[*]

Genetic Algorithms

[*]Material based on Chapter 12, Searching and Optimization, of B. Wilkinson et al., "PARALLEL PROGRAMMING. Techniques and Applications Using Networked Workstations and Parallel Computers"

Evolution and Genetic Algorithms

Genetic algorithms try to mimic evolution in a population of a species.

Based on the concept of chromosomes from biology.

Chromosomes uniquely determine characteristics of living creatures.

When two beings reproduce, portions of the chromosomes of each parent are combined to create the chromosomes of the child.

Children thus exhibit a blend of their parent's characteristics.

This inheritance is called *crossover.*

Evolution and Genetic Algorithms cont.

Other major operator is called *mutation.*

This is when a random change to the child's chromosomes occurs that may result in a child with characteristics significantly different from either parent.

Mutation typically creates significant change in the child's "viability," either for better or worse.

As more ways to decrease one's viability than increase, many mutations at once would be detrimental to a population.

Together, crossover and mutation cause the *evolution* of the population in this model.

Viability

Organism must interact with environment.

If changes to its chromosomes improve its viability compared to its parents, its more likely that it will survive to pass its new traits on.

The “strongest” and “fittest” of course do not necessarily always survive to reproduce.

A genetic algorithm uses these principles and is characterized by repeatedly applying the concepts of crossover and mutation, where the production of each new “generation” is based on the relative fitness of the members of the previous one.

This optimization method tends to produce good solutions as the “population” evolves over time.

General Structure of a Genetic Algorithm

1) Initial population of "solutions" (organisms) is created.

2) Each individual is evaluated using application-specific criteria to determine its "fitness."

3) Subset of population chosen to reproduce. Criteria that favours "more fit" individuals used.

4) Subset used to produce new generation of offspring using crossover.

5) Small number of the next generation are mutated.

6) steps 2-5 repeated to produce many generations. Continues until termination conditions met.

Sequential Genetic Algorithm

```
generation_no = 0;
initialize Population(generation_no);
evaluate Population(generation_no);
set termination condition to False;
while (not termination condition) {
  generation_no++;
  select Parents(generation_no) from
    Population(generation_no - 1);
  apply crossover to  Parents(generation_no) to get
    Offspring(generation_no);
  apply mutation to Offspring(generation_no) to get
    Population(generation_no);
  evaluate Population(generation_no);
  update termination condition;
}
```

Illustrative Example

Example: find maximum of:

$$f(x, y, z) = -x^2 + 1,000,000x - y^2 - 40,000y - z^2$$

with variables x, y, and z restricted to integer range: $-1,000,000$ to $1,000,000$

Example useful for comparison to different methods as closed solution exists.

Using algebra, f can be factored into the form:

$$f(x, y, z) = 2504 \times 10^8 - (x - 500,000)^2 - (y + 20,000)^2 - z^2$$

Clearly, the maximum 2504×10^8 occurs at $x = 500,000$, $y = -20,000$, and $z = 0$.

Illustrative Example Cont.

Exhaustive approach of evaluating f for all combinations of (x, y, z) would mean trying $(2,000,001)^3$ values.

If each evaluation took 100ns, this would take $> 22,000$ years on a single processor.

Even if parallelization sped things up by factor of $10,000$, this would take more than 2 years!

Initial Population: Data Representation

Before we select our initial population, we need to decide how to represent a solution.

For us, each solution will be represented by a binary string.

the i^{th} member of a given population would be the tuple (x_i, y_i, z_i).

As each variable is over range of $-1,000,000$ to $1,000,000$, it can take on $2,000,001$ unique values.

Each variable can thus be stored using 21 bits as:

$$2^{20} < 2,000,001 \leq 2^{21}$$

Data Representation Cont.

The solution can be represented by its “chromosome” which would be the 63 bit binary number created by the concatenation of the three variables.

For tuple $(x, y, z) = ((+262, 408), (+16, 544), (-1032))$, we can represent them individually using *sign-and-magnitude representation* as follows:

$$\begin{aligned} x &= 001000000000100001000 \\ y &= 000000100000010100000 \\ z &= 100000000010000001000 \end{aligned}$$

The chromosome for (x, y, z) is the 63 bit number:

001000000000100001000000000100000010100000\\
0100000000010000001000

We can now use a psuedorandom generator to create an initial “population” of 63 bit numbers.

Evaluation

For each generation, we need to determine the fitness of each individual.

For our current example, we can evaluate the function for each tuple. The larger the result, the more “fit” the individual is.

For $(x, y, z) = ((+262,408), (+16,544), (-1032))$, we get:

$$\begin{aligned} f(x,y,z) &= -(262,408)^2 + 1,000,000(262,408) \\ &\quad -(16,544)^2 - 40,000(16,544) - (-1032)^2 \\ &= 192,613,512,576 \end{aligned}$$

Constraints: As $2^{20} = 1,048,576$, it's possible to have magnitudes outside our $1,000,000$ range.

Such a tuple could be handled by NOT considering it as possible parent for next generation.

Could also be altered to bring back into range.

Selection Process

To mimic the biological process, we want a selection process that favours the fittest, but allows for “accidents” to happen.

Will use the *tournament selection* method to achieve this.

For each tournament, k individuals are chosen. Each member of current generation has equal chance to be chosen.

Of these k individuals, the fittest is deemed to be the winner, and will be a parent for the next generation.

There will be n tournaments, thus n parents.

Offspring Production: Single-point Crossover

Once the parents are determined, we need to combine their chromosomes to create children. Will use the *single-point crossover* method.

We make a single cut at the p^{th} bit (p is randomly chosen) in the m bit chromosome of each parent (labelled A and B).

Child number one's chromosomes are created from the first p bits of parent A's chromosomes, and the last $m - p$ bits from parent B.

Child number two's chromosomes are created from the first p bits of parent B's chromosomes, and the last $m - p$ bits from parent A. See Figure 12.2.

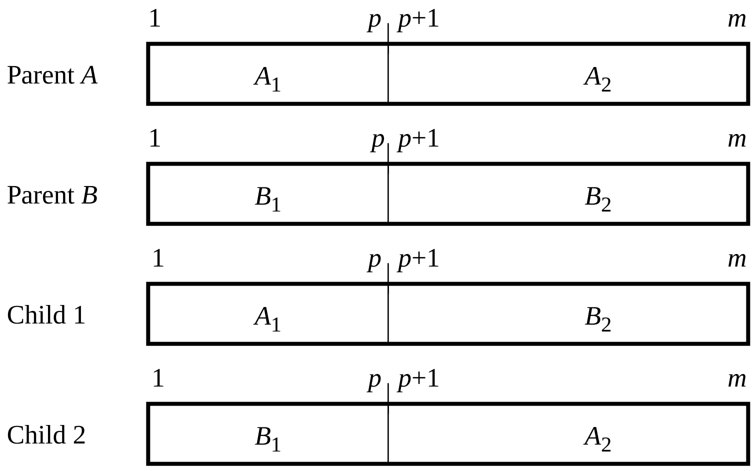

Figure 12.2 Single-point crossover.

Mutation

After crossover is used to produce offspring, a subset of the offspring is selected for mutation.

Mutations usually involve a change to only a single bit.

For the x, y, or z component, this could result in a sign change, or a magnitude change as small as (± 1) for bit zero, or as large as ($\pm 524,288$) for bit 19.

As change could be so dramatic, want to keep rate of mutation small to allow stable integration into population, or could get noncovergence of algorithm.

Termination Conditions

Usually choose a maximum number of generations as an upper limit as well as some means to measure the “quality” of the current solutions.

One way to do this is to compare successive generations, and measure the improvement between generations.

Another way is to measure similarity of population as this increases as algorithm converges towards an optimal solution.

Parallel Algorithms

Two main approaches to parallelize genetic algorithms:

1) Isolated Subpopulations with Migration

2) Common Population

In approach 1, each processor applies sequential genetic algorithm on a separate subpopulation and then they periodically (every k generations) share their "best" individuals.

Sharing referred to as *migration.*

Migration Operator

The migration operator handles the exchange of individuals between subpopulations. It takes care of:

1) Selection of the emigrants

2) Sending and receiving the emigrants

3) Integrating the emigrants

Have to be careful how to choose which individuals to send and which to replace with the new emigrants.

If simply send the fittest, and remove the weakest could push the solution towards a local optimal.

Migration Models

Migration causes communication overhead.

Need to balance variance in subpopulations, with frequency and volume of communication to achieve this variance.

Two main migration models:

- *Island Model:* Individuals allowed to migrate to any other subpopulation. See Figure 12.3.

  Permits more freedom and in some ways closer to natural processes.

  Significant more communication and delay than second method.

Migration Models Cont.

- *Stepping Stone Model:* Individuals only allowed to migrate to neighbouring populations.

  Reduces communication by limiting possible destinations, and thus number of messages. See Figure 12.4.

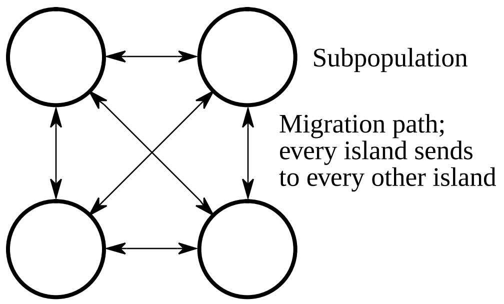

Figure 12.3 Island model.

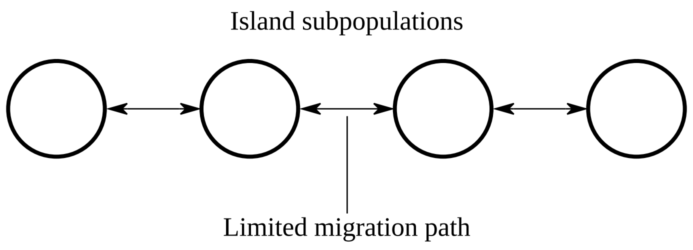

Figure 12.4 Stepping stone model

Parallel Algorithm: Subpopulations

For each slave process:

```
generation_no = 0;
initialize Population(generation_no);
evaluate Population(generation_no);
set termination condition to False;
while (not termination condition) {
  generation_no++;
  select Parents(generation_no) from
    Population(generation_no - 1);
  apply crossover to  Parents(generation_no) to get
    Offspring(generation_no);
  apply mutation to Offspring(generation_no) to get
    Population(generation_no);
  apply migration to Population(generation_no);
  evaluate Population(generation_no);
  update termination condition;
}

send best solution to master;
```

Master process chooses best solution from the subpopulations.

Subpopulations: Pros and Cons

Pros: As communication occurs only every k generations, should see speedup factor of p (for p processors) for this portion of the computation.

Migration every k generations will reduce the speedup.

This reduction will be least in “stepping stone model” as communication only to nearest neighbour.

Cons: The mostly isolated subpopulations increases chance that each subpopulation will converge to a local optimal.

Algorithm 2: Common Population

Here, the genetic operators are performed in parallel on a common population.

If each processor had information about all of the individuals, then they could perform selection, crossover, and mutation in parallel (ie. a pipeline).

If use tournament selection, they would all be operating on independent pairs of individuals. Similarly, they could evaluate independent sets for generating and evaluating next population.

Unfortunately, they would then have to communicate the results to all processors.

Common Population Cont.

For message-passing, the communication would likely be more than the computation.

For shared memory system, could store results in common memory reducing communication time considerably.

Assigning each processor a subset to process reduces memory contention.

Synchronizing after each generation reduces the parallelization, this shouldn't be too bad as only accessing memory.