Genetic Algorithms and Genetic Programming

Lecture **5**: (9/10/09)

The schema theorem (II)

Michael Herrmann

michael.herrmann@ed.ac.uk, phone: 0131 6 517177, Informatics Forum 1.42

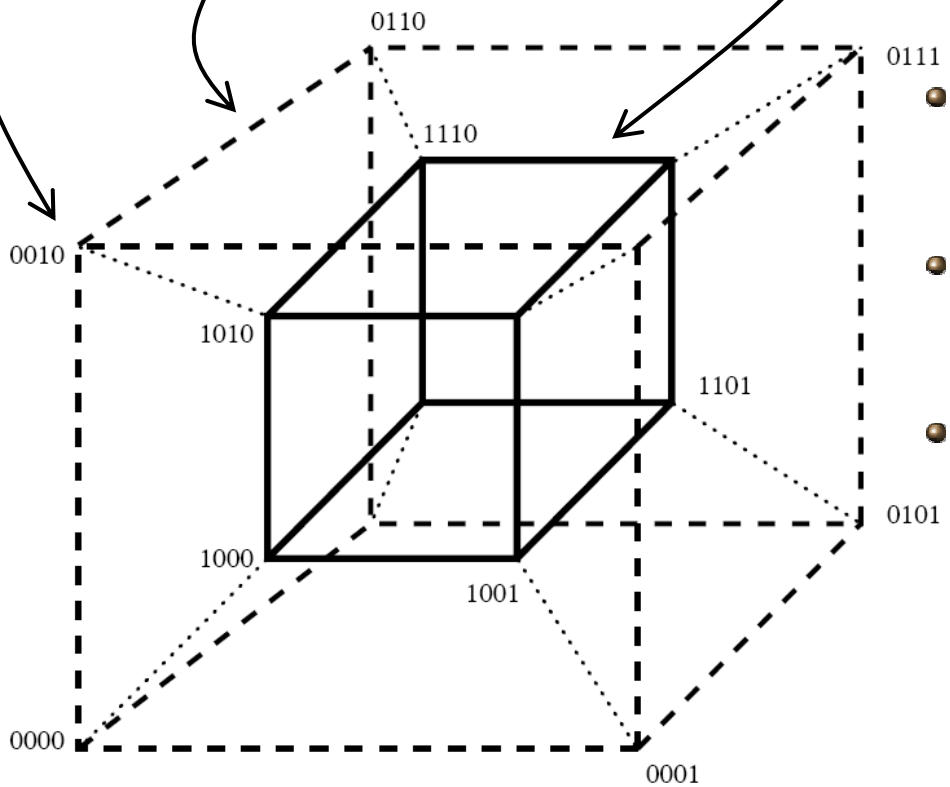
Search spaces as Hypercubes

Solution "*c*" in $\{0,1\}^L$

e.g. *c=(0,0,1,0)* for *L=4*

(1,,*,*) denotes a cube*

(0,,1,0) denotes an edge*



- A schema is a string containing wildcards "*" (but not only asterisks)
- The order of the schema is the number of bits that are actually there, e.g. **01***1 is a schema of order 3 (and length 8)
- Each chromosome is a corner of the hypercube
- Schemata correspond to hyperplanes in the hypercube
- Each chromosome is part of 2^L-1 hyperplanes. There are 3^L-1 different schemata (not counting the schema of order 0: ** … *)

How Do GAs Work?

The Schema Theorem (J. Holland, 1975)

GOAL:

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

START: $f(s_i,t)$ fitness at time t of solution s_i

$m(s_i,t)$ is the number of copies of s_i in the population at time *t*

$\bar{f}(t)$ is the average fitness of the population at time *t*

$$E(m(s_i,t+1)) = m(s_i,t)\frac{f(s_i,t)}{\Sigma_j f(s_j,t)}P$$

P denotes the population size

$E(\cdot)$ is the expected value $\frac{f(s_i,t)}{\Sigma_j f(s_j,t)}$ is the probability of selecting s_i

Writing $\Sigma_j f(s_j,t)/P$ as $\bar{f}(t)$

$$E(m(s_i,t+1)) = m(s_i,t)\frac{f(s_i,t)}{\bar{f}(t)}$$

proportion of the population that is s_i

$$\text{Prop}(s_i,t+1) = \frac{m(s_i,t)}{|P}\frac{f(s_i,t)}{\bar{f}(t)}$$

So above-average-fitness strings get more copies in the next generation and below-average-fitness strings get fewer.
Suppose s_i has, and continues to have, an above-average-fitness of $(1+c)\bar{f}$.
Then for c>0

$$E(m(s_i,t+1)) = m(s_i,t)\frac{\bar{f}+c\bar{f}}{\bar{f}} = (1+c)\,m(s_i,t)$$

If we have $m(s_i,0)$ copies at $t=0$, then we have $m(s_i,t) = (1+c)^t\,m(s_i,0)$
–this gives **exponential growth** – and decay for $c<0$.
–So fit solutions come to dominate (crossover and mutation ignored here)

RESTART:

Fitness of Schemata

If solutions s_i, s_j, s_k all sample the same schema H, we can calculate the **average fitness** $\hat{u}$ **of** H from the fitnesses of the m solutions that sample it:

$$\hat{u}(H,t) = \frac{1}{m(H,t)} \sum_{s_i \in H} m(s_i,t)\, f(s_i)$$

Given $m(H,t)$ and $\hat{u}(H,t)$, can we calculate $m(H,t+1)$?

$\hat{u}(H,t)$ is the average fitness of H at time t
$m(H,t)$ is the number of instances of H at time t
$\bar{f}(t)$ is the average fitness of the population at time t

How many instances of H will be present in P after selection?

Proportion:

$$\text{Prop}(H) = \frac{m(H,t)}{P} \frac{\hat{u}(H,t)}{\bar{f}(t)}$$

Selection of Fit Schemata – Example

What happens when we select and duplicate strings on the basis of fitness?

Does the distribution of points in each hyperplane change accordingly?

Example: Suppose s_i, s_j and s_k sample H, i.e. $s_i \in H, \quad s_j \in H, \quad s_k \in H,$

Suppose average fitness of population $= 1.0$ and

$$f(s_i,t) = 2.0, m(s_i,t) = 2$$
$$f(s_j,t) = 2.5, m(s_j,t) = 2$$
$$f(s_k,t) = 1.5, m(s_k,t) = 2$$

So, using earlier formula for samples:

$$E(m(s_i,t+1)) = 2 \times \frac{2.0}{1.0} = 4$$

$$E(m(s_j,t+1)) = 2 \times \frac{2.5}{1.0} = 5$$

$$E(m(s_k,t+1)) = 2 \times \frac{1.5}{1.0} = 3$$

All are fitter than average, all increase their number in the population.

For schema:

At t, $m(H,t) = 6, \hat{u}(H,t) = 2.0$, and

$$E(m(H,t+1)) = m(H,t)\frac{\hat{u}(H,t)}{\bar{f}(t)} = 6 \times \frac{2.0}{1.0} = 12$$

So number of samples of this hyperplane increases also, but ...

Disruption of Schemata

Crossover and mutation are both disruptive and constructive with regard to schemata. Consider only disruptive effects:

Crossover:

1 1 * * * * * * * Probability of disruption in crossover is?

1 * * * * * * * 1

Mutation:

1 1 0 0 1 0 0 1 1 1 0 1 * * Many disruptive possibilities

1 1 * * * * * * 1 1 * * * * * Only 4 disruptive possibilities

Schema Jargon

Number of defined bits is the **order** $o(H)$ of the schema H:

1 1 * * 1 1 0 * order 5
* * * * 1 1 0 * order 3

Defining length is the distance $d(H)$ between the first and last bits of the schema:

(i.e. number of potential cuts)

1 1 * * 1 1 0 * defining length 6
* * * * 1 1 0 * defining length 2

i.e. bit position of last $0/1$ − bit position of first $0/1$.

Disruptive Effects of Crossover

- 1-point crossover, probability p_c.
  $H = * * 1\ 0 * 1 * *$ $\quad d(H) = 3$
- $d(H)$ is the defining length of H.
- In a single crossover, there are $l - 1$ crossover points:
  1 0 1 0 0 1 0 0 $\quad$ 7 crossover points
- Of these, $d(H)$ points will disrupt the schema.

$$\Pr(\text{disruption}) = p_c \frac{d(H)}{l-1}$$

e.g. Suppose $p_c = 0.8, d(H) = 3, l = 100$ $\quad$ Pr(disruption) $= 0.8 \times \frac{3}{100} = 0.024$

- Better survival if $d(H)$ low.

Disruptive Effects of Mutation

- Single-point mutation, probability of applying to each bit in turn $= p_m$

$o(H)$ is the order of H

$H = * * 1\ 0 * 1 * *$ $\quad o(H) = 3$

$H = 1\ 1\ 0\ 1 * 1 * 1$ $\quad o(H) = 6$

- Probability that a bit survives is $1 - p_m$
- Flipping a defined bit always disrupts a schema, so the probability that the schema survives is:

$$\Pr(\text{survival}) = (1 - p_m)^{o(H)}$$

Best chances for surviving crossover and mutation when $d(H)$ **and** $o(H)$ **are low.**

STOPOVER:

first component of **the Schema Theorem**

after P spins : $$E(m(H,t+1)) = m(H,t)\frac{\hat{u}(H,t)}{\bar{f}(t)}$$

other parts of the Schema Theorem:

$$\Pr(\text{surviving crossover}) = 1 - p_c\frac{d(H)}{l-1}$$

l : total string length

$$\Pr(\text{surviving mutation}) = (1-p_m)^{o(H)}$$

p_m: flip mutation rate

$$E(m(H,t+1)) = m(H,t)\frac{\hat{u}(H,t)}{\bar{f}(t)}(1 - p_c\frac{d(H)}{l-1})[(1-p_m)^{o(H)}] \quad (?)$$

GOAL!

The Schema Theorem

schemata can be **created** through crossover and mutation. So we need a $\geq$.

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

Highest when

- $\hat{u}(H,t)$ is large – fit
- $d(H)$ is small – short
- $o(H)$ is small – small number of defined bits

Schema Theorem in words: short, low-order, above average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.

The Building Block Hypothesis

During crossover, these **"building blocks"** become exchanged and combined.

So the Schema Theorem identifies the building blocks of a good solution although it only addresses the disruptive effects of crossover (and the constructive effects of crossover are supposed to be a large part of why GAs work). How do we address these constructive effects?

Building Block Hypothesis: a genetic algorithm seeks optimal performance through the juxtaposition of short, low-order, high-performance schemas, called the building blocks.

Crossover combines short, low-order schemas into increasingly fit candidate solutions.

- short, low-order, high-fitness schemas
- "stepping stone" solutions which combine H_i and H_j to create even higher fitness schemas

Experimental Evidence

The Building Block Hypothesis is a **hypothesis** – so we can do an experiment to test it.

Experiment: use a problem which contains explicit building blocks and observe the population. Do the building blocks combine to give good solutions in the way the BBH predicts?

Mitchell, Forrest, Holland set up such a problem, using **Royal Road** functions.

Details: Mitchell, Chapter 4, pp127–133

Define fitness in terms of particular schemas: substrings that, if present in population, ought to be combinable into the optimal solution. They should lay out a "Royal Road" to the global optimum.

The first RR function R_1 is defined using a list of schemas s_i. Each s_i has a fitness coefficient c_i. The fitness $R_1(x)$ of some bit string x is given by:

$$R_1(x) = \sum_i c_i \delta_i(x), \ \delta_i(x) = \begin{cases} 1 & \text{if } x \in s_i \\ 0 & \text{otherwise} \end{cases}$$

Royal Road Functions

Simple example using 16 bits. Suppose:

$$s_1 = 1\ 1\ 1\ 1\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *$$
$$s_2 = *\ *\ *\ *\ 1\ 1\ 1\ 1\ *\ *\ *\ *\ *\ *\ *\ *$$
$$s_3 = *\ *\ *\ *\ *\ *\ *\ *\ 1\ 1\ 1\ 1\ *\ *\ *\ *$$
$$s_4 = *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ 1\ 1\ 1\ 1$$

and suppose $c_1 = c_2 = c_3 = c_4 = 4$ and

$$s_{\text{opt}} = 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1$$

Then $R_1(\text{opt}) = 16$

Take the string $1111\,0100\,1001\,1111$. It samples/matches s_1 and s_4.

So $\delta_1(x) = \delta_4(x) = 1$ and $\delta_2(x) = \delta_3(x) = 0$.

And $R_1(1111010010011111) = 8$.

Royal Road Functions

Several Royal Road functions defined in terms of different combinations of schemas, with building blocks at different levels, e.g. 4 contiguous 1s, 8 contiguous 1s, 16 contiguous 1s, etc.

Try to evolve the string with all 1s and compare performance of GA against a number of hill-climbing schemes:

- Steepest-ascent hill climbing (SAHC)
- Next-ascent hill climbing (NAHC)
- Random-mutation hill climbing (RMHC)

Will the GA do better?

Hill Climbing

Steepest-ascent hill climbing (SAHC)

1. Let current-best be a random string.
2. From left to right flip each bit in the string. Record fitness of each one-bit mutant.
3. If any mutant is fitter than current-best, set current-best to fittest mutant and goto 2.
4. If no fitness increase, save current-best and goto 1.
5. After N evaluations return fittest current-best.

Next-ascent hill climbing (NAHC)

1. Let current-best be a random string.
2. From left, flip each bit i in string. If no fitness increase, flip bit back. If fitness increase, set current-best to new string and continue mutating new string from 1 bit after the bit at which fitness increase was found.
3. If no fitness increase, save current-best and goto 1.
4. After N evaluations return fittest current-best.

Random-mutation hill climbing (RMHC)

1. Let current-best be a random string.
2. Flip a random bit in current-best. If no fitness decrease, set current-best to mutated string.
3. Repeat 2. until optimal string found or N evaluations completed.
4. Return fittest current-best.

See Mitchell p.129 for these algorithms.

Results

200 runs	GA	SAHC	NAHC	RMHC	
Mean	61,334	$>$ max	$>$ max	6,179	No. evaluations to
Median	54,208	$>$ max	$>$ max	5,775	find optimal string

max = 256000

Why did the GA do worse than RMHC? When do GAs perform well?

- By Markov chain analysis, RMHC's expected time is $\sim$6549 evaluations. OK.
- What's going wrong in the GA? Larger combinations of s_i in the GA get broken up by crossover and disrupted by mutation.
- And the GA suffers from **hitch-hiking**: once an instance of a high-fitness schema is discovered, the "unfit" material, especially that just next to the fit parts, spreads along with the fit material. Slows discovery of good schemas in those positions.

Hitch-hiking

- Hitch-hikers prevent independent sampling particularly in those partitions falling between two closely-spaced others, e.g. hitch-hikers with s_2 and s_4 (0s near the ends of s_2 and s_4) drowned out instances of s_3. So sampling in s_3 region not independent of sampling in s_2 and s_4 regions.

- Early convergence to wrong schemas in a number of partitions of the string limits the effectiveness of crossover.

- Sampling of the different regions is not independent.

See Mitchell Fig. 4.2 p.133 for hitch-hiking effect.

Analysis

- Easy problem, no local maxima (so hill-climbing works, RMHC is systematic).
- GA will out-perform HC on parallel machines (why?)
- GA is not sampling evenly; schema theorem does not hold. If partitions *are* sampled independently, schema theorem ought to hold.

Mitchell proposes an idealised GA (IGA):

– sample a new string s_i uniform-randomly

– if s_i contains a new desired schema, keep it and cross it over with previous best string to incorporate new schema into the solution

- IGA aims to sample each partition independently and tend to keep best schemas in each partition – **static Building Block Hypothesis**

- It works, and it's N times faster than HC
- IGA unusable in practice (why?) but gives us a lower bound on time GA needs to find optimal string
- In IGA each new string is an independent sample, whereas in RMHC each new sample differs from the previous by only one bit – so RMHC takes longer to construct building blocks

So we have some clues as to when GAs will do well...

When Do GAs Do Better Than Hill-climbing?

To act like an ideal GA and outperform hill-climbing (at least in this sort of landscape) need

- Independent samples: big enough population, slow enough selection, high enough mutation, so that no bit-positions are fixed at same value in every chromosome
- Keeping desired schemas: strong enough selection to keep desired schemas but slow enough selection to avoid hitch-hiking
- We want crossover to cross over good schemas quickly when they're found to make better chromosomes (but we don't want crossover to disrupt solutions)
- Large N/long string so speedup over RMHC is worth it.

Not possible to satisfy all constraints at once – tailor to your problem

Where Now?

- Schema theorem starts to give us an idea of how GAs work but is flawed $\rightarrow$ need better mathematical models of GA convergence . . .
- . . . but these better models don't make our GA go faster. Can we fix it empirically? Fix what, exactly?
- 1. Standard GA finds good areas, but lacks the **killer instinct** to find the globally best solution
- 2. Standard crossover often disrupts good solutions late in the run
- 3. Binary representations of non-binary problems often slow the GA down rather than allowing it to sample more freely. The "Hamming Cliff".

The Killer Instinct

Want to get from good to best individuals.

[De Jong] Say range of payoff values is [1,100]. Quickly get population with fitness say in [99,100]. Selective differential between best individuals and rest, e.g. 99.988 and 100.000, is very small. Why should GA prefer one over another?

- Dynamically scale fitness function as a function of generations or fitness range
- Use rank-proportional selection to maintain a constant selection differential. Slows down initial convergence but increases killer instinct in final stages.
- Elitism. Keep best individual found so far, or, selectively replace worst members of population

Aim is to shift balance from **exploration** at start to **exploitation** at end.

The Killer Instinct and Memetic Algorithms

- Hill-climbing local neighbourhood search is a fast single solution method which quickly gets stuck in local optima (cf. SAHC, NAHC)
- Genetic algorithms are a multi-solution technique which find good approximate solutions which are non-local optima
- Hence: try applying local search to each member of a population after crossover/mutation has been applied. We might find locally better solutions, and if near the end of run find the best/optimal solution
- GA + LS = Memetic Algorithm

Making It Better

- Change the crossover probability towards the end of the run
- Start the GA from good initial positions: **seeding**. If you know roughly where a solution might lie, use this information.
- Use a representation close to the problem: does not have to be a fixed-length linear binary string – avoid the Hamming Cliff
- Use operators that suit the representation chosen, e.g. crossover only in specific positions
- Run on parallel machines: island model GA (evolve isolated subpopulations, allow to migrate at intervals)

Reading: Mitchell Chapter 4