Nature-Inspired Computing

Genetic Algorithms

Dr. Şima Uyar
September 2006

Genetic Algorithms

- components of a GA
  - representation for potential solutions
  - method for creating initial population
  - evaluation function to rate potential solutions
  - genetic operators to alter composition of offspring
  - various parameters to control a run

Genetic Algorithms

- parameters of a GA
  - no. of generations
    - or other stopping criteria
  - population size
  - chromosome length
  - probability of applying some operators

Simple GA

Simple GA - SGA

- a.k.a. Canonical GA
- Operators of a SGA
  - selection
  - cross-over
  - mutation

SGA

```
generate initial population
repeat
  evaluate individuals
  perform reproduction
    select pairs
    recombine pairs
    apply mutation
until end_of_generations
```

Representation & Encoding

- population size constant
- individual has one chromosome (haploid)
- chromosome length constant
- individual has a fitness value
- binary genes (0/1)
- generational

Initial Population

random initial population

⇓

each gene value for each individual determined randomly to be either *0* or *1*
with equal probability

Fitness Evaluation

- fitness function
  - objective function(s)
  - constraints
- shows fitness of individual
  - degree to which solution candidate meets objective
- apply fitness function to individual

Example Problem: One-Max

Objective: maximize the number of **1**s in a string of length 5, composed only of **1**s and **0**s

⇒ *population size* = 4
chromosome length = 5
fitness function = no. of genes that are 1

Example Population

individual 1:
chromosome = 11001
fitness = 3

individual 2:
chromosome = 00001
fitness = 1

individual 3:
chromosome = 11111
fitness = 5

individual 4:
chromosome = 01110
fitness = 3

Reproduction

- consists of
  - selection
    - mating pool (size same as population)
    - possibly more than one copy of some individuals
  - cross-over
  - mutation

Selection

- uses *roulette wheel selection*
  - fitness proportionate
    - expected no. of representatives of each individual is proportional to its fitness

$$prob_i = \frac{fitness_i}{\sum_j fitness_j} \quad , j = 1 \ldots pop.size$$

Example Selection

Current Population:
i1: 11001, 3
i2: 00001, 1
i3: 11111, 5
i4: 01110, 3

Probability of each individual being selected:
prob(i1) = 3/12 = 0.25
prob(i2) = 1/12 = 0.08
prob(i3) = 5/12 = 0.42
prob(i4) = 3/12 = 0.25

Expected copies of each individual in pool:

i1: (3/12*4) 1
i2: (1/12*4) 0
i3: (5/12*4) 2
i4: (3/12*4) 1

Assume:
wheel is turned 4 times
1 copy of i1
2 copies of i3
1 copy of i4
is copied into mating pool

Example Pairing

Current mating pool:

mate 1: 11001 *(i1)*
mate 2: 11111 *(i3)*
mate 3: 11111 *(i3)*
mate 4: 01110 *(i4)*

Assume:
As a result of random drawing
(mate 1, mate 3)
and
(mate 2, mate 4)
are paired off for reproduction.

Pairs:

Pair 1:	**Pair 2:**
11001	11111
11111	01110

Recombination

- new individuals formed from pairs of parents
- one point cross-over
- probability of cross-over: p_c
  - a.k.a. cross-over rate
  - typically in range [0.5, 1.0]

One-Point Cross-Over

Example Cross-Over

Assume p_c=1.0

for pair 1:
cross-over site: 3
110 | 01 → 11011
111 | 11 → 11101

for pair 2:
cross-over site: 1
1 | 1111 → 11110
0 | 1110 → 01111

the new individuals:

i1: 11011
i2: 11101
i3: 11110
i4: 01111

Mutation

- bitwise mutation
- probability of mutation: p_m
  - a.k.a. mutation rate
  - equal probability for each gene
  - chromosome of length L; expected no. of changes: $L*p_m$
  - typically chosen to be small
    - depends on nature of problem

Bitwise Mutation

Example Mutation

Assume:
as a result of random draws,
1st gene of i1
and
4th gene of i3
are found to undergo mutation

i1: 11011 → 01011

i3: 11110 → 11100

Population Dynamics

- generational GA
  - non-overlapping populations
  - offspring replace parents

Example New Population

individual 1:
chromosome =01011
fitness = 3

individual 2:
chromosome =11101
fitness = 4

individual 3:
chromosome =11100
fitness = 3

individual 4:
chromosome =01111
fitness = 4

Stopping Criteria

- main loop repeated until stopping criteria met
  - for a predetermined no. of generations √
  - until a goal is reached
  - until population converges

Convergence

- progression towards uniformity
- *gene convergence:* when 95% of the individuals have the same value for that gene
- *population convergence:* when all genes have converged
  - average fitness approaches best

Example Stopping Criteria

Case 1:

number of generations= 250

- loop repeated 250 times
- best individual at each generation found
- overall best individual becomes solution

Example Stopping Criteria

Case 2:

objective: maximize the number of 1s

goal: to find the individual with 1 for all gene locations

- loop repeated forever
- best individual at each generation found
- terminates when goal individual found

Example Stopping Criteria

Case 3:

95% of 4 is 4

gene convergence: 4 individuals must have same value for a gene location

population convergence: 5 gene locations must be converged

Example converged populations:

Example 1:	Example 2:	Example 3:
i1: 11010	i1: 00000	i1: 11111
i2: 11010	i2: 00000	i2: 11111
i3: 11010	i3: 00000	i3: 11111
i4: 11010	i4: 00000	i4: 11111

Example Problems

Function Optimization

Objective:

Find the set of integers x_i which maximize the function **f**.

$$f(x_i)=\Sigma_i x_i^2 \qquad i=1,2,3$$

$$-512 < x_i \leq 512$$

Function Optimization

Representation:

- 1024 integers in given interval
- 10 bits needed

```
0    : 0000000000 (-511)
1    : 0000000001 (-510)
2    : 0000000010 (-509)
...        ...
1023 : 1111111111 (512)
```

Function Optimization

Individual:

- function has 3 parameters:
  x_1, x_2, x_3 **($-512 < x_i \leq 512$)**
- 10 bits for each x_i
- chromosome has 30 bits

Function Optimization

Example chromosome:

110010101011000000000000000110

x_1 x_2 x_3

$x_1 = (810 - 511) = 299$

$x_2 = (768 - 511) = 257$

$x_3 = (6 - 511) = -505$

fitness = $(299)^2 + (257)^2 + (-505)^2$

= 410475

Function Optimization

what if x_i were real numbers?

interval: $-5.12 < x_i \leq 5.12$

- possible to use binary
  - precision of 2 digits after decimal
  - use 1024 different integers (divide number by 100)
- use other representations (e.g. real)

Function Optimization

what if representation has redundancy?

e.g. interval: $-5.4 < x_i < 5.4$

0/1 Knapsack Problem

Objective:

$$\max \sum_i v_i * x_i \qquad i = 1,2,...item\_count$$

$$subject\ to \sum_i w_i * x_i \leq W$$

x_i = 0 / 1 (shows whether item i is in sack or not)

0/1 Knapsack Problem

Example item set:

(1) w= 2, v=10
(2) w= 6, v= 3
(3) w=10, v= 8
(4) w= 7, v=16
(5) w= 4, v=25

Example knapsack capacity: W = 12

Example feasible solutions:

items: {1,2,5} ⇒ weight=12, value= 38
items: {3} ⇒ weight=10, value= 8
items: {4,5} ⇒ weight=11, value= 41
items: {2} ⇒ weight=6, value= 3

0/1 Knapsack Problem

Representation:

5 items ⟹ chromosome length 5

Example chromosomes:

11001 ⇒ items {1,2,5} included in sack
00100 ⇒ items {3} included in sack
00011 ⇒ items {4,5} included in sack
01000 ⇒ items {2} included in sack

0/1 Knapsack Problem

- Can fitness be total weight of subset?
  - what if overweight?
- how to handle overweight subsets?
  - delete?
  - penalize?
    - by how much?
  - make correction?

Exercise Problem

In the Boolean satisfiability problem (SAT), the task is to make a compound statement of Boolean variables evaluate to TRUE. For example consider the following problem of 16 variables given in conjunctive normal form:

$$F = (x_5 \vee \bar{x}_{12} \vee x_{16}) \wedge (\bar{x}_4 \vee \bar{x}_6) \wedge (x_2 \vee x_{13} \vee \bar{x}_7 \vee \bar{x}_9 \vee \bar{x}_{14}) \wedge (x_1 \vee \bar{x}_8 \vee x_{11} \vee x_{15}) \wedge (x_3 \vee \bar{x}_{10})$$

Here the task is to find the truth assignment for each variable x_i *for all i=1,2,…,16* such that F=TRUE. Design a GA to solve this problem.

Genetic Algorithms: Representation of Individuals

Binary Representations

- simplest and most common
- chromosome: string of bits
  - genes: 0 / 1

example: binary representation of an integer

3: 00011
15: 01111
16: 10000

Binary Representations

problem: Hamming distance between consecutive integers may be > 1

example: 5 bit binary representation

14: 01110 15: 01111 16: 10000

Probability of changing 15 into 16 by independent bit flips (mutation) is not same as changing it into 14! (*hamming cliffs*)

√ Gray coding solves problem.

Gray Coding

- Hamming distance 1

Example: 3-bit Gray Code

integer	0	1	2	3	4	5	6	7
standard	000	001	010	011	100	101	110	111
gray	000	001	011	010	110	111	101	100

- algorithms exist for
  - gray ⇒ binary coding
  - binary ⇒ gray coding

Integer Representations

- binary representations may not always be best choice
  - another representation may be more natural for a specific problem
- e.g. for optimization of a function with integer variables

Integer Representations

- values may be
  - unrestricted (all integers)
  - restricted to a finite set
    - e.g. {0,1,2,3}
    - e.g. {North,East,South,West}

Integer Representations

- any natural relations between possible values?
  - obvious for ordinal attributes (e.g. integers)
  - maybe no natural ordering for cardinal attributes (e.g. set of compass points)

Real-Valued / Floating Point Representations

- when genes take values from a continuous distribution
- vector of real values
  - floating point numbers
- genotype for solution becomes the vector $\langle x_1, x_2, \ldots, x_k \rangle$ with $x_i \in \Re$

Permutation Representations

- deciding on sequence of events
  - most natural representation is permutation of a set of integers
- in ordinary GA numbers may occur more than once on chromosome
  - invalid permutations!
- new variation operators needed

Permutation Representations

- two classes of problems
  - based on order of events
    - e.g. scheduling of jobs
      - Job-Shop Scheduling Problem
  - based on adjacencies
    - e.g. Travelling Salesperson Problem (TSP)
      - finding a complete tour of minimal length between n cities, visiting each city only once

Permutation Representations

- two ways to encode a permutation
  - *i*th element represents event that happens in that location in a sequence
  - value of *i*th element denotes position in sequence in which *i*th event occurs

Permutation Representations

Example (TSP):

4 cities A,B,C,D and permutation [3,1,2,4] denotes the tours:

first encoding type:
[C→A→ B→ D]

second encoding type:
[B→C→ A→ D]

Genetic Algorithms: Mutation

Mutation

- a variation operator
- create one offspring from one parent
- acts on genotype
- occurs at a mutation rate: p_m
  - behaviour of a GA depends on p_m

Bitwise Mutation

- flips bits
  - 0→1 and 1→0
- setting of p_m depends on nature of problem
  - usually (expected occurence) between 1 gene per generation and 1 gene per offspring

Bitwise Mutation (Binary Representations)

Integer Representations: Random Resetting

- bit flipping extended
- acts on genotype
- mutation rate: p_m
- a permissible random value chosen
- most suitable for cardinal attributes

Integer Representations: Creep Mutation

- designed for ordinal attributes
- acts on genotype
- mutation rate: p_m

Integer Representations: Creep Mutation

- add small (positive / negative) integer to gene value
  - random value
  - sampled from a distribution
    - symmetric around 0
    - with higher probability of small changes

Integer Representations: Creep Mutation

- *step size* is important
  - controlled by parameters
  - setting of parameters important
- different mutation operators may be used together
  - e.g. "big creep" with "little creep"
  - e.g. "little creep" with "random resetting" (different rates)

Floating-Point Representations: Mutation

- allele values come from a continuous distribution
- previously discussed mutation forms not applicable
- special mutation operators required

Floating-Point Representations: Mutation Operators

- change allele values randomly within its domain
  - upper and lower boundaries
    - U_i and L_i respectively

$$< x_1, x_2, ..., x_n > \rightarrow < x_1', x_2', ..., x_n' >$$
$$where\ x_i, x_i' \in [L_i, U_i]$$

Floating-Point Representations: Uniform Mutation

- values of x_i drawn uniformly randomly from the $[L_i,U_i]$
  - analogous to
    - bit flipping for binary representations
    - random resetting for integer representations
- usually used with positionwise mutation probability

Floating-Point Representations: Non-Uniform Mutation with a Fixed Distribution

- most common form
- analogous to creep mutation for integer representations
- add an amount to gene value
- amount randomly drawn from a distribution

Floating-Point Representations: Non-Uniform Mutation

- Gaussian distribution (normal distribution)
  - with mean 0
  - user-specified standard deviation
  - may have to adjust to interval $[L_i,U_i]$

Floating-Point Representations: Non-Uniform Mutation

- Gaussian distribution
  - 2/3 of samples lie within one standard deviation of mean
    - most changes small but probability of very large changes > 0
- Cauchy distribution with same standard deviation
  - probability of higher values more than in gaussian distribution

Floating-Point Representations: Non-Uniform Mutation

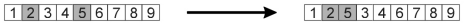
- usually
  - applied to each gene with probability 1
  - p_m used to determine standard deviation of distribution
    - determines probability distribution of size of steps taken

Permutation Representations: Mutation Operators

- not possible to consider genes independently
- move alleles around in genome
- mutation probability shows probability of a string undergoing mutation

Permutation Representations: Swap Mutation

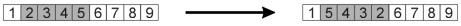
Permutation Representations: Insert Mutation



Permutation Representations: Scramble Mutation

- May act on whole string or a subset

Permutation Representations: Inversion Mutation



Genetic Algorithms: Recombination

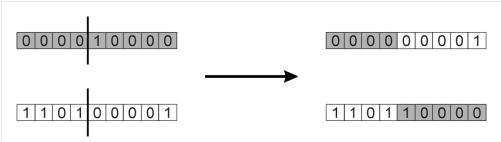
Recombination

- process for creating new individual
  - two or more parents
- term used interchangably with crossover
  - mostly refers to 2 parents
- crossover rate p_c
  - typically in range [0.5,1.0]
  - acts on parent pair

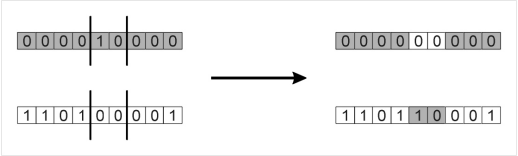
Recombination

- two parents selected randomly
- a r.v. drawn from [0,1)
- if value < p_c two offspring created through recombination
- else two offspring created asexually
  - copy of parents

Binary Representations: One-Point Crossover



Binary Representations: N-Point Crossover



Example: N=2

Binary Representations: Uniform Crossover

Assume array: [0.35, 0.62, 0.18, 0.42, 0.83, 0.76, 0.39, 0.51, 0.36]]

Binary Representations: Crossover

- positional bias
  - e.g. in 1-point crossover bias against keeping bits at head and tail of string together
- distributional bias
  - in uniform crossover bias is towards transmitting 50% of genes from each parent

Integer Representations: Crossover

- same as in binary representations
- blending is not useful
  - averaging even and odd integers produce a non-integer !

Floating-Point Representations: Recombination

- discrete recombination
  - similar to crossover operators for bit-strings
  - alleles have floating-point representations
  - offspring z, parents x and y

    value of allele i in offspring:

    $z_i=x_i$ or $z_i= y_i$ with equal probability

Floating-Point Representations: Recombination

- intermediate or arithmetic recombination
  - for each gene position
  - new allele value between those of parents

    $z_i=\alpha x_i +(1-\alpha)y_i$ where α in [0,1]
  - new allelele values
  - averaging reduces range of values in population

Floating-Point Representations: Arithmetic Recombination

- sometimes random α
- usually constant α =0.5
  - uniform arithmetic recombination
- 3 types
  - simple a. r.
  - single a. r.
  - whole a. r.

Floating-Point Representations: Simple Arithmetic Recombination

- pick random recombination point k

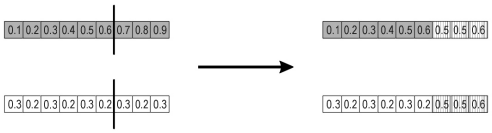
child1:

$<x_1,\ldots,x_k,\ \alpha y_{k+1}+(1-\alpha)x_{k+1},\ldots,\ \alpha y_n+(1-\alpha)x_n>$

child2:

$<y_1,\ldots,y_k,\ \alpha x_{k+1}+(1-\alpha)y_{k+1},\ldots,\ \alpha x_n+(1-\alpha)y_n>$

Floating-Point Representations: Simple Arithmetic Recombination



Example: k=8, α=0.5

Floating-Point Representations: Single Arithmetic Recombination

- pick a random allele k

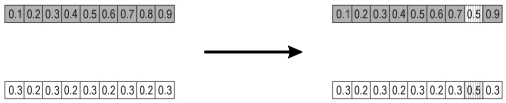
child1:

$\langle x_1, \ldots, x_{k-1}, \alpha y_k + (1-\alpha) x_k, x_{k+1}, \ldots, x_n \rangle$

child2:

$\langle y_1, \ldots, y_{k-1}, \alpha x_k + (1-\alpha) y_k, y_{k+1}, \ldots, y_n \rangle$

Floating-Point Representations: Single Arithmetic Recombination



Example: k=3, α=0.5

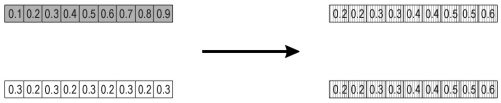
Floating-Point Representations: Whole Arithmetic Recombination

- most commonly used
- takes weighted sum of alleles from parents

$$Child1 = \alpha . \bar{x} + (1-\alpha) . \bar{y}$$

$$Child2 = \alpha . \bar{y} + (1-\alpha) . \bar{x}$$

Floating-Point Representations: Whole Arithmetic Recombination



Note: if α=0.5 two offspring are identical!

Permutation Representations: Recombination

- requires specially designed operators

- for adjacency representations
  - partially mapped crossover (PMX)
  - edge crossover
- for order based representations
  - order crossover
  - cycle crossover

Multiparent Recombination

- more than two parents
- may be advantageous for some groups of problems
- not widely used in EC
- approaches grouped as:
  - based on allele frequencies
    - generalizing uniform crossover

Multiparent Recombination

- based on segmentation and recombination (e.g. diagonal crossover)
  - generalizing n-point crossover
- based on numerical operations on real valued alleles (e.g. the center of mass crossover)
  - generalizing arithmetic recombination operators

Genetic Algorithms: Fitness Functions

Fitness

- Fitness shows how good a solution candidate is
- Not always possible to use real (raw) fitness
  - Fitness determined by objective function(s) and constraint(s)
  - Sometimes approximate fitness functions needed

Fitness

- population convergence $\Rightarrow$ fitness range decreases
  - premature convergence
    - good individuals take over population
  - slow finishing
    - when average nears best,
    - not enough diversity
    - can't drive population to optima

    i.e. best and medium get equal chances

Fitness

- Fitness remapping schemes needed
  - Fitness scaling
  - Fitness windowing

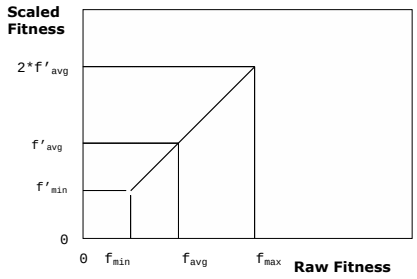
Linear Scaling

If
f: raw fitness and f':scaled fitness
then linear relationship,
$f' = af + b$

Linear Scaling

a and b chosen such that
$f'_{avg} = f_{avg}$ and $f'_{max} = C_{mult} * f_{avg}$
where
C_{mult}: expected no. of copies of best individual in population
(*Note:* Typically for populations of size 50 to 100, C_{mult}=1.2 to 2 is used.)

Linear Scaling



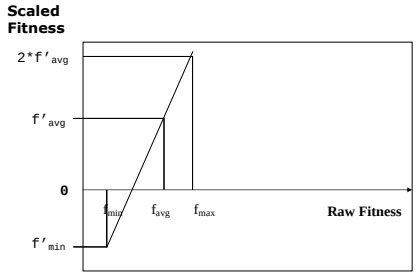
Linear Scaling

- In later runs,
  - average close to maximum
  - some very bad individuals greatly below population average

$\Rightarrow$ possible negative scaled fitnesses

- Solution: map minimum raw fitness to $f'_{min}=0$

Linear Scaling



Sigma Scaling

- developed as improvement to linear scaling
  - to deal with negative values
  - to incorporate problem dependent information into the mapping
  - population average and standard deviation

Sigma Scaling

$$f' = f + (\overline{f} - c * \sigma)$$

c: small integer (usually set to 2)
σ: population's standard deviation

if f' <0 then set f'=0

Window Scaling

- Scaling window
  f'=F-f where *F* is a constant and *F>f(x)* for all *x*
  - scaling window W: determines how often F is updated
    - W>0 ⇒ F=max{f(x)} for the last W generations
    - W=0 ⇒ infinite window size, i.e. F=max{f(x)} over all evaluations

Genetic Algorithms: Population Models

Population Models

- generational model
- steady state model

Generational Model

- population of individuals : size N
- mating pool (parents) : size N
- offspring formed from parents
- offspring replace parents
- offspring are next generation : size N

Steady State Model

- not whole population replaced
- N: population size (M≤N)
  - M individuals replaced by M offspring
- generational gap
  - percentage of replaced
  - equal to M/N
- competition based on fitness

Genetic Algorithms: Parent Selection

Fitness Proportional Selection

- FPS
- e.g. roulette wheel selection
- selection probability depends on *absolute* fitness of individual compared to *absolute* fitness of rest of the population

Selection

- *Selection scheme*: process that selects an individual to go into the mating pool
- *Selection pressure*: degree to which the better individuals are favoured
  - if higher selection pressure, better individuals favoured more

Selection Pressure

- determines convergence rate
  - if too high, possible premature convergence
  - if too low, may take too long to find good solutions

Selection Schemes

- two types:
  - proportionate
  - ordinal based

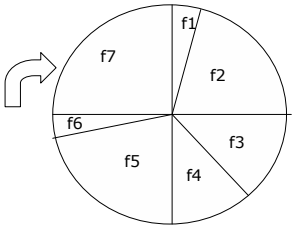
Fitness Proportionate Selection

- e.g. roulette-wheel selection (RWS)
- problems with FPS
  - premature convergence
  - almost no selection pressure when fitness values close together
  - may behave differently on transposed versions of same fitness function
    - e.g. consider f(x) and y(x)=f(x)+10;

Fitness Proportionate Selection

- solutions
  - scaling
  - windowing

Roulette-Wheel Selection



m=7

Roulette-Wheel Selection

```
begin
  set current_member=1;
  while (current_member ≤ m)do
    pick uniform r.v. r from [0,1];
    set i=1;
    while (a_i < r) do
      set i=i+1;
    od
    set mating_pool[current_member]=parents[i];
    set current_member=current_member+1;
  od
end
```

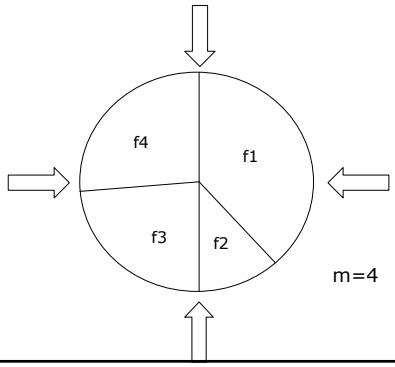
m: population size $a_i = \sum_1^i P_{sel}(i) \quad i = 1,2,...,m$

Stochastic Universal Sampling

- SUS
- one spin of wheel with m equally spaced arms
- cumulative selection probabilities

  $[a_1, a_2, \ldots\ldots, a_m]$

Stochastic Universal Sampling



m=4

Stochastic Universal Sampling

```
begin
  set current_member=i=1;
  pick uniform r.v. r from [0,1/m];
  while (current_member ≤ m) do
    while (r ≤ a[i]) do
      set mating_pool[current_member]=parents[i];
      set r=r+1/m;
      set current_member=current_member+1;
    od
    set i=i+1;
  od
end
```

m: population size $a_i = \sum_1^i P_{sel}(i) \quad i = 1,2,...,m$

Ranking Selection

- ordinal based
- population sorted by fitness
- selection probabilities based on rank
- constant selection pressure
- how to allocate probabilities to ranks
  - can be any linear or non-linear function
    - e.g. linear ranking selection (LRS)

Linear Ranking

- parameter s: $1.0 < s \leq 2.0$
  - in generational GA s: no. of expected offspring allotted to best

Assume best has rank *m* and worst *1*

- selection probability of individual with rank i:

$$p_{sel}(rank\_i) = \frac{(2-s)}{m} + \frac{2i(s-1)}{m(m-1)}$$

FPS x LRS

	Fitness	Rank	FP	LR (s=2)	LR (s=1.5)
A	1	1	0.1	0	0.167
B	5	3	0.5	0.67	0.5
C	4	2	0.4	0.33	0.33
Sum	10		1.0	1.0	1.0

Exponential Ranking

- with linear mapping
  - range of selection pressure limited
    - max s=2 (median fitness has 1 chance)
  - if wish to select above average more
    - exponential ranking

$$p_{sel}(rank\_i) = \frac{1-e^{-i}}{c}$$ c: normalization factor

Tournament Selection

- ordinal based
- RWS and SUS uses info on whole population
  - info may not be available
    - population too large
    - population distributed on a parallel system
    - maybe no universal fitness definition (e.g. game playing, evol. art, evol. design)

Tournament Selection

- TS
- relies on an ordering relation to rank any n individuals
- most widely used approach
- tournament size k
  - if k large, more of the fitter individuals
  - controls selection pressure
    - k=2 : lowest selection pressure

Tournament Selection

```
begin
  set current_member=1;
  while (current_member ≤ m)do
    pick k inividuals randomly;
    select best from k individuals;
    denote this individual i;
    set mating_pool[current_member]=i;
    set current_member=current_member+1;
  od
end
```

m: population size k: tournament size

Genetic Algorithms: Survivor Selection

Survivor Selection

- a.k.a. replacement
- determines who survives into next generation
  - reduces (m+l) to m
    - m population size (also no. of parents)
    - l no. of offspring at end of generation
- several replacement strategies

Age-Based Replacement

- fitness not taken into account
- each inidividual exists for same number of generations
  - in SGA only for 1 generation
- e.g. create 1 offspring and insert into population at each generation
  - FIFO
  - replace random (has more performance variance than FIFO; not recommended)

Fitness-Based Replacement

- uses fitness to select m individuals from (m+l) (m parents, l offspring)
  - fitness based parent selection techniques
  - replace worst
    - fast increase in population mean
    - possible premature convergence
    - use very large populations or no-duplicates
  - elitism
    - keeps current best in population
    - replaces an individual (worst, most similar, etc)