Genetic Algorithms and Genetic Programming Lecture 17

Gillian Hayes

28th November 2006

School of informatics

Selection Revisited

- Selection and Selection Pressure
- The Killer Instinct
- Memetic Algorithms
- Selection and Schemas
- Beyond the Schema Theorem
- What do Genetic Algorithms Offer?

Selection: An Example: The Knapsack Problem

Given a set of weights, W, and a target weight, T, find a subset of W whose sum is as close to T as possible.

Representation: for each $w_j \in W$, use 1 bit of the chromosome to signify whether or not w_j is a member of the solution or not.

Example:
$W = \{5, 8, 10, 23, 27, 31, 37, 41\}$
$s_i = 00101010$
means that weights 10, 27 and 37 are in the knapsack, with a total weight of 74.

The chromosome represents all subsets of W and each $s_i \in S$ where S is the set of all strings corresponding to exactly one subset of W.

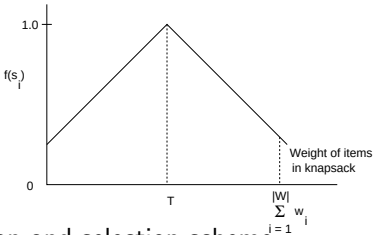
Fitness Function: The Knapsack Problem

Fitness: let sum(s_i) be the sum of all the weights in s_i. Then we want to minimise $\mid T - \text{sum}(s_i) \mid$. But our GA is a fitness maximiser, so we use:

$$f(s_i) = \frac{1}{1 + \mid T - \text{sum}(s_i) \mid}$$

Alternatives:

- $- \mid T - \text{sum}(s_i) \mid$
- Linear tent-shaped function:



Interaction between fitness function and selection scheme.

Selection 1

Let $D = |\ T - \ \mathrm{sum}(s_i)\ |$. Look at the fitness for various values of D:

$D = 0$ $\quad f(s_i) = 1.0$
$D = 1$ $\quad f(s_i) = 0.5$
$D = 2$ $\quad f(s_i) = 0.33$
$\vdots$
$D = 99$ $\quad f(s_i) = 0.01$

Fitness proportionate selection will give too much emphasis to strong solutions and will lead to premature convergence. What are the alternatives?

- Rank based selection
- Tournament selection

Selection 2

Rank based selection: put the solutions in order of fitness then select from the ordered set so that higher ranked solutions have a higher probability of being selected:

$f(s_0) = 0.5$ $\quad pref(s_0) = 1.5$
$f(s_2) = 0.1$ $\quad pref(s_2) = 1.3$
$f(s_5) = 0.025$ $\quad pref(s_5) = 1.1$
$f(s_1) = 0.01$ $\quad pref(s_1) = 0.9$
$f(s_3) = 0.005$ $\quad pref(s_3) = 0.7$
$f(s_4) = 0.001$ $\quad pref(s_4) = 0.5$

Define preference function to map rank onto some fitness value (perhaps a linear function). Divide pie in proportion to preferences.

Tournament selection: select n solutions from the population and put the fittest m (where $m < n$) into the next population,

e.g. select 2, keep the best one.

Increase $\frac{n}{m}$, increase selection pressure, e.g. select 1 from 10 (high selection pressure) vs. select 5 from 10 (lower selection pressure).

Boltzmann Selection: select proportional to $\exp^{f/T}$ where f is fitness and T is a tunable parameter. As $T \rightarrow \infty$, selection pressure is reduced, all solutions selected with equal likelihood.

What happens when we are converging on a good solution? All candidates are good, but not extremely good (or optimal) – region of optimal solution is identified but not *the* optimal.

The Killer Instinct (de Jong)

How do we get the **best** individuals (when we have good ones)?.

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

- Standard GA finds good areas but lacks the **killer instinct** to find the globally best solution.
- 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
- GA + LS = Memetic Algorithm

Making It Better

- 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
- 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)

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 focusses 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.

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

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 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.

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

What do Genetic Algorithms Offer?

- Robust problem-solving ability
- Search, optimisation, machine learning
- Good performance on dynamic problems (e.g. job-shop scheduling)
- Ease of implementation
- Hybridisation with other methods
- Anytime problem solving behaviour
- Easy to run on parallel machines
- A competitive solution for many hard problems

Reading: Mitchell Chapter 4. Skim the mathematical treatments.