COMP30030: Introduction to Artificial Intelligence

Neil Hurley

School of Computer Science
University College Dublin
neil.hurley@ucd.ie

October 25, 2018

Genetic Algorithms I

- Genetic Algorithms (GAs) are an optimization technique for functions defined over finite (discrete) domains.
- GAs are applied to a problem as follows :
  1. The search space of all possible solutions of the problem is mapped onto a set of finite strings over a finite (generally small) alphabet. That is, an encoding is chosen, such that each point in the search space is represented by exactly one string, called a chromosome. The GA works with these representations of solutions, rather than with the solutions themselves.
  2. An initial population of solutions is selected. This first generation is usually selected at random. Unlike standard optimization techniques, a GA performs a parallel search over a set of points in the search space, thus lessening the probability of being stuck in a local optimum.

Genetic Algorithms II

3. A fitness is computed for each of the individuals in the population, reflecting the way each individual is, in comparison to the others, nearer to the optimum. This value expresses the observed quality of the solution each individual represents.
4. The more fit individuals are selected according to a noisy selection, i.e. individuals are selected randomly, but with probability increasing with fitness. The GAs are thus essentially a randomized optimization technique.
5. The selected individuals form the parent set – they are crossed over (in pairs) to produce their progeny. A crossover consists of joining together non-corresponding bits of each parent in order to constitute two new individuals.
6. Another noisy selection is performed, this time biased towards the less fit individuals. These are replaced by the progeny obtained in the previous step. Unlike standard optimization techniques, the GAs proceed by replacing the weak part of a population with new individuals, rather than replacing the current best solution with a new candidate.

Genetic Algorithms III

7. A small part of the resulting population is mutated i.e. small random changes are applied to a few randomly selected individuals. In some GA applications, a small randomly chosen portion of the population is also subject to another genetic operator, the inversion — genes, while retaining their meaning, change their position inside the chromosome.
8. At this point, a new population has been constituted and the optimization process is repeated from 3 for the next generation.

Genetic Algorithms IV

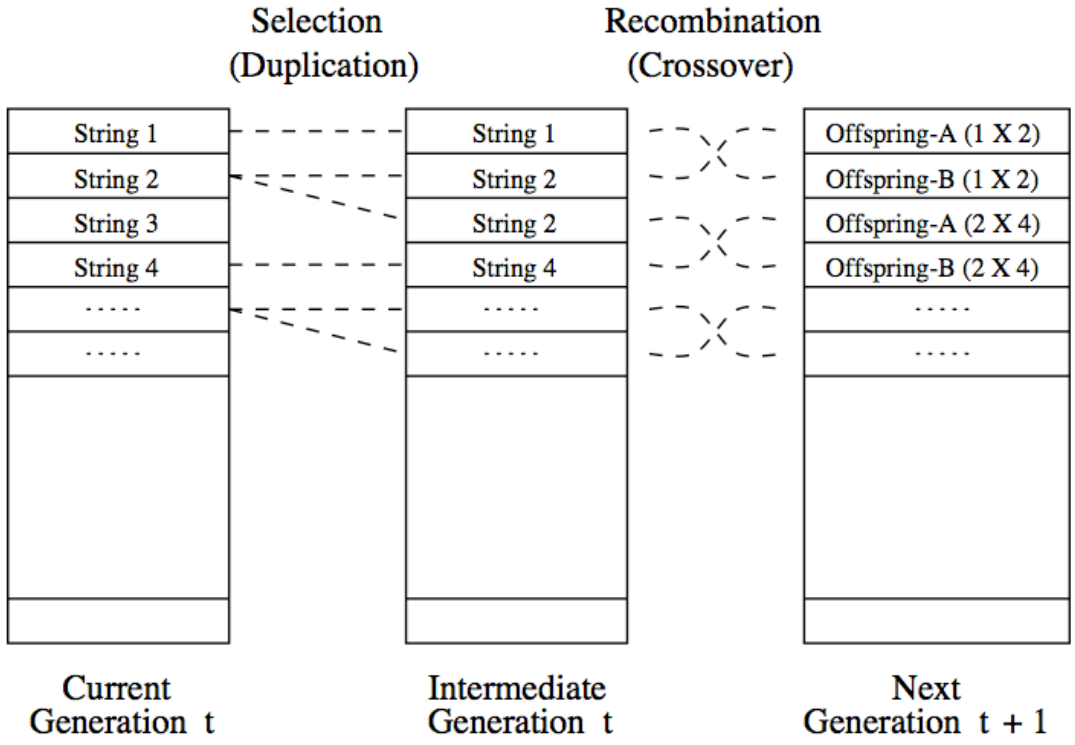

Figure: One generation is broken down into a selection phase and recombination phase. This figure shows strings being assigned into adjacent slots during selection. In fact they can be assigned slots randomly in order to shuffle the intermediate population. Mutation is not shown but can be applied after crossover

Canonical Genetic Algorithm I

- The first step is to generate a population of some size (number of members) P. Each member of this population is a binary string of length L which corresponds to the problem encoding.
- The evaluation function computes the value of the objective to be optimised.
- The fitness is defined with respect to the rest of the population by: $f_i/\bar{f}$ where f_i is the evaluation associated with string i and $\bar{f}$ the average evaluation of all strings in the population.
- Computation of the next generation can be considered as a 2-stage process:
  1. Selection is applied to the current population to create an intermediate population.
  2. Crossover (recombination) and mutation are applied to the intermediate population to create the next population

Selection I

- **Method 1** View population as mapping on to a roulette wheel, where each individual is represented by a space that proportionally corresponds to its fitness. Individuals are chosen using "stochastic sampling with replacement" to fill the intermediate population.
- **Method 2**: For each string i where $f_i/\bar{f} > 1$, the integer portion indicates how many copies of that string are directly placed in the intermediate population. All strings (including those with fitness ≤ 1 then place additional copies in the population with probability proportional to the fractional part of $f_i/\bar{f}$.

Crossover I

- Crossover is applied to randomly paired strings with probability p_c.
- Consider the string 1101001100101101 and another binary string yxyyyxyxxyyyyxyxxy in which the values 1 and 0 are denoted by x and y. Using a single randomly chosen recombination point **1-point** crossover occurs as follows

$$\begin{array}{ll} 11010 & \backslash/01100101101 \\ yxyyx & \backslash/yxxyyyyxyxxy \end{array}$$

- Swapping the fragments between the two parents produces the following offspring

$$11010yxxyyyyxyxxy \quad \text{and} \quad yxyyx01100101101$$

Mutation

- For each bit in the population, mutate with some low probability p_m, usually less than 1%
- Sometimes interpreted as randomly generating a new bit – in which case the bit value will only change 50% of the time.
- Otherwise interpreted as 'flipping' the bit value.

Representation for Graph Partitioning I

- The graph partitioning problem maps easily to a binary string representation. We can use a string of length $L = n$, the number of vertices in the graph.
- String value 1 implies that the corresponding vertex is in P_1 and string value 0 implies the corresponding vertex is in P_0.
- However, note that the ordering of the sting is completely arbitrary – depends on the labelling of the vertices in the graph.
- The effect of the 1-point crossover operator is highly dependent on the string ordering – its effect is to maintain groups of consecutively numbered vertices in the partition assignment of their parents – but if vertices are numbered randomly, this may make no intuitive sense at all.
- Note also the the 1-point crossover may generate infeasible states i.e. states where # ones $\neq$ # zeros.

Representation for Graph Partitioning II

- It therefore makes sense to move away from the canonical algorithm and choose a crossover that has a better probability of choosing high-quality children.
- One possibility is to maintain in the children the partition assignments of the vertices that both parents agree on and fill out the remainder in some way that at least preserves feasibility.
  - A basic approach is to fill the remaining vertices randomly (while ensuring feasibility).
  - A more sophisticated approach might try to choose the remaining values in a manner that tries to maximise the quality of the resulting child(ren) e.g. assign the vertices in order of maximum gain – where gain is the reduction in edge-cut resulting from the assignment of the vertex to a particular partition. This sort of approach is sometimes called a memetic algorithm.

Continuous Spaces I

- Many optimisation problems that are encounter in Machine Learning involve optimisation over a set of continuous valued parameters.
- Let $\mathbf{x}^T = (x_1, x_2, \ldots, x_n)$ be a vector in $\mathbb{R}^n$.
- Each possible value of $\mathbf{x}^T$ can be though of as a state in a continuous-valued state space.
- The optimisation (minimisation) problem now becomes:

$$\mathbf{x}^* = \arg\min f(\mathbf{x})$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$, is the objective function that maps each state to a real-valued number corresponding to the quality of the state. Our goal is to find states that attain a minimum of the objective.

- A well-known result from *multivariate calculus* is that, at a *turning* or *critical* point, the **gradient** of the function is zero,

Continuous Spaces II

- The gradient, often written as $\nabla(f)$, maps $f(.)$ to a vector of values, so $\nabla(f) : \mathbb{R}^n \rightarrow \mathbb{R}^n$. It corresponds to the set of *partial derivatives* evaluated at the point $\mathbf{x}$.

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)^T$$

- To find a critical point, we have to solve the n equations:

$$\begin{aligned} \frac{\partial f}{\partial x_1} &= 0 \\ \frac{\partial f}{\partial x_2} &= 0 \\ &\ldots \\ \frac{\partial f}{\partial x_n} &= 0 \end{aligned}$$

Continuous Spaces III

- Whether or not the critical point corresponds to a minimum depends on the matrix of second order partial derivates, called the Hessian matrix.
- It is common to formulate problems so that they have *convex* objective functions, which are guaranteed to have a single minimum point.
- Even so, for many problems, solving such as system of equations exactly is not possible. We have to use numerical methods to find an approximate solution.

Continuous Spaces IV

- Generally it is easy to compute the gradient of a function. Since the gradient is a vector it corresponds to a direction in n-dimensional space.
- In fact, $\nabla f(\mathbf{x})$ is the direction in which the function is locally most rapidly increasing at $\mathbf{x}$.
- Similarly, $-\nabla f(\mathbf{x})$ is the direction in which the function is locally most rapidly decreasing at $\mathbf{x}$
- A very common numerical strategy then, is to start at an arbitrary point $\mathbf{x}_0$ and evaluate $\nabla f(\mathbf{x}_0)$.
- We then move a small step in the direction of the negative (or positive) gradient, depending on whether we are minimising or maximising.

Continuous Spaces V

- This corresponds to the following update for minimisation:

$$\mathbf{x}_i = \mathbf{x}_{i-1} - \alpha \nabla f(\mathbf{x}_{i-1})$$

  where α is a small step size, often called the learning rate, when the optimisation is being carried out in the context of Machine Learning.
- This algorithm is known as gradient descent and is the analogue of local search for continuous problems.
- The success of gradient descent depends strongly on the learning rate – too small, and (for large problems), it may converge too slowly; too large and it may not converge at all.
- More sophisticated techniques exist e.g. that also use the Hessian matrix, that can converge in fewer iterations, but that may have to carry out more computations per iteration.

Continuous Spaces VI

- For some problems, exact computation of the gradient is expensive – instead cheaper ways to find a direction of descent (rather than the most rapid descent) are employed e.g. Stochastic Gradient Descent (SGD).

Continuous Spaces VII