MAE 552
Heuristic Optimization

Instructor: John Eddy

Lecture #12

2/20/02

Evolutionary Algorithms

Variation Operators - Mutation

- Mutation is the seemingly random alteration of genetic structure. (in biology, mutation is generally caused by mutagens such as UV radiation). We tend to consider it a random process in our algorithms.

Mutation on Vectors of Integers

- Could randomly select designs and associated variables and then:
  - Alter by a random amount (according to some distribution).
  - Randomly reassign to a new value (within bounds).

Same approaches used for real number encoding.

Mutation on Vectors of Bits

- Could use random re-assignment as in vectors of ints and reals. But clearly we would not attempt to add a random amount to a bit since they can only be 1’s and 0’s.

Mutations on Combinations

This will likely require a combination of the aforementioned mutation strategies. This case will likely require highly specialized operators (very problem dependent).

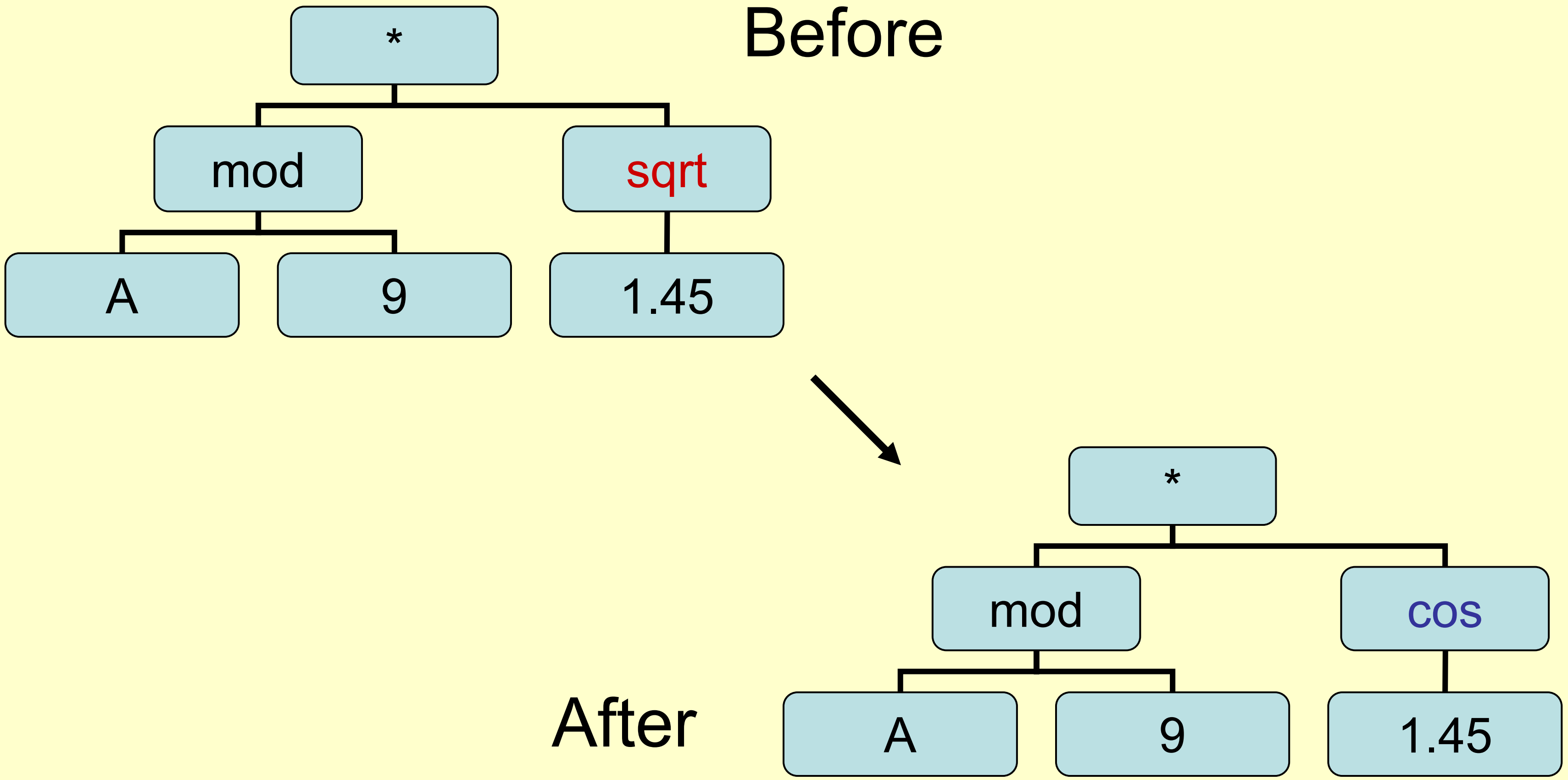
Mutations in Binary Encoding

Can be carried out in many of the aforementioned ways.

Can also perform random bit mutation where the bits of the actual design variables are changed according to some probability.

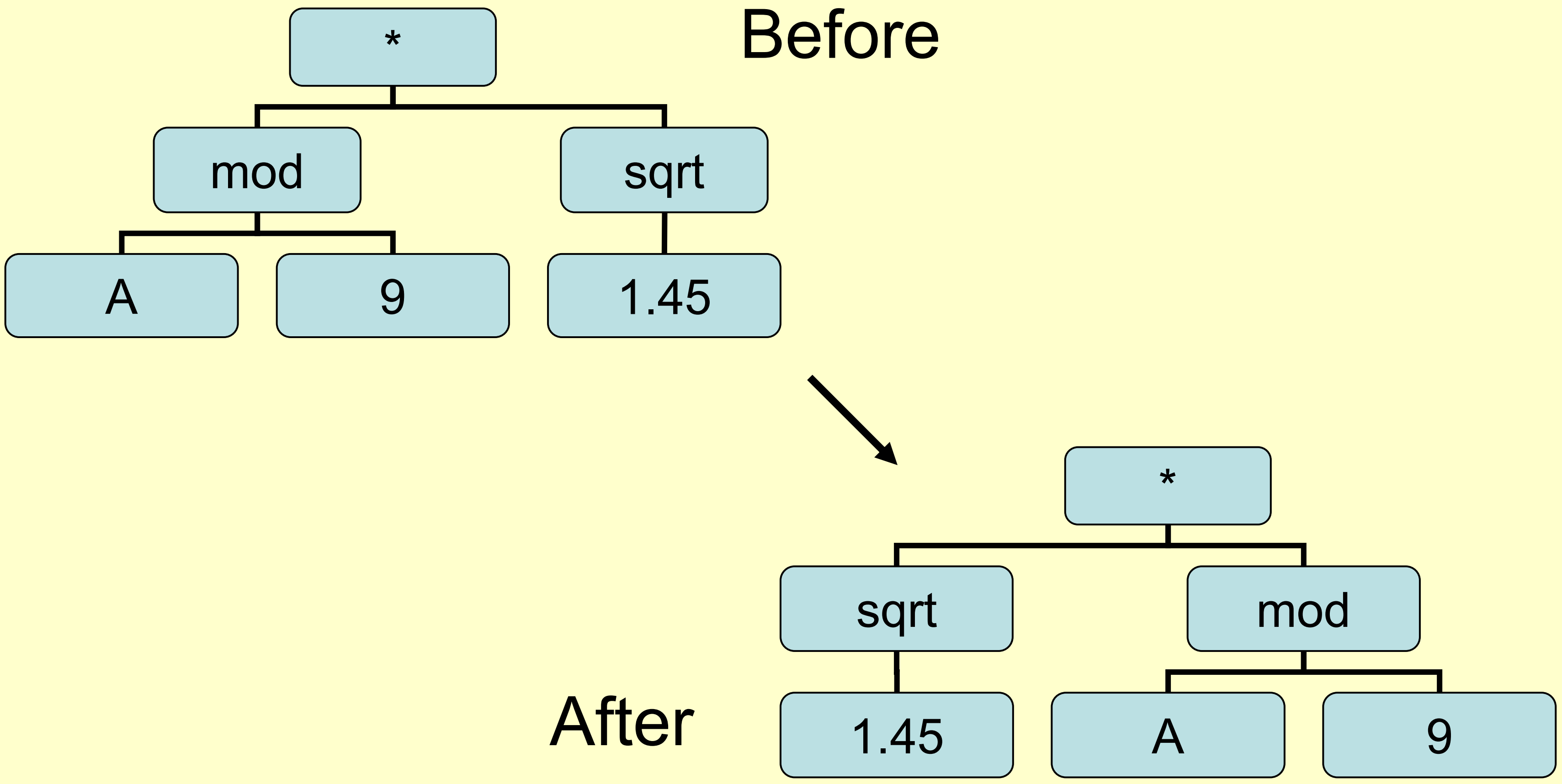
Mutations on Symbolic Exps.

One Node Mutation



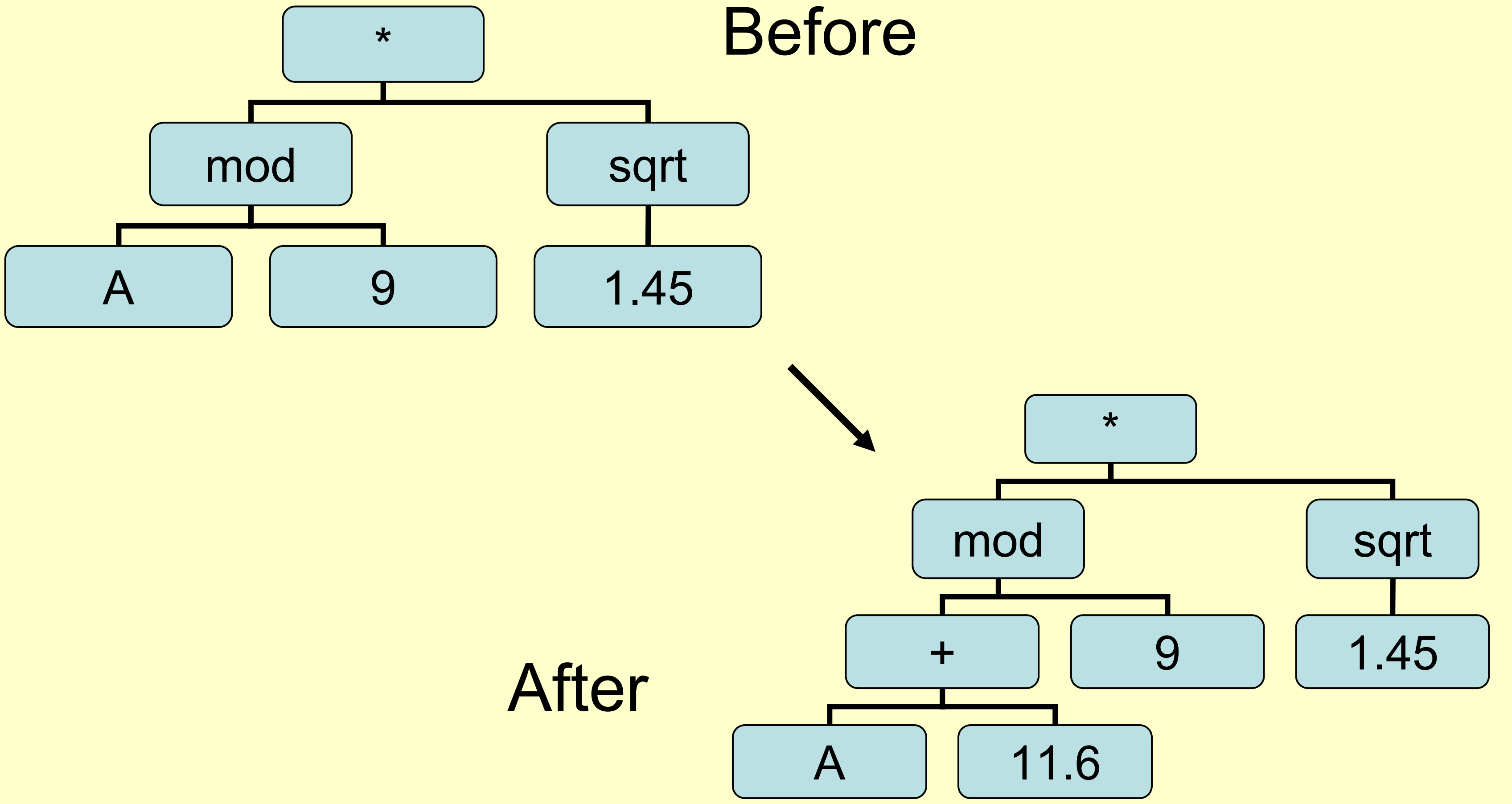
Mutations on Symbolic Exps.

Swap Mutation



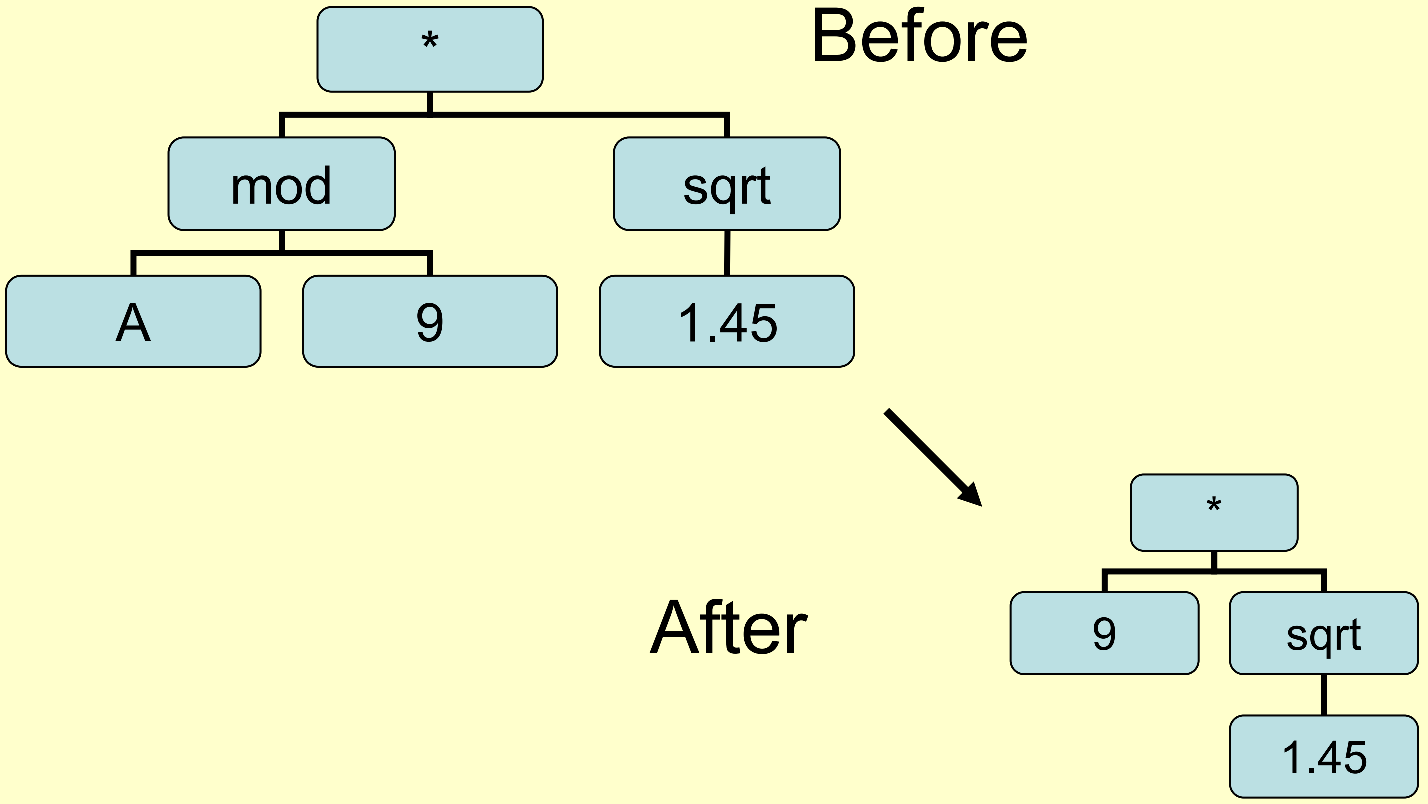
Mutations on Symbolic Exps.

Growth Mutation



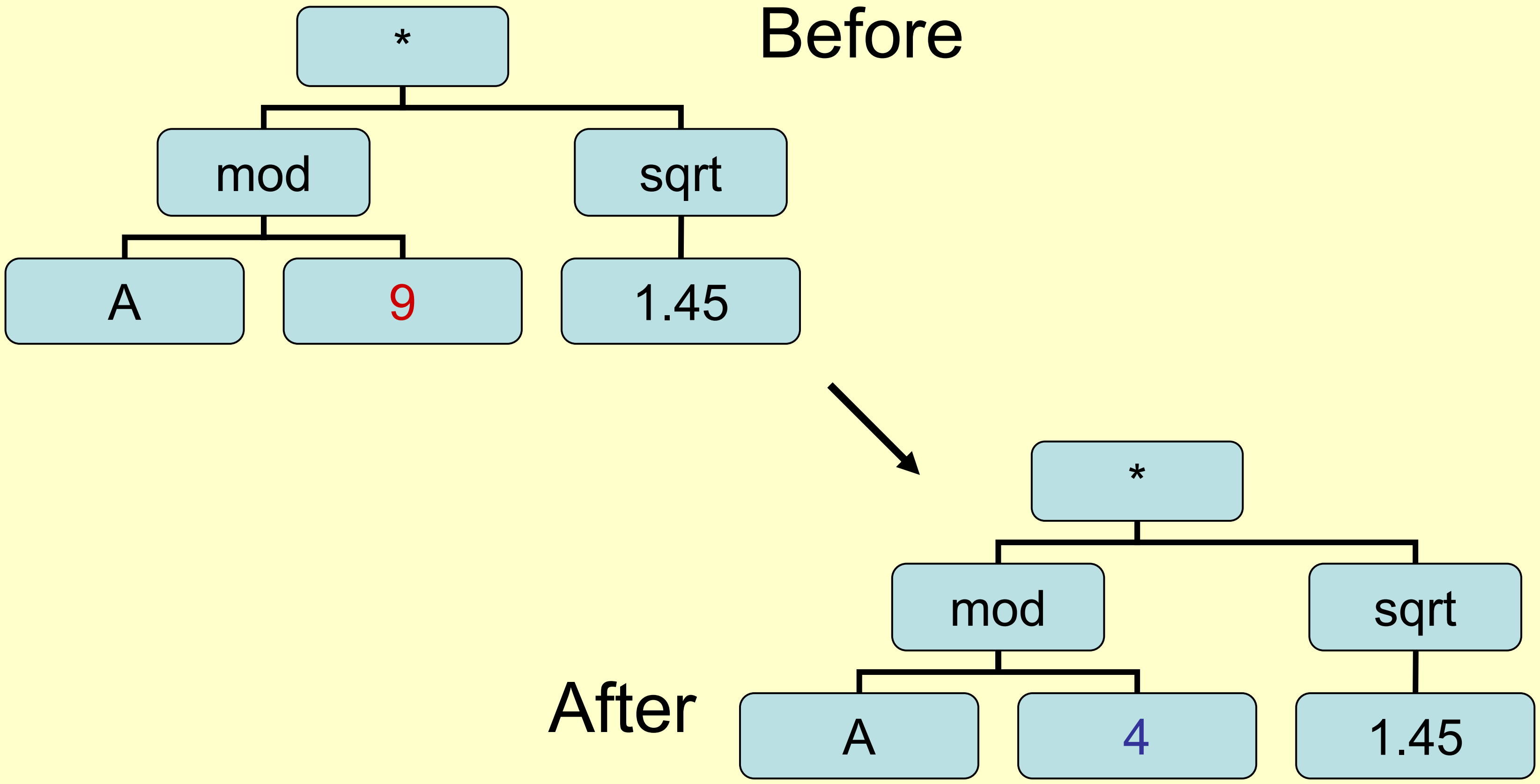
Mutations on Symbolic Exps.

Truncation Mutation



Mutations on Symbolic Exps.

Gaussian Mutation



Selection Operator

- The selection operator serves as a filter in which the algorithm will determine the makeup of the next generation.

- It acts on the individuals of the population and generally considers their quality/fitness.

Selection Operator

- Most approaches work according to one of the following schemes:
  - some designs are removed from consideration and others go on to become the members of the next generation
  - individuals are sampled and pitted against new designs and “survival” is based on their relative fitness.

Selection Operator

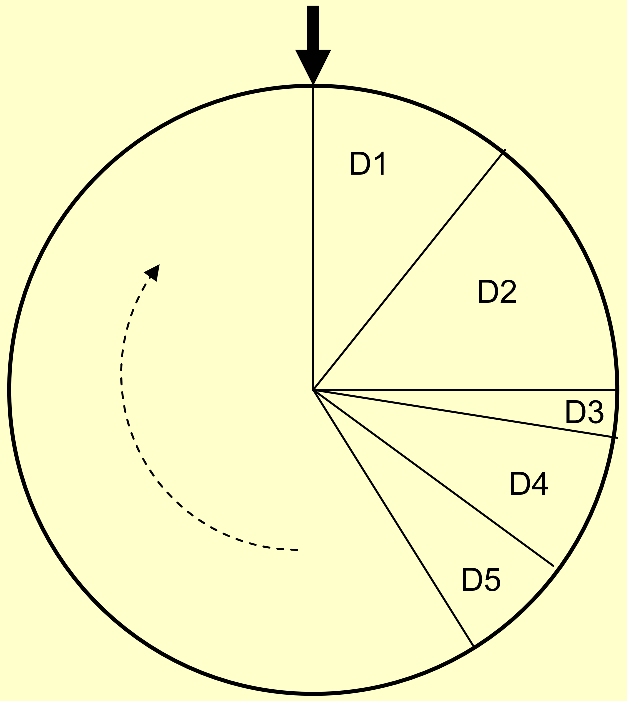
- In the first case:
  - Members of the population are simply removed and the result is a “thinning” of the population. (probability of removal is based on fitness)
  - As a result of the above, each individual has a chance for at most 1 duplicate of itself in the next population.

Selection Operator

- In the second case:
  - Members of the population are chosen (perhaps at random) to compete with each other in a “tournament”. The winner gets a copy in the next population. Tournaments are run until the next population is full.
  - As a result of the above, each individual has a chance for many duplicates in the next population (could be involved in many tournaments).

Selection Operator

- Note that the competition doesn’t have to be explicit. Consider roulette wheel selection.



Each design gets a slice of the wheel sized proportionally to its fitness.

Wheel is spun and a design is selected until next population is full.

Selection Operator

- Another common approach is to divide each members fitness by the average population fitness to achieve the number of copies sent into the next generation.

(we will see this approach later in our example).

5 Basic Components

- ***An encoded representation of solutions to the problem.***
  - Ex. binary encoding, real number encoding, integer encoding, data structure encoding.
- ***A means of generating an initial population.***
  - Ex. random initialization, patterned initialization.
- ***A means of evaluating design fitness.***
  - Need a consistent means of determining which designs are “better” than others.
- ***Operators for producing and selecting new designs.***
  - Ex. selection, crossover, mutation.
- ***Values for the parameters of the algorithm.***
  - Ex. How much crossover and mutation, how big is population.

Parameters

- As previously mentioned, the parameters define rates, sizes, etc. pertaining to the algorithm.
- Appropriate values of the parameters are very problem dependent.
- In this section, we will look at the 3 typical algorithm parameters and consider their effects.

Population Size

- A population size must be decided upon before initialization occurs.
- A very large initial population may as well be a random search.
- A very small initial population will likely converge quickly (and sub-optimally).

Crossover Rate

- A high crossover is used to encourage good mixing of the chromosomes. This encourages variation from one population to the next.
- A low crossover rate may unnecessarily slow the algorithm.
- Typical value is 75% (cross 75% of pairs)

Mutation Rate

- High mutation is a random search.
- Low mutation may prevent sufficient exploration.
- Typical values are 0.5% per bit in BE or 3-5% for any given design undergoing a mutation.

Algorithm Workings

The next set of slides will deal with some information about how the algorithm works and exploits previously learned information by means of schemata.

Schemata

Schemata are templates discovered by the algorithm (implicitly).

These templates represent similarities that exist between highly fit designs.

We will consider the case of a binary string (binary encoding or vector of bits) so that each genetic locus has 2 alleles (0, 1).

Schemata

For an alphabet with cardinality k, we will define k+1 symbols to represent our schemata.

The first k symbols will be those used by the alphabet and the k+1th symbol will be the meta-symbol (meaning a symbol that symbolized a symbol) *. * will serve as a wildcard.

Schemata

So for a set of designs with associated fitness values:

Design	Fitness
0 1 1 0 1	169
1 1 0 0 0	576
0 1 0 0 0	64
1 0 0 1 1	361

We may notice that all designs with a 1 in the first locus have high fitness values.

Schemata

Thus we may say that the schema 1**** represents a set of good designs.

So a schema represents a subset of designs and can be used to broadly classify them.

We can also determine how many designs belong to a particular schema.

Schemata

The schema 1*011 has 2 member designs:

11011 and 10011

The schema 1*01* has 4 member designs:

10010, 10011, 11010, and 11011

In General, a schema has 2^s member designs where s is the number of wildcards (*'s).

Schemata

So how many schema are there?

For our binary example, each location in a schema can be a 0, 1, or *. Therefore, there are 3^n possible combinations where n is the number of locations.

So for 5 bit long genomes, there are 3^5 or 243 different possible schema.

Schemata

For an alphabet of cardinality k, there are then $(k+1)^n$ schemata where n is again the length of a genome.

So what? What am I supposed to do with this? I cannot possibly test for all of these schemata.

Schemata

The algorithm will process these schemata automatically. It turns out that the number of schemata usefully processed each generation is something like n^3 where n is the population size.

Some schemata will have difficulties propagating through the generations and others will not. Those that do not and represent designs of good fitness will become building blocks that the algorithm uses to create designs.

Schemata

Which schemata will propagate well and which will not?

This depends a great deal on your encoding method and your operators.

I will present an example using our binary paradigm.

Schemata

When using a single point crossover, a schema like:

1 * * * 0

Is very likely to be disrupted while a schema like:

* * * 1 0

Is not. See why?

Schemata

It is also possible for mutation to disrupt or destroy a schema for a design but it is not likely because of the low probability of mutation.

So highly fit, short, well placed schemata will propagate well and the occurrences of them will increase exponentially as the algorithm progresses.