Evolutionary Algorithms

Part 3

- *Sources*

Martijn Schut, schut@cs.vu.nl,

Jaap Hofstede, Beasly, Bull, Martin

Andrea G. B. Tettamanzi

Joost N. Kok and Richard Spillman

KMT Software, Inc. -- http://www.kmt.com

Dan Kiely
Ran Shoham
Brent Heigold

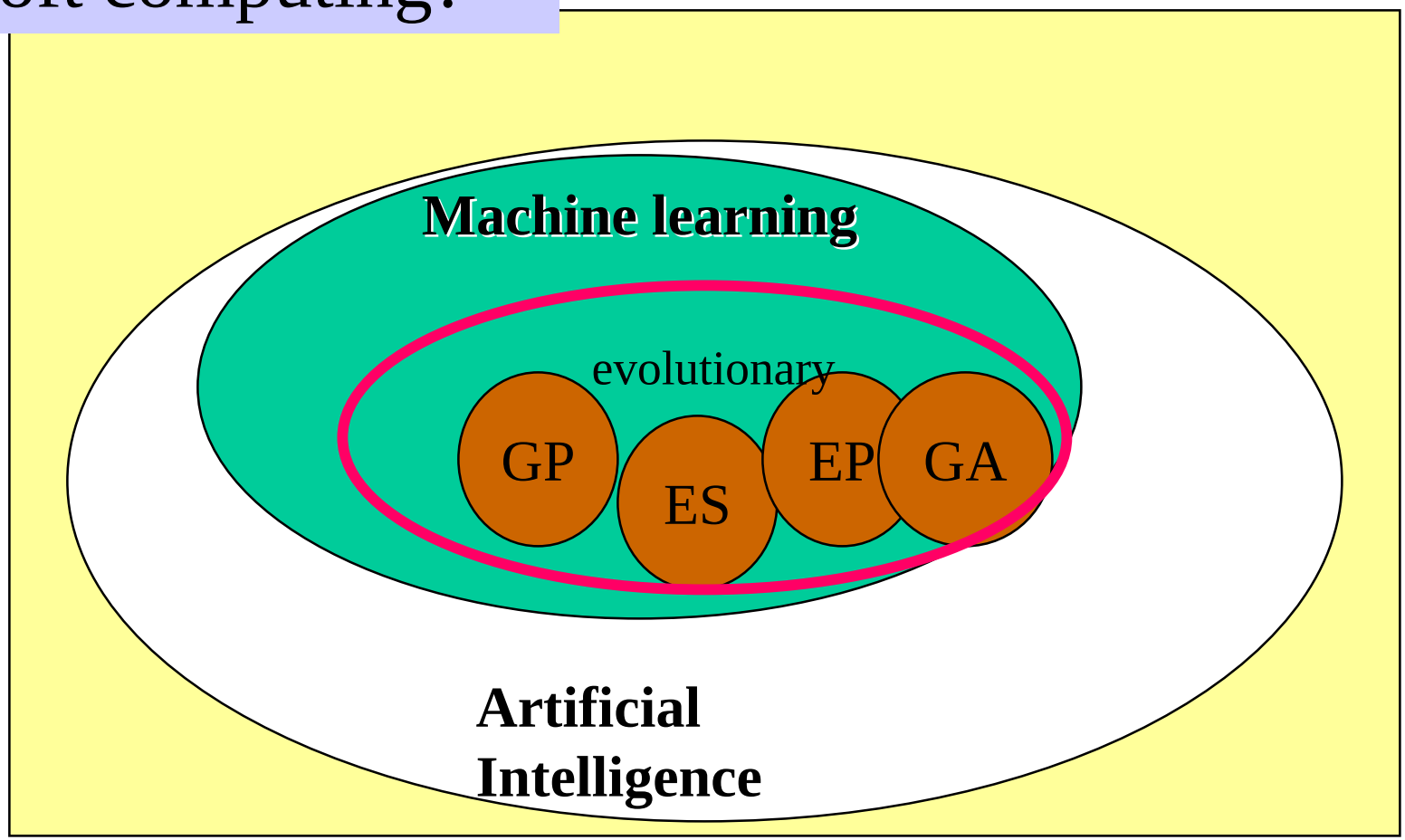
Genetic Programming

OUTLINE

- Genetic Programming Structure
- Additional GP Operations
- Hardware Evolution

What is Genetic Programming(GP)?

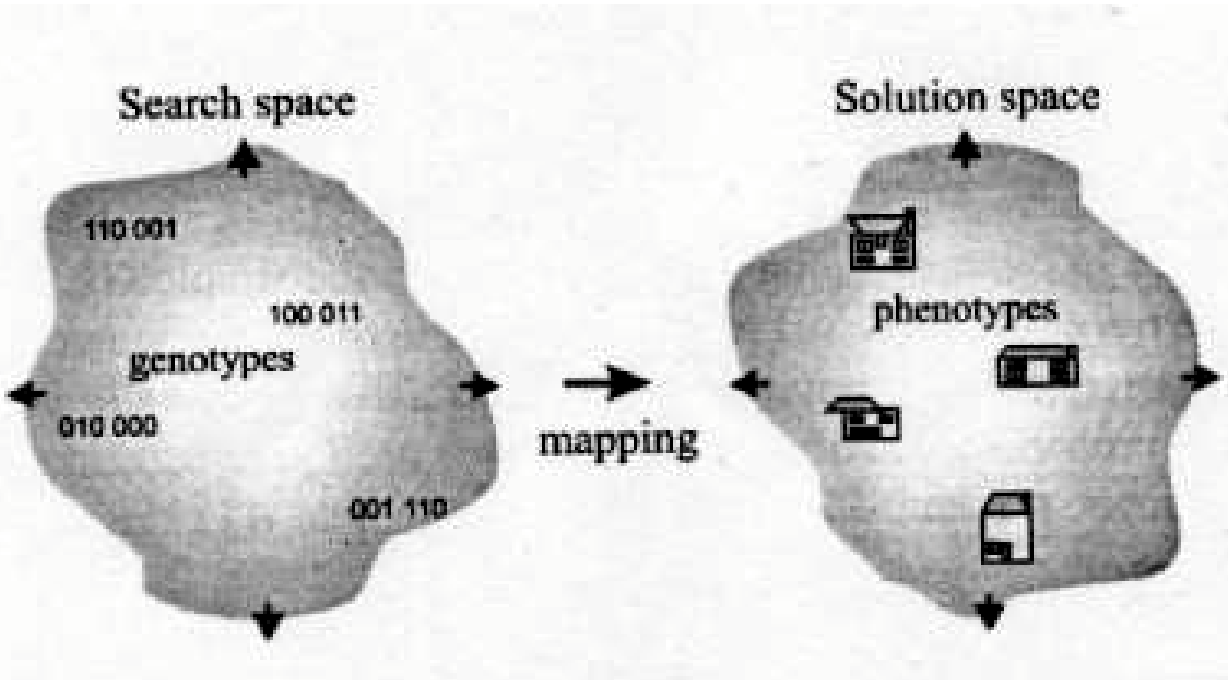
Soft computing?



Genetic Algorithms-review

- Most widely used
- **Robust**
- uses 2 separate spaces
  - search space - coded solution (genotype)
  - solution space - actual solutions (phenotypes)

Genotypes must be mapped to phenotypes before the quality or fitness of each solution can be evaluated



New Idea -Evolutionary Strategies (ES)

- ES are like GP -- ***no distinction*** between search and solution space
- Individuals are represented as **real-valued** vectors.
- Simple ES
  - one parent and one child
  - Child solution generated by randomly mutating the problem parameters of the parent.
- Susceptible to **stagnation** at local optima

Evolutionary Strategies (cont'd)

- Slow to converge to optimal solution
- More advanced ES
  - have pools of parents and children
- Unlike GA and GP, ES
  - Separates parent individuals from child individuals
  - Selects its parent solutions deterministically

Evolutionary Programming

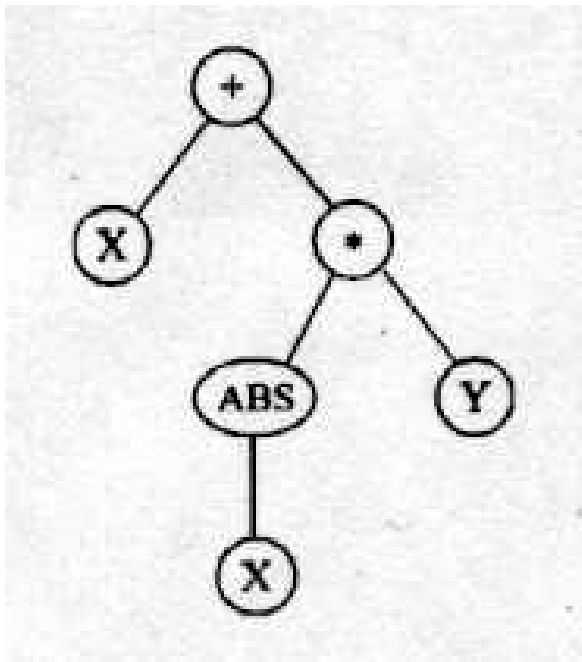
- Resembles ES, developed independently
- Early versions of EP applied to the evolution of transition table of finite state machines
- One population of solutions, reproduction is by mutation only
- Like ES operates on the decision variable of the problem directly (ie **Genotype = Phenotype**)
- Tournament selection of parents
  - better fitness more likely a parent
  - children generated until population doubled in size
  - everyone evaluated and the half of population with lowest fitness deleted.

GP Agenda

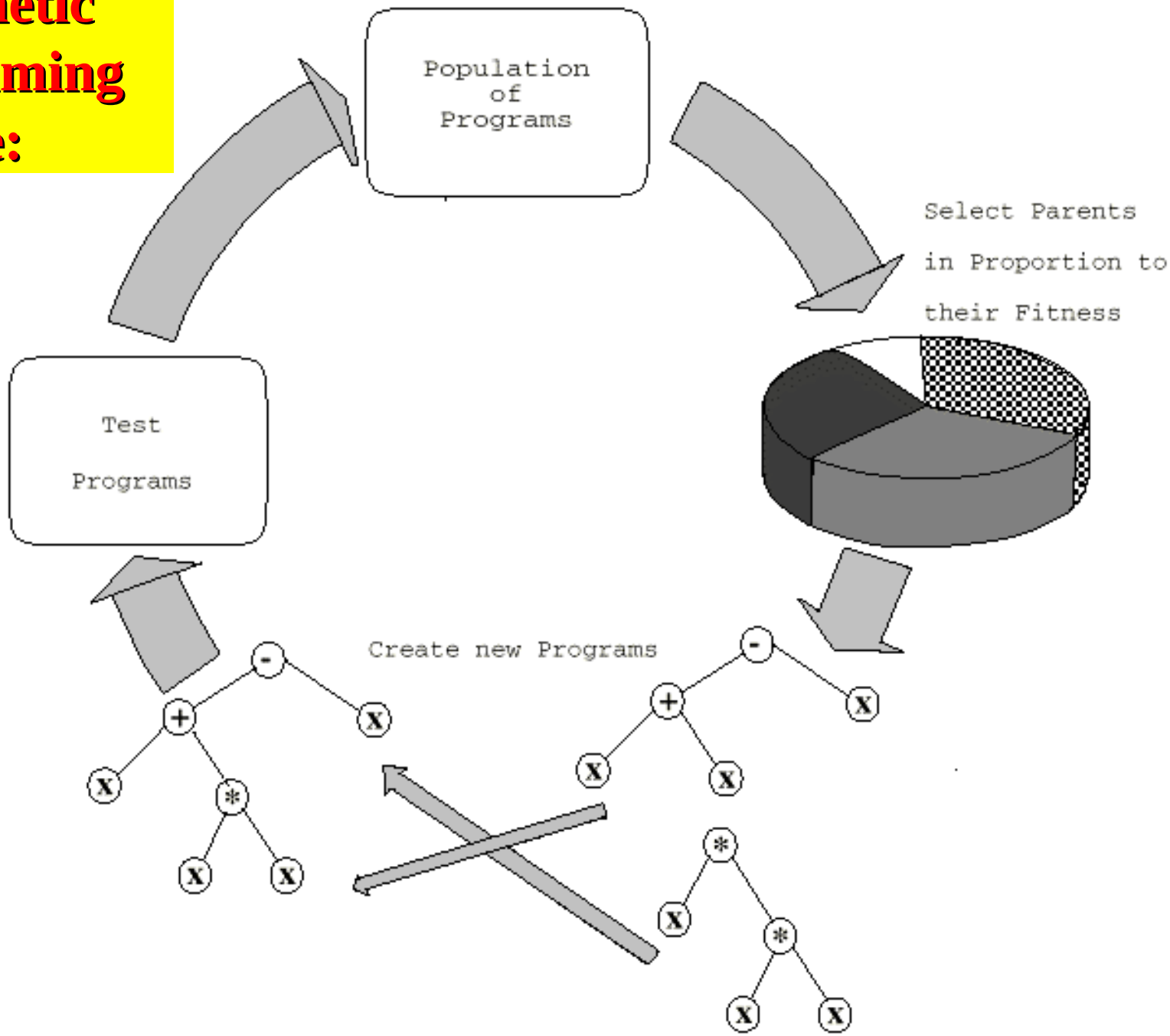
- What is Genetic Programming?
- Background/History.
- Why Genetic Programming?
- How Genetic Principles are Applied.
- Examples of Genetic Programs.
- Future of Genetic Programming.

Genetic Programming

- **Specialized (or, generalized?) form of GA**
- **Manipulates a very specific type of solution using modified genetic operators**
- **Original application was to design computer programs**
- **Now applied in alternative areas such as Analog Circuits**
- **Does not make distinction between search and solution space.**
- **Solution represented in very specific hierarchical manner.**



The genetic programming cycle:



Background/History

- By John R. Koza, Stanford University.
- 1992, Genetic Programming Treatise - “Genetic Programming. On the Programming of Computers by Means of Natural Selection.” - *Origin of GP*.
- Combining the idea of machine learning and evolved tree structures.

Why Genetic Programming?

- It saves time by freeing the human from having to design complex algorithms.
- Not only designing the algorithms but creating ones that give optimal solutions.
- Again, link to Artificial Intelligence, but also systematic design, CAD, optimization, future computing.

What Constitutes a Genetic Program?

- Starts with "What needs to be done"
- Agent figures out "How to do it"
- Produces a computer program - “Breeding Programs”
- Fitness Test
- Code reuse
- Architecture Design - Hierarchies
- Produce results that are competitive with human produced results (well, sometimes)

How are Genetic Principles Applied?

- “Breeding” computer programs.
- Crossovers.
- Mutations.
- Fitness testing.

"Breeding" Computer Programs

- Start off with a large "pool" of random computer programs.
- Need a way of coming up with the best solution to the problem using the programs in the "pool"
- Based on the definition of the problem and criteria specified in the fitness test, mutations and crossovers are used to come up with new programs which will solve the problem.

Genetic Programming Structure

- Given the graph representation, Genetic Programming becomes a straightforward implementation of a Genetic Algorithm

- A GP system must have:

Initial Population Generation Method

Parent Selection Method

CrossOver and Mutation Operators

Initial Population

- The initial population for a GP run consists of a randomly generated set of "rooted, point-labeled trees with ordered branches"

- Begin by selecting one of the functions from the set of functions (F) to be labeled as a root

  the number of branches from the root is given by the number of arguments required by the selected function

  An element is randomly selected from the combined function and terminal set (F and T) and placed on each of the branches

  if a function is selected, branches are created for its arguments

Example

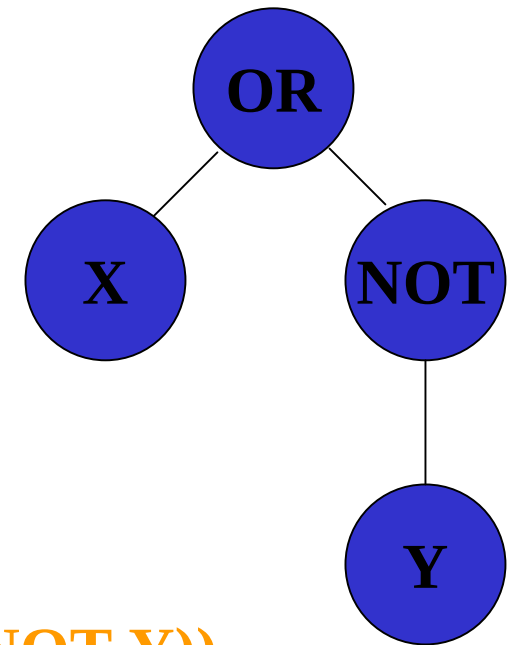
- Let F = {AND,OR,NOT} and T = {X, Y}

Randomly select OR

Randomly select X

Randomly select NOT

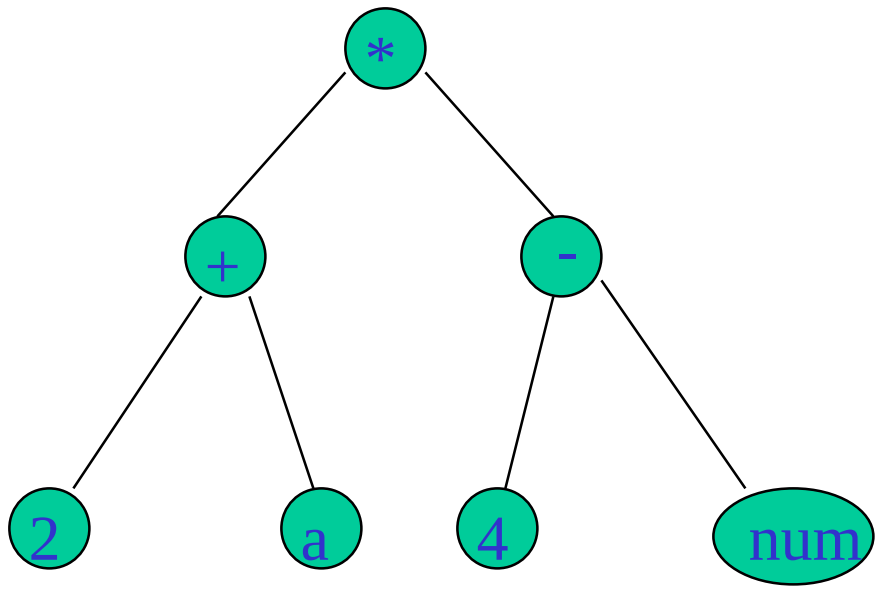
Randomly select Y



(OR X (NOT Y))

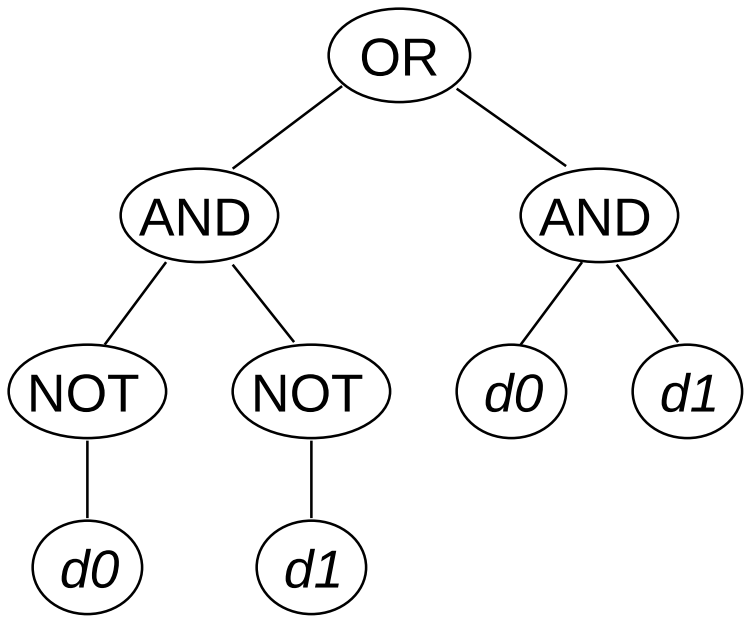
Computer Programs as Trees

- Infix/Postfix
- (2 + a)*(4 - num)



Genetic Programming: The Individuals

subset of LISP S-expressions



(OR (AND (NOT *d0*) (NOT *d1*)) (AND *d0 d1*))

Generative Method

- **GOAL:** Start with an initial mixture of trees (programs) of various sizes and shapes

The "full" generation method creates a population of trees in which the length of every path is equal to a specified maximum depth

PROCESS: Restrict the selection of nodes at depths less than the maximum to the function set and select only terminal nodes at the maximum depth

DEFINITION: The depth of a tree is the length of the longest path from the root to an endpoint

Growth Method

- Create an initial population consisting of trees with a variable depth up to some maximum

PROCESS: **Allow both functions and terminals for all nodes below the maximum depth and only terminals for nodes at the maximum depth**

Best Method

- The best method combines these two approaches

a maximum depth is specified (say 2 to 6)

PROCESS: an equal number of trees at each maximum depth is constructed - half using the full method and half using the grow method

Mutations in Nature

Properties of mutations

- Ultimate source of genetic variation.
- **Radiation, chemicals** change genetic information.
- Causes new genes to be created.
- One chromosome.
- Asexual.
- Very rare.

Before:

acgtactggctaa

After:

acatactggctaa

Mutations in Programs

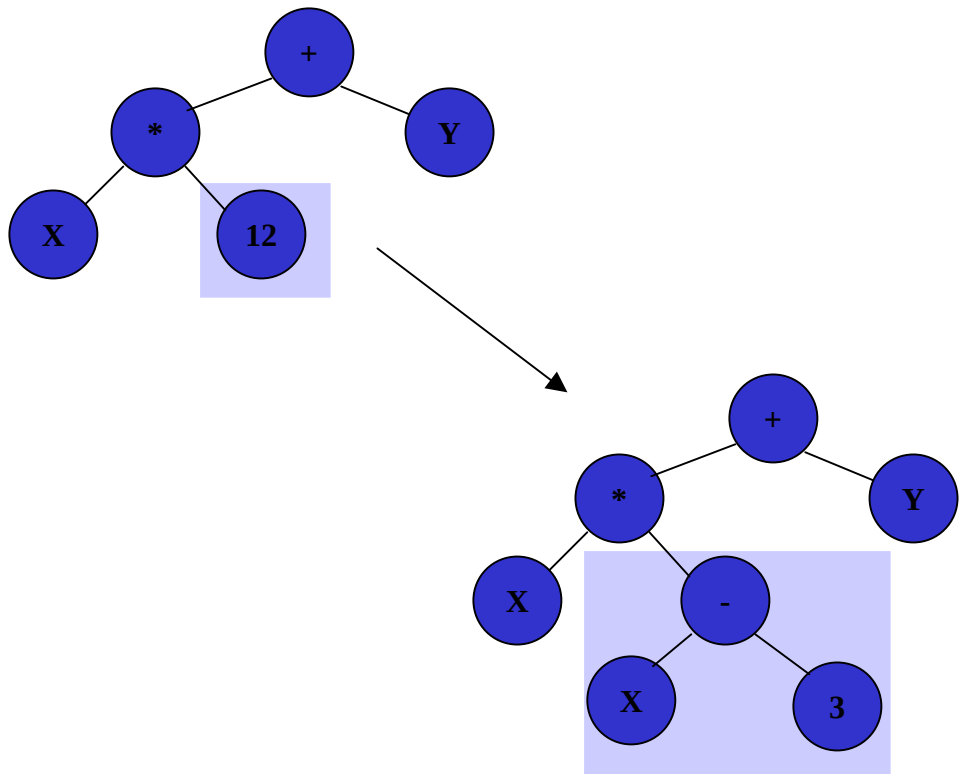
- Single parental program is ***probabilistically selected*** from the population based on fitness.
- **Mutation** point randomly chosen.
  - the subtree rooted at that point is deleted, and
  - a **new subtree is grown** there using the same random growth process that was used to generate the initial population.
- **Asexual operations** (mutation) are typically performed sparingly:
  - with a *low probability* of,
  - probabilistically selected from the population based on fitness.

Mutation inPrograms

- Determine if a population element will be subjected to a mutation
- Select a random point in the program tree
- Remove the subtree at that point
- Insert a randomly generated subtree at that point

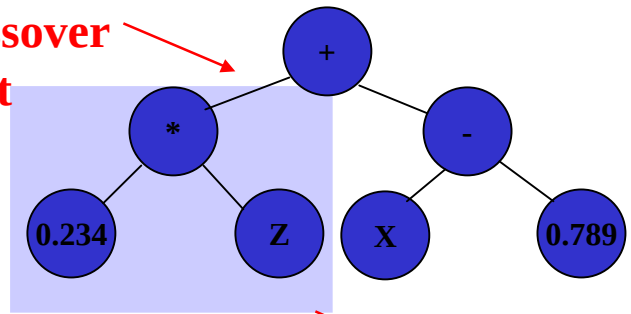
Example of Mutation in GP

- Apply mutation to:

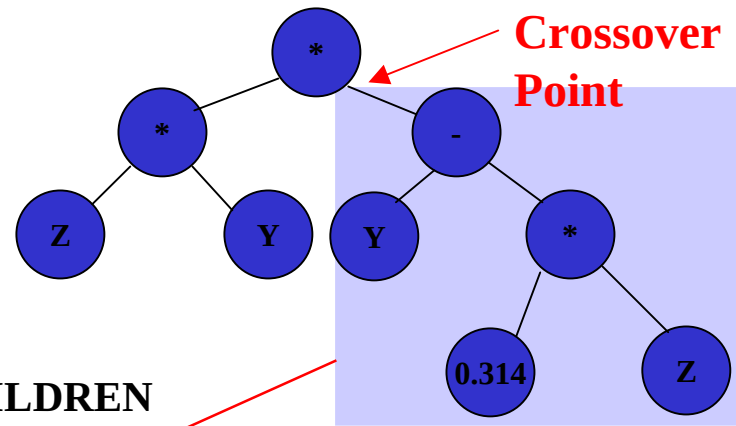


Crossover Example in GP

- Given two parents:



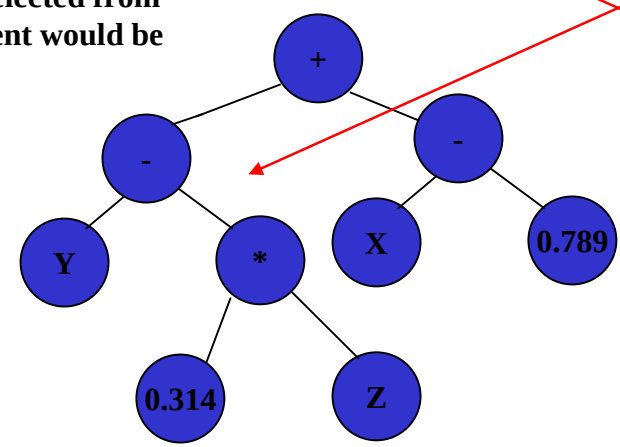
(+ (* 0.234 Z) (- X 0.789)

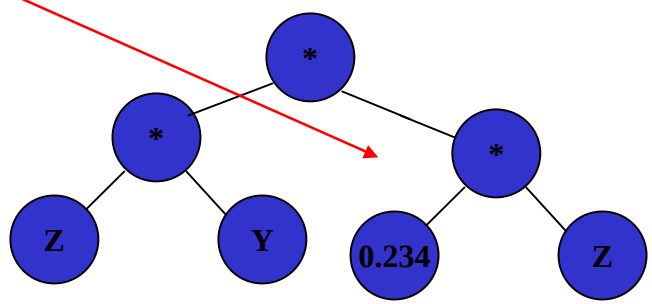


(* (* Z Y) (+ Y (* 0.314 Z)))

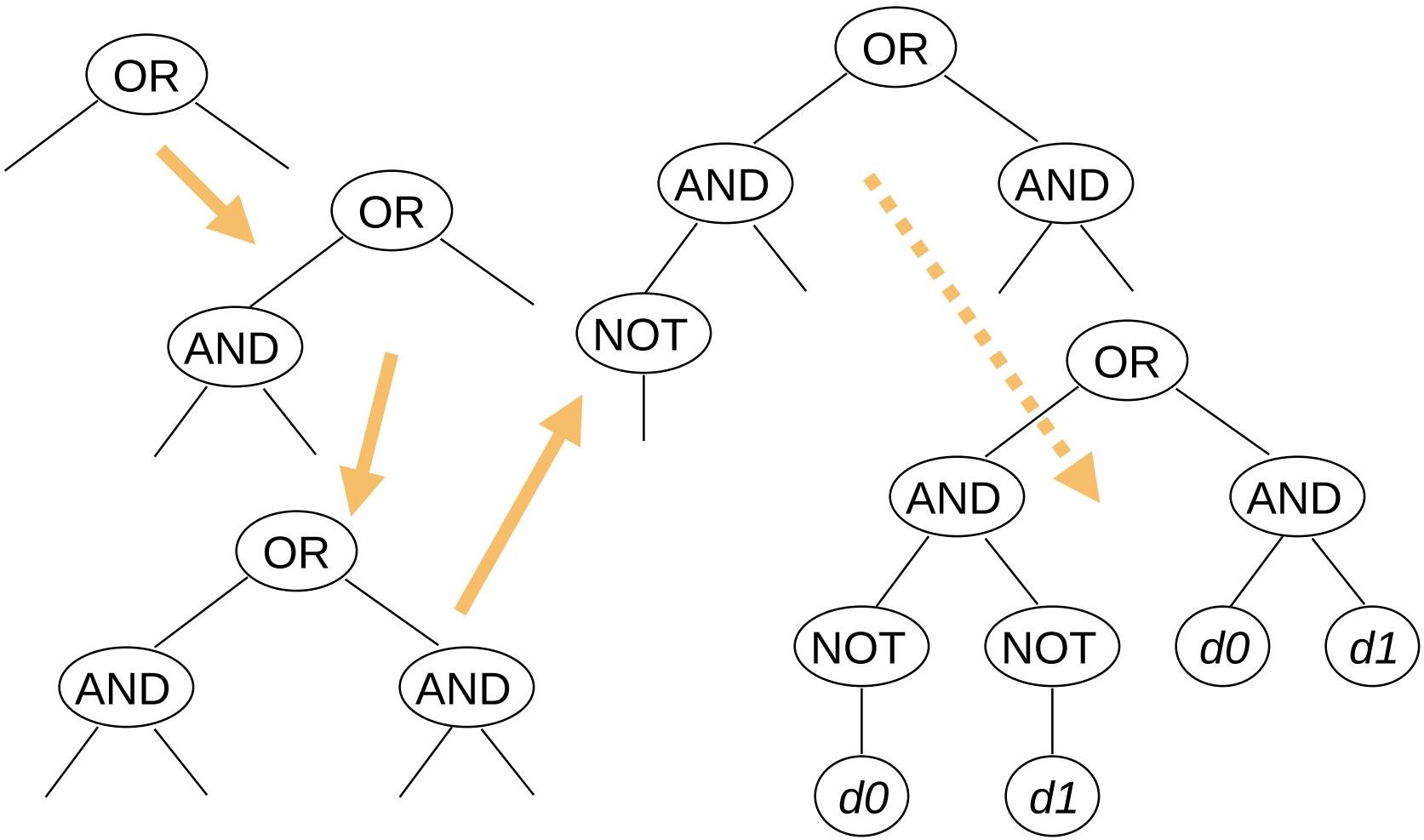
CHILDREN

NOTE: The root could be selected from one parent so the entire parent would be placed at the crossover site

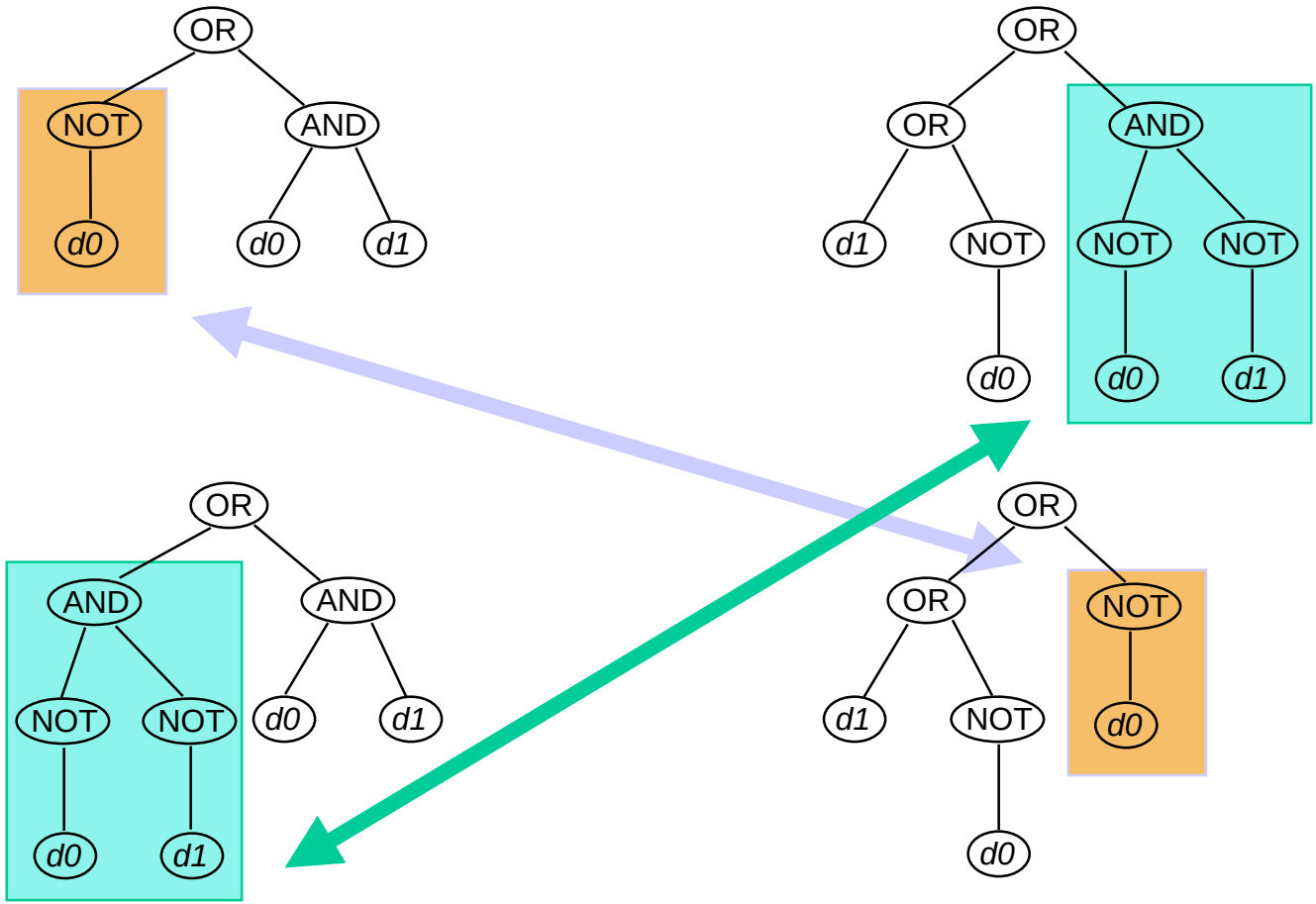




Genetic Programming: Initialization



Genetic Programming: Crossover



Crossovers in Programs

- Two ***parental programs*** are selected from the population based on fitness.
- A crossover point is randomly chosen in the first and second parent.
  - The first parent is called receiving
  - The second parent is called contributing
- The subtree rooted at the crossover point of the first parent is deleted
- It is replaced by the subtree from the second parent.
- **Crossover** is the *predominant operation* in genetic programming (and genetic algorithm) research
- It is performed with a high probability (say, 85% to 90%).

Examples of Genetic Programs

- 1. Symbolic Regression -
  - the process of discovering:
    - the functional form of a target function
    - and all of its necessary coefficients,
    - or at least an approximation to these.
- 2. Analog circuit design
  - Embryo circuit is an initial circuit which is **modified** to create a new circuit according to functionality criteria.

Genetic Programming in the Future

- Speculative.
- Only been around for 8 years.
- Is very successful.
- Discovery of new algorithms in existing projects.

Growth Control

- Since crossover selects randomly sized subtrees to place at random locations in an existing tree, it is possible for these trees to grow quite large

SOLUTION: If a crossover operation creates a tree that is too large then abort the crossover and select one of the parents to pass to the next generation

Parent Selection

- Since all the reproduction methods are based on the fitness measure, they can be used for genetic programming without a change

Roulette Wheel Selection

Tournament Selection

Fitness

- Overall, the fitness calculation for genetic programming requires running each program in the population and evaluating the results

- EXAMPLE: the problem is to develop a program which implements a function that passes through a set of 5 given points

run the program for each of the 5 points and compare how far the output is from the point it is trying to match

The Fitness Test

- Identifying the way of evaluating how good a given computer program is at solving the problem at hand.
- How good can a program cope with its environment.
- Can be measured in many ways, i.e. error, distance, time, etc…

Fitness Test Criteria

- Time complexity a good criteria.
  - i.e. **n^2** vs. **n*log n**.
- Accuracy - Values of variables.
- Combinations of criteria may also be tested.

Other GP Functions

Editing

- Simplify functions as genetic programming is running
- Implements a set of rules for simplification
- RULE ONE: If any function that has no side effects and only constants as arguments is found, it is replaced by the results of its evaluation

Other Editing Rules: **(- X X) is replaced by 0**

(/ X 1) is replaced by X

(and X X) is replaced by X

Encapsulation

- GOAL: identify an important operation and protect it from disruption by crossover by creating a new function

- Begin by randomly selecting a population element in the same way they are selected to become parents - bias to the best
  - randomly select a subtree and create a function without arguments which implements the subtree
  - replace the subtree with a single node that calls this new function

Constants

- Many programs involve the implementation of an algorithm which contains mathematical expressions some of which contain constants

- METHOD: a special terminal is added to the terminal set, R, a random constant
  - some characteristics are specified such as possible range and type (real, integer. . .)
  - every time R is selected, a random version of R is placed in the tree

Initial Population Alternative

- To avoid prejudicing the initial population towards full or grow trees, do *ramped- half-and- half* using user-determined ***minDepth*** and ***maxDepth*** parameters.
  - At each depth *D* between minDepth and maxDepth, create:
    - popsize/(maxDepth- minDepth)/ 2 full trees
    - popsize/(maxDepth- minDepth)/ 2 grow trees each of depth D.

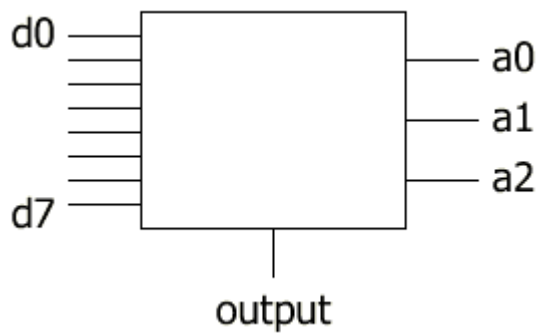
Selection Alternative in GP

- As in GA, GP can use fitness proportional selection and tournament selection.

- *fitness-overselection* is a technique used to increase selection pressure on large populations.
  - the population of trees is sorted in fitness order
  - those trees that are part of the top 32% of the population's fitness are selected for 80% of the time,
  - the remaining 68% of the population is selected for only 20% of the time.

Example of GP in circuit design

- Evolve code which implements an 8-to-1 Mux
  - function set = {and (2 args), or (2 args), not (1arg), if (3 args: condition, true exp, false exp)}
  - terminal set = {d0 ... d7, a0 ... a2}
  - fitness = each tree is evaluated on all 2048 (2^{11}) cases of possible input, fitness is how many cases the GP tree gets right.

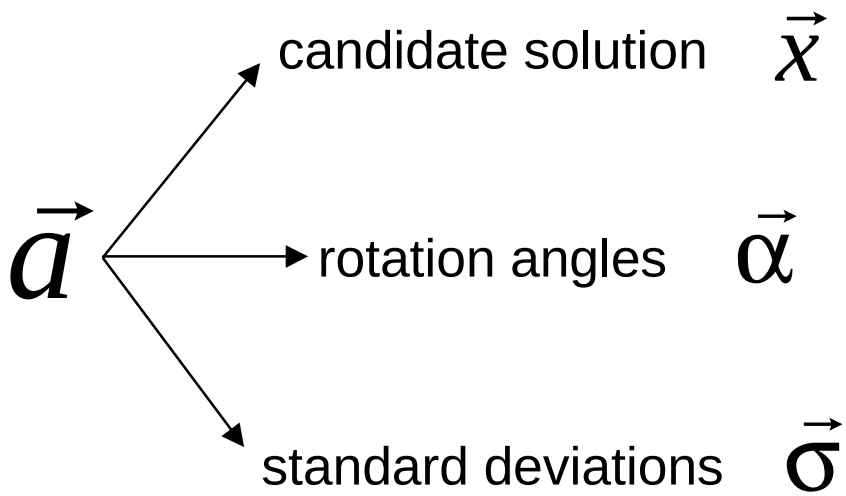


Typical Program in GP (LISP data)

- An early program might look like:

```
(if (or a2 a0)
    (if (or a2 a0)
        (if a2 d7 d1) a0)
    (or a0
        (or (or (if (not (and d3 d7))
                    (if (not d5)
                                (or d1
                                    (if a2 d7 d1))
                            (or d2 d2))
                    (and a1
                         (if a1 a1 d1))) d2)
            (and (and a0 d5)
                 (or d7 d0)))))
```

Evolution Strategies: Individuals



$$\alpha_{ij} = \frac{1}{2}\arctan\frac{2\,\mathrm{cov}(i,j)}{\sigma_i^2 - \sigma_j^2}$$

Evolution Strategies: Mutation

$$\sigma_i' = \sigma_i \exp(\tau' N(0,1) + \tau N_i(0,1))$$

$$\alpha_j' = \alpha_j + \beta N_j(0,1)$$

self-adaptation

$$\vec{x}' = \vec{x} + \vec{N}(\vec{0}, \vec{\sigma}', \vec{\alpha}')$$

Hans-Paul Schwefel suggests:

$$\tau \propto \left(\sqrt{2\sqrt{n}}\right)^{-1}$$

$$\tau' \propto \left(\sqrt{2n}\right)^{-1}$$

$$\beta \approx 0.0873 = 5°$$

Genetic Programming

- **Program induction**
- **LISP (historically), math expressions, machine language, ...**
- **Applications:**
  - **optimal control;**
  - **planning;**
  - **sequence induction;**
  - **symbolic regression;**
  - **modelling and forecasting;**
  - **symbolic integration and differentiation;**
  - **inverse problems**

Genetic Programming: Other Operators

- Mutation: replace a terminal with a subtree
- Permutation: change the order of arguments to a function
- Editing: simplify S-expressions, e.g. (AND X X) $\rightarrow$ X
- Encapsulation: define a new function using a subtree
- Decimation: throw away most of the population

Genetic Programming: Fitness

Fitness cases: $j = 1, \ldots, N_e$

"Raw" fitness: $$r(\gamma) = \sum_{j=1}^{N_e} \left| \text{Output}(\gamma, j) - C(j) \right|$$

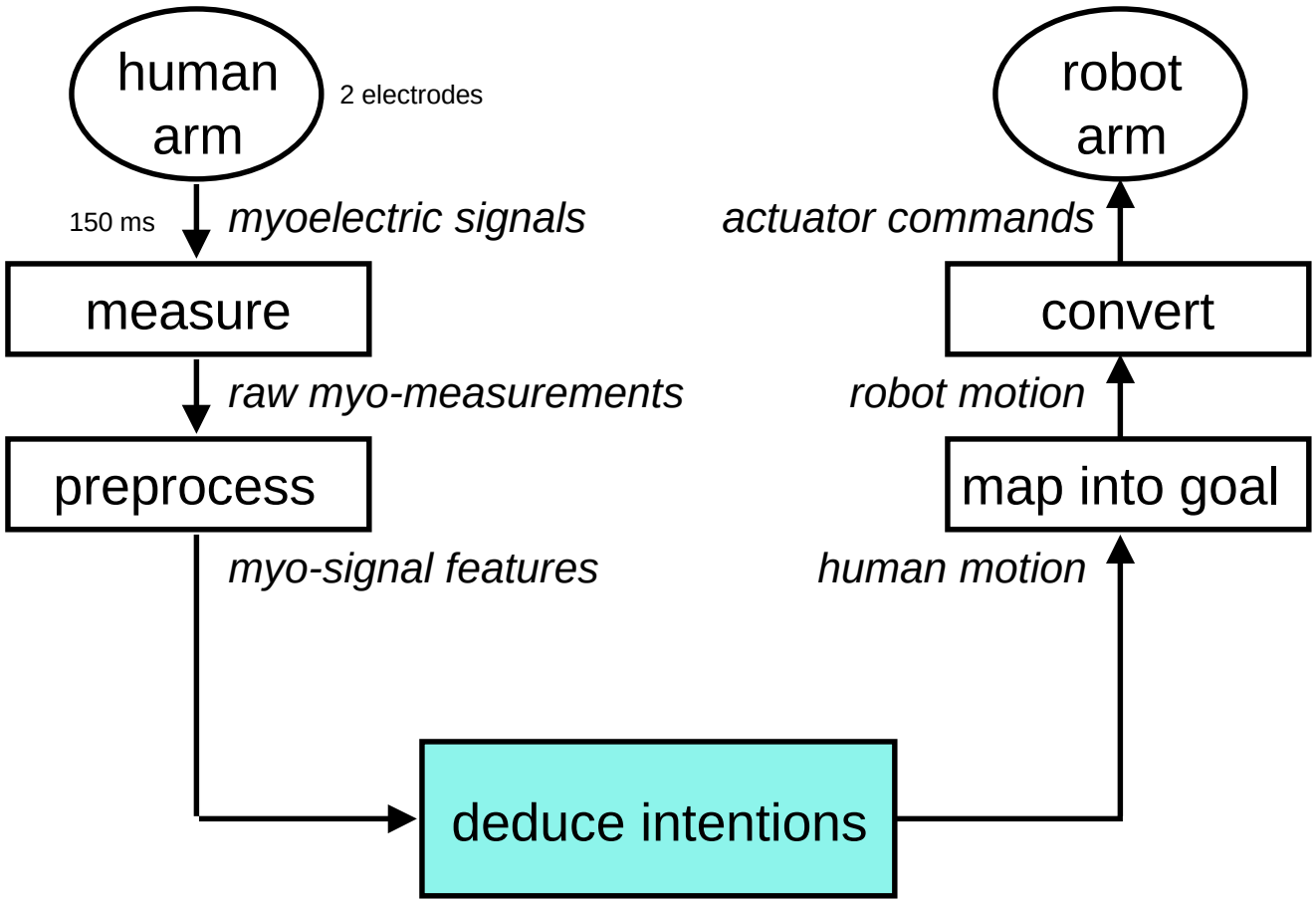
"Standardized" fitness: $s(\gamma) \in [0, +\infty)$

"Adjusted" fitness: $$a(\gamma) = \frac{1}{1 + s(\gamma)}$$

Example of Application: Myoelectric Prosthesis Control

- Control of an upper arm prosthesis
- Genetic Programming application
- Recognize thumb flection, extension and abduction patterns

Prosthesis Control: The Context



Prosthesis Control: Terminals

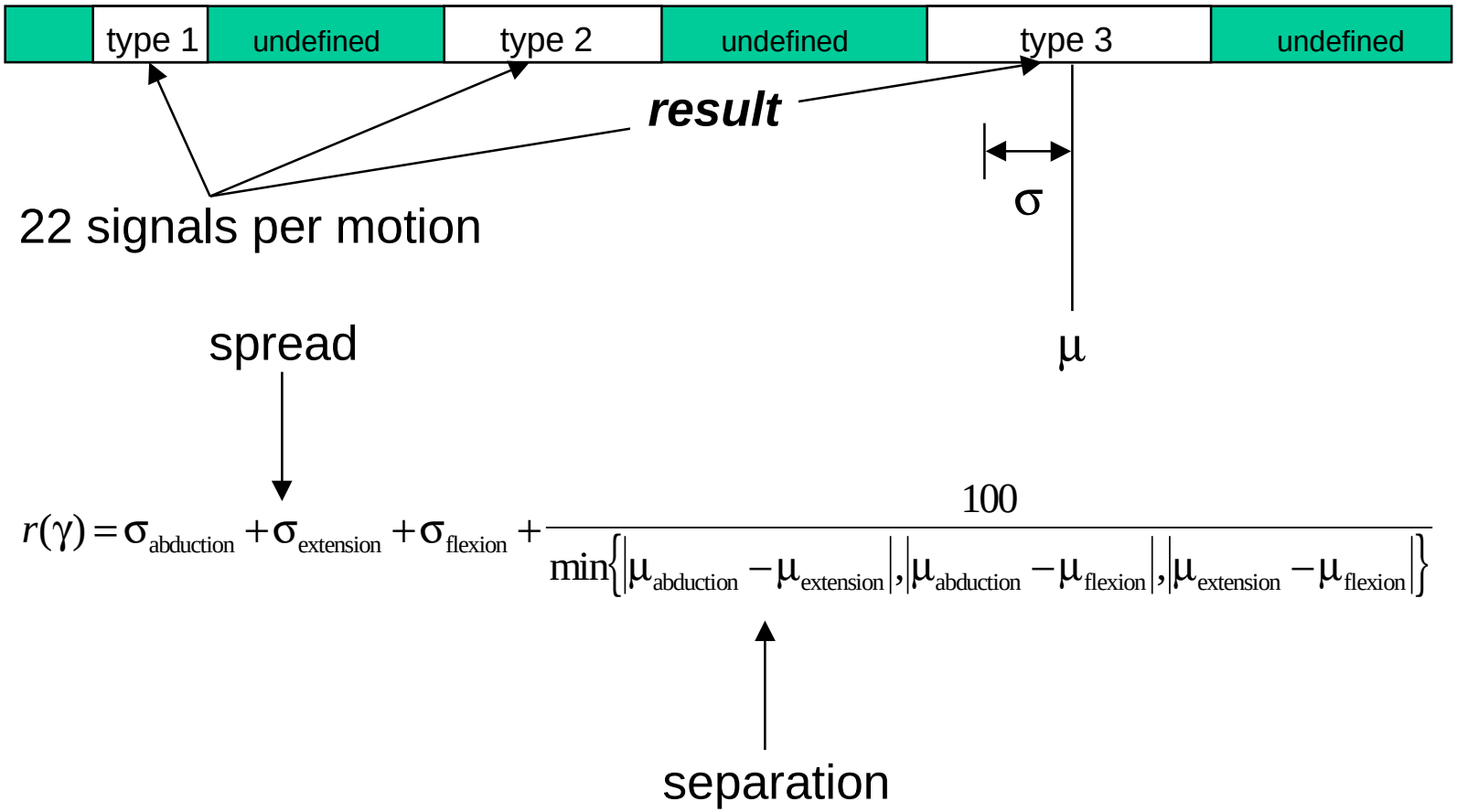
Features for electrodes 1, 2:

- **Mean absolute value (*MAV*)**
- **Mean absolute value slope (*MAVS*)**
- **Number of zero crossings (*ZC*)**
- **Number of slope sign changes (*SC*)**
- **Waveform length (*LEN*)**
- **Average value (*AVG*)**
- **Up slope (*UP*)**
- **Down slope (*DOWN*)**
- ***MAV1/MAV2, MAV2/MAV1***
- ***0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 0.01, -1.0***

Prosthesis Control: Function Set

Addition	x + y	
Subtraction	x - y	
Multiplication	x * y	
Division	x / y	(protected for y=0)
Square root	sqrt(|x|)	
Sine	sin x	
Cosine	cos x	
Tangent	tan x	(protected for x=π/2)
Natural logarithm	ln |x|	(protected for x=0)
Common logarithm	log |x|	(protected for x=0)
Exponential	exp x	
Power function	x ^ y	
Reciprocal	1/x	(protected for x=0)
Absolute value	|x|	
Integer or truncate	int(x)	
Sign	sign(x)	

Prosthesis Control: Fitness



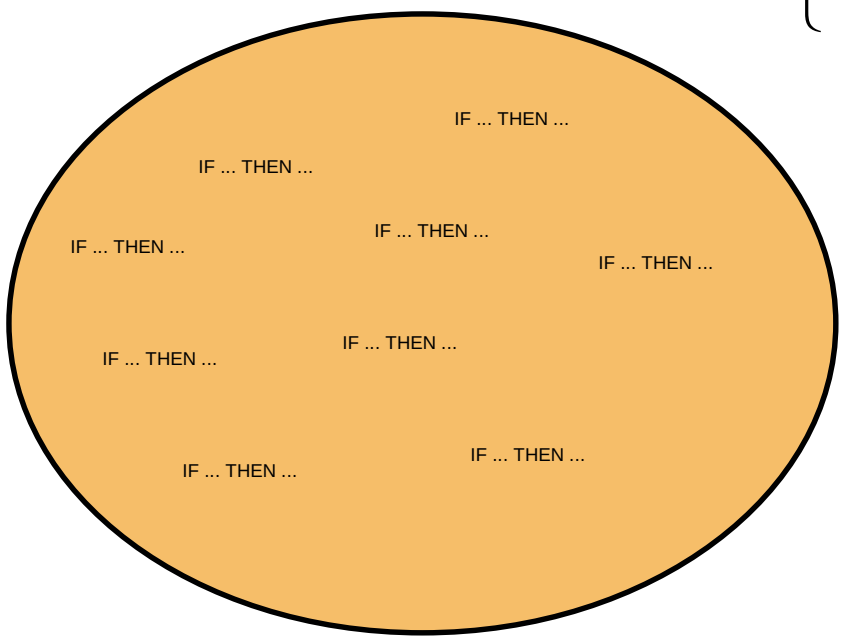
Myoelectric Prosthesis Control Reference

- Jaime J. Fernandez, Kristin A. Farry and John B. Cheatham. “Waveform Recognition Using Genetic Programming: The Myoelectric Signal Recognition Problem. GP ‘96, The MIT Press, pp. 63–71

Classifier Systems (Michigan approach)

individual: IF X = A AND Y = B THEN Z = D

$$f_{n+1}(\gamma) = \begin{cases} (1-e)\, f_n(\gamma) + r & \gamma(n) = \text{class}(n) \\ (1-p)\, f_n(\gamma) & \gamma(n) \neq \text{class}(n) \end{cases}$$



where $r \propto (1 - gN_{\gamma})R$

number of attributes in antecedent part

Practical Implementation Issues

- from elegant academia to not so elegant but robust and efficient real-world applications, evolution programs
- handling constraints
- hybridization
- parallel and distributed algorithms

Evolution Programs

Slogan:

Genetic Algorithms + Data Structures = Evolution Programs

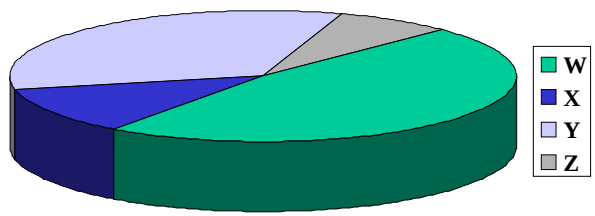
Key ideas:

- use a data structure as close as possible to object problem
- write appropriate genetic operators
- ensure that all genotypes correspond to feasible solutions
- ensure that genetic operators preserve feasibility

Encodings: “Pie” Problems

W	X	Y	Z
128	32	90	20
0–255	0–255	0–255	0–255

X = 32/270 = 11.85%



Encodings: “Permutation” Problems

Adjacency Representation

(2, 4, 8, 3, 9, 7, 1, 5, 6)

Ordinal Representation

(1, 1, 2, 1, 4, 1, 3, 1, 1)

Path Representation

(1, 2, 4, 3, 8, 5, 9, 6, 7)

Matrix Representation

0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1
0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	0	0	1	1	0	0

Sorting Representation

(-23, -6, 2, 0, 19, 32, 85, 11, 25)

1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7

Handling Constraints

- Penalty functions

  Risk of spending most of the time evaluating unfeasible solutions, sticking with the first feasible solution found, or finding an unfeasible solution that scores better of feasible solutions

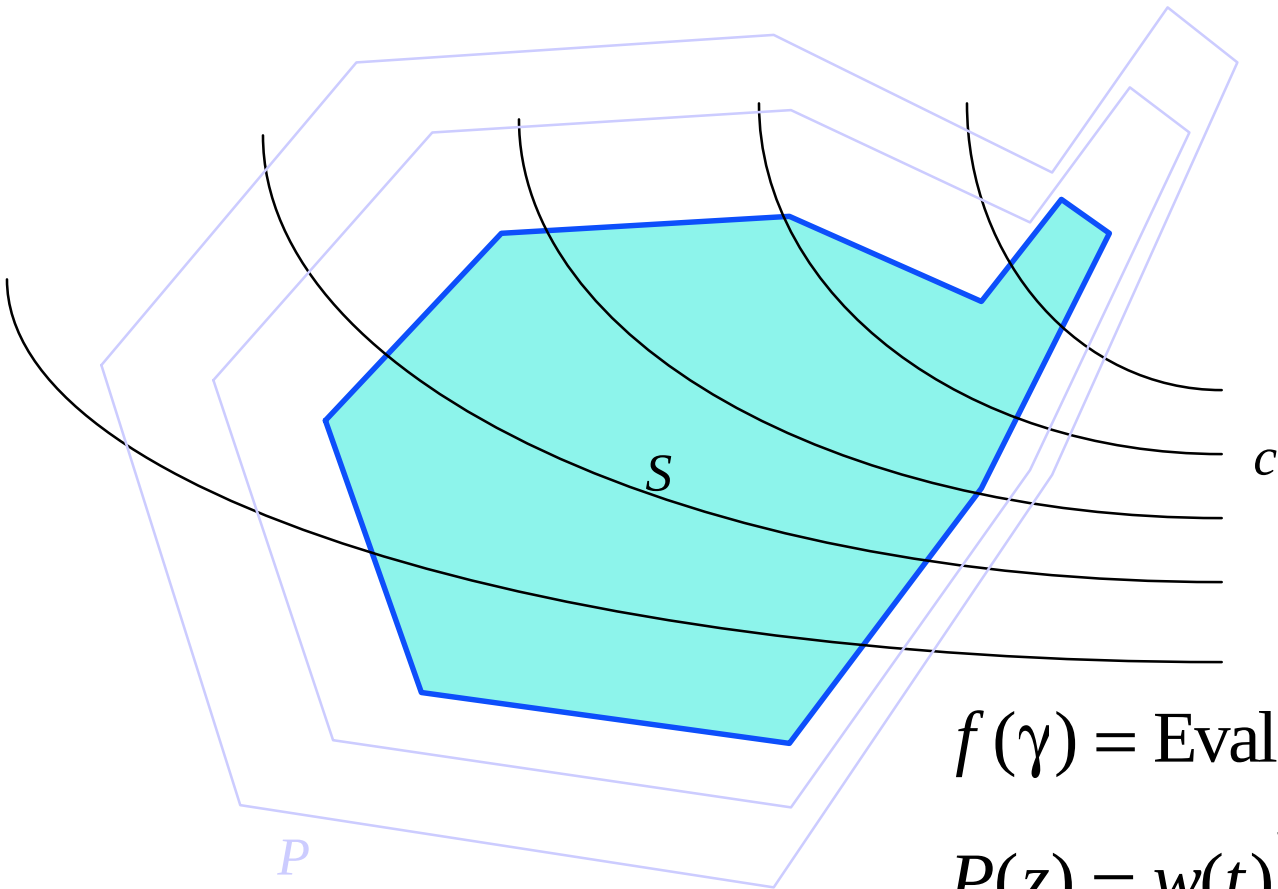
- Decoders or repair algorithms

  Computationally intensive, tailored to the particular application

- Appropriate data structures and specialized genetic operators

  All possible genotypes encode for feasible solutions

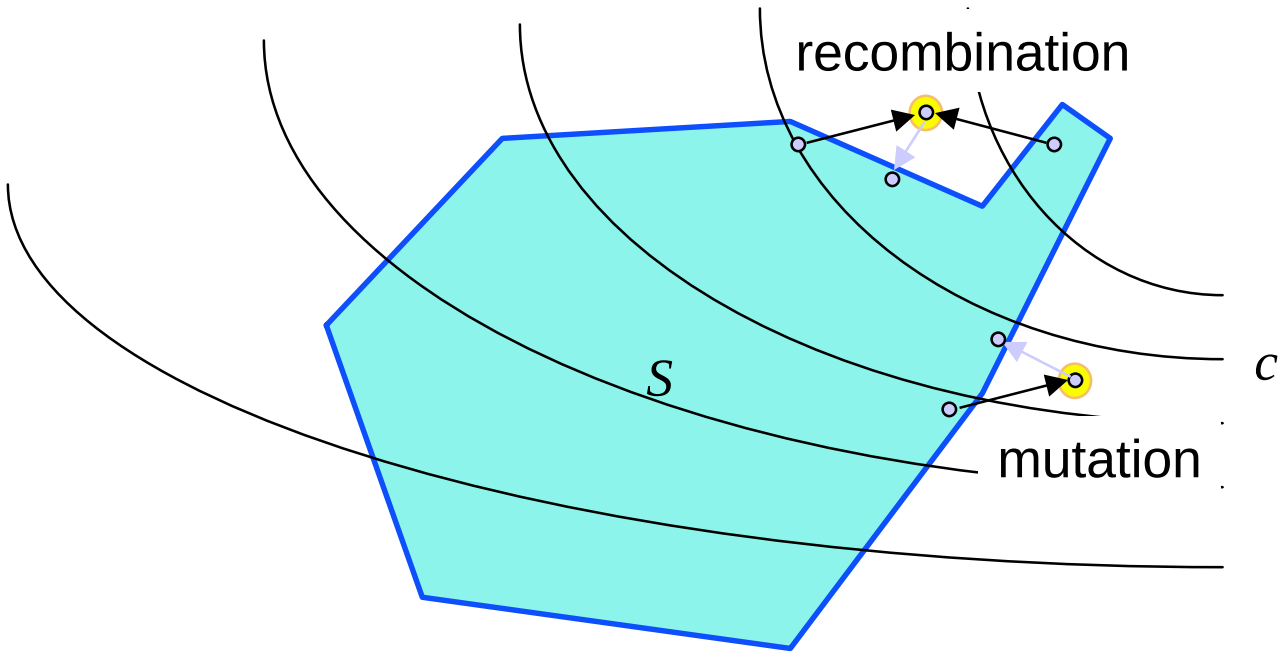
Penalty Functions



$$f(\gamma) = \text{Eval}(c(z)) + P(z)$$

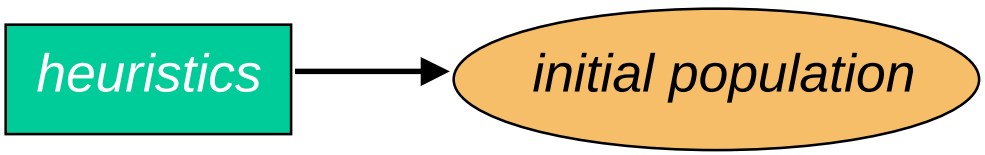
$$P(z) = w(t)\sum_i w_i \Delta_i(z)$$

Decoders / Repair Algorithms



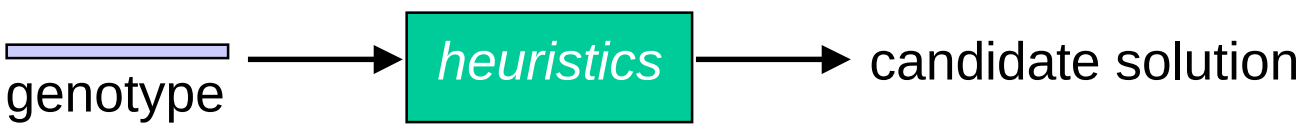
Hybridization

1) Seed the population with solutions provided by some heuristics



2) Use local optimization algorithms as genetic operators
(Lamarckian mutation)

3) Encode parameters of a heuristics

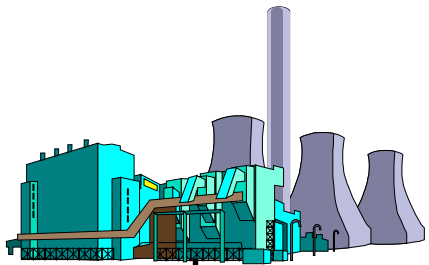


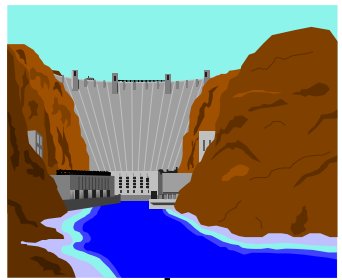
Example of Application: Unit Commitment

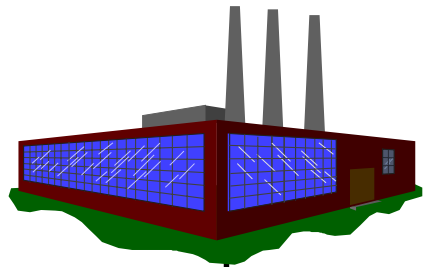
- Multiobjective optimization problem: cost VS emission
- Many linear and non-linear constraints
- Traditionally approached with dynamic programming

- Hybrid evolutionary/knowledge-based approach
- A flexible decision support system for planners
- Solution time increases linearly with the problem size

The Unit Commitment Problem





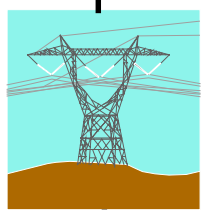


$$z_E = \sum_{i=1}^{n} E_i(P_i)$$

$$E_i(P_i) = \sum_{j=1}^{m} E_{ij}(P_i)$$

$$E_{ij}(P_i) = \alpha_{ij} + \beta_{ij} P_i + \gamma_{ij} P_i^2$$

Emissions

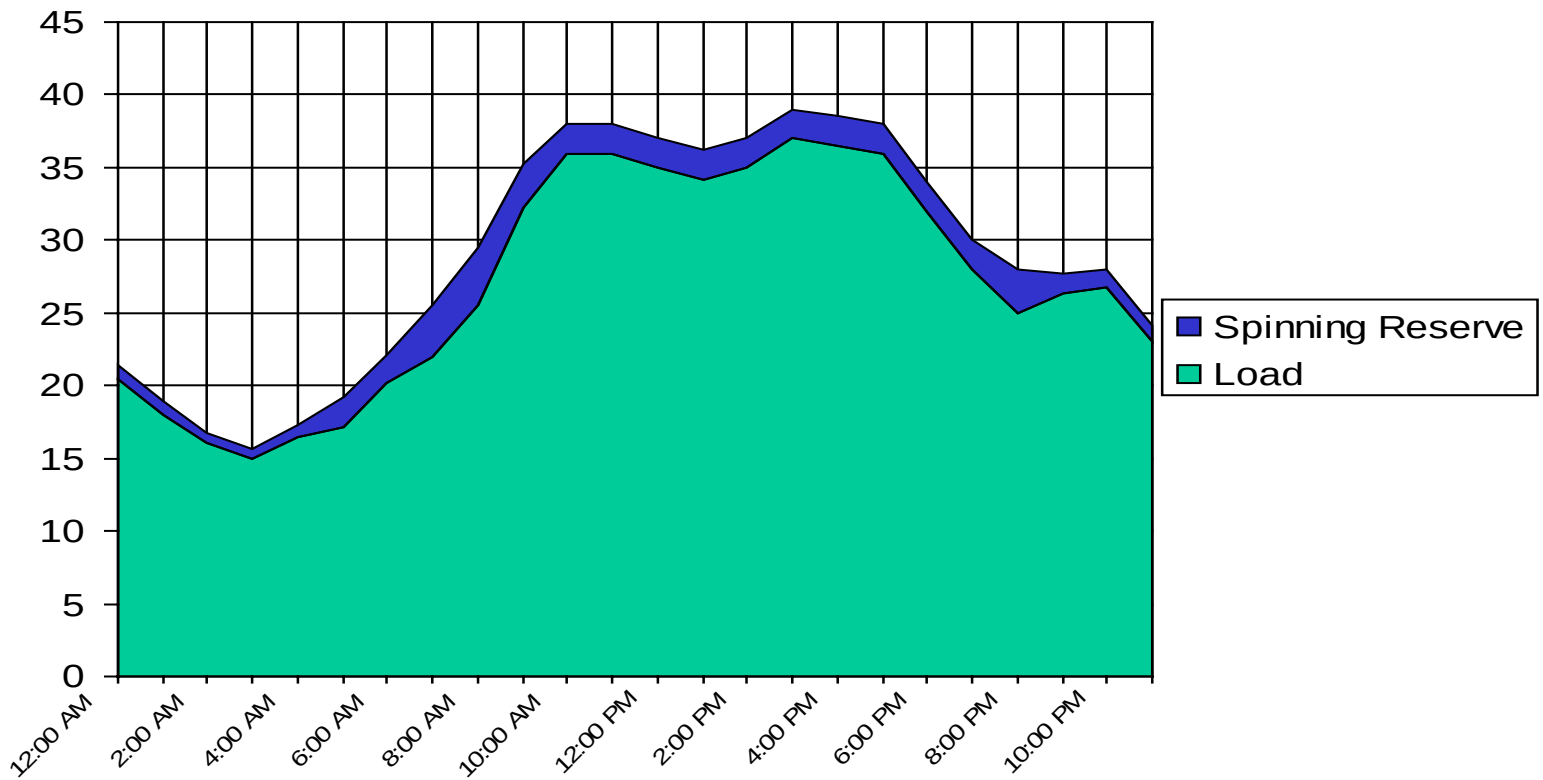


$$z_{\$} = \sum_{i=1}^{n} \left(C_i(P_i) + SU_i + SD_i + HS_i \right)$$

$$C_i(P_i) = a_i + b_i P_i + c_i P_i^2$$

Cost

Predicted Load Curve

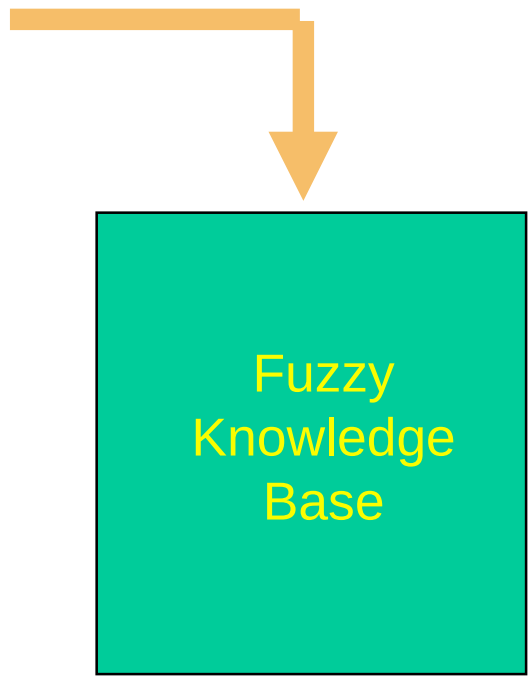


Unit Commitment: Constraints

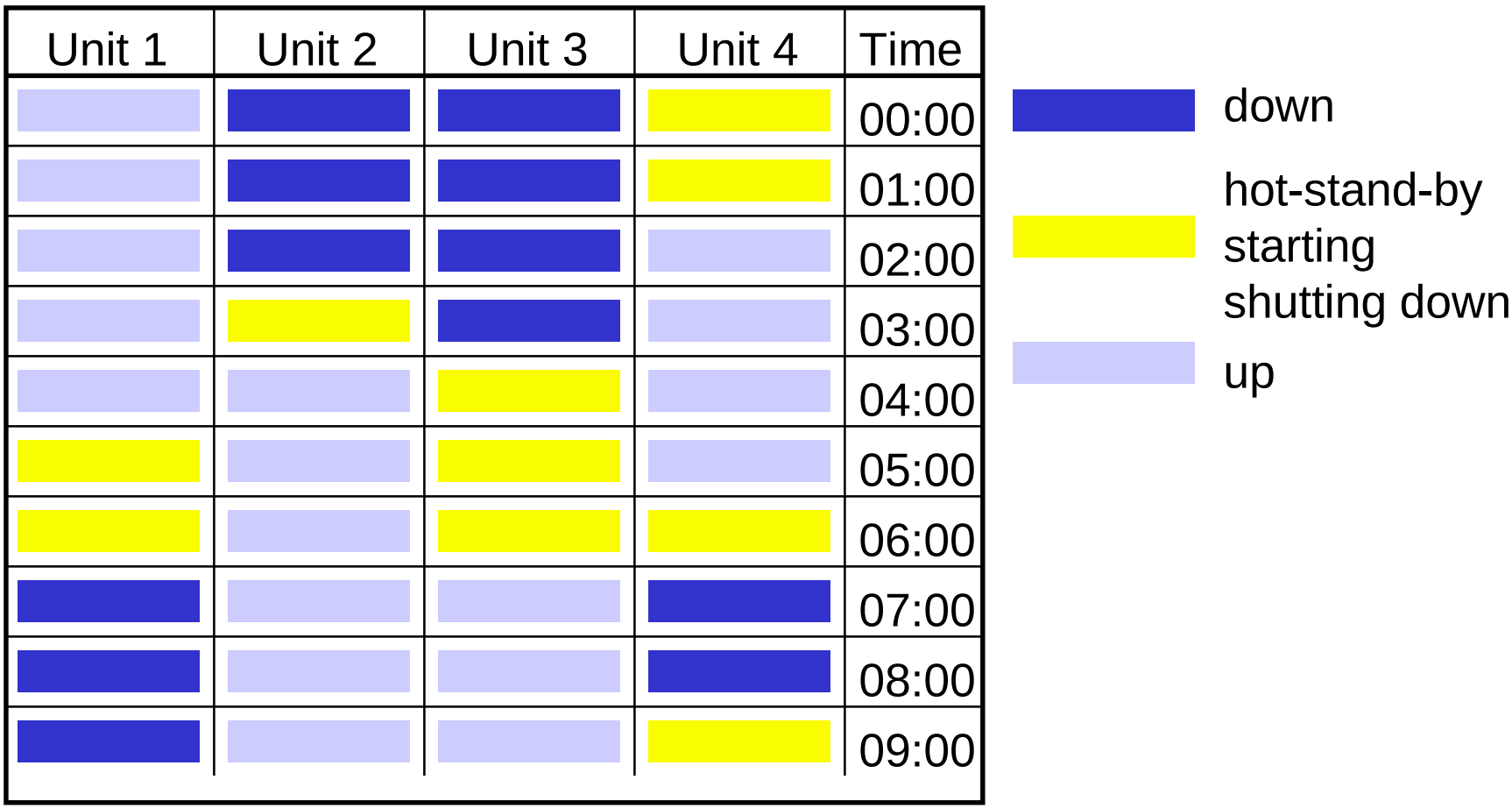
- Power balance requirement
- Spinning reserve requirement
- Unit maximum and minimum output limits
- Unit minimum up and down times
- Power rate limits
- Unit initial conditions
- Unit status restrictions
- Plant crew constraints
- ...

Unit Commitment: Encoding

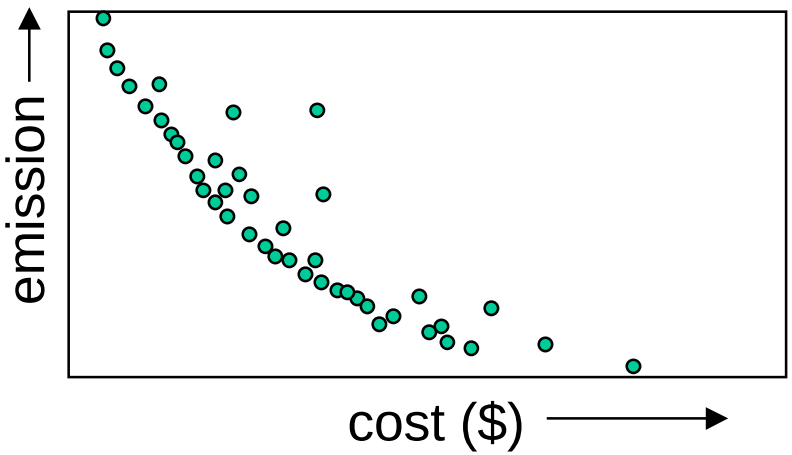
Unit 1	Unit 2	Unit 3	Unit 4	Time
1.0	0.8	0.2	0.15	00:00
0.9	1.0	0.2	1.0	01:00
0.0	1.0	0.8	0.2	02:00
0.0	0.5	1.0	0.8	03:00
1.0	0.65	0.8	1.0	04:00
0.8	0.8	0.25	1.0	05:00
1.0	0.4	0.2	1.0	06:00
0.0	0.0	1.0	0.75	07:00
0.5	1.0	1.0	0.8	08:00
1.0	0.5	0.0	0.0	09:00



Unit Commitment: Solution



Unit Commitment: Selection



competitive selection:

$507,762	$516,511
213,489 £	60,080 £

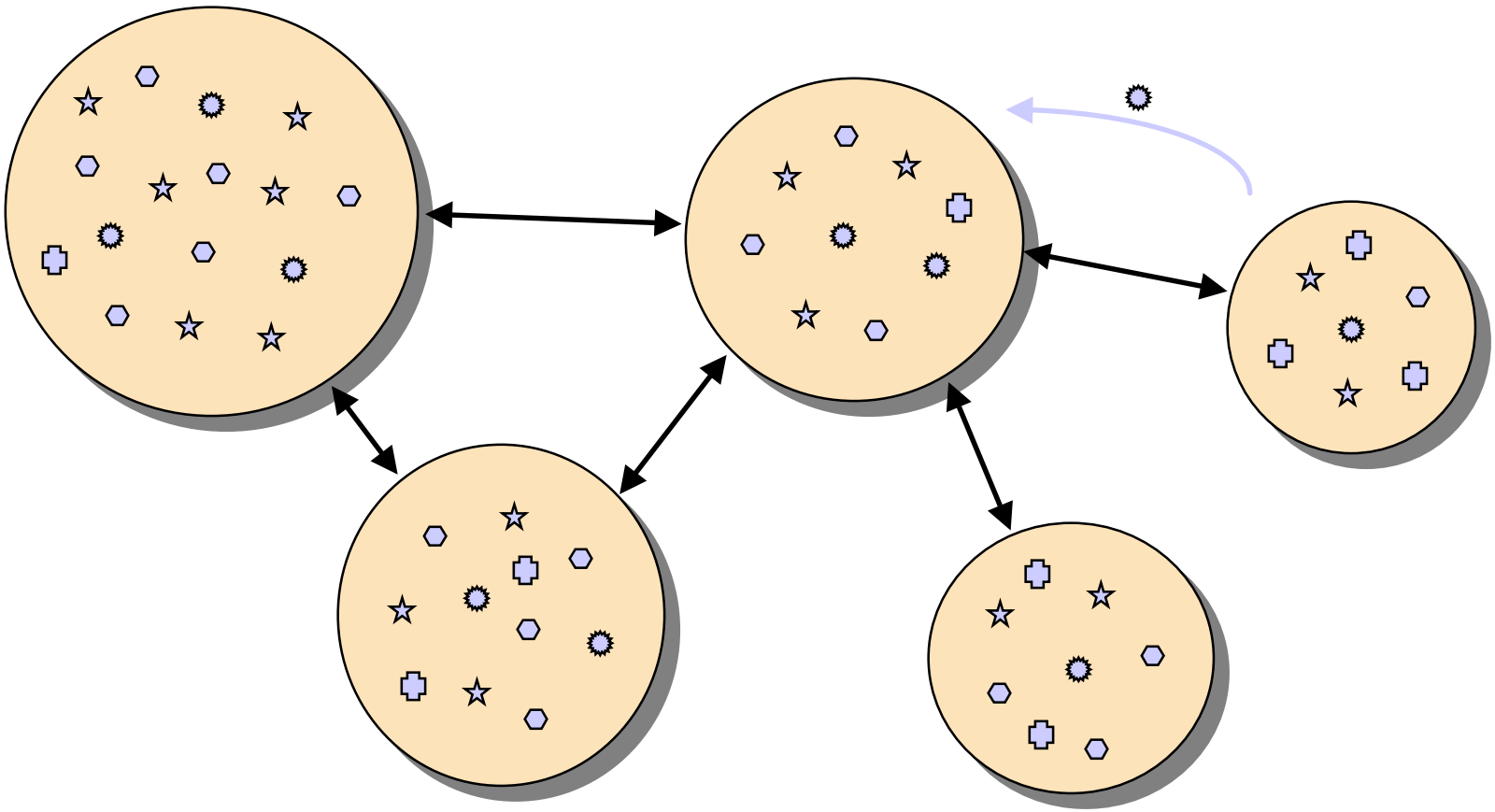
Unit Commitment References

- D. Srinivasan, A. Tettamanzi. “An Integrated Framework for Devising Optimum Generation Schedules”. In *Proceedings of the 1995 IEEE International Conference on Evolutionary Computing (ICEC ‘95)*, vol. 1, pp. 1-4.
- D. Srinivasan, A. Tettamanzi. *A Heuristic-Guided Evolutionary Approach to Multiobjective Generation Scheduling*. IEE Proceedings Part C - Generation, Transmission, and Distribution, 143(6):553-559, November 1996.
- D. Srinivasan, A. Tettamanzi. *An Evolutionary Algorithm for Evauation of Emission Compliance Options in View of the Clean Air Act Amendments*. IEEE Transactions on Power Systems, 12(1):336-341, February 1997.

Parallel Evolutionary Algorithms

- Standard evolutionary algorithm is sequential...
- … but evolutionary algorithms are intrinsically parallel
- Various models:
  - cellular evolutionary algorithm,
  - fine grain parallel evolutionary algorithm,
  - course grain parallel evolutionary algorithm, (isand)
  - Master-slave sequential evolutionary algorithm with parallel fitness function calculation

Island Model



Example Application: Protein Folding

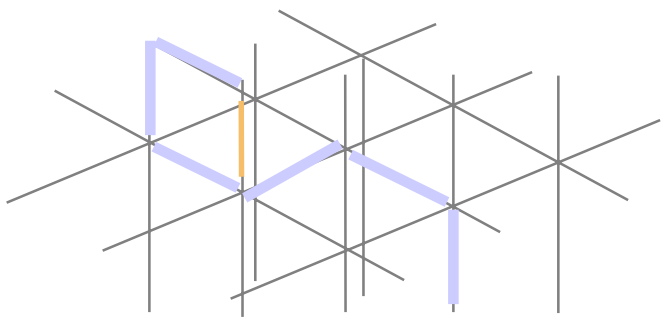
- Finding 3-D geometry of a protein to understand its functionality
- Very difficult: one of the “grand challenge problems”
- Standard GA approach
- Simplified protein model

Protein Folding: The Problem

- **Much of a proteins function may be derived from its conformation (3-D geometry or "tertiary" structure).**
- **Magnetic resonance & X-ray crystallography are currently used to view the conformation of a protein:**
  - **expensive in terms of equipment, computation and time;**
  - **require isolation, purification and crystallization of protein.**
- **Prediction of the final folded conformation of a protein chain has been shown to be NP-hard.**
- **Current approaches:**
  - **molecular dynamics modelling (brute force simulation);**
  - **statistical prediction;**
  - **hill-climbing search techniques (simulated annealing).**

Protein Folding: Simplified Model

- 90° lattice (6 degrees of freedom at each point);
- Peptides occupy intersections;
- No side chains;
- Hydrophobic or hydrophilic (no relative strengths) amino acids;
- Only hydrophobic/hydrophilic forces considered;
- Adjacency considered only in cardinal directions;
- Cross-chain hydrophobic contacts are the basis for evaluation.



Protein Folding: Representation

relative move encoding:

UP	DOWN	FORWARD	LEFT	UP	RIGHT

...

preference order encoding:

UP LEFT RIGHT DOWN FORWARD	DOWN LEFT UP FORWARD RIGHT	FORWARD UP DOWN LEFT RIGHT	LEFT DOWN FORWARD UP RIGHT

...

Protein Folding: Fitness

Decode: plot the course encoded by the genotype.

Test each occupied cell:

- any collisions: **-2**;
- no collisions AND a hydrophobe in an adjacent cell: **1**.

Notes:

- for each contact: **+2**;
- adjacent hydrophobes not discounted in the scoring;
- multiple collisions (>1 peptides in one cell): **-2**;
- hydrophobe collisions imply an additional penalty (no contacts are scored).

Protein Folding: Experiments

- Preference ordering encoding;
- Two-point crossover with a rate of 95%;
- Bit mutation with a rate of 0.1%;
- Population size: 1000 individuals;
- crowding and incest reduction.

- Test sequences with known minimum configuration;

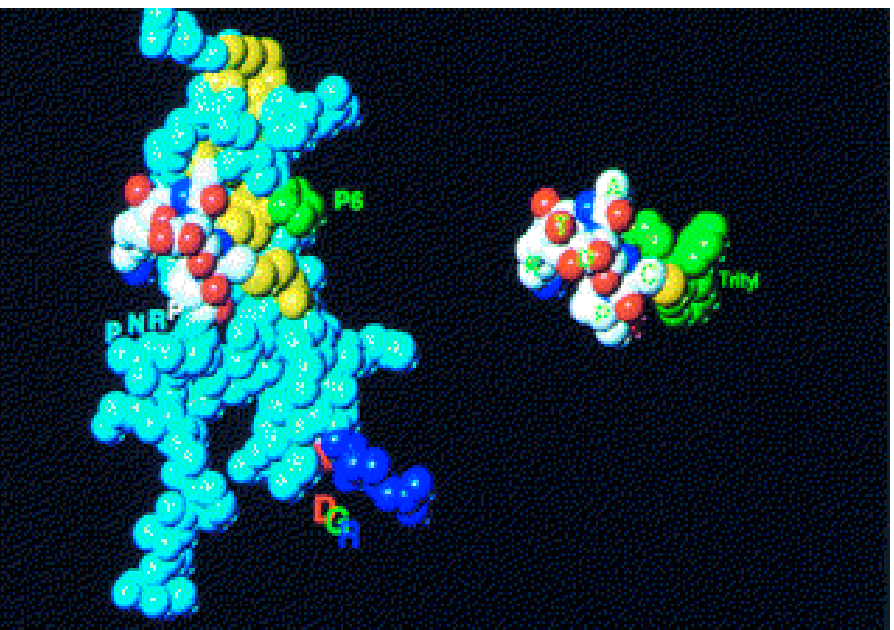
Protein Folding References

- S. Schulze-Kremer. “Genetic Algorithms for Protein Tertiary Structure Prediction”. PPSN 2, North-Holland 1992.
- R. Unger and J. Moult. “A Genetic Algorithm for 3D Protein Folding Simulations”. ICGA-5, 1993, pp. 581–588.
- Arnold L. Patton, W. F. Punch III and E. D. Goodman. “A Standard GA Approach to Native Protein Conformation Prediction”. ICGA 6, 1995, pp. 574–581.

Example of Application: Drug Design

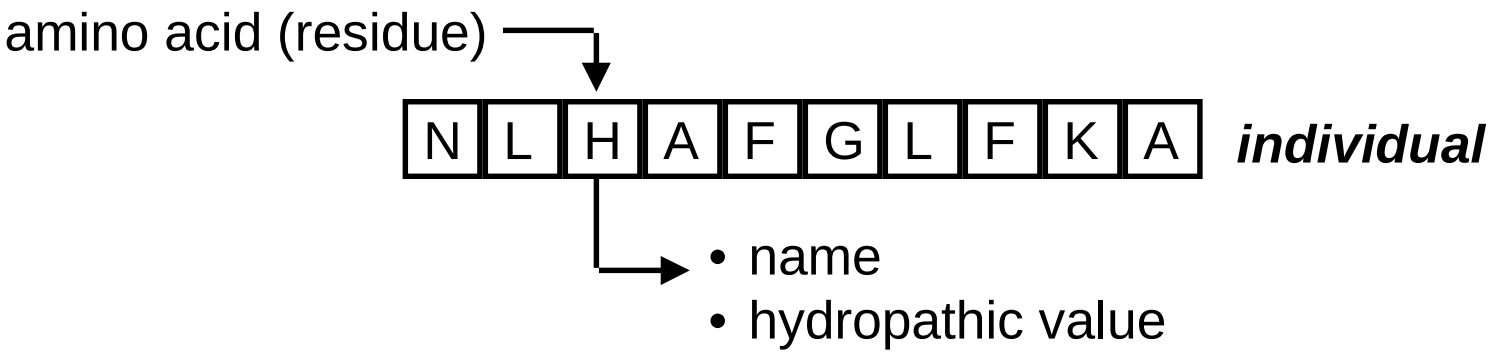
Purpose: given a chemical specification (activity), design a tertiary structure complying with it.

Requirement: a quantitative structure-activity relationship model.



Example: design ligands that can bind targets specifically and selectively. Complementary peptides.

Drug Design: Implementation



Operators:

- Hill-climbing Crossover
- Hill-climbing Mutation
- Reordering (no selection)

implicit selection

Drug Design: Fitness

target *a*

complement *b*

$$a_k = \sum_{i=k-s}^{k+s} h_i$$

moving average hydropathy

$$b_k = \sum_{i=k-s}^{k+s} g_i$$

hydropathy of residues

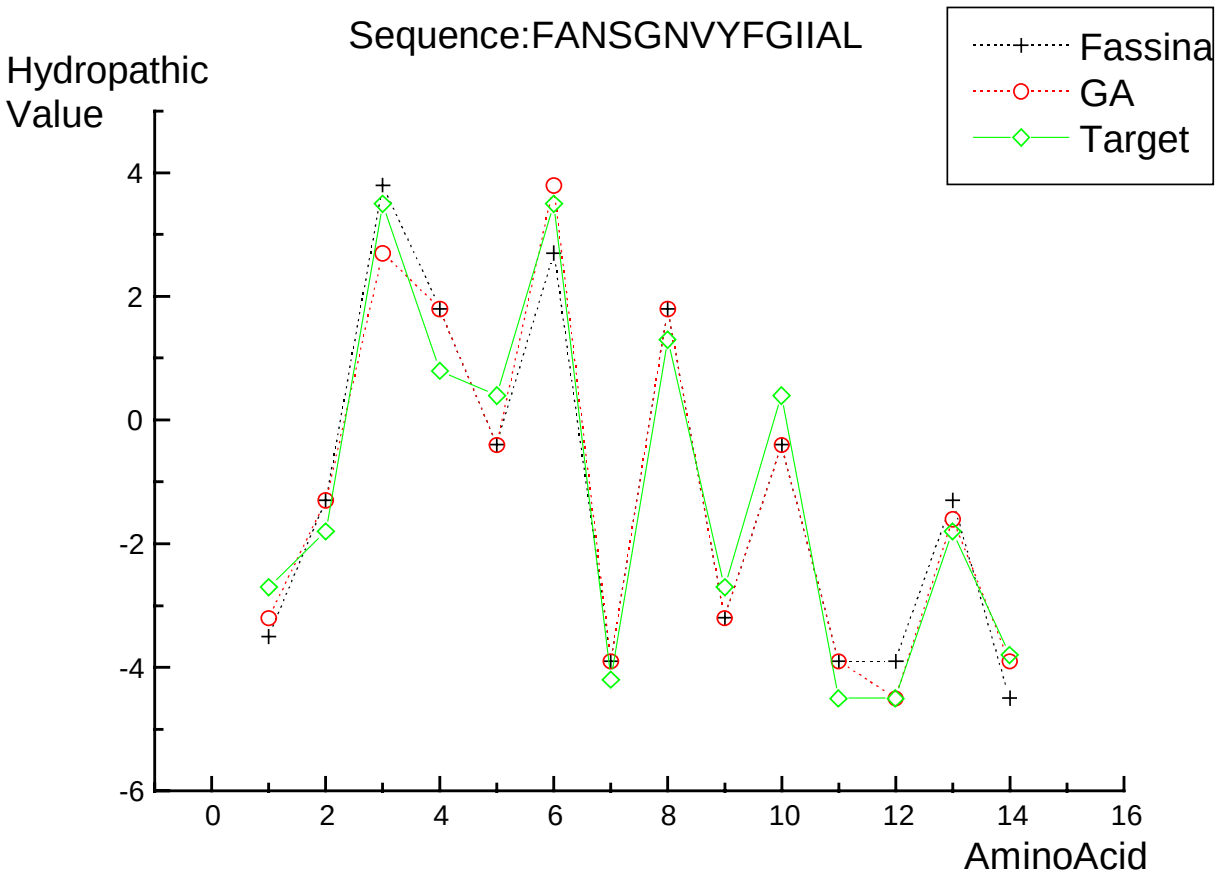
$k = s, \ldots, n - s$

n: number of residues in target

$$Q = \sum_i \sqrt{\frac{(a_i + b_i)^2}{n - 2s}}$$

(lower Q = better complementarity)

Drug Design: Results



Drug Design References

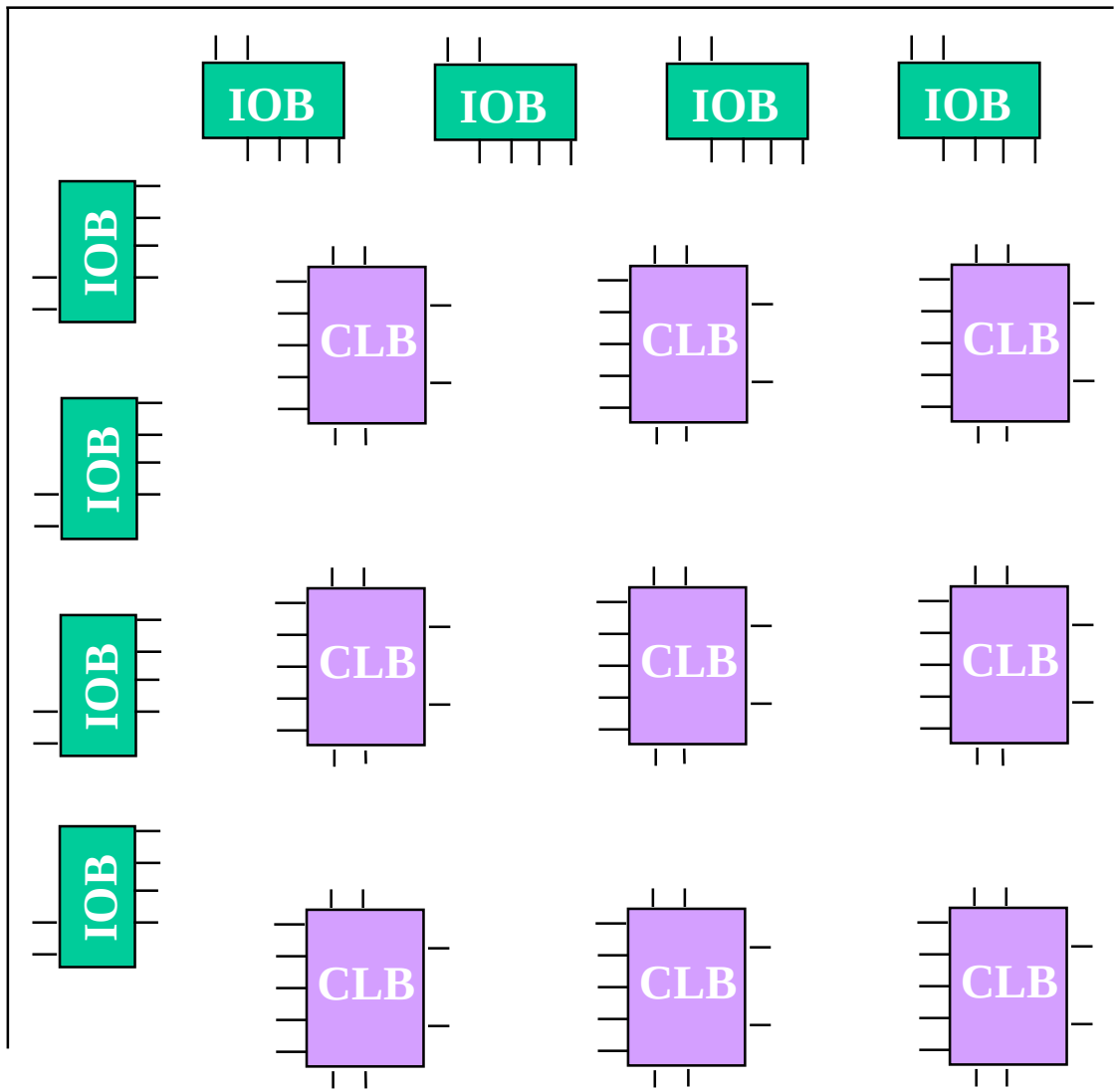
- T. S. Lim. *A Genetic Algorithms Approach for Drug Design.* MS Dissertation, Oxford University, Computing Laboratory, 1995.
- A. L. Parrill. *Evolutionary and Genetic Methods in Drug Design.* Drug Discovery Today, Vol. 1, No. 12, Dec 1996, pp. 514–521.

Evolution in Hardware, or Evolvable Hardware (EHW)

- A genetic algorithm can be used to design hardware as well as software
- This requires some form of programmable hardware
  - FPGA structure

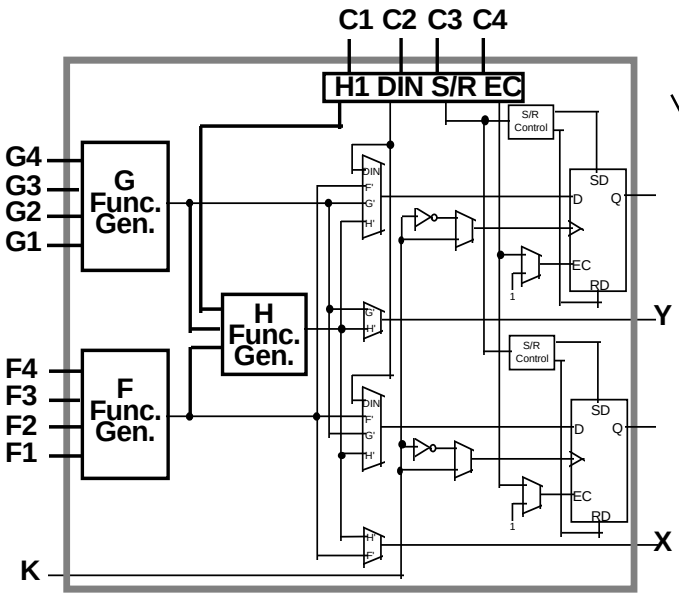
Reconfigurable Hardware

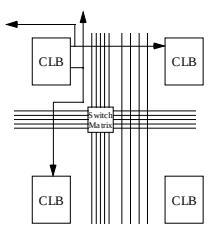
- FPGAs consist of logic blocks and programmable wire paths:



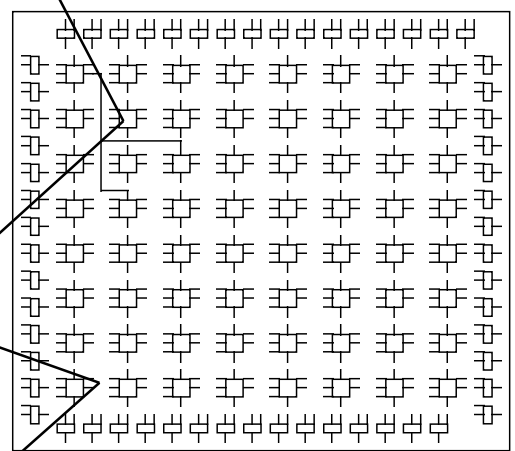
Xilinx XC 4000 Structure

Configurable Logic Blocks (CLBs)





Programmable Interconnect



- **High Density -> 1M System Gates**
- **SRAM Based LUT**
- **ASIC-like array structure**

Close up view:

Programmable Interconnect Points, PIPs (White)

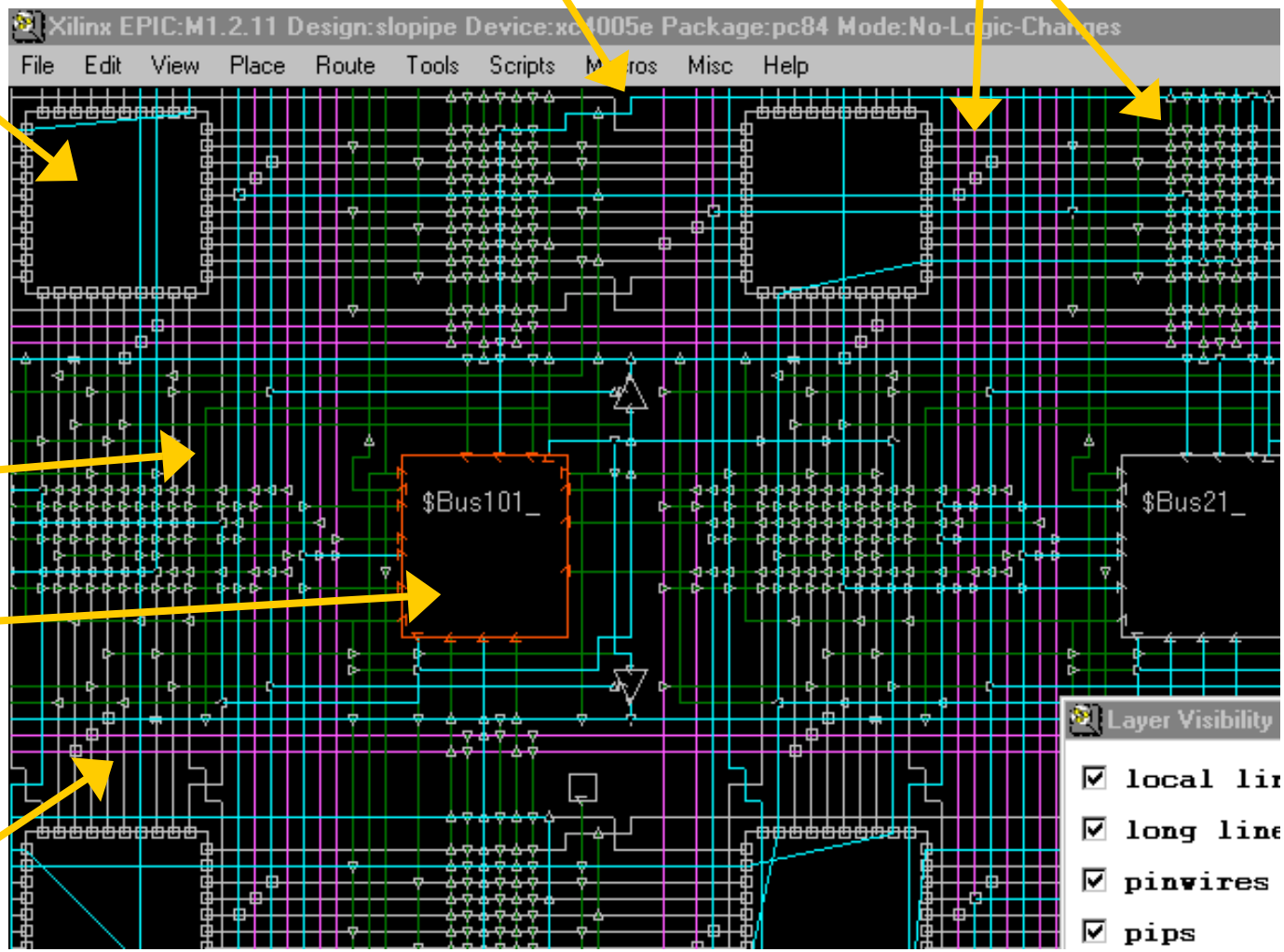
Routed Wires (Blue)

Switch Matrix

Direct Interconnect (Green)

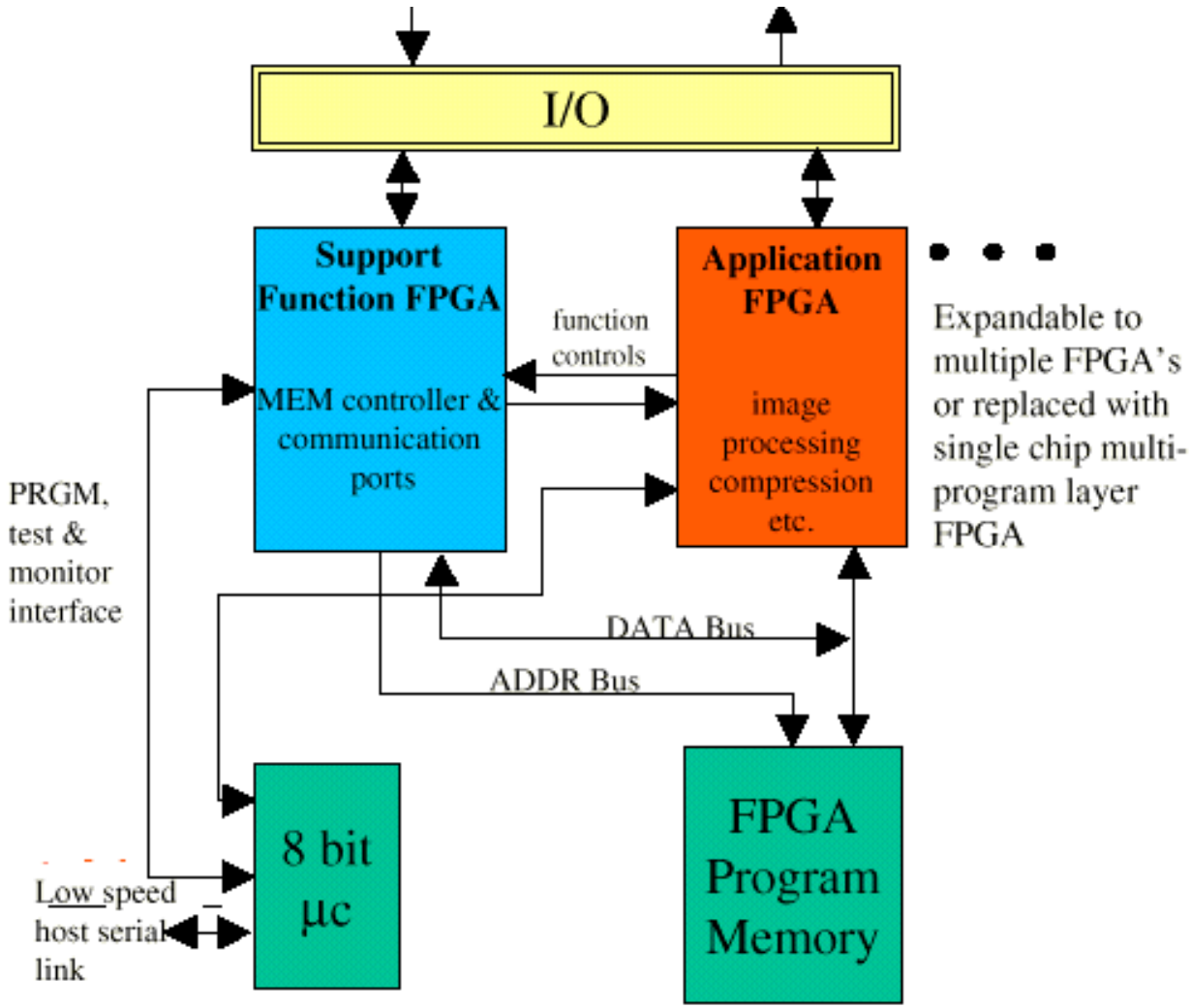
CLB (Red)

Long Lines (Purple)



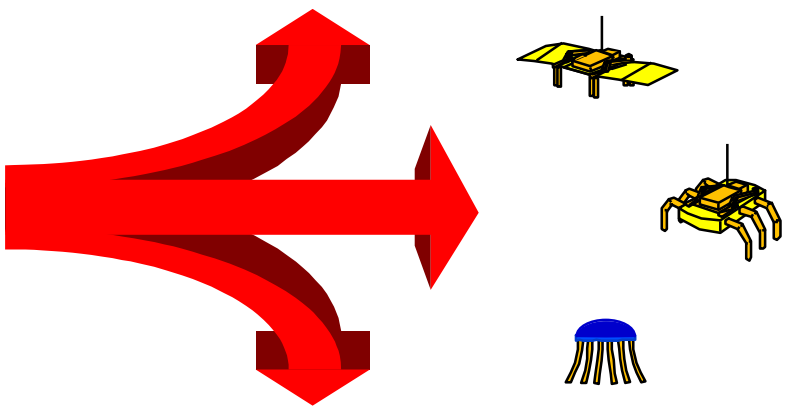
Evolvable Architecture

- A group of FPGA under the control of a microprocessor:



EHW Concept

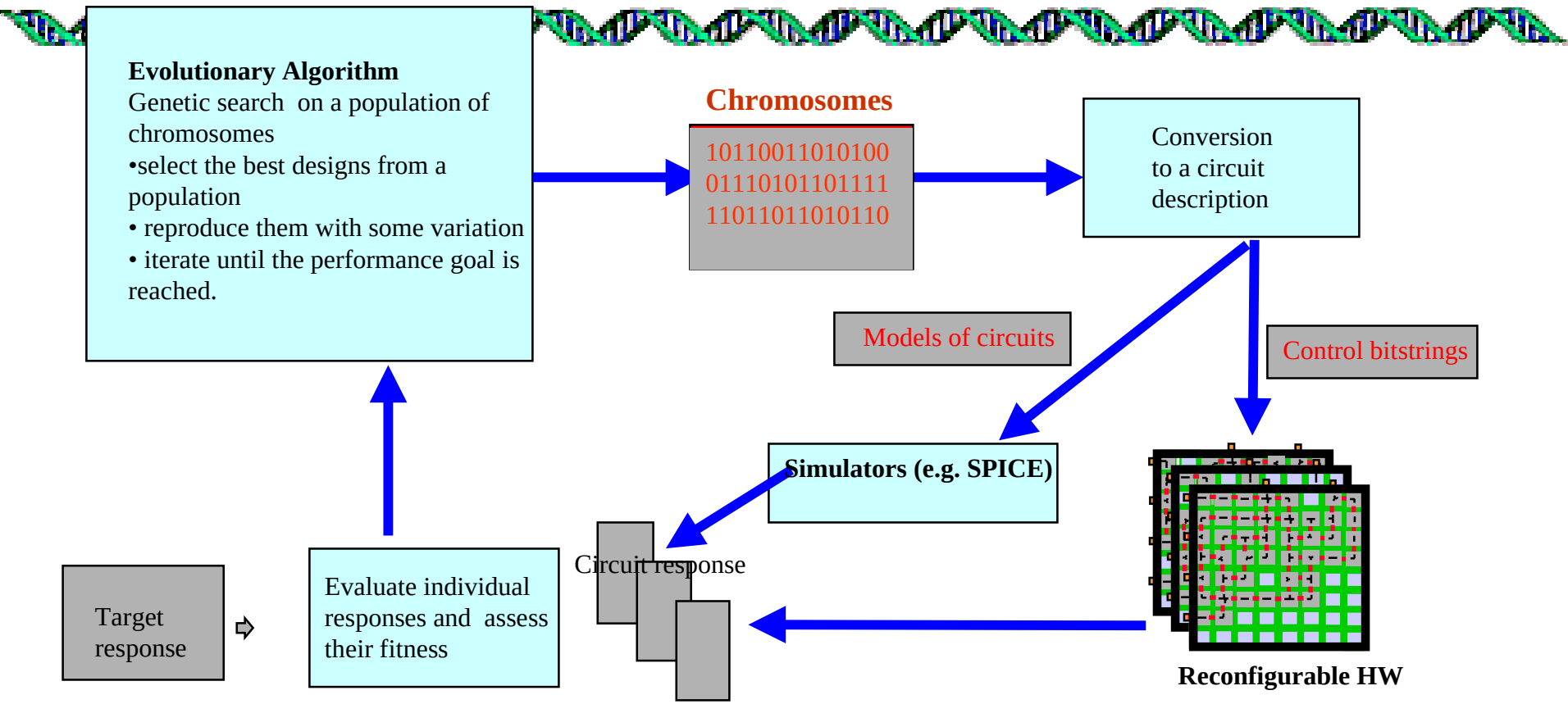
- Create new hardware devices:



Accelerated evolution, ~ seconds for electronics

- **Potential designs and implementations compete;**
  - **the best ones are slightly modified to search for even more suitable solutions**

GA Process



Potential designs/implementations compete; the best ones are slightly modified to search for even more suitable solutions

Powerful Ideas

- Reversible Logic
- Evolutionary Algorithms
- Cellular Automata
- FPGA
- Reconfigurable Hardware
- Hardware Evolution

SUMMARY

- Genetic Programming Structure
- Additional GP Operations
- Hardware Evolution

Summary

- Field of study in Machine Learning.
- Created by John Koza in 1992.
- Save time while creating better programs.
- Based on the principles of genetics.
- Symbolic Regression/Circuit Design.
- Future uncertain.

Student Projects

- Both Genetic Programming and EHW offer interesting opportunities for projects.
  - Select a software algorithm and try to produce it using GP
  - Target a hardware device and use EWH to design it

Possible Quiz Questions

- Remember that even though each quiz is worth only 5 to 10 points, the points do add up to a significant contribution to your overall grade

- If there is a quiz it *might* cover these issues:
  - What is the best method to produce an initial population for a GP system?
  - What is reconfigurable hardware?
  - What does EHW stand for?