Machine Learning and Data Mining

8. Genetic Algorithms

Luc De Raedt

Virtually all slides taken from Tom Mitchell,
some material also from Milanie Mitchell's book

Contents

- Evolutionary computation
- Prototypical GA
- An example: GABIL
- Schema theorem
- Genetic Programming

Evolutionary Computation

- Computational procedures patterned after biological evolution
- Search procedure that probabilistically applies search operators to set of points in the search space

Biological Evolution

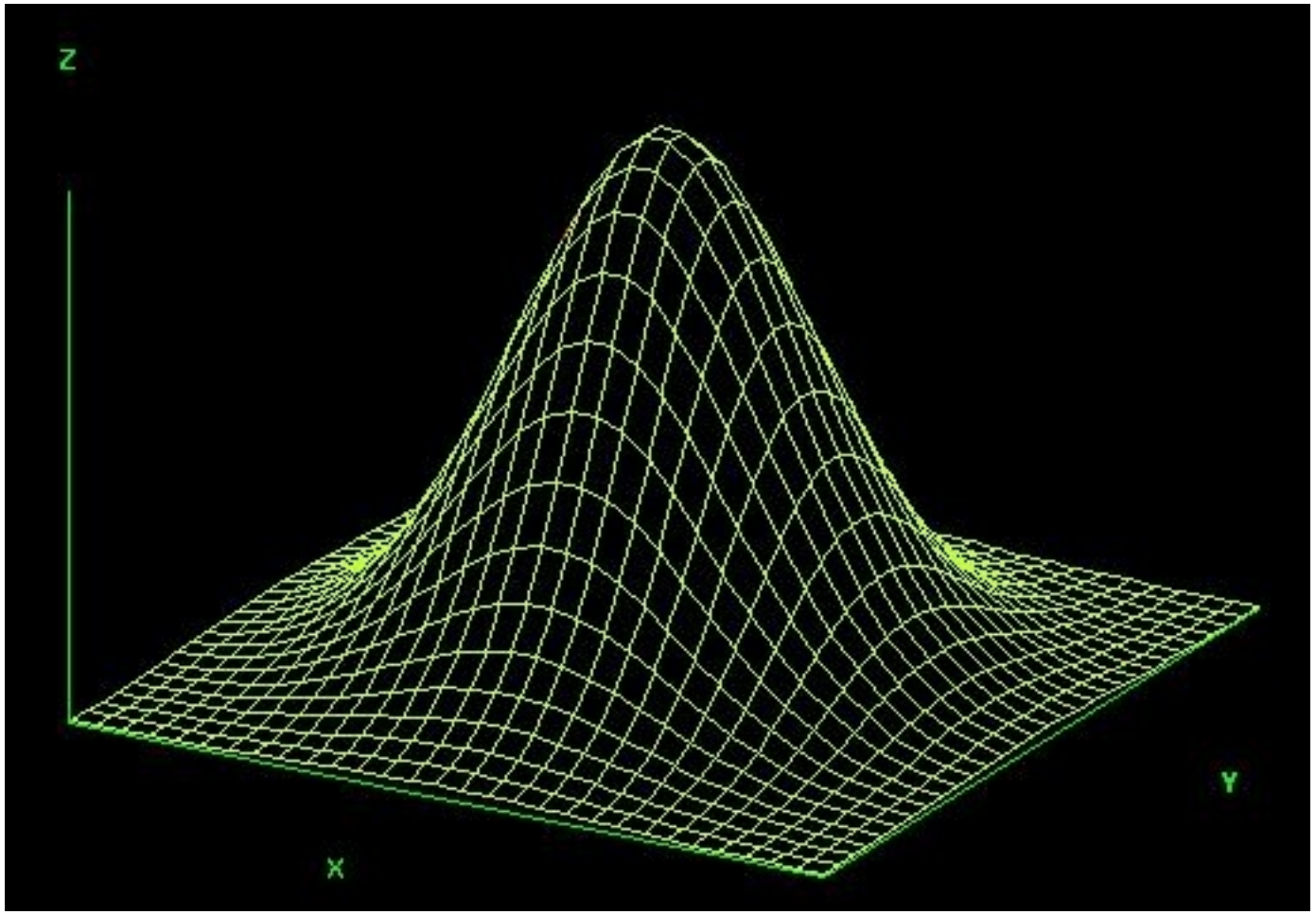
- Lamarck and others:
  - Species ``transmute'' over time
- Darwin and Wallace:
  - Consistent, heritable variation among individuals in population
  - Natural selection of the fittest
- Mendel and genetics:
  - A mechanism for inheriting traits
  - genotype to phenotype mapping

Appeal of Evolution

- Search through vast search spaces
- Parallelism
- Simple rules of random variation (mutation, recombination and others) and natural selection responsible for extraordinary variety and complexity

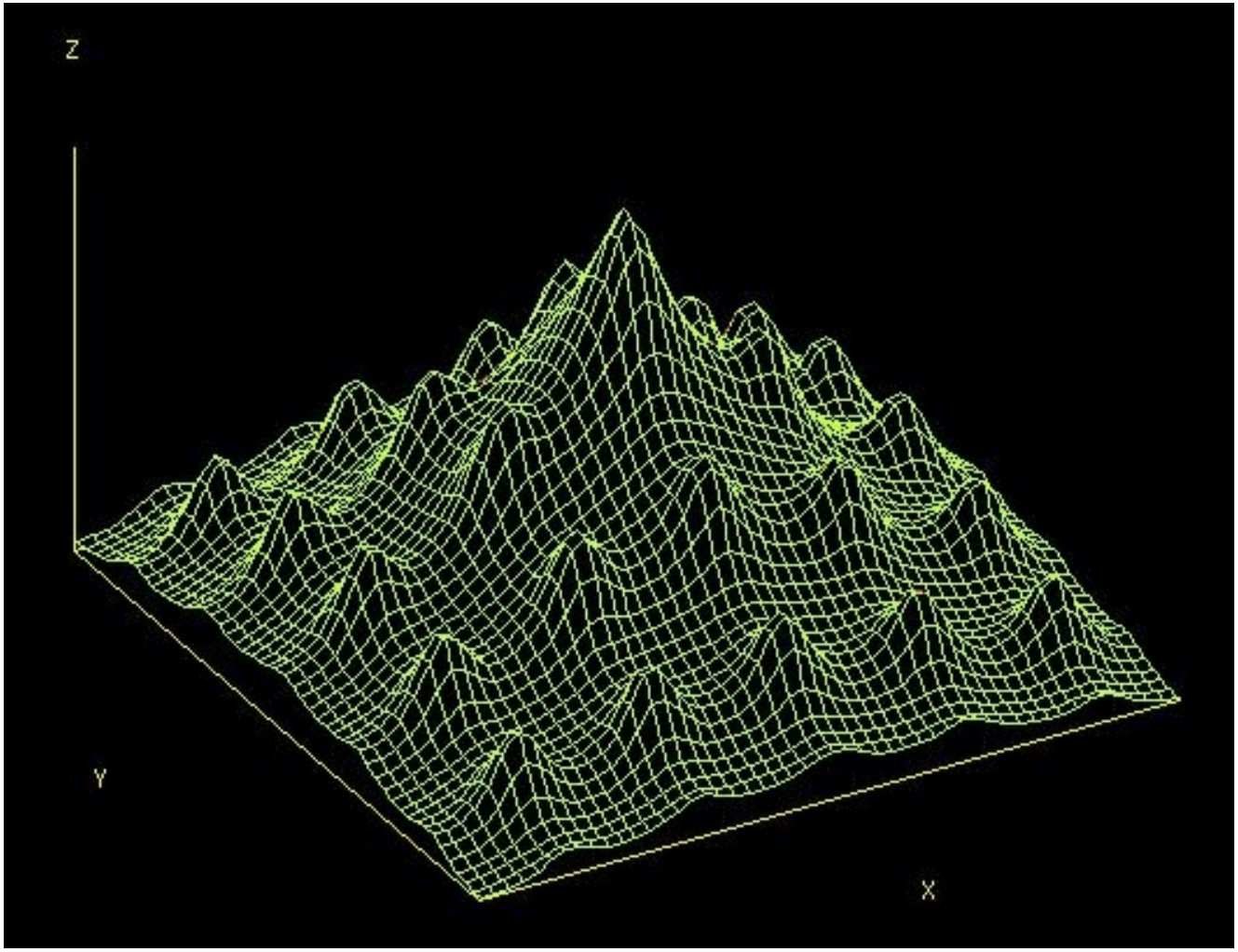
Fitness Landscapes:

Smooth Hills



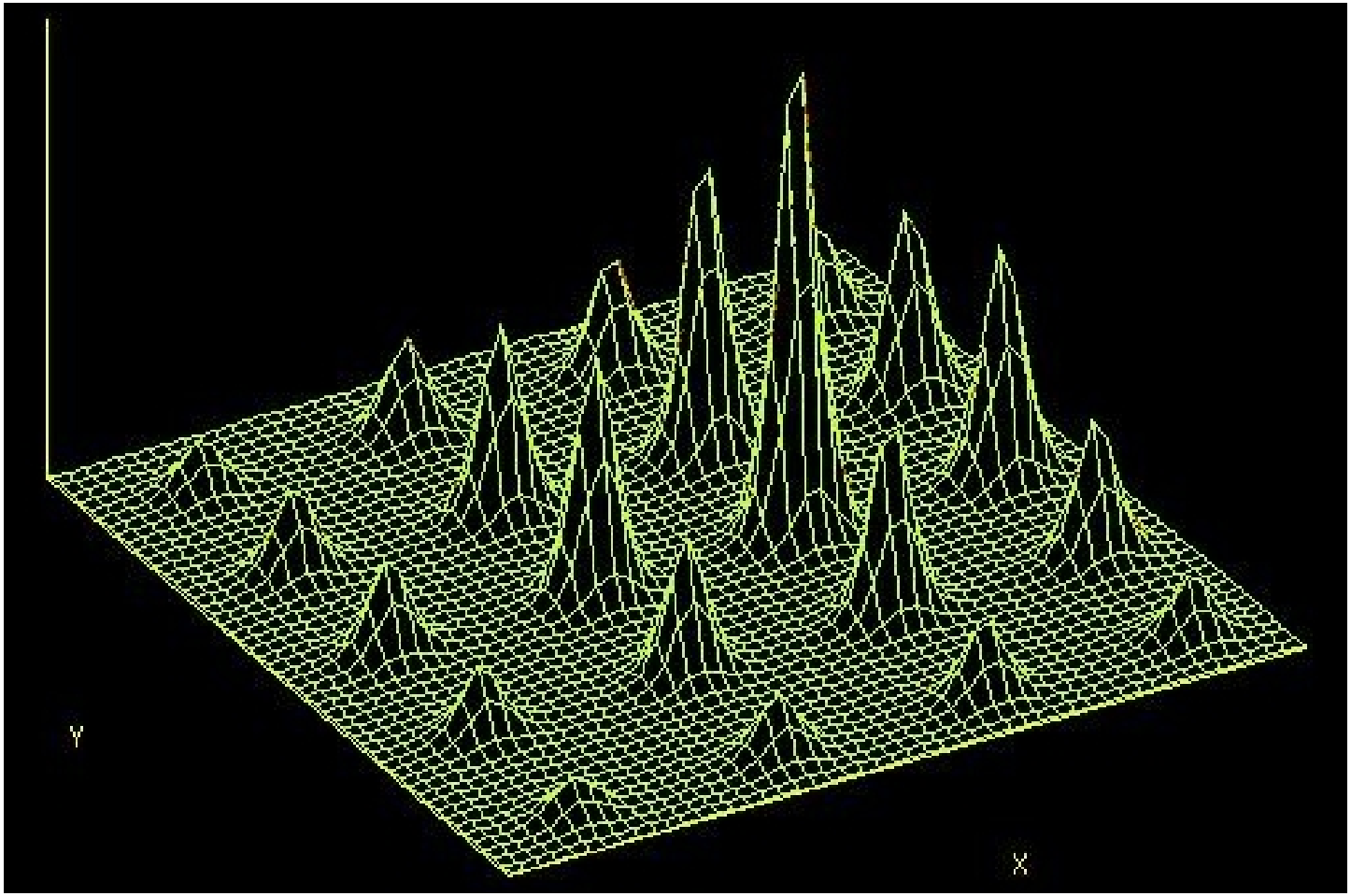
Fitness Landscapes:

Unimodal Hill, Fine Local Texture



Fitness Landscapes:

Coarse Local Structure



GA Operators

- *Selection*:
  chooses chromosomes in population for reproduction
- *Crossover*:
  randomly chooses locus; exchanges subsequences before and after locus
- *Mutation*:
  randomly flips some of the bits in a chromosome

GA$(Fitness, Fitness\_threshold, p, r, m)$

- *Initialize:* $P \leftarrow p$ random hypotheses
- *Evaluate:* for each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\_threshold$

  1. *Select:* Probabilistically select $(1 - r)p$ members of P to add to P_S.

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

  2. *Crossover:* Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P. For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add all offspring to P_s.
  3. *Mutate:* Invert a randomly selected bit in $m \cdot p$ random members of P_s
  4. *Update:* $P \leftarrow P_s$
  5. *Evaluate:* for each h in P, compute $Fitness(h)$

- Return the hypothesis from P that has the highest fitness.

Representing hypotheses

Represent

$$(Outlook = Overcast \vee Rain) \wedge (Wind = Strong)$$

by

$Outlook$	$Wind$
011	10

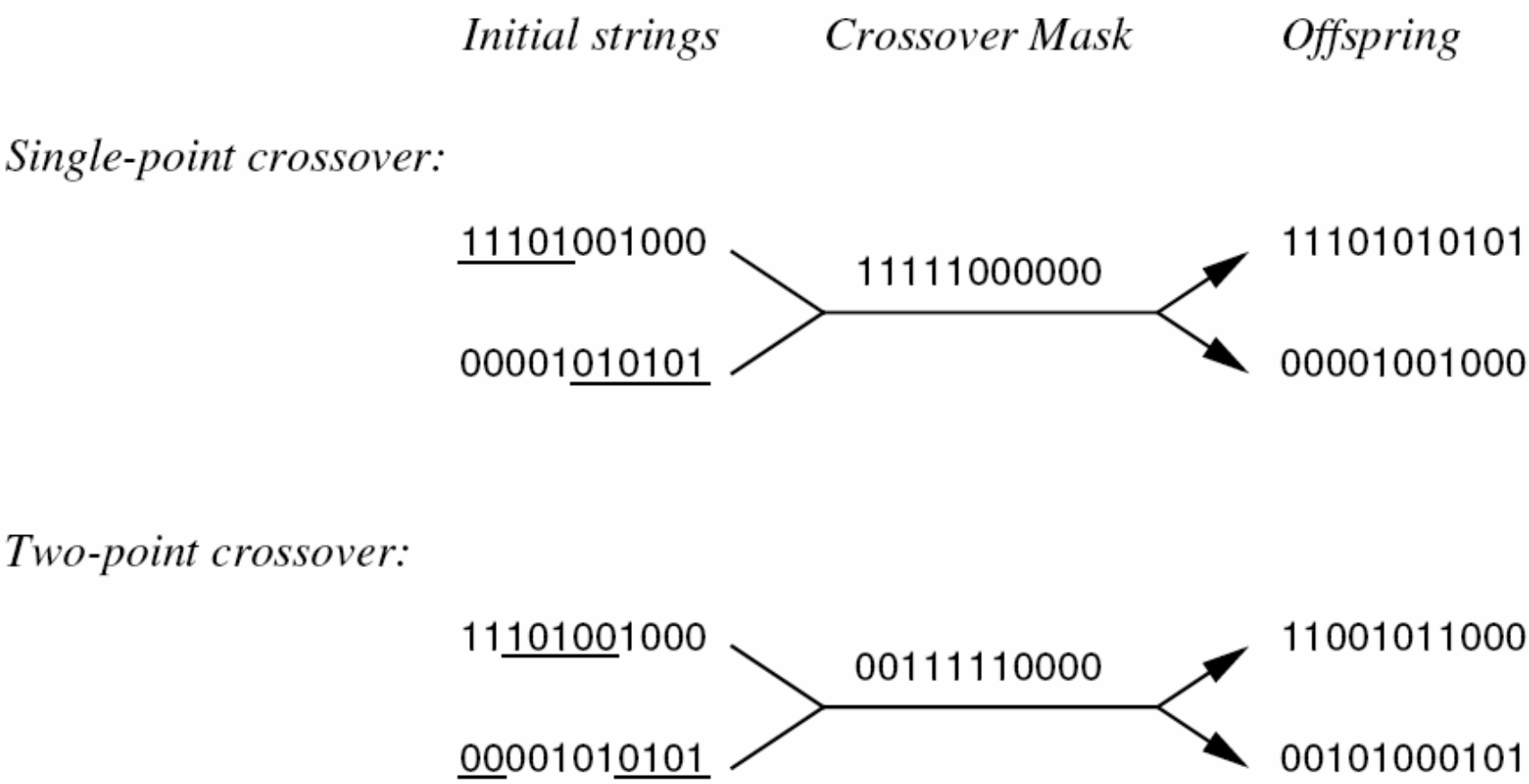
Represent

$$\text{IF} \quad Wind = Strong \quad \text{THEN} \quad PlayTennis = yes$$

by

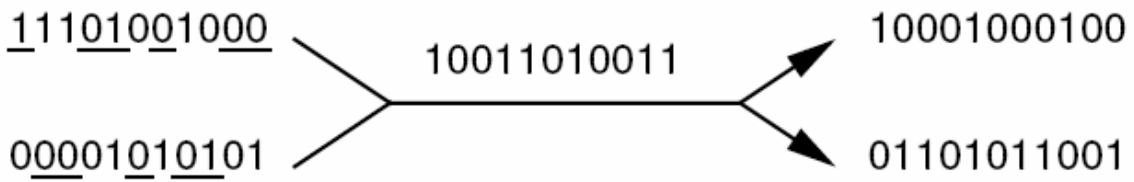
$Outlook$	$Wind$	$PlayTennis$
111	10	10

Operators for Genetic Algorithms



Operators for Genetic Algorithms

Uniform crossover:



Point mutation:

Selecting Most Fit Hypotheses

- Fitness proportionate selection:

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

  ... can lead to *crowding*

- Tournament selection:
  - Pick h1 and h2 at random with uniform prob.
  - With probability p select the more fit.
- Rank selection:
  - Sort all hypotheses by fitness
  - Prob of selection is proportional to rank

GABIL (De Jong et al. 93)

- Learn disjunctive set of propositional rules, competitive with C4.5

Fitness:

$$Fitness(h) = (correct(h))^2$$

Representation:

$$\text{IF } a_1 = T \wedge a_2 = F \text{ THEN } c = T; \text{ IF } a_2 = T \text{ THEN } c = F$$

represented by

a_1	a_2	c		a_1	a_2	c
10	01	1		11	10	0

Genetic operators

- want variable length rule sets
- want only well-formed bitstring hypotheses

Crossover with Variable-Length Bitstrings

Start with

	a_1	a_2	c		a_1	a_2	c
h_1 :	10	01	1		11	10	0
h_2 :	01	11	0		10	01	0

1. choose crossover points for h_1, e.g., after bits 1, 8
2. now restrict points in h_2 to those that produce bitstrings with well-defined semantics, e.g., $\langle 1, 3\rangle$, $\langle 1, 8\rangle$, $\langle 6, 8\rangle$.

if we choose $\langle 1, 3\rangle$, result is

	a_1	a_2	c
h_3 :	11	10	0

	a_1	a_2	c		a_1	a_2	c		a_1	a_2	c
h_4 :	00	01	1		11	11	0		10	01	0

GABIL Extensions

- Add new genetic operators, also applied probabilistically:
  - *AddAlternative*: generalize constraint on a_i by changing a 0 to 1
  - *DropCondition*: generalize constraint on a_i by changing every 0 to 1
- And, add new field to bitstring to determine whether to allow these

a_1	a_2	c	a_1	a_2	c	AA	DC
01	11	0	10	01	0	1	0

- So now the learning strategy also evolves!

GABIL Results

- Performance of GABIL comparable to symbolic rule/tree learning methods C4.5, ID5R, AQ14
- Average performance on a set of 12 synthetic problems:
  - GABIL without AA and DC operators: 92.1% accuracy
  - GABIL with AA and DC operators: 95.2% accuracy
  - symbolic learning methods ranged from 91.2 to 96.6

Another Example

- The Traveling Salesman problem
  - Given
    - N cities and their distances
  - Find
    - the shortest path that connects them all
- How to represent hypotheses ?
  - a sequence of cities ?
  - ABCD ?

Genetic operators ?

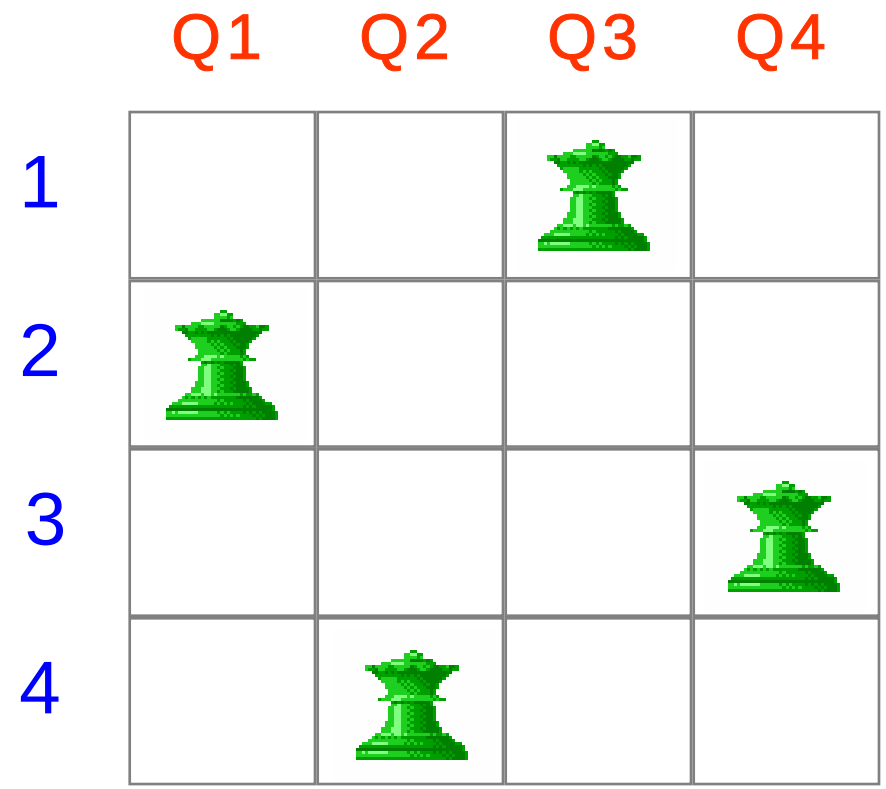
- Cross-over
  - 135 | 26478 yields 135 | 54321
  - 876 | 54321 876 | 26478
- Better ?
  - 135 | 26478 yields 135 | 42876
  - 876 | 54321 876 | 24135
- Mutation
  - 13526478 gives 13726458
- There is nothing “sacred” about Gen. Operators

n-Queens

Place 4 queens on a 4 x 4 chessboard so that none can take another.

Four variables Q1, Q2, Q3, Q4 representing the row of the queen in each column. Domain of each variable is {1,2,3,4}

One solution! -->



The 8 Queens Problem ?

- Similar Encoding and Operators
  - Order matters
- Fitness ?
  - Of a single queen : - number of queens it attacks
  - Of a configuration : sum of fitness queens
- Selection
  - Roulette-Wheel

Schema Theorem

- How to characterize evolution of population in GA?
- Schema = string containing 0, 1, * (``don't care'')
  - Typical schema: 10**0*
  - Instances of above schema: 101101, 100000, ...
- Characterize population by number of instances representing each possible schema
  - m(s,t) = number of instances of schema s in pop at time t

Consider only Selection

- $\bar{f}(t) =$ average fitness of pop. at time t
- $m(s,t) =$ instances of schema s in pop at time t
- $\hat{u}(s,t) =$ ave. fitness of instances of s at time t

Probability of selecting h in one selection step

$$\Pr(h) = \frac{f(h)}{\sum_{i=1}^{n} f(h_i)} = \frac{f(h)}{n\bar{f}(t)}$$

Probabilty of selecting an instance of s in one step

$$\Pr(h \in s) = \sum_{h \in s \cap p_t} \frac{f(h)}{n\bar{f}(t)} = \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t)$$

Expected number of instances of s after n selections

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

Schema Theorem

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t) \left(1 - p_c \frac{d(s)}{l-1}\right) (1-p_m)^{o(s)}$$

- $m(s,t) =$ instances of schema s in pop at time t
- $\bar{f}(t) =$ average fitness of pop. at time t
- $\hat{u}(s,t) =$ ave. fitness of instances of s at time t
- $p_c =$ probability of single point crossover operator
- $p_m =$ probability of mutation operator
- $l =$ length of single bit strings
- $o(s)$ number of defined (non "*") bits in s
- $d(s) =$ distance between leftmost, rightmost defined bits in s

Genetic Programming (Koza, 1992):
Evolving Lisp Programs

- GAs to produce computer programs
- Example:
  - program computing orbital period of planet (Kepler‘s third law: $P^2 = cA^3$)

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

The square of the sidereal period of an orbiting planet is directly proportional to the cube of the orbit's semimajor axis.

Parse Tree for Expression

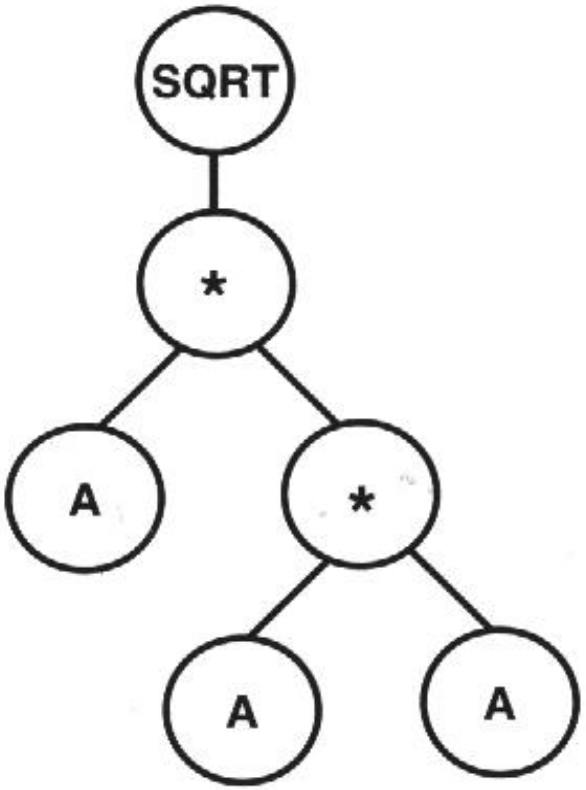


Figure 2.1 Parse tree for the Lisp expression (SQRT (∗ A (∗ A A))).

Koza‘s Algorithm

1. Choose set of functions and terminals
2. Generate initial population of random trees
3. Calculate fitness by running programs on „fitness cases“
4. Apply selection, cross-over, mutation to form a new population
5. Go to step 3

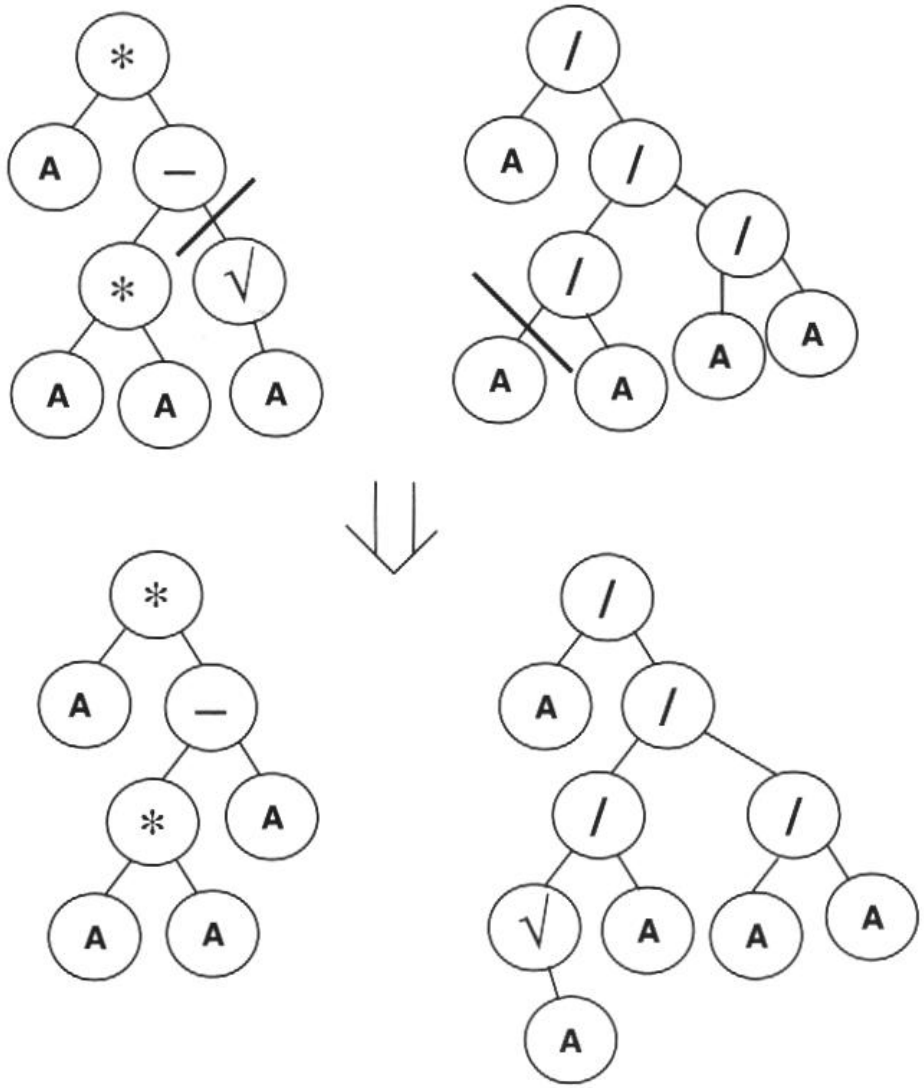
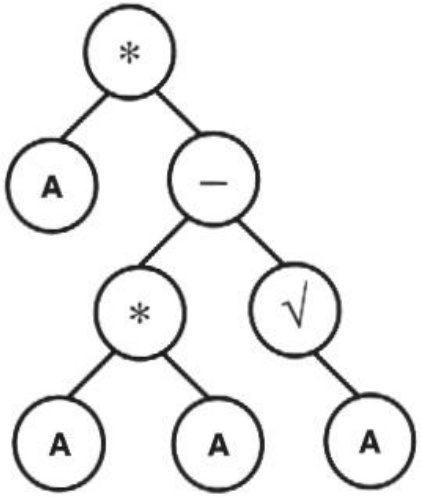
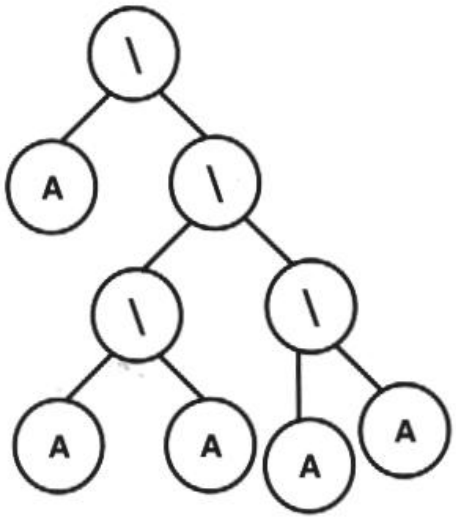


Figure 2.3 An example of crossover in the genetic programming algorithm. The two parents are shown at the top of the figure, the two offspring below. The crossover points are indicated by slashes in the parent trees.



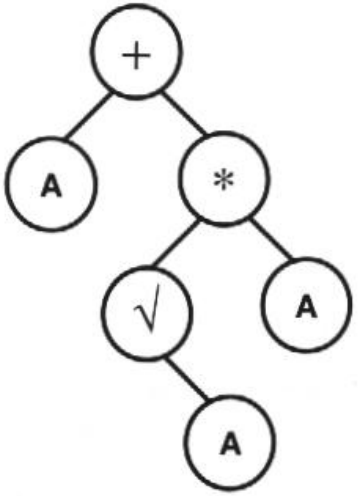
A ∗ [(A ∗ A) – √ A]

f = 1



A \ [(A \ A) \ (A \ A)]

f = 3



A + (√ A ∗ A)

f = 0

Fitness Cases:

Planet	A	Correct Output (P)
Venus	0.72	0.61
Earth	1.00	1.00
Mars	1.52	1.84
Jupiter	5.20	11.9
Saturn	9.53	29.4
Uranus	19.1	83.5

Block-Stacking Problem

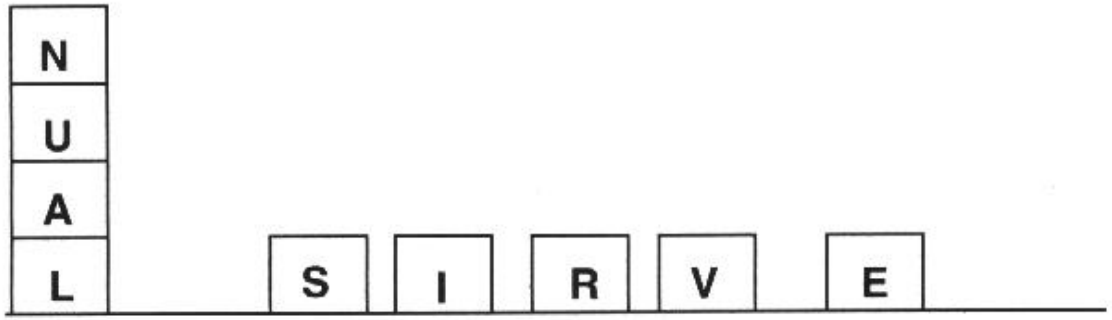


Figure 2.4 One initial state for the block-stacking problem (adapted from Koza 1992). The goal is to find a plan that will stack the blocks correctly (spelling "universal") from any initial state.

Block-Stacking Problem

- Terminals:
  - `CS` („current stack“)
  - `TB` („top correct block“)
  - `NN` („next needed“)
- Functions:
  - `MS(`*`x`*`)` („move to stack“)
  - `MT(`*`x`*`)` („move to table“)
  - `DU(`*`expression1, expression2`*`)` („do until“)
  - `NOT(`*`expression`*`)`
  - `EQ(`*`expression1, expression2`*`)`

Block-Stacking Problem

- Generation 1:
  `(EQ (MS NN) (EQ (MS NN)(MS NN)))`
- Generation 5:
  `(DU (MS NN) (NOT NN))`
- Generation 10:
  `(EQ (DU (MT CS) (NOT CS)) (DU (MS NN) (NOT NN)))`

Genetic Programming:
Discussion

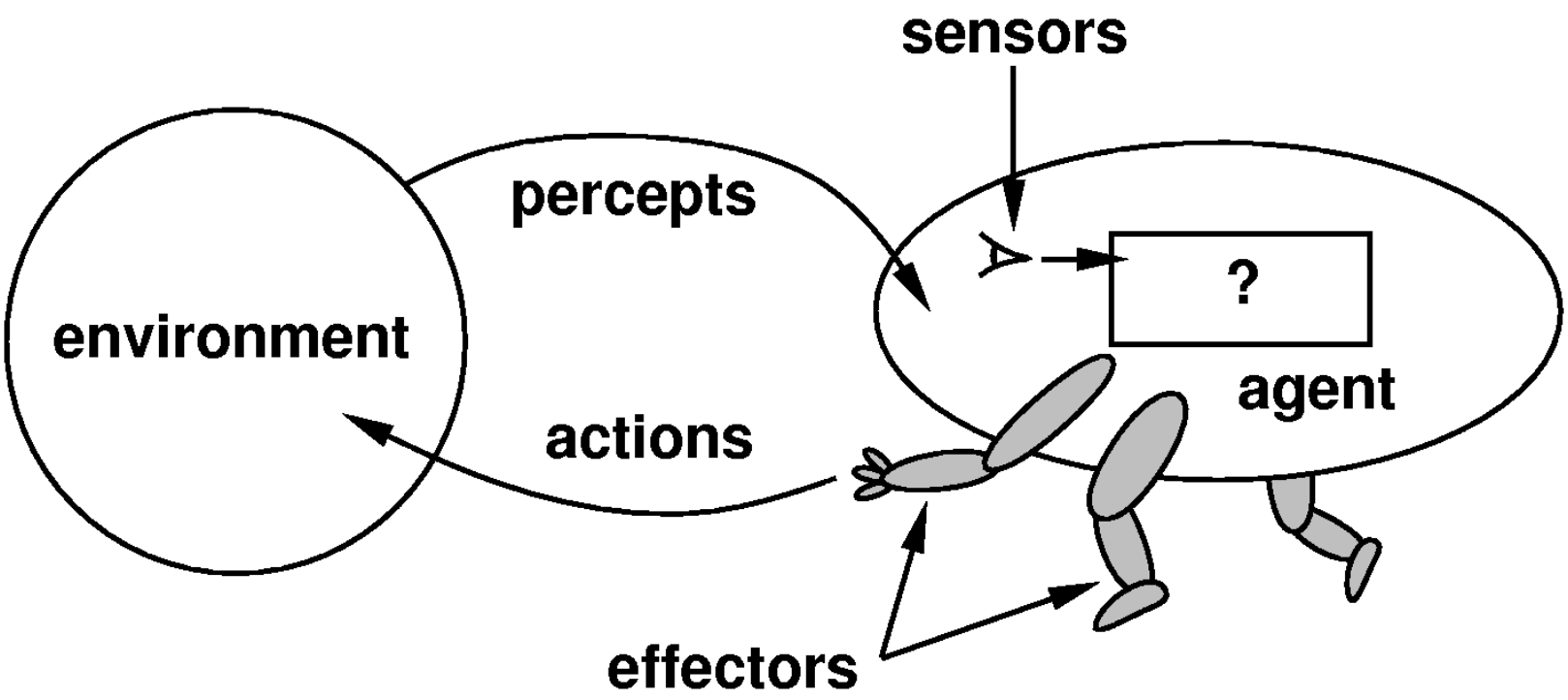
- Block-stacking: simple domain, high-level terminals and functions
- Additional feature: encapsulation (chunking) of useful subtrees
- Density of solutions or useful building blocks in Lisp expressions?
- Comparison with other methods?
- Scalability?
- Generalization performance?

Classifier Systems

- Mixing ideas from
  - Genetic Algorithms
  - Rule based Systems
- An early form of reinforcement learning
- Now connected to Artificial Life

- Source:
  - The Hitch-Hiker's Guide to Evolutionary Computation, Q.1.4
  - Better: John Holland, Escaping Brittleness, in Machine Learning: An AI Approach, 1986

Agent Architecture



Control

- Rules = Classifier
  - IF condition THEN action
- Conditions and Actions are strings over {0,1,#}
- Message passing:
  - inputs to system are messages (bitstrings)
  - actions are messages (bitstrings)
  - conditions match current message
- Production system
  - quite powerful, cf. Post Correspondence Problem from Theoretical Computer Science

Kermit

IF small, flying object to the left THEN send @
IF small, flying object to the right THEN send %
IF small, flying object centered THEN send $
IF large, looming object THEN send !
IF no large, looming object THEN send *
IF * and @ THEN move head 15 degrees left
IF * and % THEN move head 15 degrees right
IF * and $ THEN move in direction head pointing
IF ! THEN move rapidly away from direction head pointing

Kermit in CS

IF	THEN	conditions
0000,	00 00 00 00	0000 small object
0000,	00 01 00 01	1111 large
0000,	00 10 00 10	00 flying object
1111,	01 ## 11 11	01 looming object
~1111,	01 ## 10 00	00 left
1000,	00 00 01 00	01 right
1000,	00 01 01 01	10 center
1000,	00 10 01 10	~ not
1111,	## ## 01 11	# don’t care

Kermit the Classifier System

IF	THEN	actions
0000,	00 00 00 00	0000 @
0000,	00 01 00 01	0001 %
0000,	00 10 00 10	0010 $
1111,	01 ## 11 11	1111 ! (danger)
~1111,	01 ## 10 00	1000 * (safe)
1000,	00 00 01 00	0100 (move left)
1000,	00 01 01 01	0101 (move right)
1000,	00 10 01 10	0110 (move ahead)
1111,	## ## 01 11	0111 (move away)

Classifier System Algorithm (no learning)

- t := 0;
- initMessageList ML (t);
- initClassifierPopulation P (t); (Random)
- while not done (test for fitness) do
  - t := t + 1;
  - ML := readDetectors (t);
  - ML' := matchClassifiers ML,P (t);
  - ML := sendEffectors ML' (t);

Learning CFS

- t := 0;
- ML (t);
- initClassifierPopulation P (t);
- while not done do
  - t := t + 1;ML := readDetectors (t);
  - ML' :=matchClassifiers ML,P (t);
  - ML' := selectMatchingClassifiers ML',P (t);
  - ML' :=taxPostingClassifiers ML',P (t);
  - ML := sendEffectors ML' (t);)
  - C := receivePayoff (t);
  - P' :=distributeCredit C,P (t);
  - At some points in time
    - P := generateNewRules P'(t);.

Idea of Reinforcement Learning

- Bucket-Brigade
- Classifiers have strength (fitness)
- Complex Bidding System
  - before classifier is allowed to post message
    - bidding
  - highest bidder wins but has to pay with fitness
  - reinforcement received from environment is divided across successful bidders
  - fitness goes down with time

Genetic Algorithm

- Crossover among rules
  - the higher strength/fitness the more likely it is of being selected
  - the lower the strength/fitness the more likely the classifier is replaced
- Many variants, ideas, …
- We: only the very basics

Evolutionary Programming

- Conduct randomized, parallel, hill-climbing search through H
- Approach learning as optimization problem (optimize fitness)
- Nice feature: evaluation of Fitness can be very indirect