Adaptive Agenten

Complex Adaptive Systems

Prof. Dr. Michael Köhler-Bußmeier

HAW Hamburg, Department Informatik

Version vom 13. September 2016

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Agent

- Agenten-Programmierung durch Wenn-Dann-Regeln
- Die Ähnlichkeit zu PROLOG ist nicht zufällig.
- Regel-Beispiel in JASON:

```
+bomb(Terminal, Gate, BombType) : skill(BombType)
    <- !go(Terminal, Gate);
    disarm(BombType).
```

- Lies: **Wenn** ich den *belief* `bomb(Terminal, Gate, BombType)` hinzufüge (+) **und** ich glaube, dass mein Keintext (Wissen usw.) `skill(BombType)` gilt,
  dann führe ich die Plan-Aktionen `!go(Terminal, Gate)` und `disarm(BombType)` aus.
- The conjunction of literals in the context must be a logical consequence of that agent's current beliefs.

Anforderungen an das Regelsystem

Fragen:

- Wie kann Flexibilität erreicht werden?
- Muss man eine Regel für jede denkbare Situation definieren?
- Wie können Agenten sich anpassen/lernen?

Wir folgen hier den Ideen von John H. Holland: *Hidden order: How Adaptation builds complexity* [Hol95]

Agenten

- Regelform: Wenn *Trigger*, dann *Aktion*.
- Jeder Trigger ist eine Aussagenlogische Formel.
- Die Atome sind dann Signale
  – entweder **Sensor-Input** oder **interne Signale**.
- Wir nehmen an, dass Trigger in DNF vorliegen:

$$\textbf{Wenn } (\neg A \wedge B) \vee (B \wedge C) \vee (A \wedge \neg B \wedge \neg D) \vee (C), \textbf{ dann } action$$

- Aus jeder Co-Klausel: $(\neg A \wedge B)$ usw. machen wir eine eigene Regel.
- Diese existeren dann als alternative Regel:

$$\begin{aligned}
&\textbf{Wenn } (\neg A \wedge B), \textbf{ dann } action\\
&\textbf{Wenn } (B \wedge C), \textbf{ dann } action\\
&\textbf{Wenn } (A \wedge \neg B \wedge \neg D), \textbf{ dann } action\\
&\textbf{Wenn } (C), \textbf{ dann } action
\end{aligned}$$

- Einfaches Regeln der Form: Wenn *konj. Trigger*, dann *Aktion*.
- Jeder Trigger ist dann eine Konjunktion von Literalen (Atome bzw. negierte Atome).

Regelüberlappung

- Einfaches Regelsystem: Wenn *Trigger*, dann *Aktion*.
- Dies hat interessante Konsequenzen:
- Niemand hat verboten, dass mehrere Regeln gleichzeitig aktiv sind.
- Beispiel:

$$\textbf{Wenn } (A \wedge B), \textbf{ dann } action_1$$
$$\textbf{Wenn } (A \wedge C), \textbf{ dann } action_2$$
$$\textbf{Wenn } (A \wedge B \wedge C), \textbf{ dann } action_3$$

- Wenn der Agent die Signale A, B und C erhält, dann sind alle drei Regeln aktiviert.

Regelüberlappung: Orientierung

Example

Ein Frosch mit zwei überlappenden Regeln:

Wenn `moving object`, **dann** `flee`
Wenn (`moving object` $\wedge$ `small object`, **dann** `hunt`

Regel 1 verhindert es, gefressen zu werden.
Regel 2 sorgt für Futter.
Der Frosch sollte die Regel so ausbalancieren, dass seine Überlebenswkt. maximiert wird.

- Wenn wir mehrere aktivierte Regeln haben, welche wollen wir dann auswählen?
- Wir benötigen eine **Auswahlstrategie**.

Regelüberlappung: Orientierung

- Die Überlappung von Regeln hat einen großen Vorteil:
- Ein Agent ist bei völlig neuen Situationen nicht unbedingt unvorbereitet.
- Er kann sich seine Reaktionen für die Spezialsituation aus den Reaktionen zusammenbasteln, die er in den allgemeiner Situationen verwenden würde.
- Der Agent hat keine Regel, wenn er sich in einer Situation $(A \wedge B \wedge C)$ befindet. Er hat aber zwei Regel für die allgemeineren Situationen $(A \wedge B)$ bzw. $(A \wedge C)$:

  Wenn $(A \wedge B)$, **dann** $action_1$
  Wenn $(A \wedge C)$ **dann** $action_2$

  Meine Reaktion könnte jetzt also „irgendwie“ aus $action_1$ und $action_2$ zusammensetzen.
- Natürlich kann dieses Vorgehen auch mal schiefgehen.
- Meist sind aber die Aktionen für die Allgemeinsituationen aber „bewährt“.
- Offen ist noch wie man eine Reaktion aus $action_1$ und $action_2$ zusammensetzen kann.

Regelüberlappung

Zur Kombination von Reaktionen aus Bewährtem.

Example (nach John Holland)

Wie reagiert man in einer Situation:

(having a flat tire)(while driving) (a red saab)(on the expressway)

Man greift zurück auf Bewährtes:

Wenn (having a flat tire)(while driving), **dann** slow down
Wenn (having a flat tire) (on the expressway), **dann** pull to trouble lane

Aus diesen Regeln generiert man dann eine Reaktion.

Regelüberlappung: Auswahlstrategie 1

Verschiedene Auswahlstrategien sind denkbar:

- Es wird die erste im Listing gewählt.
  - Vorteil: leicht zu programmieren
  - Nachteil: Dies verhindert viele Optimierungen.
- Irgendeine Regel wird per Zufall gewählt.
  - Vorteil: Konzeptionell einfach
  - Nachteil: Dadurch wird Erfahrungslernen unmöglich.
- Es wird eine möglichst spezifische Regel gewählt.
  - Vorteil: Man kann eine Hierarchie von Regeln definieren.
  - Nachteil, dass ist i.a. immer noch nicht eindeutig.
    Beispiel: Die Signale A, B und C aktivieren alle Regeln:

    Wenn $(A \wedge B)$, **dann** $action_1$
    Wenn $(A \wedge C)$ **dann** $action_2$
- ...

Regelüberlappung: Auswahlstrategie 2

- ...
- Regeln werden gewichtet; die stärkste Regeln gewinnt.
  - Vorteil: Durch ein Anpassen der Gewichte kann gelernt werden.
  - Nachteil: Gewichte müssen verwaltet und angepasst werden.
- Regeln werden gewichtet; es wird gewürfelt.
  Die Auswahl-Wkt. einer Regel ist ihr Anteil am Gesamtgewicht alle aktivierten Regeln.
  - Vorteil: Das System reagiert (meist) robuster, da auch „kleine“ Regeln eine Chance haben.
    Auch wäre dies ein Mechanismus, um in Spezialsituationen zu reagieren.
  - Nachteil: Noch mehr Aufwand
- usw.

Bewertungen

- Man will Überlappung der Trigger, damit der Programmierer nicht auf Disjunktheit achten muss.
- Mein Favorit: Regeln werden gewichtet; es wird gewürfelt. Die Auswahl-Wkt. einer Regel ist ihr Anteil am Gewicht alle aktivierten Regeln.
- Vorteil: Wir können uns auf sich verändernde Umgebungen einstellen (lernen).
- Vorteil: Wir können robust auch mal vom „Trott“ abweichen.
- Vorteil: Wir können auch in Speziallsituationen auf Erfahrungswissen zurückgreifen und stehen nicht ratlos da.

- Im folgenden Abschnitt: Wie werden Gewichte angepasst?
  (Lernen als Anpassung an die Umgebung)
- Danach: Wie werden aus erfolgreichem Regeln neue generiert?
  (Lernen über die Anpassung hinaus)

Bewertungen: Regelkaskaden

- Wir gehen davon aus, dass die Umgebung unser Verhalten bewertet (reward based learning).
- Beispiel: Der Frosch erhält als Belohnung für sein Verhalten seine Jagdbeute.
- Die Belohnung ist sogar in der Einheit „Kalorien“ quantifiziert.
- Problem: Normalweise führt ein sensorischer Input nicht direkt zu einer motorischen Antwort.
- Stattdessen aktiviert eine Regel interne Signale, die eine Kaskade von Regeln auslöst
- Analog bei Brettspielen usw.

Bewertungen: Beispiel

Example

Frosch:

1 : **Wenn** small object from left, **dann** jump to left
2 : **Wenn** small object from left, **dann** move head left
3 : **Wenn** small object ahead, **dann** spit
4 : **Wenn** small object ahead, **dann** tongue
5 : **Wenn** tasty object, **dann** yum-yum

Wenn eine Fliege von links kommt, dann könnte die Kaskade so aussehen:

2 : **Wenn** small object from left, **dann** move head left
4 : **Wenn** small object ahead, **dann** tongue
5 : **Wenn** tasty object, **dann** yum-yum

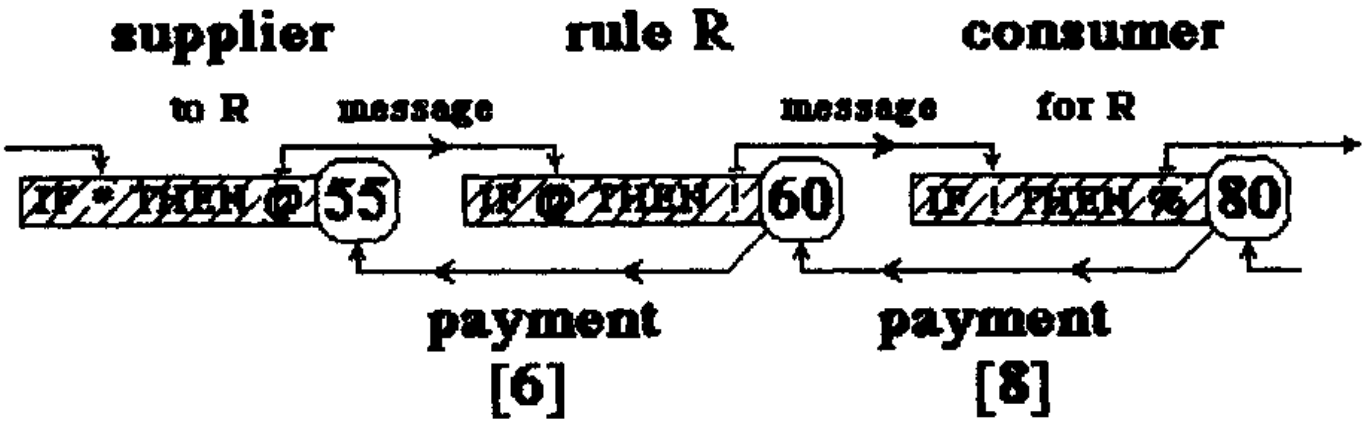
Hier sollte jetzt nicht die Regel 5 belohnt werden, sondern eher die Auflösung des Konfliktes von 1 und 2, die der Frosch hier zugunsten von Regel 2 entschieden hat.

Bewertungen

- Problem: Belohnung und die richtige Auswahl ist nicht direkt zuzuordnen.
- Es existiert eine ganze Handlungskette zwischen Reiz und Reaktion.
- Frage: Welche Regel in dieser Kette ist wie zu belohnen?
- Das Beispiel hat gezeigt: Wir können nicht einfach die letzte Regel belohnen, denn vielleicht war es ja die erste Auswahl zu Beginn der Kette, die wichtig war (und der Rest der Kette war im Extremfall sogar eindeutig).
- In diesem Fall wäre es also unproduktiv, die letzte Regel zu belohnen.
- Stattdessen ist die gesamte Kette zu belohnen.
- Aber: Wie ist die Belohnung aufzuteilen?

Bewertungen: Regelkaskaden

- John Holland schlägt ein producer-consumer Bezahlsystem vor.
- Wenn eine Regel auf ein Signal reagiert, dann „zahlt“ sie der Signal-auslösenden Regel sofort eine Belohnung (d.h. auch ohne eine Belohnung durch die Umgebung).
- Dafür darf die letzte Regel die Umgebungsbelohnung behalten.
- Die Währungseinheit ist das Regelgewicht.
- Die Umgebungsbelohnung fällt auch abhängig vom Gewicht aus.



- mögliche Umsetzung: Eine Regel zahlt 10% seines Gewichtes an den Signalauslöser (den Verkäufer).
- Die Umwelt (der Käufer) zahlt 10% des Gewichtes derjenigen Regel, die eine zu belohnende Aktion in der Umwelt vornimmt.

Bewertungen: Gewichtungsdynamik

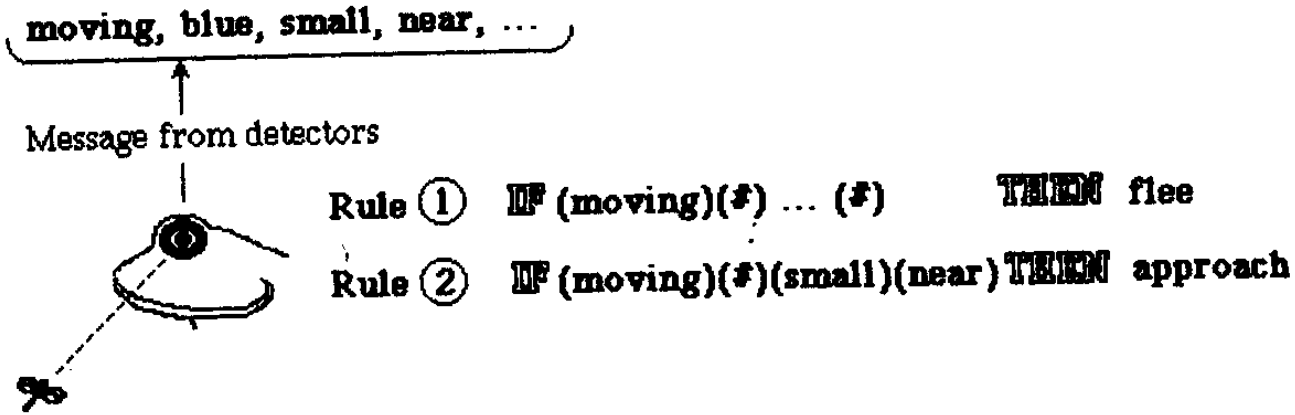
- direkte Konsequenz: Eine starke Regel wird oft den Zuschlag erhalten und wird dann schwächer.
- Dies „lohnt“ sich für die Regel nur, wenn sie teuer „weiterverkaufen“ kann.
- Sprich: Nur wenn die letzte Regel von der Umwelt belohnt wird, dann haben sich die Käufe entlang der Kette gelohnt.
- Sollte die Belohnung ausfallen, dann machen die Regeln entlang der Lieferkette Verlust.
- Das ist auch gewümscht, denn so gut scheinen sich die Regeln ja nicht zu bewähren.
- Die Hoffnung ist daher, dass alternative Regel sich vielleicht besser schlagen.
- Verlieren die gewichtigsten Regeln nun durch Miserfolg an Gewicht, dann erhöht sich dadurch das relative Gewicht der alternativen Regeln.
- Sollten sich diese bewähren, dann machen sie den Gewinn. Dadurch erhöht sich ihr relatives Gewicht noch weiter usw.

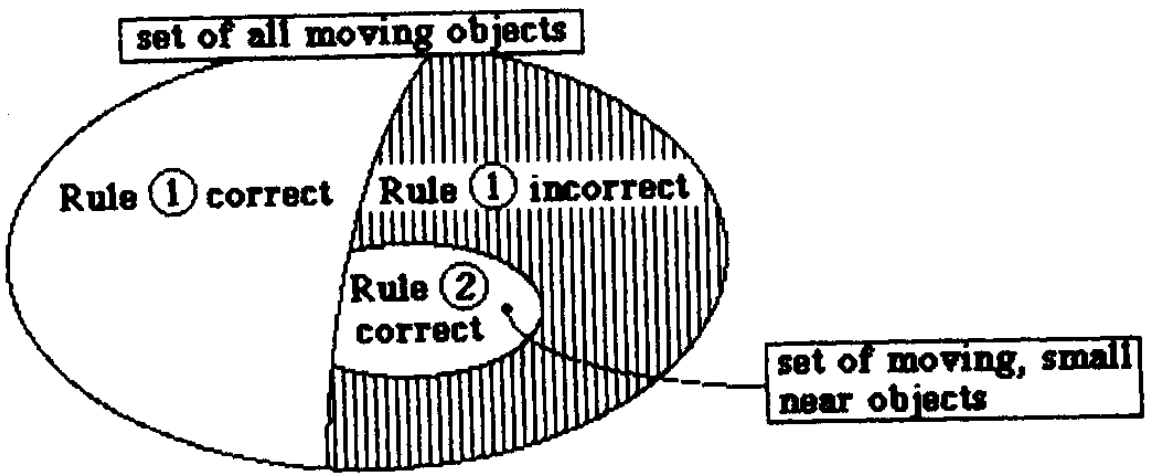
Bewertungen: Spezifizität

- Wir können Ausweertung von Bewertungsketten noch erweitern.
- Wir bauen eine Regel-Hierarchie auf:
- Spezifischere Regeln gehen vor.
  $A \wedge B \wedge C$ ist spezifischer als $A \wedge B$.
- Spezifizität = Anzahl der Atome im Trigger
- Daher vergeben wir den Zuschlag nicht per Würfelwurf proportional zum Gewicht der passenden Regeln.
- Vielmehr ist die Wkt. proportional zum Produkt aus Gewicht und Spezifizität.
- Spezifischere Regeln werden dadurch bevorzugt.

Frosch mit Regel-Hierarchie:

- Kleine Objekte in der Nähe sind leichte Beute.
- Kleine Objekte in der Ferne sind Beute, aber unsichere.
- Große Objekte sind eher gefährlich.





Bewertungen: Spezifizität

- Bislang: Agenten lernen durch Adaption.
- Jetzt: Unsere Agenten sollen Neues lernen.
- Situation = Menge an Signalen
- Bislang: Zu einer Situation besitzt ein Agent eine Menge von passenden Regeln, die (je nach Gewicht) ausgewählt werden.
- Nun: Erzeugen neuer Regeln
- Wir wollen dabei auf Bewährtes (= starke Regeln) zurückgreifen,
- Annahme: eine „Gruppe an Trigger-Bedingung“, die in allen starken Regeln vorkomt, ist relevant.
- Neues Konzept: ein **Schema** einer Regel
- Ein Schema beschreibt nun formal diese „Gruppe“.

Regeln: Bausteine

Angenommen wir haben die Signale $A_1, \ldots, A_n$.

- Ein Trigger ist eine Konjunktion: bspw. $A_1 \wedge \neg A_4 \wedge A_6$
- Notation als Sequenz der Länge n ist dann: $1\#\#0\#1\#\ldots\#$
- 1 = Atom, 0 = negiertes Atom, # = “don’t care”

- Ein **Schema** definiert nun Muster für solche Trigger-Sequenzen.
- Positionen, an denen der Wert 0/1/# egal ist, werden mit dem Symbol * markiert.

Example

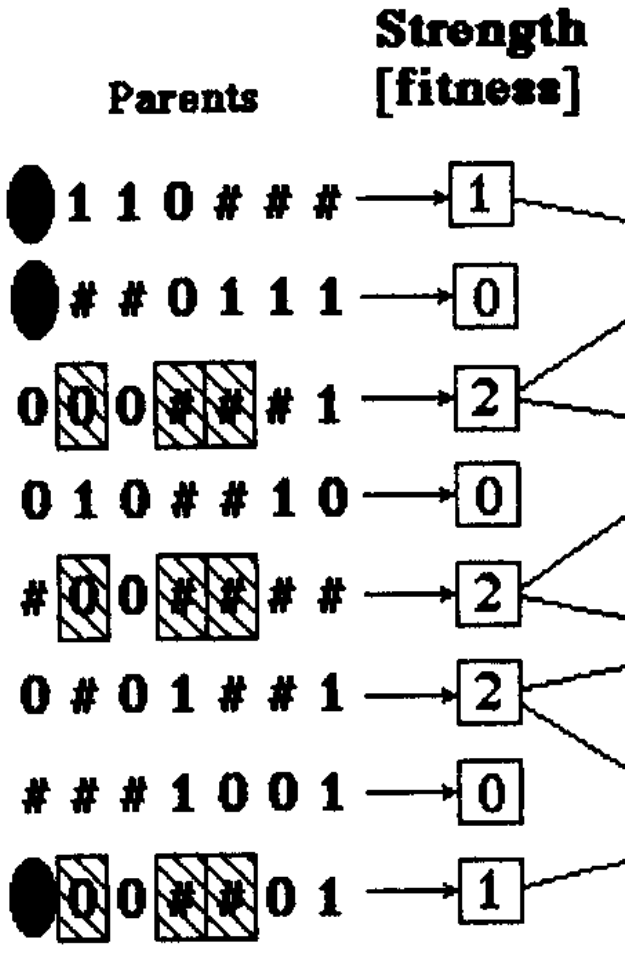
- Beispiel: $1\#*0****\ldots*$ ist ein Schema, das die ersten 3 Positionen definiert.
- Der Trigger $1\#\#0\#1\#\ldots\#$ passt zu diesem Baustein.
- Der Trigger $11\#0\#1\#\ldots\#$ passt nicht.

Bausteine: Bewertung eines Schemas

- Angenommen wir haben eine Situation durch eine Schema b identifiziert.
- Alle Regeln, die zu b passen, sollen jetzt als Grundlage für neue dienen?
- Frage: Wie? bzw: Was wollen wir denn herbeizüchten?
- Heuristik: Wir nehmen die Stärken (Gewichte) als Indikatoren.
- Regeln sollen proportional zur Stärke einfließen.

- Frage: Welches Schema b sollen wir nun heranziehen, um neue (bessere) Regeln zu generieren?
- Reformuliert: Wann gehören denn Regeln zusammen, d.h. bilden einen Baustein?
- Idee: Wir bewerten ein Schema nach seiner durchschnittlichen Stärke $S(b)$.
- $S(b) =$ durchschnittliche Stärke der Regeln, die zu b passen.
- Ein Schema b ist gut, wenn seine Stärke $S(b)$ besser als die durchschnittliche Stärke aller Regeln A ist.

Bausteine: Bewertung eines Schemas



- Wir haben $n = 7$ Signale und 8 Regeln.
- Durchschnittliche Stärke
  $A = (1+0+2+0+2+2+0+1)/8 = 8/8 = 1$
- Auf das Schema $b_1 = 1******$ passen 3 Regeln.
- Wegen $S(b_1) = (1+0+1)/3 = 2/3$ ist dieses Schema unterdurchschnittlich.
- Auf das Schema $b_2 = *0**\#\#***$ passen 3 Regeln.
- Wegen $S(b_2) = (2+2+1)/3 = 5/3$ ist dieses Schema überdurchschnittlich.

Bausteine: Bewertung eines Schemas

Idee 1

- Wir berechnen die durchschnittliche Stärke A aller Regeln
- Wir betrachten alle Schemata b.
- Berechnen jeweils $S(b)$ und vergleichen mit A.
- Für Schemata b mit $S(b) > A$ generieren wir aus allen Regeln, die zu b passen, neue.

- Diese Idee ist leider idiotisch, denn:
- Die Anzahl der Schemata wächst exponentiell in der Signalanzahl n:
- An jeder Position können wir das Muster $*$ wählen oder einen festen Wert 0/1/#.
- Für $n = 100$ Signale haben wir 2^{100} Kandidaten für b zu testen.
- Wenn wir 1 Million Schemata pro Sekunde prüfen könnten, dann hätten wir beim Urknall anfangen müssen zu rechnen, um alle Kandidaten zu betrachten.

Genetische Algorithmen

Idee 2:

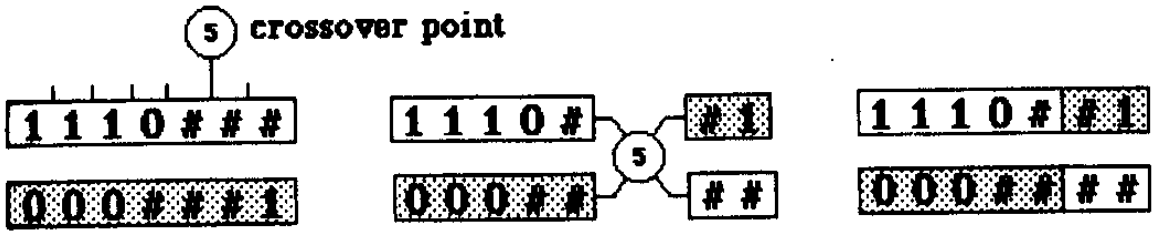
- John Holland schlägt stattdessen einen anderen Ansatz vor: genetische Algorithmen.
- Hier nur ein Appetithappen.
- Mehr zu Lernen in der VL *Intelligente Systeme*.

Wir betrachten die Regeln als Population.

- In jeder Runde „züchten“ wir aus der aktuellen Population Nachfahren.
- Regeln mit einer höheren **fitness** haben eine größere Reproduktionswkt. (= mehr Nachkommen).
- Die Nachkommen (=Regeln) enstehen durch **Rekombination** (cross-over).
- Die Nachfahren ersetzen dann die aktuelle Population zum Teil oder komplett.

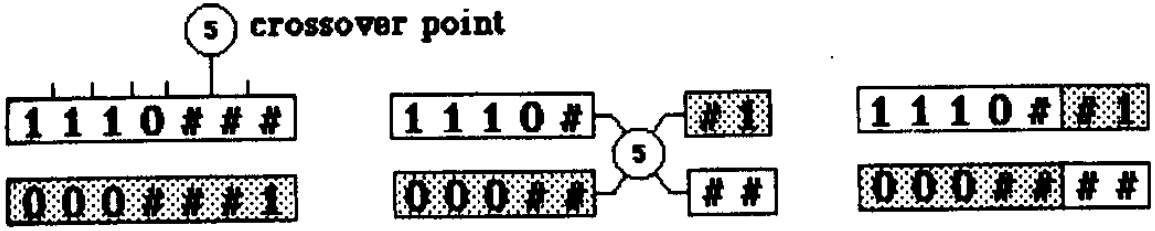
Genetische Algorithmen: Cross-Over

Beim cross-over werden zwei „Gene“ über Kreuz vererbt.

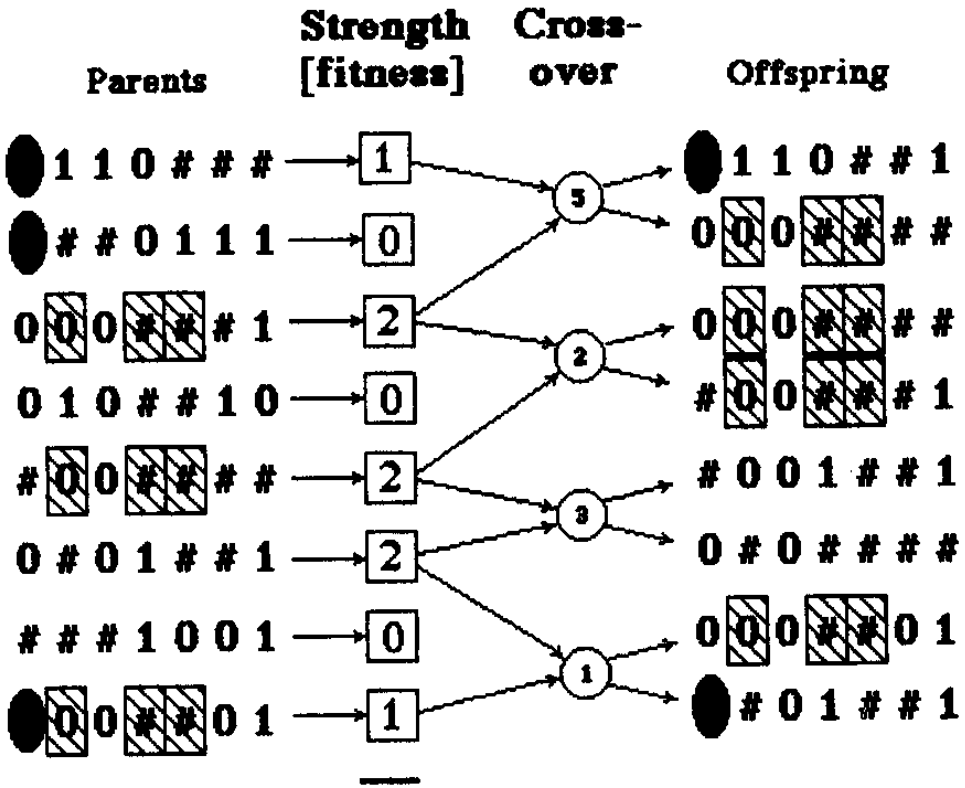


- Beim cross-over kommt es i.a. auf die Wahl des **cross-over point** an.
- Liegt der cross-over point in einem charakterisierenden Baustein, so wird der Trigger „zerrissen“.
- Baustein = der Abschnitt in einem Schema zwischen den * links und rechts.
- Bei Schemata mit kurzer Bausteinlänge ist ein solches Zerreißen aber eher unwahrscheinlich.
- Bei langen Bausteinen können wir darauf hoffen, dass sich diese i.w. aus kurzen Teilbausteinen zusammengesetzt haben.
- vgl. Beispiel: Fahren auf der Autobahn in einem roten Saab, wenn der Reifen platzt

Crossover Operator



Genetic Algorithm



Ave. Strength of All Indivs. = 1

Ave. Strength of Instances of ●****** = (1+0+1)/3 = 2/3

Ave. Strength of Instances of *0*##** = (2+2+1)/3 = 5/3

- Die Anzahl der über-durchschnittlichen Schemata-Regeln hat zugenommen.
- Genau: von 3 auf 4.
- Hingegen hat die Anzahl der unter-durchschnittlichen Schemata-Regeln abgenommen.
- Genau: von 3 auf 2.
- Dies ist eine direkte Folge von der Vermehrungsregel:
- Fitte Regeln haben mehr Kinder.

Genetische Algorithmen

Zusammenfassung

- Regeln mit einer höheren **fitness** haben eine größere Reproduktionswkt. (= mehr Nachkommen).
- Dadurch vermehren sich die Schemta b mit $S(b) > A$, ohne dass wir $S(b)$ zuvor berechnen müssten.
- Die Nachkommen (=Regeln) enstehen durch **Rekombination** (cross-over).
- Dadurch entstehen auch mal neue Regeln.
- Dies ist neu, denn zuvor wurden ja bewährte Regeln lediglich bestärkt. Dadurch können wir uns an wandelnde Umgebungen anpassen, aber nichts Neues erfinden.
- Genetische Programmierung erzeugt aus fitten Regeln durch Rekombination neue.

Literatur I

John H. Holland.
Hidden Order: How Adaptation builds complexity.
Hekix Books, 1995.