NSTC

Genetic Algorithms
GA application : chip fabrication

• chip manufacture at the sub-micron level
  ▪ so small, lines so close, that get “crosstalk” between lines
  ▪ can reduce effect by running at a lower voltage
  ▪ but then the transistors are driven less, so harder to meet timing constraints

• post fabrication adjustment to timing characteristics
  ▪ programmable (fusible) delays in clock lines
  ▪ use GA to smooth the variance, per chip
  ▪ can run faster, meet timing constraints
  ▪ so can run at lower power
  ▪ or increase chip yields at given power

spacecraft antennae

- evolved using a LOGO-like (turtle graphics) antenna constructing programming language

“Antenna ST5-3-10 achieves high gain (2-4dB) across a wider range of elevation angles. This allows a broader range of angles over which maximum data throughput can be achieved. Also, less power from the solar array and batteries may be required.”

-- [http://ic.arc.nasa.gov/projects/esg/research/antenna.htm]
basic genetic algorithm

initialise population

breed new generation

breed offspring by crossover, mutation

evaluate population

select next generation

stopping criterion?

no

yes

finished
schemas

• consider binary chromosomes
  ▪ eg 10100, 11001

• a schema is a set of “similar” chromosomes
  ▪ plural: “schemas”, or “schemata”
  ▪ eg 1*100 = {10100, 11100}, 1*10* = {10100, 11100, 10101, 11101}
    ▪ the * is a “don’t care” symbol
    ▪ the 1 and 0 are “fixed” symbols

• there are $2^n$ possible binary strings of length $n$: $\{0,1\}^n$
• there are a total of $2^{2^n}$ subsets of these strings: $2^{\{0,1\}^n}$
• there are only $3^n$ possible schemas of length $n$: $\{0,1,*\}^n$
  ▪ so not every subset is definable as a schema

hyperplanes (1)

- a schema defines a \textbf{hyperplane} in the \textit{n}-dimensional string space
- the number of *s = the dimension of the hyperplane

\[
0*0 = 0(0+1)0 = \{000,010\}
\]
\[
**1 = (0+1)(0+1)1 = \{001,011,101,111\}
\]
• a given binary string of length $n$ is a member of $2^n$ schemas

• a member of all the hyperplanes that contain it

$$(1+*)(1+*)(0+*) = \{110, 11*, 1*0, *10, 1**, *1*, **0, ***\}$$
some definitions

• \( f(x) \) = fitness of string \( x \) in the population

• \( n(S, t) \) = number of instances (strings \( x \)) of schema \( S \) in population at time \( t \), with \( n > 0 \)

• average fitness of instances of \( S \) in population at time \( t \)
  \[ \hat{f}(S, t) = \frac{\sum_{x \in S} f(x)}{n(S, t)} \]

• \( \bar{f}(t) \) = average fitness of population at time \( t \)
evolution of schemas (1)

- $f(x)/\overline{f}(t)$ = number of offspring of $x$ at $t + 1$ (roulette wheel selection)
  - assumption: population large enough for probabilistic reasoning
- for now, ignore effects of crossover and mutation, which can destroy (and create) strings of $S$
- then the expected number of instances of $S$ at time $t + 1$ is the total number of offspring of instances of $S$

$$E\left(n(S,t+1)\right) = \sum_{x \in S} f(x) / \overline{f}(t)$$
evolution of schemas (2)

- this can be rewritten using the average fitness of $S$ as

$$E(n(S, t + 1)) = n(S, t) \frac{\hat{f}(S, t)}{\bar{f}(t)}$$

- so the expected number of instances of $S$ at time $t + 1$ is proportional to the number at time $t$
  - note: the fitnesses are also functions of time

- so, if $S$ is fitter than the average (the “constant” of proportionality is greater than 1), then the number of instances of $S$ increases exponentially
  - $\bar{f}(t)$ will also increase with time (if the search is converging!)
effect of disruption

• however, this growth result assumes no effect of the evolutionary operators

• so, now let’s include the destructive effect of crossover and mutation, to get a lower bound on this expected number, $E$
  - lower bound, because there will be some constructive effects, too
effect of mutation (1)

• $S$ will “survive” a *mutation* provided that it does not occur at a fixed bit
  - **1****1**: survives mutation at “*”
  - **1****1**: destroyed by mutation at “1”

• let $p_m = \text{probability that a single bit is mutated}$
  - assume probability is independent of position in string
  - parameter of the execution

• the probability that a particular bit will not be mutated is $1 - p_m$
  - prob that no bits are mutated = $(1 - p_m)^{\#str}$
effect of mutation (2)

• **order**: $o(S) =$ number of fixed symbols in schema $S$
  - $o(*1*1001) = 5$
  - $o(**1****) = 1$

• the probability that no *fixed* bits will be mutated, hence the probability of $S$ surviving mutation

$$= (1 - p_m)^{o(S)}$$

• note: the probability of survival is highest for small $o$
  - the fewer number of fixed bits, the less likelihood of disruption
effect of crossover (1)

- \( S \) will “survive” a *single point crossover* provided that the crossover point is not between the first and last fixed bits
  - \[**|0**01**\] : survives crossover at “|”
  - \[**0*|01**\] : may be destroyed by crossover at “|”
  - \[**0**01*|\] : survives
    - it *may* also survive between these bits, by chance, depending on the other parent

- let \( p_c = \) probability that a single point crossover is applied to a string \( x \)
  - assume probability is independent of position in string
effect of crossover (2)

- for a string of length \( l \), there are \( l - 1 \) crossover points

- **defining length**: \( d(S) \) = distance between first and last fixed symbols of schema \( S \)
  - \( d(*1**01) = 4 \)
  - \( d(**10**) = 1 \)

- between the first and last fixed bits, the number of crossover points is the defining length \( d(S) \)
effect of crossover (3)

• the probability of the crossover point being between the first and last fixed bits

\[
= \frac{d(S)}{l - 1}
\]

• example: * | * | 0 | * | * | 1 | * : \(d = 3, l - 1 = 6\)
  - half the crossover points are (potentially) disruptive,
  - half are not (for two parents making two children)
effect of crossover (4)

• the probability of being destroyed by crossover is

\[ \leq p_c \frac{d(S)}{l-1} \]

- fuller accounts include probability of both parents having $S$, hence of $S$ surviving crossover

• hence the probability of surviving crossover is

\[ \geq 1 - p_c \frac{d(S)}{l-1} \]

• note: the probability of survival is highest for small $d$
  - the shorter the schema, the less likelihood of disruption
building blocks

- **building blocks** (BBs) = schemas of
  - low order $o$ (best survival of mutation disruption)
  - short defining length $d$ (best survival of crossover disruption)
  - high fitness $f$ ("fitness coefficient" greater than 1)
    - hence what is a BB is a function of the current fitness

- analogy between BBs and (biological) genes
Schema Theorem [Holland 1975]

- putting these survival probabilities into the simple original equation, we get the bound

\[
E(n(S, t + 1)) \geq n(S, t) \frac{\hat{f}(S, t)}{f(t)} \left(1 - p_c \frac{d(S)}{l - 1}\right) \left(1 - p_m\right)^{o(S)}
\]

- (fitness growth x crossover disruption x mutation disruption)

- BBs (fit, short, low order schemas) will have a “growth coefficient” \( k > 1 \)

\[
E(n(S, t + 1)) \geq k n(S, t)
\]

- **Schema Theorem**: BBs increase (roughly) exponentially in the population
Building Block hypothesis [Goldberg 1989]

• **implicit parallelism**: all BBs (all fit schemas) grow in parallel

• what of constructive effects of crossover?
  - crossover can combine strings containing individual BBs into strings containing several BBs

• **Building Block hypothesis**: GA’s power comes from the mechanism of crossover combining BBs to produce better solutions
  - *assumption*: a string with multiple BBs is fitter than a string with fewer BBs
  - mutation – supplies diversity
BB assumption: deception and epistasy

• deception
  ▪ the fit building blocks are not schemas of the global maximum
    ◦ \( f(00) = 7, f(01) = 5, f(10) = 1, f(11) = 9 \)
    ◦ \( f(0^*) = 6, f(1^*) = 5 \), yet solution is \( f(11) \)
  ▪ the small building block stepping stones are deceptive

• epistasy
  ▪ interaction of different genes
  ▪ if highly epistatic, there are no small building blocks to act as stepping stones
    ◦ multiple genes must have the right values to give high fitness
**BB hypothesis : cGAs**

- **cGAs (Compact Genetic Algorithms)** represent the population as a probability vector, rather than as a set of bit strings
  - the $i$th component of the probability vector gives the probability that the $i$th bit of an individual’s string is a 1
- **very compact representation**
  - store a single vector of numbers, v. a population of bit strings
  - suitable for small memory, real time evolution in hardware
  - works (surprisingly?) well in practice
- **does not rely on BB hypothesis**
  - probability vector loses information about correlations between “genes”, or BBs
  - “1-bit optimiser”

---

$k$-armed bandit theory

- “fruit machine”, “slot machine” = a “one-armed bandit”
  - because it has one arm, and it steals your money!
- consider a $k$-armed bandit, where each arm has an average payoff, per go (per “trial”) of $\mu_i$, with a variance $\sigma^2_i$
- you are allowed $M$ trials, but you do not know these payoffs
- what is your optimum strategy for assigning trials to arms, to maximise your total return (including during the trials)?
  - a classic problem in statistics
2-armed bandits and GAs

• Holland solved this analytically for $k = 2$
  ▪ as the trials progress, exponentially increase the probability of selecting the better-performing arm over the poorer-performing arm

• the GA implicitly adopts this strategy, via its “implicit parallelism” – so might it be optimal?
  ▪ it’s not that simple!
  ▪ $k$-armed bandit’s arms are independent variables, but schemas interact/overlap, and so the GA does not sample them independently

• but it is a suggestive approximation
GA analyses

- Schema Theorem
  - concise intuitive descriptions
  - coarse grained: relatively crude averaging over schemas

- Markov chain models
  - future of process depends only on the present state, not the past history
    - search algorithm depends only on current state, not search trajectory
    - EA progression depends only on current population
  - more detailed fine grained models
  - useful for convergence results
  - lead to computationally intractable equations
GA parameter tweaking

- initialisation
  - usually random individuals, spanning the parameter space
- population size
  - a few 100 to a few 1000
- mutation rates, genetic operators
  - lots of variants possible!
  - adaptively evolve these parameters, too
- diversity
  - loss of diversity may cause premature convergence to a non-optimal solution
co-evolution

• as a way of increasing selection pressure
• as the solution improves, change other parameters
  ▪ evolve (change) fitness landscape from initial one that is easier to search (or cheaper to calculate) to one closer to real problem
  ▪ co-evolving competing populations
    ❖ each forms the other’s landscape
  ▪ co-evolve population and test cases
    ❖ as the solution improves, the test cases get harder
  ▪ co-evolve population of simulated robots, and fitness simulator
    ❖ simulator need cover only the aspects of solution being explored
• Hillis : coevolved a sorting algorithm and its test cases
  ▪ fitness of test cases based on how *poorly* they were sorted by the algorithm

niching and parallelism

• parallel implementation

• subpopulations evolve in parallel with little interaction
  ▪ non-competing “species”
  ▪ helps maintain diversity between the sub-populations

• interaction controlled at a few points
  ▪ limited amount of competition

• more parameters to play with
  ▪ number/size of sub-populations, migration rate and policy, …
  ▪ vary parameters across populations
Genetic Programming

- GP is a kind of genetic algorithm where each chromosome is a computer program
- execute each program to evaluate its fitness
  - the program may itself be the artefact of interest
    - execute it on test cases to see how well it performs its task
  - often the artefact of interest is built, or “grown”, by executing the program
    - eg, a program that draws a circuit diagram
      - evaluate the resulting circuit by, eg, simulation
    - a second order encoding
      - can include iteration, choice, etc, as well as terminal values
  - so there can be a big distance between genotype (program) and phenotype (result of execution)
sample applications

- Koza, *Genetic Programming*, vols I-IV
  - “human-competitive” results
  - patented electrical circuits

- quantum circuits
  - GP trees encode rules for describing circuits

- control applications
  - robot motor controls
    - the program itself is the artefact of interest

- clustering applications
  - data mining, rule induction, bioinformatics
bioinformatics applications

• vast amounts of biological data being generated
  ▪ gene sequencing
  ▪ gene expression data (which genes are affecting which others)
  ▪ protein data
  ▪ metabolic pathways and networks
  ▪ complex diseases
  ▪ ...

• this data is
  ▪ noisy
  ▪ very complicated
    ✷ mechanisms evolved over billions of years
    ✷ highly non-linear

• GP can evolves programs (rules) that “explain” the data
  ▪ in terms that make sense to biologists
GP : representing programs (1)

• the program is usually represented as a tree
  ▪ the parse tree of the programming language
    ▪ mathematical operators for function evaluation
    ▪ “turtle graphics” language for drawing circuits
    ▪ ...

• variants use lists (eg, assembly language models), graphs, and other structures
  ▪ with suitably adapted operators

• these trees are of variable sizes
  ▪ compared to fixed length bit string chromosomes
  ▪ usual to put a limit on the depth
GP : representing programs (2)

• syntactically correct programs
  ▪ design the language to maximise syntactically correct trees
    ♦ Lisp was an early favourite choice, for this reason
  ▪ constrain evolution operators, to ensure correct generation
  ▪ “fix-up” trees before evaluation

• GP computational metaphor, whilst keeping the underlying idea of evolution = inheritance, variation, selection, has moved far from the biological model of linear chromosomes
  ▪ since its representation (tree chromosomes) is far from biology, maybe its variation operators should be similarly far from biology? Use something other than crossover?
EAs: unexpected solutions

• some apocryphal(?) tales about evolving a (simulated) robot (brain and body) to “travel as far as possible”:
  ▪ One solution was very tall. All it did was fall over. The way fitness was defined, this counted as going a very long horizontal distance.
  ▪ One solution found an error in the physics simulation code, and learned to “fly”.
  ▪ One robot started vibrating very fast, and generated a divide-by-zero. This counted as infinite fitness, because it had “travelled” infinitely far.

• These kind of things happens *all the time* with EAs!