Training Large GAGP Problems, Schema Theorem Revisited, Evolution Strategies

- Training Large GAGP Problems: RSS and DSS
- Schema Theorem Revisited and Beyond
- What Evolution Strategies are
- Simplest Evolution Strategy (1+1)-ES
- (μ + λ) and (μ, λ) ESs – multimembered populations
- ESs vs. GAs
- A Games/ALife Example (MENL)

Training Large GAGP Problems

Usually wish to run GAs/GPs on several instances of a problem, e.g. lettuce classifier on several batches of lettuces

Need a lot of data to train them

Fitness evaluations take a long time - most compute-intensive part of the GA/GP.

How to reduce amount of data needed for training? How to maintain performance on unseen data/problems?

- Choose subsets of training data
- Subsets must be representative of the whole population
- Allow us to generalise

Random Subset Selection RSS

How to choose subsets from training data?

- Choose a smaller subset, select uniform-randomly from training set
- Always choose same size of subset, each generation

Test out members of GA/GP population on this subset of training data (rather than on full set of available training data) to get a fitness score for each chromosome
Dynamic Subset Selection DSS

Alternative method of subset choice

- Focus on difficult cases; e.g. in classification problem those that are frequently misclassified – could be in a GP symbolic regression problem

- Also focus on those items that haven’t been in a subset for a while – age

DSS Details

- Each generation, assign a weight to each piece of training data $i$:
  \begin{align*}
  W_i &= D_i^d + A_i^a \\
  \end{align*}

  $D$ is difficulty, $A$ is no. generations since last selected, $a$ and $d$ are constants (parameters of problem)

- Probability of $i$ being selected for subset of size $S$:
  \begin{align*}
  P_i &= \frac{W_i S}{\sum_j W_j}
  \end{align*}

  where $j$ are all members of training set.

If $i$ selected, set $D_i = A_i = 0$, else $D_i$ unchanged, $A_i = A_i + 1$.

$D_i$ starts at 0; each time $i$ is picked for a subset but is misclassified, $D_i = D_i + 1$. Age $A_i$ starts at 0.

Select members of subset according to $P_i$. Gives a subset of size $S$ on average.

Eventually all members of training set are picked.

Evaluate current generation on this subset, not on all training set.

Ross and Gathercole on Thyroid Data

Used GP to classify thyroid data (healthy thyroid, unhealthy thyroid, based on values of a number of parameters)

Training set of 2000, $S = 400$ gave good performance

- DSS outperformed RSS; both outperformed regular GP – need fewer generations to solution (i.e. best performance on test set of data), fewer tree evaluations

- Performance drops off below some value of $S$ (200) – need to select sufficient training data to get a representative selection of examples

- Performance doesn’t increase much above $S$ of 400 – don’t overfit to training data

Can also add in parsimony term to fitness to penalise large trees – speeds up fitness evaluation.

(Link to Ross and Gathercole on webpage)
What Process is the Schema Theorem Describing?

Individual chromosomes sample $2^l$ schemas each. So calculate fitness for $N$ members of population, but get an estimate of fitnesses of $2^l$ schemas.

**Selection** focuses search on areas of space with above-average fitness (because we get exponentially increasing nos. of schemas that are of above-average fitness). (We may find it hard to differentiate between them – killer instinct.)

**Crossover** puts together high fitness building blocks (but often disrupts good solutions late in the run – affects killer instinct).

**Mutation** makes sure genetic diversity is not lost (affects killer instinct).

Holland: two-armed bandit problem. After many trials, we start to build up a picture of the payoff of each arm. To optimise the amount of payoff as we’re learning (“online”) it is best to exponentially increase the probability of choosing the arm that, from our estimates, appears to be best.

Gillian Hayes  
GAGP Lecture 17  
24th November 2008

Optimal Online Learning

So if we cast GAs as an online learning problem, they are following the optimal **online learning strategy** – by exponentially increasing the number of samples of a schema in proportion to its average observed fitness. This seems good!

**BUT**

Suppose schema 111***** has fitness 2, 0******* has fitness 1, and the rest have fitness 0.

Then 1****** has fitness 0.5 and 0******* has fitness 1.

But with a GA, 1****** will dominate population quickly with fitness close to 2, in form of many instances of 111*****

This doesn’t follow the schema theorem.

Gillian Hayes  
GAGP Lecture 17  
24th November 2008

GAs and Schemas

How, Why?

In bandit problems, the payoff from one arm is independent of the payoff from another arm. In GAs, the schemas are not independent of each other – we’re not sampling them independently. So measured average fitness is not necessarily the same as the true average fitness.

And conversely to above example, we could end up in a good part of the search space, but not the best (suppose now 0****1*** had fitness 3...)

Also, the fitness of 1******* has high variance – our GA cannot make an accurate estimate of its fitness from a few samples.

So **non-uniform sampling** and **high fitness variance** prevent the schema theorem from giving an accurate prediction/explanation of the GA’s performance.

Gillian Hayes  
GAGP Lecture 17  
24th November 2008

Beyond the Schema Theorem

- We would still like to know how the search proceeds through the search space and how many instances of a given schema there will be at the end of the search.
- Other models of GAs account for finite populations and so non-uniform sampling (Markov chains) and constructive effects of crossover and mutation – but calculations intractable for even small populations.
- Statistical mechanics of spin glasses: a binary chromosome is a string of spins (spin up and spin down, +1 and -1 or 1 and 0). Can model how one chromosome can change into another chromosome after selection and mutation by relating the fitness of the chromosome to a “spin energy” – and eventually model the distribution of chromosomes throughout the search space as a function of generation number – so we don’t know about individual chromosomes but we know about large collections or ensembles of them.
- And, of course, non-binary alphabets.

Gillian Hayes  
GAGP Lecture 17  
24th November 2008
Evolution Strategies (ES)

- Originally Rechenberg and Schwefel (1960s) in TU Berlin (so ~ “the European GA”)
- Real-valued function optimisation – improving the shape of jet nozzles
- An individual is a vector of real numbers; and we have a vector of mutation variances (one for each gene)
- Mutate each gene by an amount drawn from a normal distribution whose variance is given by the corresponding position in the mutation-variance vector
- If mutated child fitter than parent, replace parent
- Adapt the mutation variances according to some rule

(1 + 1)-ES: one parent produces one child

Suppose we are trying to minimise $f(x)$

- Initial individual is $x = (x_1, x_2, \ldots, x_n)$, e.g. (0.1, 0.7, 0.4, 0.2)
- Initial variance is $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, e.g. (0.01, 0.3, 0.2, 0.05)
- Mutate: $y_i = x_i + N(0, \sigma_i)$ for each $i$
- If $f(y) < f(x)$ then $x = y$, else $x = x$ and mutate again
- Until termination condition reached

How do we vary $\sigma$?

Rechenberg analysed some simple problems: what value of $\sigma$ gives fastest convergence rate?

- Rule of thumb: ratio $p$ of successful mutations (i.e. child fitter than parent) to all mutations should be 1:5 – the “1:5-rule”
- If ratio is greater, e.g. 1.10, then mutation may be too exploitative (steps are too small) and $\sigma$ should be increased
- If ratio is smaller, then mutation may be too explorative (too big steps) and $\sigma$ should be decreased

$\sigma = \sigma/c$ if $p < 1/5$ where $c = 0.85^{1/n} < 1$, $n$ is no. dimensions in problem

$\sigma = \sigma.c$ if $p > 1/5$

unchanged if $p = 1.5$

Example

Maximise $f(x_1, x_2) = 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2)$

$-3.0 \leq x_1 \leq 12.0$ and $4.0 \leq x_2 \leq 6.0$

Individual is $x$, variance $\sigma$, e.g. (5.3,4.9) and (1.0,1.0)

Mutation: $y_1 = x_1 + N(0,1.0) = 5.3 + 0.4 = 5.7$

$y_2 = x_2 + N(0,1.0) = 4.9 - 0.3 = 4.6$

Fitness of $x = 18.38$, fitness of $y = 24.85$ (and still within bounds)
So replace $x$ by $y$ (we are maximising, child is fitter than the parent)

Real example: $x$ could be a vector representing the radii of a jet nozzle at equally-spaced points along its axis (cylinder: all radii the same; ornate chair leg shape: all radii different), fitness would be amount of fluid flowing through nozzle
\((\mu + \lambda)\)-ES, \((\mu, \lambda)\)-ES

- Population of \(\mu\) individuals, they produce \(\lambda\) offspring.

- In “+” cut down joint population of \((\mu + \lambda)\) members to \(\mu\) for next generation: choose the best \(\mu\) individuals

- In “,” cut down child population of \(\lambda\) members to \(\mu\) for next generation (so \(\lambda > \mu\)): choose the best \(\mu\) individuals

Crossover is applied to two parents: at each gene (\(x\) and \(\sigma\)) choose value from one or other parent OR average parents’ values

---

**Self-Adapting Mutation**

Can also allow the mutation parameters to adapt

This allows the ES to search along the axes of the space, with steps of different size along the individual axes – step size governed by \(\sigma\)

Modern ESs introduce another parameter \(\alpha\), a set of rotation angles, that allow mutation to search along arbitrary directions in the search space

---

**ESs vs GAs**

- ES originally designed for numerical optimisation, GAs for discrete and real domains

- Both maintain populations, both use “survival of the fittest”

- ES representation real-valued vectors, (canonical) GAs use binary vectors

- Selection in GAs allows even the weakest members to go into the next generation. In ESs the weakest members are replaced. In GAs, best individual may be selected several times, in ES no repetitions

- In ESs, selection follows mutation/crossover; vice versa in GAs

---

- In ESs, genetic operators self adapt, in (canonical) GAs they are fixed

- Constraints: in GAs, if individual goes out of limits, impose a high penalty in fitness function. In ESs, don’t add it to the new population, and if there are many such illegal offspring the control parameters will adapt

- BUT, each does borrow from the other
Example of ESs in Games/ALife

Multi-agent based evolutionary ANN (artificial neural network) for general navigation in unknown environments – Fang Wang, UEd

Task: Build a controller for an “artificial creature” (grid world)

- Each creature has a multi-agent controller
- Each agent is an ANN
- At each time step, the agents get input, e.g. from a visual sensor, and they produce an output, e.g. a movement
- The output of the fittest agent is executed
- If it fails, the output of the next fittest is executed, and so on
- If all agents fail evolutionary learning is triggered
- MENL – Multi-agent Evolutionary Neural-network with Lifetime-learning

Single Agent

ANN: Input from visual field in an angle in front of agent, representing occupancy of several squares to front of agent

Output: one of 11 actions

Hidden nodes

ES vector: weights and biases for neural network, sigmas, (but no alphas)

Fitness function to learn exploration: \( \Sigma \) (result of all previous decisions - no. times the next grid point has been visited). So fittest agents have done well previously and are visiting new states

Result of a decision: high value if agent moves, medium value if agent turns, zero value if agent stationary, negative value if agent hits something

Other fitness functions for other behaviours

Combining Behaviours

Multi-agent starts by exploring an Environment1, learns to explore

It then learns to travel to a goal position, starting off with the ANNs that can already explore

It then goes back to explore again, etc.

-- Shaping

-- So a multi-agent controller with lifetime learning

-- But each learning step is some number of applications of evolution

Could apply to evolve/learn the parameters of other controllers, e.g. with more complex visual sensors and actuators