

Master of Technology

Computational Intelligence II

Genetic Algorithms: Practical Issues and Theoretical Foundations

Dr. Zhu Fangming
Institute of Systems Science,
National University of Singapore
Email: isszfm@nus.edu.sg

© 2018 NUS. The contents contained in this document may not be reproduced in any form or by any means,
without the written permission of ISS, NUS other than for the purpose for which it has been supplied.

Objectives

- Understand issues affecting fitness
- Understand different genetic operators and the effects of parameter settings on outcome
- Understand why GA works (Schema Theorem)

Topics

- Issues affecting fitness
- GA control parameters
- GA theory - Schema Theorem
- Appendix: Problem-specific operators

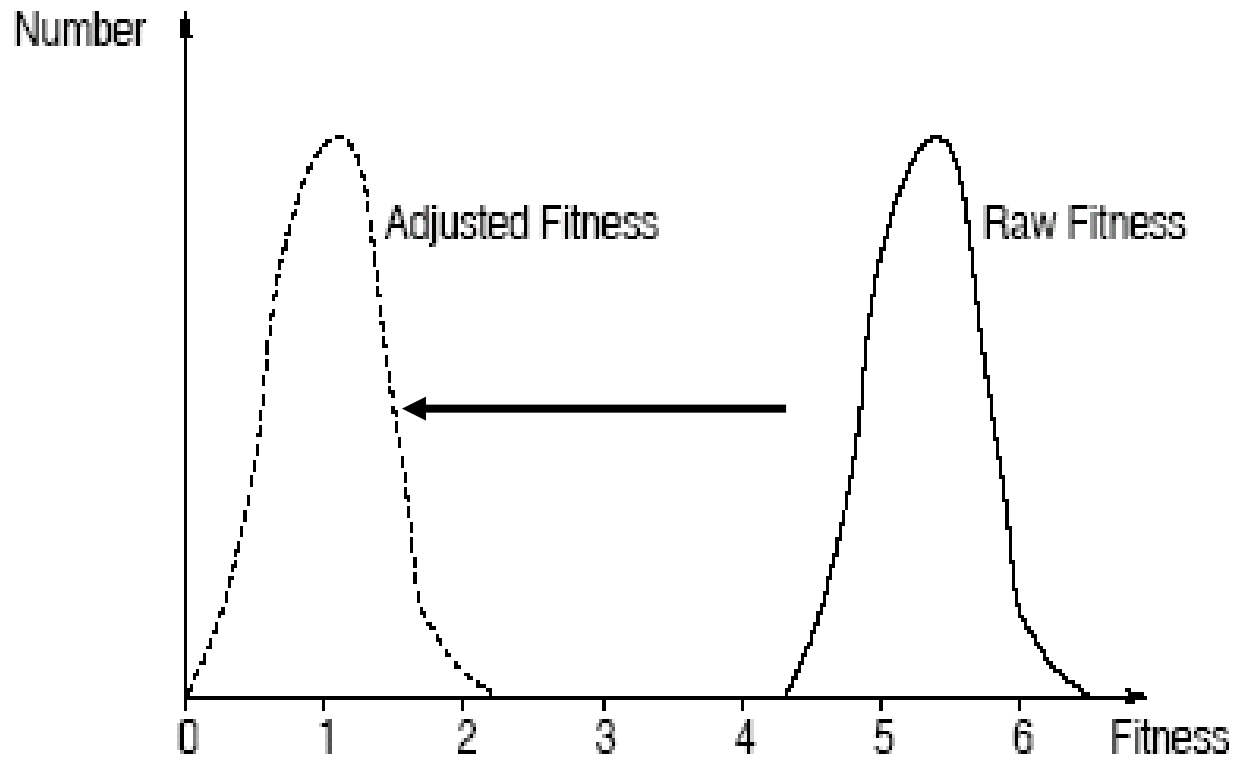
Fitness Values: the scatter matters

- The scatter of the fitness values in the population affects selection pressure, thus would have an effect on GA process.
 - Suppose there are two strings with fitness scores 1 and 2, respectively. The second string will get twice as many reproductive opportunities as the first.
 - If the two strings will now score 11 and 12, a ratio is only 1.09.
 - Selection pressure would be significantly reduced.

Fitness Values: scatter wide or scatter close

- Suppose we optimise a function with a fitness range of 0-10. Initially the random population might score mostly in the range between 0 and 1.
 - A lucky individual with a score of 3 will then be given a large selection advantage.
 - It will take over the population, reducing, and eventually removing, the genetic diversity.
 - It is a potential hazard as the fitness of the population might not improve to 9-10.
- Strong selection pressure (biased selection) has a tendency toward the premature convergence of the GA search.
- Weak selection pressure can make the search ineffective.

Fitness Scaling / Shifting



Taken from: Beasley, D. R. Bull, R. R. Martin. An Overview of Genetic Algorithms: Part 1, Fundamentals. University Computing 15:4 (1993) 170-181.

Simple Scaling

- Use the worst fitness value as a baseline for the window, then subtract this value from all other fitness scores
- Change this baseline every **w** generations so that the baseline becomes the worst fitness value in the **w** most recent generations.
 - o Typically **w** is between 2-10.

$$\tilde{f}(x) = f(x) - f_{min,w}$$

Sigma Scaling

- Setting the baseline to s standard deviations below the mean.
 - Individuals with fitness values below this baseline are assigned a fitness of zero.
 - By definition, the average fitness of the scaled population will be within s standard deviations.
- Typically the number of standard deviations s is between 2-5.
- Keeps the baseline nearer the “average performers”
 - Overcome a potential problem with particularly poor individuals which, with windowing, would put the baseline very low.

$$\tilde{f}(x) = \max\{0, f(x) - (\bar{f} - s \cdot \sigma_f)\}$$

Fitness Proportionate Selection Methods

- Variations of selection methods are attempts to strike a balance between exploitation and exploration.
- Fitness proportionate methods
 - Choosing of chromosome for reproduction according to their relative fitness.
- Elitist model
 - Always preserve a copy of the **best chromosome** in the new population.
 - If the best chromosome A generated up to generation t is not in generation $(t+1)$ then *add* A to generation $(t+1)$ as the N -th or $(N+1)$ -th chromosome in the population



Fitness Proportionate Selection Methods

- Expected value model
 - Reduces the stochastic errors of selection routine. This is done by introducing a **count k** for each chromosome v_i which is set initially to $k = f(v_i)/\underline{F(t)}$ the ratio of the chromosome's fitness to the average population fitness.
 - The count **k** is decreased by 0.5 or 1 when the chromosome is selected for reproduction.
 - When the count falls below zero block further selection of the chromosome.
 - Tends to reduce stochastic selection errors and bias.

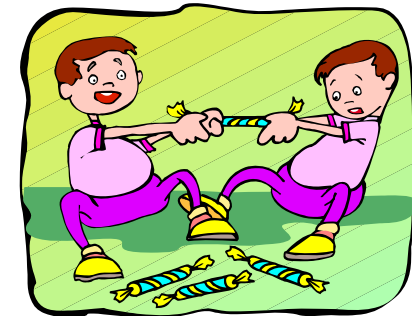
Other Selection Methods

- **Tournament selection method**

- Choosing individuals for reproduction randomly. This slows down selection pressure.

- **Binary tournament**

- o Two individuals are chosen at random and the better of the two individuals is selected.



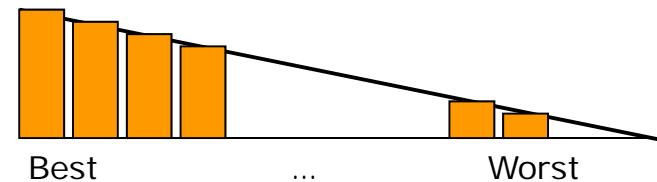
- **Larger or N-tournament**

- o N individuals are chosen at random from the population, with the best being selected for reproduction.

Other Selection Methods

- **Rank selection method**

- The population is ordered according to the measured fitness values. A new fitness value is then ascribed, inversely related to their rank. This also helps slow down selection pressure.



- **Linear ranking**

- o Each chromosome in the population is ranked in increasing order of fitness, from 1 to N, and assigned a new **subjective fitness** using a ranking function with constant selection pressure.

- **Exponential ranking**

- o First chromosome assigned fitness of 1;
the second fitness of s (typically 0.99);
the third fitness of s^2 and so on.

Tuning Control Parameters

- Many control parameters used to tune genetic search are really indirect means of affecting selection pressure and population diversity.
 - Population size
 - Fitness function
 - Selection method
 - Probability of crossover
 - Probability of mutation
 - Generation Gap

Tuning Control Parameters

- These parameters typically interact with one another non-linearly, so they cannot be optimized one at a time.
- There is a great deal of discussion of parameter settings and approaches to parameter adaptation in literature but there are no conclusive results on what is best.

Control Parameters (Fitness Function)

- The fitness function should communicate what is desired in terms of performance. It is the only chance to communicate your intentions to the powerful GA process. The fitness function must be a non-negative figure of merit.
- The fitness function should not only give an accurate score on performance but also permit differentiation between the performance of different individuals.
- The scaling of fitness function for better fitness sensitivity is sometimes required.
- When fitness function cannot be well defined, some form of subjective measure is acceptable.

Control Parameters (Pop. Size)

- How many chromosomes to use?
- Small population sizes do not give sufficient sampling of solution space.
- Large populations give more informed search, but require more evaluations per generation and possibly slower convergence
- Guideline
 - Population sizes ranging between 30 and 500 give best performance.

Control Parameters (Generation Gap)

- The generation gap (survival policy) controls the percentage of the population $\text{Pop}(t)$ that is replaced at generation $t+1$.
- $N^*(1-G)$ population members at generation t are chosen at random to survive intact in $\text{Pop}(t+1)$.
- A value of $G = 1.0$ means the entire population is replaced each generation.
- $G = 0.5$ means that half of the individuals survive into the next generation.
- For best results keep $G \geq 0.6$

Control Parameters (Selection Method)

- The selection method determines how and where the search is directed.
- Better solutions get selected and propagated more frequently directing the search to promising regions in solution space.
- If a few individuals are selected too often, premature convergence may occur \approx bias / favoritism
- Many selection methods are designed to avoid premature convergence and wasting computational time.
- Selection methods may involve picking the individuals in proportion to fitness values, tournament and ranking.

Control Parameters (Crossover Rate)

- Considerations
 - If it is too high, high performance structures are discarded faster than selection can make improvements.
 - If too low, the search may stagnate.
- For small population (10-40) and single crossover,
 - high crossover rates (0.85) work well
- For medium size populations (30-90)
 - lower rates are better (0.6)
- For large populations
 - small rates (0.3) are best

Control Parameters (Mutation Rate)

- Mutation increases the variability or diversity of the population.
- Considerations
 - A high rate yields essentially a random search.
 - A low rate may cause some bit positions to remain forever stuck.
- For best results
keep $p_m \leq 0.05$

Control Parameters (De Jong's recommended settings)

- These settings are widely used in the GA community
 - Best population size 50-100 individuals
 - Best single-point crossover approximately 0.6 per pair of parents
 - Best mutation rate 0.001 per bit
- As always, your mileage may vary! 😊

Trade-off Considerations on Control Parameters

- **Increasing the crossover probability** increases recombination of schemata, but it also increases the **disruption** of good chromosomes.
- **Increasing the mutation probability** tends to transform the genetic search into a **random search**, but it also helps re-introduce lost genetic material.
- **Increasing the population size** increases its diversity and reduces the probability that the GA will prematurely converge to a local optimum, but it also **increases the time** required for the population to converge to the optimal regions in the search space.

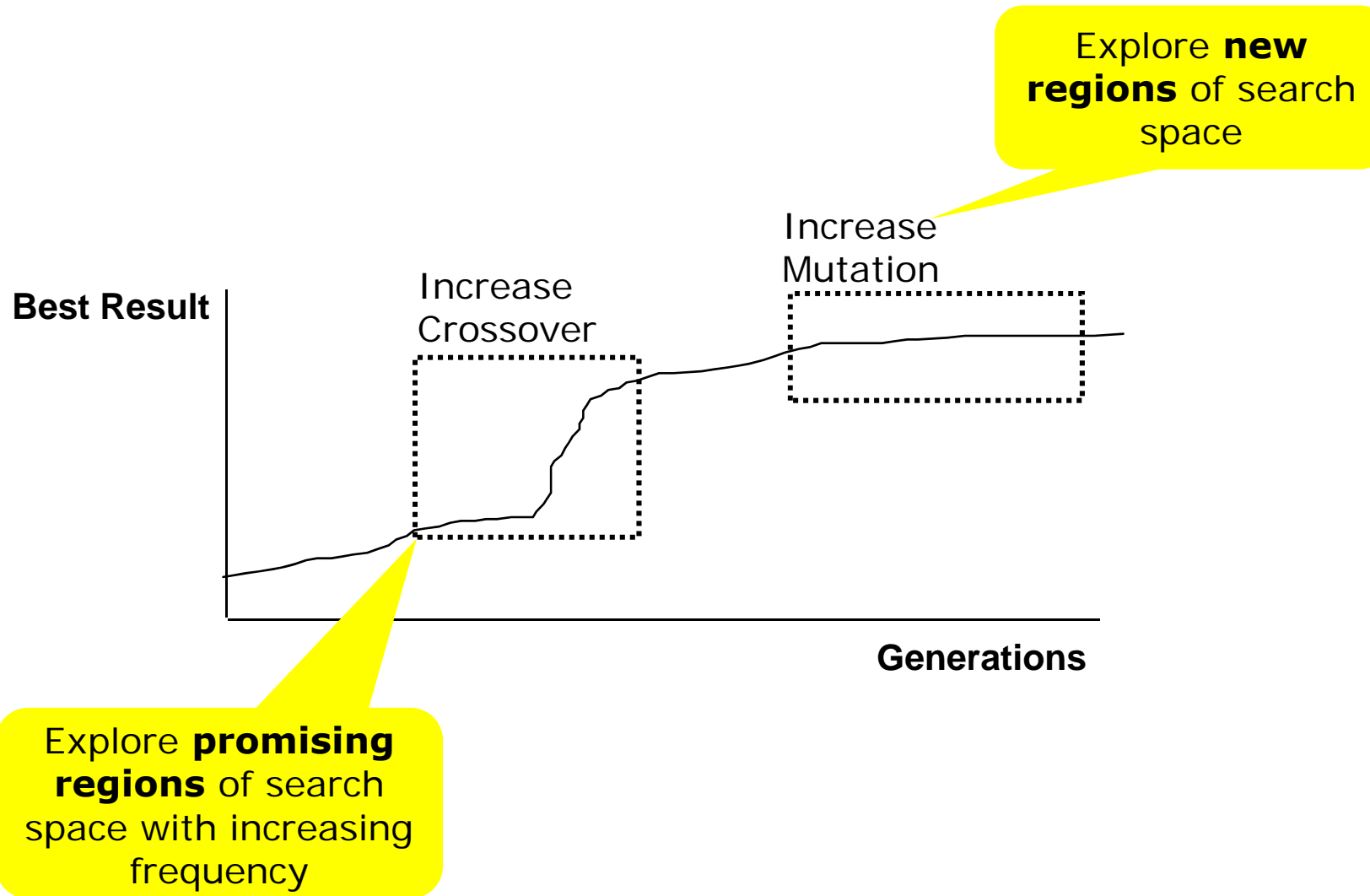
Adaptive GAs

- Static configurations of control parameters and encodings in GAs have some drawbacks.
- Parameter settings optimal in the earlier stages of the search typically become inefficient during the later stages.
- **Population size, crossover and mutation rates can be adaptively changed** during a run to maintain a balanced combination of exploration of new regions in the search space and exploitation of already sampled regions.

Adaptive GAs

- As a large fraction of the population has converged, crossover becomes ineffective in searching for better chromosomes
 - Typically low mutation rates (0.001 to 0.01) are inadequate for continuing exploration.
 - Increasing mutation rate to explore new regions of search space
- Monitor the improvements of the fitness with respect to the genetic operator settings.
- Dynamically modify the rates at which the various genetic operators are used based on the improvements over the last 50 operations.
- Helps solve new problems without any pre-determined operator settings.

Adaptive GAs



GA Theory – Schema Theorem

- GA theory provides some explanation why, for a given problem formulation, it is possible to:
 1. Obtain better results in subsequent generations
 2. Obtain convergence to some optimal point
- The **schema theorem** is typically described in the historical context using binary coding with a healthy dose of helpful assumptions

Schema = String Pattern

- Think of a binary string representation of solution
- We introduce the notion of **schema**
 - A schema is a string pattern that describes similar chromosomes, regardless of their desirability
 - A schema is built by introducing a ***don't care*** symbol, written *****, into the alphabet of genes.

One schema = how many strings?

- **A schema represents all strings which match it on all positions other than ***
- **One** don't care symbol
 - The schema (* 1 1 0 1 0 0) matches **two strings**
 1. (0 1 1 0 1 0 0)
 2. (1 1 1 0 1 0 0)
- **Two** don't care symbols
 - The schema (* 1 * 1 1 0 0 1) matches **four strings**
 1. (0 1 1 1 1 0 0 1)
 2. (1 1 1 1 1 0 0 1)
 3. (0 1 0 1 1 0 0 1)
 4. (1 1 0 1 1 0 0 1)
- So, every schema matches exactly **2^r strings**, where **r** is the number of don't care symbols * (recall that we are considering binary numbers with only 0's and 1's)

One string = how many schemata?

- But wait a minute – in GA, we don't model schemata but chromosomes in the form of strings with actual values.
- So, how many schemata does a single string match?

Examples

- A string (0) is matched by two schemata
 1. (0)
 2. (*)
- A string (01) is matched by four schemata
 1. (01) - exact copy of string
 2. (0*)
 3. (*1)
 4. (***) - all don't care symbols

One string = how many schemata?

- So how many schemata is a single **string of length m** matched by?
 - **2^m schemata**
- This is termed **implicit parallelism**
 - Through a single chromosome, many schemata are being sampled
 - Allows GA to sample a significantly larger number of schemata than the population size may suggest
- So search as many schemata as possible!

Population of Schemata

- In a population of size n , between 2^m and $n \cdot 2^m$ different schemata may be expressed
- Furthermore, consider a schema of length **m** where the value in each position may be 0, 1 or *
- Therefore, for **schemata of length m** , there are in total **3^m possible schemata**

Example: 27 possible schemata for 3-bit chromosome

(*, *, *) ✓ ✗ ☒
 (*, *, 0) ✗ ☒
 (*, *, 1) ✓
 (*, 0, *) ✓ ✗
 (*, 1, *) ☒
 (0, *, *) ✓ ☒
 (1, *, *) ✗
 (*, 0, 0) ✗
 (*, 0, 1) ✓
 (*, 1, 0) ☒
 (*, 1, 1)
 (0, *, 0) ☒
 (0, *, 1) ✓
 (1, *, 1)

(1, *, 0) ✗
 (0, 0, *) ✓
 (0, 1, *) ☒
 (1, 0, *) ✗
 (1, 1, *)
 (0, 0, 0)
 (0, 0, 1) ✓
 (0, 1, 0) ☒
 (0, 1, 1)
 (1, 0, 0) ✗
 (1, 0, 1)
 (1, 1, 0)
 (1, 1, 1)

(0, 0, 1) ✓
 (1, 0, 0) ✗
 (0, 1, 0) ☒

3 strings in fact sample
19 of 27 schemata

→ implicit parallelism

→ no need for massive population

Building blocks

- A **building block** is something which tends to “add value” because the solutions in which it appears are **on average** better than the solutions in which it does not appear.
- In the binary-coded case it will be something that is passed on from generation to generation, so perhaps it is a particular setting of certain bits e.g. “*10*1011***00” in a 16-bit problem

Building blocks

- Goldberg (1989) writes
“A genetic algorithm achieves high performance through the juxtaposition of **short, low order, highly fit schemata** or building blocks”
- Highly fit schemata are those whose instances in the population with an average fitness higher than the overall average fitness of the population
- But what's this “short” and “low order” business?

Defining Length

- The distance between the first and the last fixed positions in the schema
 - $(* * * \underline{0} 0 1 * 1 1 \underline{0}) \rightarrow \text{defining length is } 10 - 4 = 6$
 - $(\underline{1} 1 1 0 1 * * 0 0 \underline{1}) \rightarrow \text{defining length is } 10 - 1 = 9$

Why bother?

- It is useful in calculating survival probabilities of the schema after **crossover** which 'chops' up a chromosome. It is related to the **compactness of information** contained in a schema.
- **Shorter** defining lengths survive crossovers better!

Example: Effects of Defining Length

Consider the schemata

S1 (* 0 1 1 *)

S2 (* * * 1 1)

If the crossover point is
between 2nd and 3rd gene
then the offspring may not
match S1

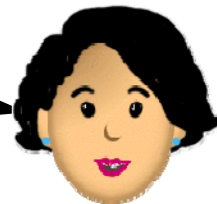
Before crossover

1	0	1	1	0	← based on S1 (* 0 1 1 *)
0	1	0	1	1	← based on S2 (* * * 1 1)

After crossover

1	0	0	1	1	← still matches S2 (* * * 1 1)
0	<u>1</u>	1	1	0	← no longer matches S1 (* <u>0</u> 1 1 *)

What's the
defining length
of S1 and S2?



Order

- The number of 0 and 1 positions
i.e. fixed positions present in the schema
 - $(* * * \underline{001} * \underline{110}) \rightarrow$ order is 6
 - $(\underline{11101} * * \underline{001}) \rightarrow$ order is 8

Again, why bother?

- Well, it is useful in calculating survival probabilities of the schema after **mutations** which arbitrarily changes values in fixed positions.
- **Lower order** schemata survive mutations better!

Example: Effects of Order

Consider schema S1

(*) (0) (1) (1) (*)

Consider this instance of S1

(1) (1) (1) (1) (1)

If any of the 3 bits in positions 2 to 4 is mutated then the string will not match S1 – we've lost schema S1!

Consider schema S2

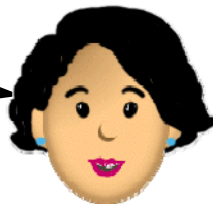
(*) (1) (*) (*) (1)

Consider this instance of S2

(1) (0) (1) (1) (1)

If the bit in position 2 or 5 is mutated then the string will not match S2 – we've lost schema S2!

What is the order of S1 and S2?



Overview of Schema Theorem

- Helps explain why GA works
 - Can GA really get better solutions in later generations?
 - Can GA converge on some optima?
- Three parts to the equation
 1. Effects of selection
 - **above average fitness** solutions are favoured
 2. Effects of crossover and defining length of schemata
 - **shorter** survive crossovers better
 3. Effects of mutation and order of schemata
 - **lower order** survive mutations better

Selection and Schema Survival

- For a given schema S
 - $E(S,t)$ is the number of strings in a population at the time t matching the schema S
 - $f(S,t)$ is the fitness of schema S at time t defined as follows:
 $f(S,t) = \sum_j f(v_j, t)/p$
where there are p strings $\{v_1, \dots, v_p\}$ in the population matching or representing the schema S at time t
- Let us now *assume* the use of fitness proportionate selection method

Selection and Schema Survival

- That is, the probability of a chromosome being selected is proportionate to its relative fitness
i.e. Roulette Wheel selection
- The chances of survival of the schema S is therefore expressed in terms of its fitness relative to the average population fitness and is written as follows:

$$\mathbf{f(S,t)/F(t)}$$

where $\underline{F(t)} = \sum_{\text{pop_size}} f(v_j, t) / \text{pop_size}$

the average fitness of the whole population at time t

Basic Schema Growth Equation

- The number of strings to be selected depends on the replacement strategy i.e. how many strings in the current generation need to be replaced
- After the selection step, we expect the number of strings matching the schema S to change as follows:

$$E(\mathbf{S}, t+1) = E(\mathbf{S}, t) \times \frac{f(\mathbf{S}, t)}{\underline{F}(t)} \quad \leftarrow \text{due to selection}$$

where $\underline{F}(t) = \sum_{\text{pop_size}} f(v_j, t) / \text{pop_size}$
the average fitness of the whole population at time t

Basic Schema Growth Equation (cont.)

- Since $\bar{F}(t)$ is the average fitness of the population at time t then we can make the claims:
 - $E(s,t)$ grows according to the *ratio of the fitness of the schema to the average fitness of the population*.
 - An “above average” schema receives an increasing number of strings in the next generation.
 - A “below average” schema receives decreasing number of strings.
- This is survival of the fittest at work!

Basic Schema Growth Equation (cont.)

- Let us rewrite the growth ratio $f(S,t)/\underline{F(t)}$ expressed exclusively using the average population fitness $\underline{F(t)}$
- To that end, let us define the constant ε to denote the growth constant for the schema S
- We can now rewrite the growth ratio $f(S,t)/\underline{F(t)}$ as follows:

$$\begin{aligned}\mathbf{f(S,t)/\underline{F(t)}} &= (\underline{F(t)} + \varepsilon \underline{F(t)}) / \underline{F(t)} \\ &= \mathbf{1 + \varepsilon}\end{aligned}$$

Modified Schema Growth Equation

- We can now rewrite the schema growth equation using the growth constant ε .
- We expect the number of strings matching the schema S to change as follows:

$$E(S, t+1) = E(S, t) \times (1 + \varepsilon) \quad \leftarrow \text{due to selection}$$

where $f(S, t)/\underline{F(t)} = 1 + \varepsilon$

- *Assumes* that the schema S remains above average by a constant amount ε (another simplification!)

Modified Schema Growth Equation

- And so we have: $E(\mathbf{S},1) = E(\mathbf{S},0) \times (1+\epsilon)$

$$\begin{aligned} E(\mathbf{S},2) &= E(\mathbf{S},1) \times (1+\epsilon) \\ &= (E(\mathbf{S},0) \times (1+\epsilon)) \times (1+\epsilon) \end{aligned}$$

- The long-term effect of this rule is

$$E(\mathbf{S},t) = E(\mathbf{S},0) \times (1+\epsilon)^t$$

- An “above average” schema, which defines a promising part of the search space, receives an **exponentially increasing** number of strings in the next generation.

Effects Of Genetic Operators

- Recombination takes the responsibility of introducing new individuals in the population through two genetic operators

recap

Defining length

Distance between first and last fixed positions in the schema

1. Crossover

→ recall that **shorter** survive crossovers better

recap

Order

Number of fixed positions – 0's and 1's – present in the schema

2. Mutation

→ recall that **lower order** survive mutations better

Crossover and Survival

- The **defining length** of a schema plays a significant role in the probability of its destruction and survival due to crossover.
- Let us suppose that a single crossover site is selected randomly among $(n-1)$ positions for chromosomes with length n
- The probability of **destruction** of a schema S is

$$p_{\text{dest}}(\mathbf{S}) = \delta(\mathbf{S}) / (n-1)$$

where $\delta(S)$ is the defining length of S

Crossover and Schema Survival

- That is, the more widely spread the fixed points in a schema, the higher the chances it would be destroyed due to crossover.
- Since only some chromosomes undergo crossover and the selection probability of crossover is p_c , the modified probability of destruction of a schema is

$$p_{\text{dest}}(\mathbf{S}) = p_c \times \delta(\mathbf{S}) / (n-1)$$

where $\delta(S)$ is the defining length of S

Schema Growth with Effects of Crossover

- Consequently, the probability of **survival** of a schema S is

$$\begin{aligned} p_{\text{surv}}(\mathbf{S}) &= \mathbf{1} - p_{\text{dest}}(\mathbf{S}) \\ &= \mathbf{1} - p_c \times \delta(\mathbf{S}) / (n-1) \end{aligned}$$

- The combined effect of selection and crossover gives a new form of the reproductive schema growth equation

$$\begin{aligned} E(\mathbf{S}, t+1) &= E(\mathbf{S}, t) \times (1 + \varepsilon) \times \quad \leftarrow \text{due to selection} \\ &\quad \left(\mathbf{1} - p_c \times \delta(\mathbf{S}) / (n-1) \right) \leftarrow \text{due to crossover} \end{aligned}$$

Mutation and Schema Survival

- A mutation operator randomly changes a single position within a chromosome with probability p_m . This is also the probability of **destruction** at a *single* position in the chromosome.
- Therefore, the probability of **survival** for a *single* position in the chromosome is $(1 - p_m)$.
- *All fixed positions* of a schema must remain unchanged if the schema is to survive mutation.

Mutation and Schema Survival

- A single mutation is independent from other mutations, so the probability of a schema S surviving mutation is

$$p_{\text{surv}}(\mathbf{S}) = (1 - p_m)^{o(\mathbf{S})}$$

where $o(\mathbf{S})$ is the order of \mathbf{S}

recap

Order
Number of fixed positions – 0's
and 1's – present in the schema

- Since $p_m \ll 1$, this probability can be simplified:

$$p_{\text{surv}}(\mathbf{S}) = (1 - p_m)^{o(\mathbf{S})}$$

$$\approx 1 - o(\mathbf{S}) \times p_m \quad \text{where } p_m \ll 1$$

- That is, the probability of survival is *penalised* by the amount $o(\mathbf{S}) \times p_m$

Schema Theorem Equation

- The combined effect of selection, crossover and mutation gives a new form of the reproductive schema growth equation also known as the

Schema Theorem:

Exponentially increasing or decreasing number of instances depending on relative fitness

$$E(S, t+1) = E(S, t) \times (1 + \varepsilon) \times \left(1 - p_c \times \delta(S)/(n-1) - o(S) \times p_m \right)$$

← due to selection

← due to crossover

← due to **mutation**

Probability of survival of schema due to crossover and mutation

Summary of Schema Theorem

1. Effects of **selection**

→ **above average fitness** solutions are favoured
in an exponential manner

2. Effects of **crossover** and defining length of schemata

→ **shorter** survive crossovers better

- crossover has the potential to destroy schemata
- the longer, the more prone to destruction

3. Effects of **mutation** and order of schemata

→ **lower order** survive mutations better

- mutation has the potential to destroy schemata
- the more fixed positions, the more prone to destruction

Let's Get Real!

- There are problems with the idealistic form of the Schema Theorem
 - Growth due to selection is exponential in the constant $(1 + \varepsilon)$.
 - Considers only *worst-case* scenario. It does not take into consideration the positive effects of genetic operators.
 - Focuses on the *number* of schema surviving. It does not say *which* schema will survive.
- Nonetheless, it's a milestone in explaining why GA works intuitively

Example - 1

The schema $S (*1* * *)$ has the properties: $\delta(S)=2-2=0$ and $o(S)=1$

Generation 1
 $\sum F = \underline{1415}$

String	Fitness
00110	36
✓ 0 <u>1</u> 010	100
10000	256
✓ 0 <u>1</u> 001	81
00011	9
00100	16
00001	1
✓ 0 <u>1</u> 100	144
✓ 1 <u>1</u> 000	576
✓ 0 <u>1</u> 110	196

$E(S,1) = 5$ (number of strings matching S at $t=1$)

Example - 2

The average fitness of the schema S (*1* * *) at t=1 is 219.4

Generation 1

$\sum F = \underline{1415}$

String	Fitness
00110	36
✓0 <u>1</u> 010	100
10000	256
✓0 <u>1</u> 001	81
00011	9
00100	16
00001	1
✓0 <u>1</u> 100	144
✓1 <u>1</u> 000	576
✓0 <u>1</u> 110	196

$$E(S,1) = 5$$

$$\begin{aligned} f(S,1) &= (100+81+144+576+196) / 5 \\ &= 1097/5 \\ &= 219.4 \end{aligned}$$

We can now calculate $E(S,2)$ the expected number of instances of schema S at t=2.

Example - 3

Given $\delta(S)=0$
 $o(S)=1$
 $E(S,1) = 5$
 $f(S,1) = 219.4$

Generation 1
 $\sum F = \underline{1415}$

String	Fitness
00110	36
✓ 0 <u>1</u> 010	100
10000	256
✓ 0 <u>1</u> 001	81
00011	9
00100	16
00001	1
✓ 0 <u>1</u> 100	144
✓ 1 <u>1</u> 000	576
✓ 0 <u>1</u> 110	196

Let us further assume the settings

$p_c = 0.6$ (crossover rate)

$p_m = 0.1$ (mutation rate)

Now, we can calculate $E(S,2)$.

$$E(S,2) = E(S,1) \times \frac{f(S,t)}{F(t)} \times (1 - p_c \times \delta(S)/(n-1) - o(S) \times p_m)$$

$$= 5 \times (219.4 / 141.5) \times (1 - 0.6 \times 0/4 - 1 \times 0.1)$$

= 6.98 instances of schema S at t=2

Example - 4

The Schema Theorem predicted 6.98 instances at $t=2$

Generation 1

$$\sum F = \underline{1415}$$

$$E(S,1) = 5$$

String	Fitness
00110	36
✓ 0 <u>1</u> 010	100
10000	256
✓ 0 <u>1</u> 001	81
00011	9
00100	16
00001	1
✓ 0 <u>1</u> 100	144
✓ 1 <u>1</u> 000	576
✓ 0 <u>1</u> 110	196

String Fitness

✓ 1 <u>1</u> 000	576
✓ 0 <u>1</u> 110	196
✓ 1 <u>1</u> 000	576
✓ 1 <u>1</u> 000	576
✓ 1 <u>1</u> 000	576
10001	289
✓ 1 <u>1</u> 100	784
✓ 1 <u>1</u> 000	576
✓ 0 <u>1</u> 000	64
✓ 1 <u>1</u> 110	900

Generation 2

$$\sum F = \underline{5113}$$

$$E(S,2) = 6.98$$

$$E'(S,2) = 9$$

The schema $S (*1* * *)$ actually has *nine* instances in the population at $t=2$
 → this is due to sampling errors arising from stochastic nature of GA

Practical Issues: why practice \neq theory

- There is a limit on the hypothetically unlimited number of iterations and population size
 - Estimates based on finite samples have a sampling error and lead to search trajectories much different from those theoretically predicted. This may lead to premature convergence to a local optimum.

Summary

- Issues affecting fitness
 - Fitness distribution
 - Different selection methods
- GA control parameters
 - Interaction between parameters
- Schema Theorem
 - Combined effects of selection, crossover survivability and mutation survivability

Appendix: Problem-Specific Operators

Special Problem Representations (Decoding Method)

- Consider the Travelling Salesman Problem (TSP)
- The traditional crossover & mutation operators which work well when solutions are coded as bit strings do not work when the solutions are coded as **sequences** of pre-defined, non-mutable values e.g. (C F B D E A)
- What operators do we need?

Unary Operators

Mutation

- Requires only one chromosome to generate one or more child chromosomes
- Commonly used for **mutation** but can be used for generating new chromosomes in general
- Some examples
 - Swap (C **F** B D E **A**) → (C **A** B D E **F**)
 - Slice (C F | **B D** | **E A**) → (C F | **E A** | **B D**)

Binary Operators

Order Crossover

- Creates children which preserve the order and position of symbols in a sub-sequence of one parent
- It also preserves the relative order of the remaining symbols from the second parent
- Example:
 - P1: (A B C | D E F | G H I)
 - P2: (H A E | **C B G** | I D F)yields
 - C1: (C B G | D E F | I H A)
 - C2: (D E F | **C B G** | H I A)


Binary Operators

Order Crossover

- Algorithm
 1. First the symbols between the cut-points are copied from P1 into the child.
 2. Then starting just after the second cut-point in P2, the symbols are copied from P2 into the child. Any symbols already copied are omitted.
 3. Then wrapping around to the first symbol in P2 until all of the symbols have been copied into the child. Any symbols already copied are omitted.
- The second child can be constructed by switching the roles of P1 and P2

Binary Operators

Order Crossover - Example

- Start with parent chromosomes
 - P1: (A B C | **D E F** | G H I)
 - P2: (H A E | C B G | I D F)
- First version of C1 is copied from P1's mid-section
 - C1: (_ _ _ | **D E F** | _ _ _)
- Consider P2 (H A E | C B G | I D F) from the second cut-point

 - Note that (D F) are both already in C1

Binary Operators

Order Crossover - Example

- Copy symbols from P2 (H A E | C B G | **I D F**)
not already in C1
 - C1: (_ _ _ | D E F | **I** _ _)
 - Only (I) copied since (D F) already in C1
- Consider P2 (**H A E** | C B G | I D F) from the start
 - C1: (_ _ _ | D E F | I **H A**)
 - Only (H A) copied since we have reached end of C1
- So, let's start filling up from beginning of C1
 - Only (C B G) copied since (E) already in C1
 - C1: (**C B G** | D E F | I H A)


DONE!

Binary Operators Cycle Crossover

- Creates children so that the position of each symbol in a child is always determined by one of its parent.
- Unlike order crossover, cycle crossover does not rely on cut points.
- Consider the parent chromosomes
 - P1: (**A** B C D E F G **H** I)
 - P2: (**H** A E C B G I **D** F)
- Compare the symbols of first parent P1 and second parent P2:
A displaces **H**, **H** displaces **D**, **D** displaces **C**, **C** displaces **E**,
E displaces **B**, **B** displaces **A**.

Binary Operators

Cycle Crossover - Example

- This is our initial cycle – **A H D C E B**
- The symbols **A H D C E B** assume the same position in the child C1 as they appear in P1
 - P1: (**A B C D E** F G **H I**)
 - C1: (**A B C D E** _ _ **H** _)
- Now, find the first symbol in second parent P2 not present in C1. It happens to be *G*.
 - C1: (**A B C D E** _ _ **H** _)
 - P2: (**H A E C B** *G* I D F)


Binary Operators Cycle Crossover - Example

- Let's continue our cycle, this time, starting with G in P2
 - P1: (A B C D E **F** G H **I**)
 - P2: (H A E C B **G** I D **F**)
- Compare the symbols of second parent P2 and first parent P1:
G in P2 displaces F in P1, **F** displaces I, **I** displaces G.
- This is our new cycle – **G F I**
- The positions of **G F I** are determined by P2
 - P2: (H A E C B **G I** D **F**)
 - C1: (A B C D E **G I** H **F**)

DONE!

Binary Operators

Partially Mapped Crossover

- Creates children which preserve the order and position of symbols in a sub-sequence of one parent while preserving the order and position of many of the remaining symbols from the other parent.
- These operators are especially useful for solving permutation problems like travelling salesman problem and scheduling

Binary Operators

Partially Mapped Crossover - Example

- Consider the parent chromosomes
 - P1: (A B C | D E F | G H I)
 - P2: (H A E | **C B G** | I D F)
- Copy the symbols between cut-points from the second parent P2.
 - C1: (_ _ _ | **C B G** | _ _ _)
- The remaining symbols are determined in a two-step process.

Binary Operators

Partially Mapped Crossover - Example

- First, the symbols in the first parent P1 *outside* the cut-points (and not already in C1) are copied to the corresponding positions of the child.
 - P1: (**A** B C | D E F | G **H** **I**)
 - C1: (**A** _ _ | C B G | _ **H** **I**)
- Next, each symbol in the first parent P1 *between* the cut points is placed in the child at the position occupied in the first parent P1 by the symbol from the second parent P2 that displaced it.

- P1: (A B C | **D** E F | G H I)
 - P2: (H A E | C B G | I D F)
 - C1: (A _ **D** | C B G | _ H I)

Binary Operators

Partially Mapped Crossover - Example

- Continued...

- P1: (A B C | D **E** F | G H I)
- P2: (H A E | C B G | I D F)
- C1: (A **E** D | C B G | _ H I)

- And finally

- P1: (A B C | D E **F** | G H I)
- P2: (H A E | C B G | I D F)
- C1: (A E D | C B G | **F** H I)

- We now have our first child chromosome

- C1: (A E D | C B G | F H I)