bachelor (2 V + 2 Ü, 5 CP)
- Evolutionäre Algorithmen — Bachelor (2 V + 2 Ü, 5 CP)
- Neuronale Netze — Bachelor (2 V + 2 Ü, 5 CP)
- Fuzzy-Systeme — Master (2 V + 2 Ü, 6 CP)
- Bayes'sche Netze — Master (2 V + 2 Ü, 6 CP)
- Intelligente Datenanalyse — Master (2 V + 2 Ü, 6 CP)
- (Pro-)Seminare: Information Mining, Computational Intelligence

Forschungsbeispiele:

- Entdeckung & Visualisierung interessanter Muster (M. Steinbrecher)
- Data Mining in der Landwirtschaft (G. Ruß)
- Temporales Data Mining in Medizin & SE (C. Moewes)

Zur Vorlesung

- Vorlesungstermine: Mo., 13:15–14:45 Uhr, G29-307
- Vorlesungsausfälle: 25.04.2011 (Ostern), 13.06.2011 (Pfingsten)
- Vorlesungsende: 04.07.2011
- Informationen zur Vorlesung:
  `http://fuzzy.cs.ovgu.de/wiki/pmwiki.php?n=Lehre.EA2011`
  - wöchentliche Vorlesungsfolien als PDF
  - Übungsblätter ebenfalls
  - wichtige Ankündigungen und Termine!

Inhalte und Lernziele der Vorlesung

- Einführung in biologische Grundlagen der Evolution und Genetik
- Ausgestaltung genetischer Operatoren (z.B. Selektion, Kreuzung, Rekombination, Mutation)
- Überblick verschiedener Arten evolutionärer Algorithmen und genetischer Programmierung
- Erläuterung von Vor- und Nachteilen dieser Algorithmen anhand von Beispielen
- Behandlung verwandter Verfahren (z.B. simuliertes Ausglühen)
- Anwendungsbeispiele

Zur Übung

Zielsetzung:

- Anwendung von adäquaten Modellierungstechniken zum Entwurf von evolutionären Algorithmen
- Anwendung der Methoden der numerischen Optimierung zur Problemlösung
- Bewertung und Anwendung evolutionärer Programmierung zur Analyse komplexer Systeme
- Befähigung zur Entwicklung von evolutionären Algorithmen

Ihre Aufgabe:

- Nacharbeiten des Vorlesungsstoffs
- Bearbeitung der Übungsaufgaben
- aktive Teilnahme an den Übungen
- alternativ: Teilnahme am Programmierwettbewerb

Durchführung der Übungen

- Sie werden aktiv und erklären Ihre Lösungen!
- Tutor macht auf Fehler aufmerksam und beantwortet Fragen
- das „Vorrechnen“ der Aufgaben ist nicht Sinn der Übung
- ganz bewusst: keine ausgearbeiteten Musterlösungen
- Tutor: Christian Moewes `mailto:cmoewes@ovgu.de`
- Raum G29-019, Sprechstunde: immer, wenn Bürotür offen steht

Übung: 2 Termine zur Auswahl

- Mi., 11:15–12:45 Uhr in Raum G29-E037
- Mi., 13:15–14:45 Uhr in Raum G22A-218
- Anmeldung:
  `https://iws.cs.uni-magdeburg.de:8443/frs/subscribe`
- erste Übung am 06.04.2011 entfällt!
- stattdessen: Experiment zur Programmiererfahrung (Janet Feigenspan)
  - `http://wwwiti.cs.uni-magdeburg.de/ feigensp/peEA/`
  - zu den jeweiligen Zeiten am 06.04.2011 in Raum G29-114
  - bei nachgewiesener Teilnahme: ein zusätzlicher Votierungspunkt
  - fakultativ: um zahlreiche Teilnahme wird gebeten
- regulärer Übungsbeginn: 13.04.2011 mit Übungsblatt 1

Zur Prüfung

- schriftliche Klausur: 120 Minuten
- voraussichtlich Mitte Juli
- Termine, Räume etc. werden in Vorlesung u. WWW angekündigt
- Durchführung ohne Hilfsmittel
- nur Schreibmaterial (Stifte/Füller, die blau oder schwarz schreiben)
- Bekanntgabe der Ergebnisse: HISQIS
- Einsichtnahme in die Klausur ist möglich (Termin im WWW)

Schein- und Prüfungskriterien: 1. Möglichkeit

Studenten, die den Kurs mit Prüfung oder benotetem Schein beenden wollen, müssen

- regelmäßig und gut in Übungen mitarbeiten,
- mind. 2/3 der Aufgaben votieren,
- mind. 2x Lösung zu schriftlicher Aufgabe präsentieren,
- Klausur nach dem Kurs bestehen

Bestehen der Klausur: Erhalt eines unbenoteten Scheines möglich

Schein- und Prüfungskriterien: 2. Möglichkeit

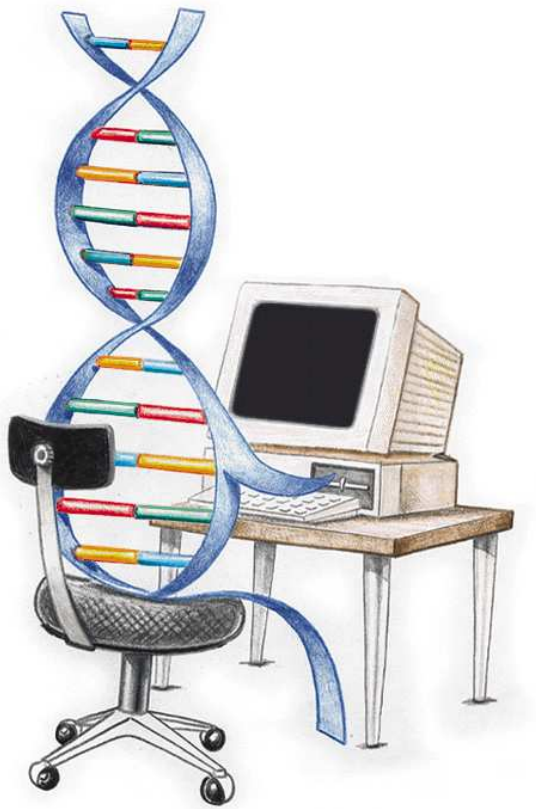
Studenten, die den Kurs mit Prüfung oder benotetem Schein beenden wollen, können alternativ

- am **Programmierwettbewerb** erfolgreich teilnehmen und
- Klausur nach dem Kurs bestehen

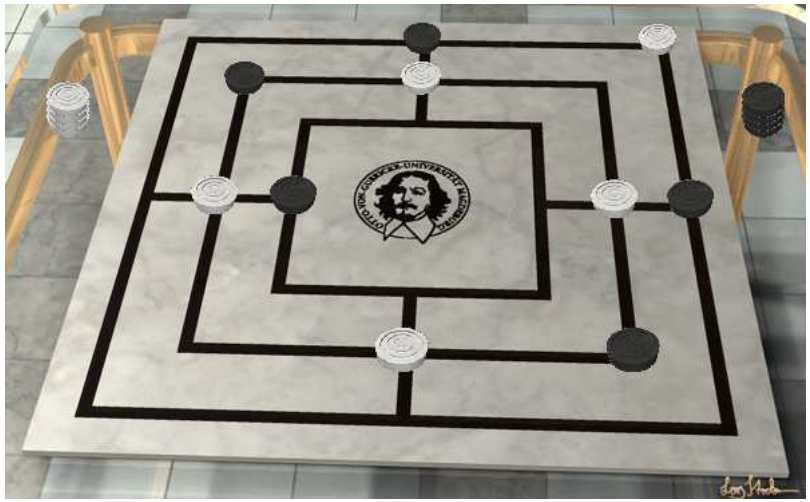
Übungsteilnahme ist hier somit fakultativ
Programmierwettbewerb gilt auch als Prüfungszulassung

Zum Programmierwettbewerb

Dankeschön an Andrea Unger und Thomas Steube



Quelle: `http://www.genetic-programming.com/`

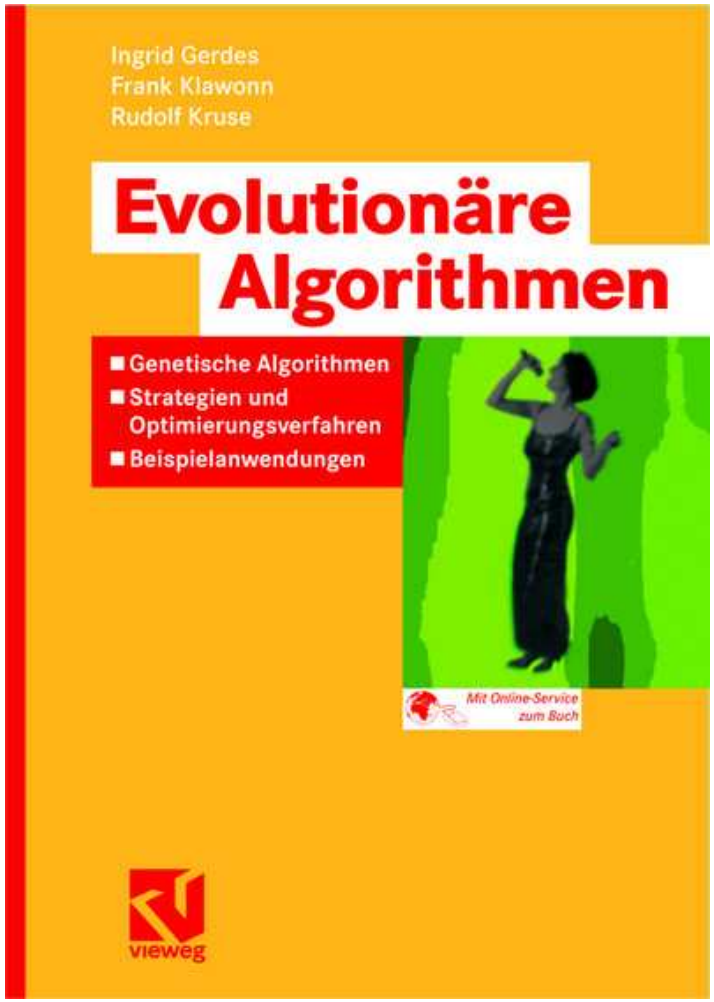


- Idee: EAD-Vorlesung 2003
- Ziel: Entwicklung Mühle-KI
- hier: Optimierung von Spielstrategien mittels EA
- Anmeldebeginn: 06.04.2011 ab 11:00 Uhr

Inhalt der Vorlesung

1. Einführung
2. Metaheuristiken und verwandte Optimierungsverfahren I/II
3. Kodierung, Fitness, Selektion
4. Variation und genetische Operatoren
5. Metaheuristiken und verwandte Optimierungsverfahren II/II
6. Das Schematheorem
7. Genetische Programmierung
8. Evolutionsstrategien und Verhaltenssimulation
9. No Free Lunch, Parallelisierung, Zufallszahlen
10. Mehrkriterienoptimierung
11. Anwendungsbeispiele

Buch zur Vorlesung



Literatur zur Lehrveranstaltung I

Bäck, T. and Schwefel, H. (1993).

An overview of evolutionary algorithms for parameter optimization.

Evolutionary Computation, 1(1):1–23.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998).

Genetic Programming — An Introduction: On the Automatic Evolution of Computer Programs and Its Applications.

Morgan Kaufmann Publisher, Inc. and dpunkt-Verlag, San Francisco, CA, USA and Heidelberg, Germany.

Darwin, C. (1859).

On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.

John Murray, London, United Kingdom.

Literatur zur Lehrveranstaltung II

Dawkins, R. (1986).
The Blind Watchmaker.
Norton, New York, NY, USA.

Dawkins, R. (1989).
The Selfish Gene.
Oxford University Press, United Kingdom, 2nd edition.

Dawkins, R. (1990).
Der blinde Uhrmacher: ein neues Plädoyer für den Darwinismus.
Deutscher Taschenbuch-Verlag, Munich, Germany.

Dawkins, R. (1998).
Das egoistische Gen.
Rowohlt, Reinbek bei Hamburg, Germany.

Literatur zur Lehrveranstaltung III

Dorigo, M. and Stützle, T. (2004).
Ant Colony Optimization.
MIT Press, Cambridge, MA, USA.

Gerdes, I., Klawonn, F., and Kruse, R. (2004).
Evolutionäre Algorithmen.
Vieweg, Wiesbaden, Germany.

Michalewicz, Z. (1996).
Genetic Algorithms + Data Structures = Evolution Programs.
Springer-Verlag, New York, NY, USA, 3rd (extended) edition.

Nissen, V. (1997).
Einführung in evolutionäre Algorithmen: Optimierung nach dem Vorbild der Evolution.
Vieweg, Braunschweig/Wiesbaden, Germany.

Literatur zur Lehrveranstaltung IV

Vollmer, G. (1995).

Der wissenschaftstheoretische status der evolutionstheorie. einwände und gegenargumente.

In Vollmer, G., editor, *Biophilosophie*, page 92–106. Reclam, Stuttgart, Germany.

Weicker, K. (2007).

Evolutionäre Algorithmen.

Teubner Verlag, Stuttgart, Germany, 2nd edition.

Übersicht

Einordnung: Computational Intelligence

Computational Intelligence = **Neuronale Netze** (im Sommer)
+ **Fuzzy-Systeme** (im Winter)
+ **Evolutionäre Algorithmen**

Computational Intelligence ist charakterisiert durch:

- meist „modellfreie“ Ansätze
- Approximation statt exakte Lösung
- schnelles Finden einer brauchbaren Lösung

Lösen von Optimierungsproblemen

Definition (Optimierungsproblem)

Ein *Optimierungsproblem* $(\Omega, f, \succ)$ ist gegeben durch einen Suchraum Ω, eine Bewertungsfunktion $f : \Omega \to \mathbb{R}$, die jedem Lösungskandidaten einen Gütewert zuweist, sowie einer Vergleichsrelation $\succ \in \{<, >\}$. Dann ist die *Menge der globalen Optima* $\mathcal{H} \subseteq \Omega$ definiert als

$$\mathcal{H} = \{x \in \Omega \mid \forall x' \in \Omega : f(x) \succeq f(x')\}.$$

- gegeben: ein Optimierungsproblem $(\Omega, f, \succ)$
- gesucht: ein Element $x \in \Omega$, das die Funktion f (global) optimiert

Prinzipielle Lösungsansätze

analytische Lösung:

- sehr effizient, aber nur seltenen anwendbar

vollständige Durchforstung:

- sehr ineffizient, daher nur bei sehr kleinen Suchräumen anwendbar

blinde Zufallssuche:

- immer anwendbar, aber meist sehr ineffizient

gesteuerte Suche:

- Voraussetzung: Funktionswerte ähnlicher Elemente aus Suchraum Ω sind sich ähnlich

Anwendungsgebiete

Beispiele für Optimierungsprobleme I

Parameteroptimierung

- z.B. Krümmung von Rohren für minimalen Widerstand
- allgemein: Finden eines Parametersatzes, sodaß gegebene reellwertige Funktion ein (möglichst globales) Optimum annimmt

Packprobleme

- z.B. Füllen eines Rucksacks mit maximalem Wert
- oder Packen möglichst weniger Kisten mit gegebenen Gütern

Wegeprobleme

- z.B. Problem des Handlungsreisenden (z.B. Bohren von Platinen)
- Reihenfolge anzufahrender Ziele, Fahrtroutenoptimierung, Verlegen von Leiterbahnen auf Platinen/integrierten Schaltkreisen

Anwendungsgebiete

Beispiele für Optimierungsprobleme II

Anordnungsprobleme

- z.B. Steinerproblem (engl. facility allocation problem):
- Positionierung von Verteilerknoten z.B. in einem Telefonnetz

Planungsprobleme

- z.B. Ablaufpläne (Scheduling), Arbeitspläne, Operationenfolgen
- (auch Optimierung in Compilern – Umordnung der Befehle)

Strategieprobleme

- z.B. Gefangenendilemma und andere Modelle der Spieltheorie
- Verhaltensmodellierung von Akteuren im Wirtschaftsleben

biologische Modellbildung

- z.B. Netspinner (beschreibende Regeln zum Spinnennetz-Bau)
- EA optimiert Parameter, Vergleich mit Realität $\Rightarrow$ gutes Modell

Übersicht

Motivation

- EAs basieren auf **biolog. Evolutionstheorie** [Darwin, 1859].
- empfehlenswert: [Dawkins, 1986, Dawkins, 1989] (Englisch), [Dawkins, 1990, Dawkins, 1998] (Deutsch)

- grundsätzliches Prinzip:
  - **Durch zufällige Variation entstehende, vorteilhafte Eigenschaften werden durch natürliche Auslese ausgewählt.**
  - Individuen mit vorteilhaften Eigenschaften haben bessere Fortpflanzungs- und Vermehrungschancen – *„differentielle Reproduktion“*
- Evolutionstheorie erklärt Vielfalt und Komplexität der Lebewesen
- erlaubt Vereinigung aller Disziplinen der Biologie

Prinzipien der organismischen Evolution I

nach [Vollmer, 1995]

Diversität

- alle Lebewensen (sogar in derselben Art) sind *verschieden*
- jedes Erbgut ist anders $\Rightarrow$ *Vielfalt des Lebens*
- jetzt existierende Lebewesen = winziger Bruchteil aller möglichen

Variation

- *neue Varianten* durch Mutation und genetische Rekombination (sexuelle Fortpflanzung)

Vererbung

- Variationen sind *erblich*, wenn sie in Keimbahn gelangen
- werden also genetisch an nächste Generation weitergegeben
- i.A. keine Vererbung erworbener Eigenschaften (*Lamarckismus*)

Prinzipien der organismischen Evolution II

Artbildung

- *genetische Divergenz* von Individuen und Populationen

$\Rightarrow$ neue *Arten* (Indiv. nicht fruchtbar miteinander kreuzbar)

- charakt. Verzweigungen des phylogenet. (stammesgesch.) Baumes

Überproduktion, fast alle Lebewesen:

- *nicht alle Nachkommen* können Reproduktionsreife erreichen

Anpassung / natürliche Auslese / differentielle Reproduktion

- im Durchschnitt: erbliche Variationen der Überlebenden einer Pop.

$\Rightarrow$ *erhöht Anpassung* an lokale Umgebung

- Herbert Spencers Slogan "survival of the fittest" ist irreführend
- besser „unterschiedl. Tauglichkeit $\Rightarrow$ unterschiedl. Vermehrung"

Prinzipien der organismischen Evolution III

Zufälligkeit / blinde Variation

- Variationen werden *zufällig ausgelöst/bewirkt/verursacht*
- keine Ausrichtung auf best. Merkmale/günstige Anpassungen
- *nicht teleologisch*, von griech.: $\tau\varepsilon\lambda o\varsigma$ — Ziel, Zweck

Gradualismus

- Variationen erfolgen in vergleichsweise *kleinen Stufen* (gemessen am gesamten Informationsgehalt/Komplexität des Organismus)

$\Rightarrow$ phylogenetische Veränderungen = *graduell* und langsam (Gegensatz: Saltationismus — große Entwicklungssprünge)

Evolution / Transmutation / Vererbung mit Modifikation

- Anpassung an Umgebung $\Rightarrow$ Arten *entwickeln* sich allmählich
- Evolutionstheorie im Gegensatz zum Kreationismus (Behauptung der Unveränderlichkeit der Arten)

Prinzipien der organismischen Evolution IV

diskrete genetische Einheiten

- Speichern/Übertragen/Ändern des Erbguts in diskreten Einheiten
- keine kontinuierliche Verschmelzung von Erbmerkmalen
- ansonsten: durch Rekombination zum *Jenkins nightmare* (völliges Verschwinden jeglicher Verschiedenheit in Population)

Opportunismus

- evolutive Prozesse arbeiten nur mit dem, was vorhanden ist
- nicht mit dem, was es einmal gab oder geben könnte
- bessere/optimale Lösungen werden nicht gefunden, wenn benötigte evolutiven Zwischenstadien „schlechter“ wären

evolutionsstrategische Prinzipien

- nicht nur Organismen werden optimiert, sondern auch Mechanismen der Evolution: Vermehrungs- und Sterberaten, Lebensdauern, Anfälligkeit gegenüber Mutationen, Mutationsschrittweiten, Evolutionsgeschwindigkeit, etc.

Prinzipien der organismischen Evolution V

ökologische Nischen

- konkurrierende Arten können einander tolerieren, wenn sie versch. Ökonischen („Lebensräume“) besetzen/schaffen
- trotz Konkurrenz und natürlicher Auslese: Artenvielfalt möglich

Irreversibilität

- Lauf der Evolution ist irreversibel und unwiederholbar

Nichtvorhersagbarkeit

- Lauf der Evolution: nicht determiniert, nicht programmiert, nicht zielgerichtet $\Rightarrow$ nicht vorhersagbar

wachsende Komplexität

- biologische Evolution führt i.A. zu immer komplexeren Systemen
- Problem: Wie misst man Komplexität von Lebewesen?

Übersicht

Grundbegriffe und ihre Bedeutung I

Begriff	Biologie	Informatik
Individuum	Lebewesen	Lösungskandidat
Chromosom	DNS-Histon-Protein-Strang	Zeichenkette
	legt „Bauplan“ bzw. (Teil der Eigenschaften) eines Individuums in kodierter Form fest	
	meist mehrere Chromsomen je Individuum	meist nur ein Chromosom je Individuum
Gen	Teilstück eines Chromosoms	ein Zeichen
	grundlegende Einheit der Vererbung, die eine (Teil-)Eigenschaft eines Individuums festlegt	
Allel (Allelomorph)	Ausprägung eines Gens	Wert eines Zeichens
	je Chromosom gibt es nur eine Ausprägung eines Gens	
Locus	Ort eines Gens	Position eines Zeichens
	in einem Chromosom gibt es an jedem Ort genau ein Gen	

Grundbegriffe und ihre Bedeutung II

Begriff	Biologie	Informatik
Phänotyp	äußeres Erscheinungsbild eines Lebewesens	Umsetzung/Implementierung eines Lösungskandidaten
Genotyp	genetische Konstitution eines Lebewesens	Kodierung eines Lösungskandidaten
Population	Menge von Lebewesen	Familie/Multimenge von Chromosomen
Generation	Population zu einem Zeitpunkt	
Reproduktion	Erzeugen von Nachkommen aus einem oder mehreren (meist zwei) Lebewesen	Erzeugen von (Kind-)Chromosomen aus 1 oder mehreren (Eltern-)Chromosomen
Fitness	Tauglichkeit/Angepaßtheit eines Lebewesens	Güte/Tauglichkeit eines Lösungskandidaten
	bestimmt Überlebens- und Fortpflanzungschancen	

Elemente eines evolutionären Algorithmus I

Kodierungsvorschrift für Lösungskandidaten

- problemspezifische Kodierung der Lösungskandidaten
- keine allgemeinen Regeln
- später: einige Aspekte, zur Beachtung bei Wahl einer Kodierung

Methode, die **Anfangspopulation** erzeugt

- meistens Erzeugen zufälliger Zeichenketten
- auch komplexere Verfahren je nach gewählter Kodierung

Bewertungsfunktion (Fitnessfunktion) für die Individuen

- stellt Umgebung dar und gibt Güte der Individuen an
- meist identisch mit zu optimierender Funktion
- enthält auch zusätzliche Elemente (z.B. Nebenbedingungen)

Elemente eines evolutionären Algorithmus II

Auswahlmethode basierend auf Fitnessfunktion

- bestimmt Individuen für Erzeugung von Nachkommen
- wählt Individuen unverändert in nächste Generation

genetischen Operatoren, die Lösungskandidaten ändern

- *Mutation* — zufällige Veränderung einzelner Gene
- *Crossover* — Rekombination von Chromosomen
  - richtig: "crossing over" (Meiose-Vorgang, Zellteilungsphase)
  - Chromsomen werden zerteilt und dann überkreuzt zusammengefügt

Parameterwerte (Populationsgröße, Mutationsw'keit, etc.)

Abbruchkriterium, z.B.

- festgelegte Anzahl von Generationen berechnet
- festgelegte Anzahl von Generationen keine Verbesserung
- vorgegebene Mindestlösungsgüte erreicht

Formale Definitionen: Dekodierungsfunktion

nach [Weicker, 2007]

- für jedes Optimierungsproblem: unterschiedliche Darstellung der Lösungskandidaten
- EAs trennen Suchraum Ω (sog. Phänotyp) von Darstellung des Lösungskandidaten in Individuum (sog. Genotyp $\mathcal{G}$)
- Bewertungsfunktion f ist definiert auf Ω
- Mutation und Rekombination sind auf $\mathcal{G}$ definiert
- Bewertung eines im Genotyp vorliegenden Individuums durch Abbildung in Ω

Definition (Dekodierungsfunktion)

Eine Dekodierungsfunktion $\text{dec} : \mathcal{G} \rightarrow \Omega$ ist eine Abbildung vom Genotyp $\mathcal{G}$ auf den Phänotyp Ω.

Formale Definitionen: Individuum

Individuum besteht i.A. aus drei Dingen:

1. *Genotyp* $A.G \in \mathcal{G}$ eines Individuums A
2. *Zusatzinformationen* oder *Strategieparameter* $A.S \in \mathcal{Z}$
   - z.B. Parametereinstellungen für Operatoren
   - Raum $\mathcal{Z}$ aller möglichen Zusatzinformationen
   - $A.S$ sind ebenso wie $A.G$ durch Operatoren modifizierbar
3. *Güte* oder *Fitness* $A.F \in \mathbb{R}$

Definition (Individuum)

Ein *Individuum* A ist ein Tupel $(A.G, A.S, A.F)$ bestehend aus dem eigentlichen Lösungskandidaten (dem Genotyp $A.G \in \mathcal{G}$), den optionalen Zusatzinformationen $A.S \in \mathcal{Z}$ und dem Gütewert $A.F = f(\text{dec}(A.G)) \in \mathbb{R}$.

Formale Definitionen: Operatoren

- Ξ („Xi“): Menge aller Zustände des Zufallszahlengenerators
- keine Beschreibung der Veränderung des aktuellen Zustands $\xi \in \Xi$

Definition (Operatoren)

Für ein durch $\mathcal{G}$ kodiertes Optimierungsproblem und $\mathcal{Z}$ wird ein *Mutationsoperator* definiert durch die Abbildung

$$\text{Mut}^{\xi} : \mathcal{G} \times \mathcal{Z} \to \mathcal{G} \times \mathcal{Z}.$$

Analog wird ein *Rekombinationsoperator* mit $r \geq 2$ Eltern und $s \geq 1$ Kindern ($r, s \in \mathbb{N}$) definiert durch die Abbildung

$$\text{Rek}^{\xi} : (\mathcal{G} \times \mathcal{Z})^r \to (\mathcal{G} \times \mathcal{Z})^s.$$

Formale Definitionen: Selektionsoperator

- Eingabe: Population mit r Individuen, wobei s gewählt werden
- Selektion verändert/erfindet keine neuen Individuen
- Selektion bestimmt Indizes der Individuen nur durch Gütewerte

Definition (Selektionsoperator)

Ein *Selektionsoperator* Sel wird auf eine Population $P = \left\langle A^{(1)}, \ldots, A^{(r)} \right\rangle$ angewandt:

$$\mathrm{Sel}^{\xi} : (\mathcal{G} \times \mathcal{Z} \times \mathbb{R})^r \to (\mathcal{G} \times \mathcal{Z} \times \mathbb{R})^r$$

$$\left\langle A^{(i)} \right\rangle_{1 \leq i \leq r} \mapsto \left\langle A^{(\mathrm{IS}^{\xi}(c_1,\ldots,c_r)_k)} \right\rangle_{1 \leq k \leq s} \quad \text{mit } A^{(i)} = (a_i, b_i, c_i).$$

Die dabei zugrunde gelegete Indexselektion hat die Form

$$\mathrm{IS}^{\xi} : \mathbb{R}^r \to \{1, \ldots, r\}^s.$$

Beispiel zum Selektionsoperator

- Elternpopulation bestehe aus Individuen $A^{(1)}, A^{(2)}, \ldots, A^{(5)}$
- jeweiligen Gütewerte der Individuen seien
  1. $A^{(1)}.F = 2.5$
  2. $A^{(2)}.F = 1.9$
  3. $A^{(3)}.F = 3.7$
  4. $A^{(4)}.F = 4.1$
  5. $A^{(5)}.F = 2.4$
- Selektion wählt mit $\mathrm{IS}^{\xi} : \mathbb{R}^5 \rightarrow \{1, \ldots, 5\}^3$ Indizes 4, 3 und 1 bzw. Individuen $A^{(4)}, A^{(3)}$ und $A^{(1)}$

Generischer Grundalgorithmus

Definition

Ein *einfacher evolutionärer Algorithmus* zu einem Optimierungsproblem $(\Omega, f, \succ)$ ist ein 8-Tupel $(\mathcal{G}, \text{dec}, \text{Mut}, \text{Rek}, \text{IS}_{\text{Eltern}}, \text{IS}_{\text{Umwelt}}, \mu, \lambda)$. Dabei bezeichnen μ die Anzahl an Individuen in der Elternpopulation und λ die Anzahl der erzeugten Kinder pro Generation. Ferner gelten

$$\text{Rek} : (\mathcal{G} \times \mathcal{Z})^k \to (\mathcal{G} \times \mathcal{Z})^{k'},$$

$$\text{IS}_{\text{Eltern}} : \mathbb{R}^\mu \to (1, \ldots, \mu)^{\frac{k}{k'} \cdot \lambda} \qquad \text{mit } \frac{k}{k'} \cdot \lambda \in \mathbb{N},$$

$$\text{IS}_{\text{Umwelt}} : \mathbb{R}^{\mu+\lambda} \to (1, \ldots, \mu + \lambda)^\mu.$$

Generischer Grundalgorithmus

Algorithm 1 EA-Schema

Input: Optimierungsproblem $(\Omega, f, \succ)$

$t \leftarrow 0$
$\text{pop}(t) \leftarrow$ erzeuge Population der Größe μ
bewerte $\text{pop}(t)$
while Terminierungsbedingung nicht erfüllt {
  $\text{pop}_1 \leftarrow$ selektiere Eltern für λ Nachkommen aus $\text{pop}(t)$
  $\text{pop}_2 \leftarrow$ erzeuge Nachkommen durch Rekombination aus pop_1
  $\text{pop}_3 \leftarrow$ mutiere die Individuen in pop_2
  bewerte pop_3
  $t \leftarrow t + 1$
  $\text{pop}(t) \leftarrow$ selektiere μ Individuen aus $\text{pop}_3 \cup \text{pop}(t-1)$
}
return bestes Individuum aus $\text{pop}(t)$

Genetischer vs. Evolutionärer Algorithmus

Begriffliche Trennung

genetischer Algorithmus:

- Kodierung: Zeichenkette aus Nullen und Einsen

$\Rightarrow$ Chromosomen ist *Bitstring* (Wort über Alphabet $\{0,\ 1\}$)

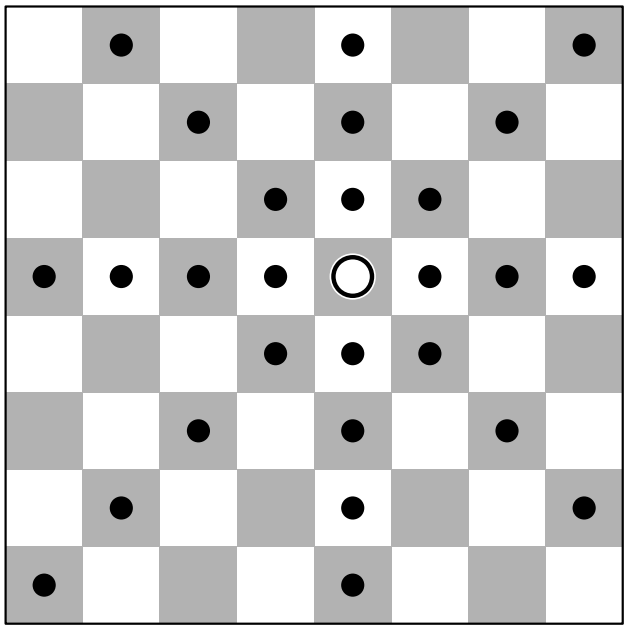
evolutionärer Algorithmus:

- Kodierung: problemspezifisch
  (Zeichenkette aus Buchstaben, Graphen, Formeln, etc.)
- genetische Operatoren: anhand Kodierung und Problems definiert

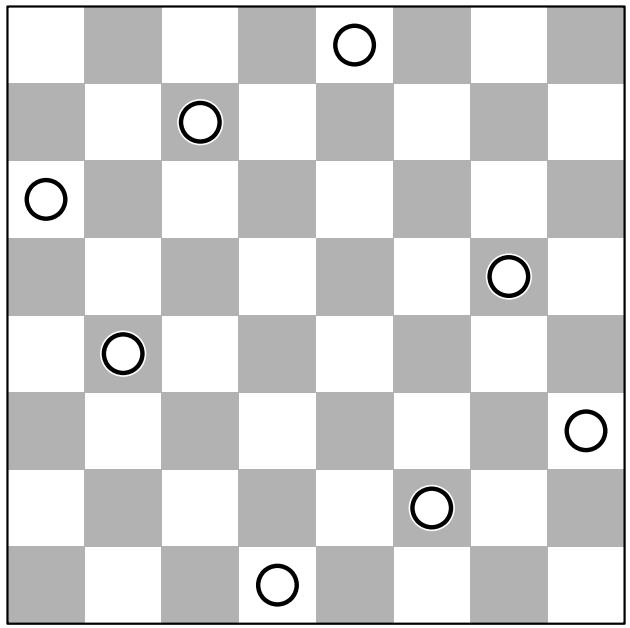
Übersicht

Das n-Damen-Problem

platziere n Damen auf $n \times n$ Schachbrett, sodass in keiner Reihe, keiner Linie und keiner Diagonale > 1 Dame
oder: Platziere Damen, sodass keine Dame einer anderen im Weg



Zugmöglichkeiten einer Dame



Lösung des 8-Damen-Problems

Lösung durch Backtracking

1. platziere Damen zeilenweise von unten nach oben
   (oder spaltenweise von links nach rechts, o.ä.)
2. betrachte jede Zeile wie folgt:
   - setze in 1 Zeile 1 Dame der Reihe nach von links nach rechts auf Felder der Zeile
   - für jede Damenplatzierung: prüfe, ob Kollision mit Damen in tieferliegenden Zeilen
   - falls nicht, bearbeite rekursiv nächste Zeile
   - anschließend: rücke Dame 1 Feld weiter nach rechts
3. gebe Lösung aus, falls Dame in oberster Zeile ohne Kollision platziert

Lösung durch Backtracking

```
int search (int y)
{                                   /* --- depth first search */
  int x, i, d;                      /* loop variables, buffer */
  int sol = 0;                      /* solution counter */

  if (y >= size) {                  /* if a solution has been found, */
    show(); return 1; }             /* show it and abort the function */
  for (x = 0; x < size; x++) {      /* traverse fields of the current row */
    for (i = y; --i >= 0; ) {       /* traverse the preceding rows */
      d = abs(qpos[i] -x);          /* and check for collisions */
      if ((d == 0) || (d == y-i)) break;
    }                               /* if there is a colliding queen, */
    if (i >= 0) continue;           /* skip the current field */
    qpos[y] = x;                    /* otherwise place the queen */
    sol += search(y+1);             /* and search recursively */
  }
  return sol;                       /* return the number of */
}  /* search() */                   /* solutions found */
```

Direkte Berechnung

falls nur *eine* Lösung (eine Platzierung der Damen) erforderlich, dann Berechnung der Positionen für alle $n > 3$ wie folgt:

- $n \bmod 2 = 1 \Rightarrow$ setze 1 Dame auf $(n-1, n-1)$ und $n \leftarrow n-1$
- $n \bmod 6 \neq 2 \Rightarrow$ setze Damen
  in Zeilen $y = 0, \ldots, \frac{n}{2} - 1$ in Spalten $x = 2y + 1$,
  in den Zeilen $y = \frac{n}{2}, \ldots, n-1$ in Spalten $x = 2y - n$
- $n \bmod 6 = 2 \Rightarrow$ setze Damen
  in Zeilen $y = 0, \ldots, \frac{n}{2} - 1$ in Spalten $x = (2y + \frac{n}{2}) \bmod n$,
  in Zeilen $y = \frac{n}{2}, \ldots, n-1$ in Spalten $x = (2y - \frac{n}{2} + 2) \bmod n$

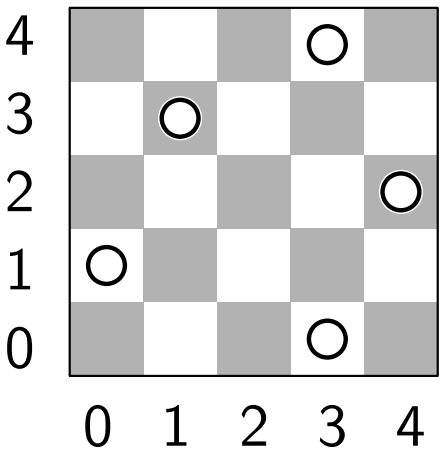
daher: EA nicht zweckmäßig zur Lösung dieses Problems
aber: gute Verdeutlichung einiger EA-Aspekte an diesem Problem

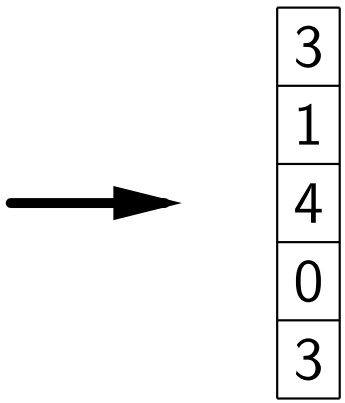
EA: Kodierung

- Darstellung: 1 Lösungskandidat = 1 Chromosom mit n Genen
- jedes Gen: 1 Zeile des Brettes mit n mögliche Allelen
- Wert des Gens: Position der Dame in zugehöriger Zeile

Lösungskandidat ($n = 5$)

Phänotyp





Chromosom

Genotyp

- Lösungskandidaten mit > 1 Dame je Zeile ausgeschlossen

$\Rightarrow$ kleinerer Suchraum

EA: Datentypen

- Datentyp für 1 Chromosom, der auch Fitness speichert
- Datentyp für 1 Population mit Puffer für „Zwischenpopulation“ und Merker für bestes Individuum

```
typedef struct {                /* --- an individual --- */
  int fitness;                  /* fitness (number of collisions) */
  int cnt;                      /* number of genes (number of rows) */
  int genes[1];                 /* genes (queen positions in rows) */
} IND;                          /* (individual) */


typedef struct {                /* --- a population --- */
  int size;                     /* number of individuals */
  IND **inds;                   /* vector of individuals */
  IND **buf;                    /* buffer for individuals */
  IND *best;                    /* best individual */
} POP;                          /* (population) */
```

EA: Hauptschleife

zeigt Grundform eines EA:

```
pop_init(pop);                  /* initialize the population */
while ((pop_eval(pop) < 0)      /* while no solution found and */
&&     (--gencnt >= 0)) {       /* not all generations computed */
  pop_select(pop, tmsize, elitist);
  pop_cross (pop, frac);        /* select individuals, */
  pop_mutate(pop, prob);        /* do crossover, and */
}                               /* mutate individuals */
```

Parameter:

`gencnt`	maximale Anzahl (noch) zu berechnender Generationen
`tmsize`	Größe des Turniers für die Individuenauswahl
`elitist`	zeigt an, ob bestes Individuum immer übernommen wird
`frac`	Anteil der Individuen, die Crossover unterworfen werden
`prob`	Mutationswahrscheinlichkeit

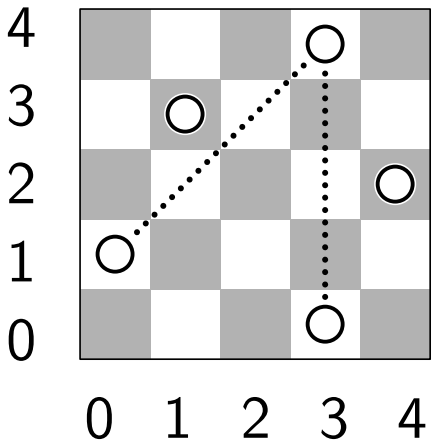
EA: Initialisierung

erzeuge zufällige Folgen von n Zahlen aus $\{0, 1, \ldots, n-1\}$

```
void ind_init (IND *ind)
{                                   /* --- initialize an individual */
  int i;                            /* loop variable */

  for (i = ind->n; --i >= 0; ) /* initialize the genes randomly */
    ind->genes[i] = (int)(ind->n *drand());
  ind->fitness = 1;                 /* fitness is not known yet */
}  /* ind_init() */


void pop_init (POP *pop)
{                                   /* --- initialize a population */
  int i;                            /* loop variable */

  for (i = pop->size; --i >= 0; )
    ind_init(pop->inds[i]);         /* initialize all individuals */
}  /* pop_init() */
```

EA: Bewertung

- Fitness: negierte Zahl der Spalten und Diagonalen mit ≥ 1 Dame (negierte Zahl, weil zu maximierende Fitness)



2 Kollisionen $\rightarrow$ Fitness $= -2$

- bei ≥ 2 Damen in 1 Spalte/Diagonale: zähle jedes Paar (einfacher zu implementieren)
- Fitnessfunktion ergibt unmittelbar Abbruchkriterium:
  Lösung hat (maximal mögliche) Fitness 0
- auch: garantierte Terminierung durch maximale Generationenzahl

EA: Bewertung

zähle Kollisionen durch Berechnungen auf Chromosomen:

```
int ind_eval (IND *ind)
{                                   /* --- evaluate an individual */
  int i, k;                         /* loop variables */
  int d;                            /* horizontal distance between queens */
  int n;                            /* number of collisions */

  if (ind->fitness <= 0)            /* if the fitness is already known, */
    return ind->fitness;            /* simply return it */
  for (n = 0, i = ind->n; --i > 0; ) {
    for (k = i; --k >= 0; ) {       /* traverse all pairs of queens */
      d = abs(ind->genes[i] -ind->genes[k]);
      if ((d == 0) || (d == i-k)) n++;
    }                               /* count the number of pairs of queens */
  }                                 /* in the same column or diagonal */
  return ind->fitness = -n;         /* return the number of collisions */
}  /* ind_eval() */
```

EA: Bewertung

- Fitness aller Individuen der Population wird berechnet
- gleichzeitig: Bestimmung des besten Individuums
- bestes Individuum Fitness 0 $\Rightarrow$ Lösung gefunden

```
int pop_eval (POP *pop)
{                                   /* --- evaluate a population */
  int i;                            /* loop variable */
  IND *best;                        /* best individual */

  ind_eval(best = pop->inds[0]);
  for (i = pop->size; --i > 0; )
    if (ind_eval(pop->inds[i]) >= best->fitness)
      best = pop->inds[i];          /* find the best individual */
  pop->best = best;                 /* note the best individual */
  return best->fitness;             /* and return its fitness */
}  /* pop_eval() */
```

EA: Auswahl von Individuen

Turnierauswahl:

- betrachte `tmsize` zufällig bestimmte Individuen
- bestes dieser Individuen „gewinnt“ Turnier und wird ausgewählt
- je höher Fitness, desto größer Chance, ausgewählt zu werden

```
IND* pop_tmsel (POP *pop, int tmsize)
{                                  /* --- tournament selection */
  IND *ind, *best;                 /* competing/best individual */

  best = pop->inds[(int)(pop->size *drand())];
  while (--tmsize > 0) {           /* randomly select tmsize individuals */
    ind = pop->inds[(int)(pop->size *drand())];
    if (ind->fitness > best->fitness) best = ind;
  }                                /* det. individual with best fitness */
  return best;                     /* and return this individual */
}  /* pop_tmsel() */
```

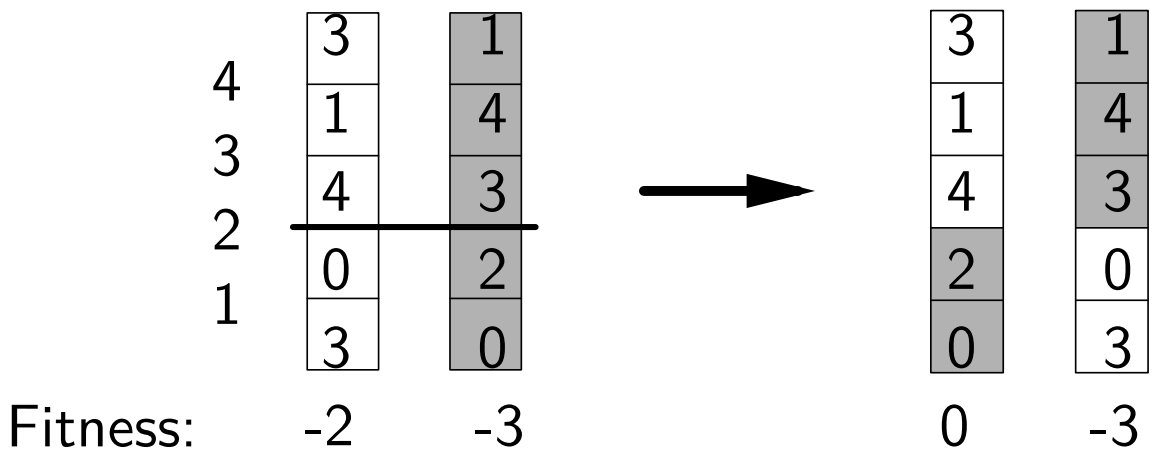
EA: Auswahl von Individuen

- Turnierauswahl für Individuen der nächsten Generation
- evtl. bestes Individuum übernehmen (und nicht verändern)

```
void pop_select (POP *pop, int tmsize, int elitist)
{                                   /* --- select individuals */
  int i;                            /* loop variables */
  IND **p;                          /* exchange buffer */

  i = pop->size;                    /* select 'popsize' individuals */
  if (elitist)                      /* preserve the best individual */
    ind_copy(pop->buf[--i], pop->best);
  while (--i >= 0)                  /* select (other) individuals */
    ind_copy(pop->buf[i], pop_tmsel(pop, tmsize));
  p = pop->inds; pop->inds = pop->buf;
  pop->buf  = p;                    /* set selected individuals */
  pop->best = NULL;                 /* best individual is not known yet */
}  /* pop_select() */
```

EA: Crossover

- Tausch eines Chromosomenstücks (oder ausgewählter Teilmenge der Gene) zweier Individuen
- hier: sogenanntes **Ein-Punkt-Crossover**
  - wähle zufällig eine Trennstelle zwischen zwei Genen
  - tausche Gensequenzen auf einer Seite der Trennstelle aus
  - Beispiel: wähle Trennstelle 2



EA: Crossover

Austausch eines Chromosomenstücks zwischen zwei Individuen

```
void ind_cross (IND *ind1, IND *ind2)
{                                   /* --- crossover of two chromosomes */
  int i;                            /* loop variable */
  int k;                            /* gene index of crossover point */
  int t;                            /* exchange buffer */

  k = (int)(drand() *(ind1->n-1)) +1; /* choose a crossover point */
  if (k > (ind1->n >> 1)) { i = ind1->n; }
  else                    { i = k; k = 0; }
  while (--i >= k) {                /* traverse smaller section */
    t              = ind1->genes[i];
    ind1->genes[i] = ind2->genes[i];
    ind2->genes[i] = t;             /* exchange genes */
  }                                 /* of the chromosomes */
  ind1->fitness = 1;                /* invalidate the fitness */
  ind2->fitness = 1;                /* of the changed individuals */
}  /* ind_cross() */
```

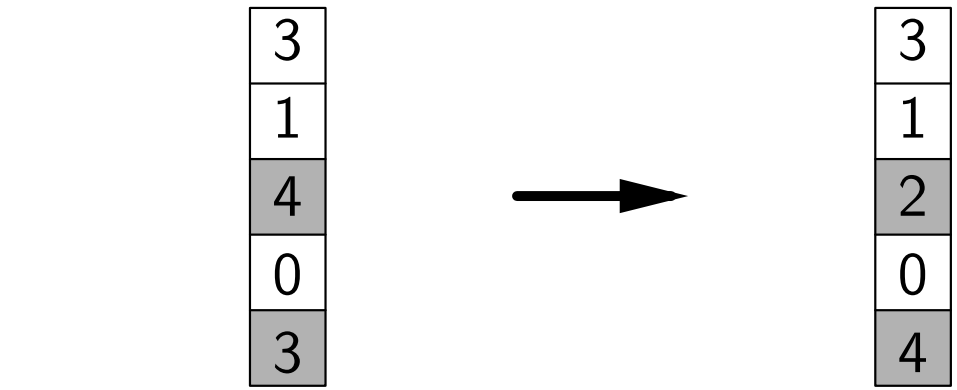
EA: Crossover

- gewisser Anteil der Individuen wird Crossover unterworfen
- Aufnehmen beider Crossoverprodukte in neue Population
- „Elternindividuen“ gehen verloren
- kein Crossover mit bestem Individuum (falls übernommen)

```
void pop_cross (POP *pop, double frac)
{                                   /* --- crossover in a population */
  int i, k;                         /* loop variables */

  k = (int)((pop->size -1) *frac) & ~1;
  for (i = 0; i < k; i += 2)    /* crossover of pairs of individuals */
    ind_cross(pop->inds[i], pop->inds[i+1]);
}  /* pop_cross() */
```

EA: Mutation

- Ersetzen zufällig gewählter Gene (Ändern der Allele)
- u.U. zufällige Anzahl der zu ersetzenden Gene
  (Zahl ersetzter Gene sollte klein sein)



- meisten Mutationen sind schädlich (verschlechtern Fitness)
- nicht vorhandene Allele können durch Mutation entstehen

EA: Mutation

- entscheide, ob (weiter) mutiert werden soll
- bestes Individuum (falls übernommen) wird nicht mutiert

```
void ind_mutate (IND *ind, double prob)
{                                  /* --- mutate an individual */
  if (drand() >= prob) return; /* det. whether to change individual */
  do ind->genes[(int)(ind->n *drand())] = (int)(ind->n *drand());
  while (drand() < prob);          /* randomly change random genes */
  ind->fitness = 1;                /* fitness is no longer known */
}  /* ind_mutate() */


void pop_mutate (POP *pop, double prob)
{                                  /* --- mutate a population */
  int i;                           /* loop variable */
  for (i = pop->size -1; --i >= 0; )
    ind_mutate(pop->inds[i], prob);
}  /* pop_mutate() */              /* mutate individuals */
```

Programme

- die besprochenen Verfahren zur Lösung des n-Damen-Problems,
  - Backtracking,
  - direkte Berechnung,
  - evolutionärer Algorithmus,

  sind zu finden auf der Vorlesungsseite als Kommadozeilenprogramme (C-Programme `queens.c` und `qga.c`)
- Aufruf ohne Parameter: Liste von Programmoptionen
- beachte: EA findet nicht immer Lösung (Fitness < 0)
- Eigenschaften der Verfahren in einigen Übungsaufgaben