Fitness Uniform Deletion:
A Simple Way to Preserve Diversity

Shane Legg & Marcus Hutter

IDSIA
Galleria 2
6928 Manno-Lugano
Switzerland

The problem of population diversity

- Diversity tends to collapse over time
- Especially bad for deceptive problems or when high selection pressure is used
- Can be difficult to measure how similar individuals are based on their genes, e.g. neural nets and in genetic programming
- Diversity loss occurs when we delete uncommon individuals

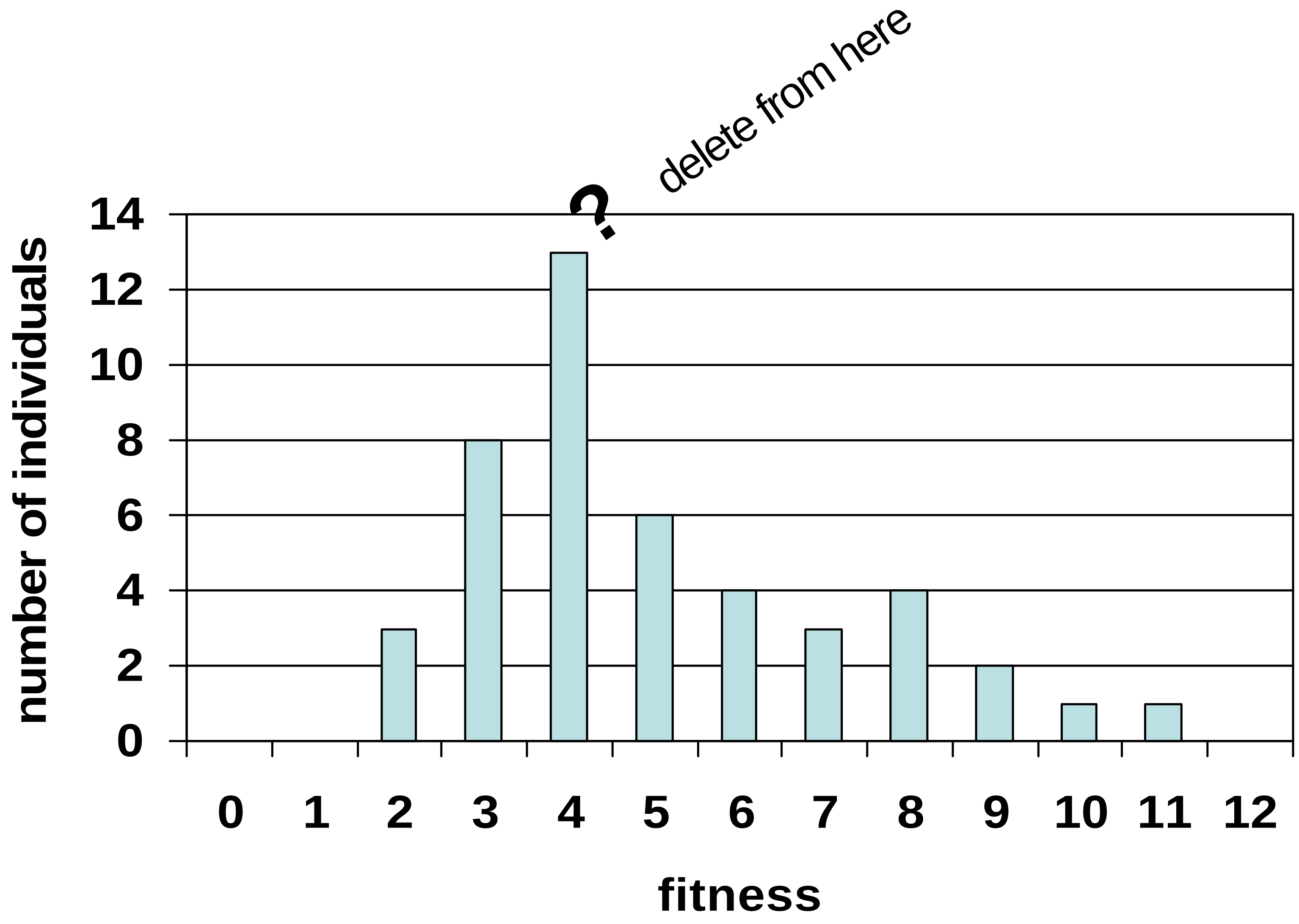
Fitness Uniform Deletion Scheme (FUDS)

The intuition:

- If we delete an individual which has a unique fitness value we must be losing population diversity
- Conversely, if we delete an individual with a very common fitness value we are probably not losing much diversity

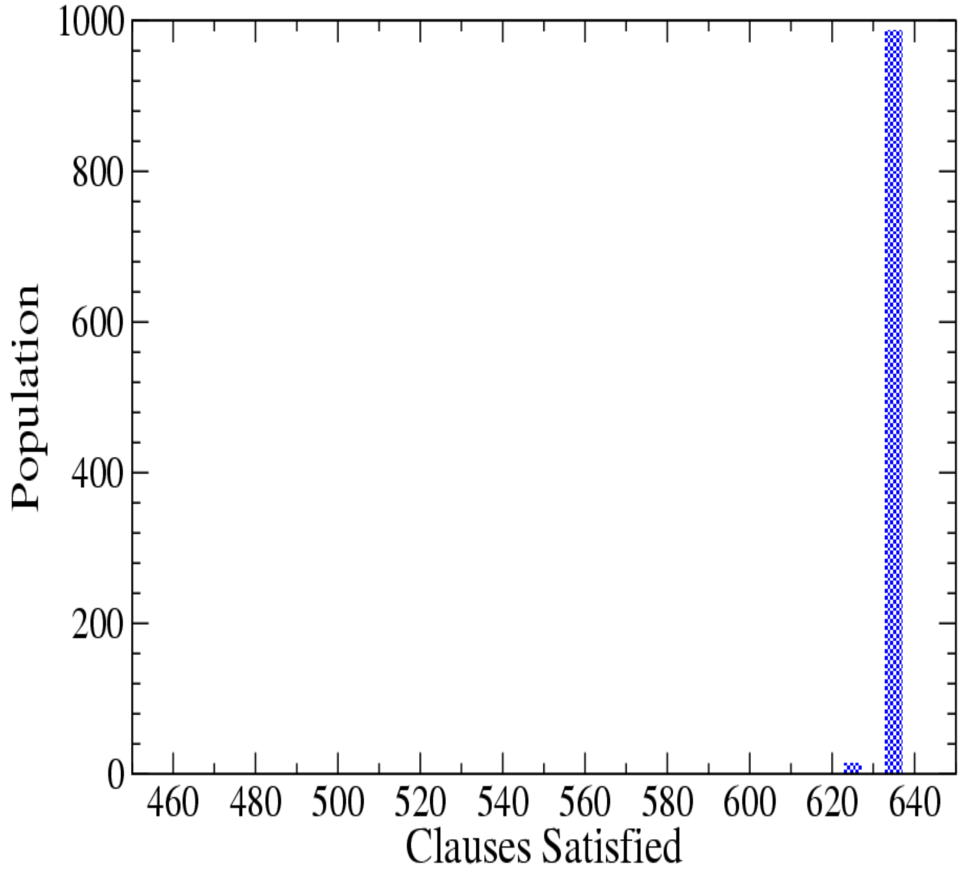
Thus, if we must delete, delete individuals with commonly occurring fitness values. This should help maintain population diversity and allow easy escape from local optima.

Simple Implementation!

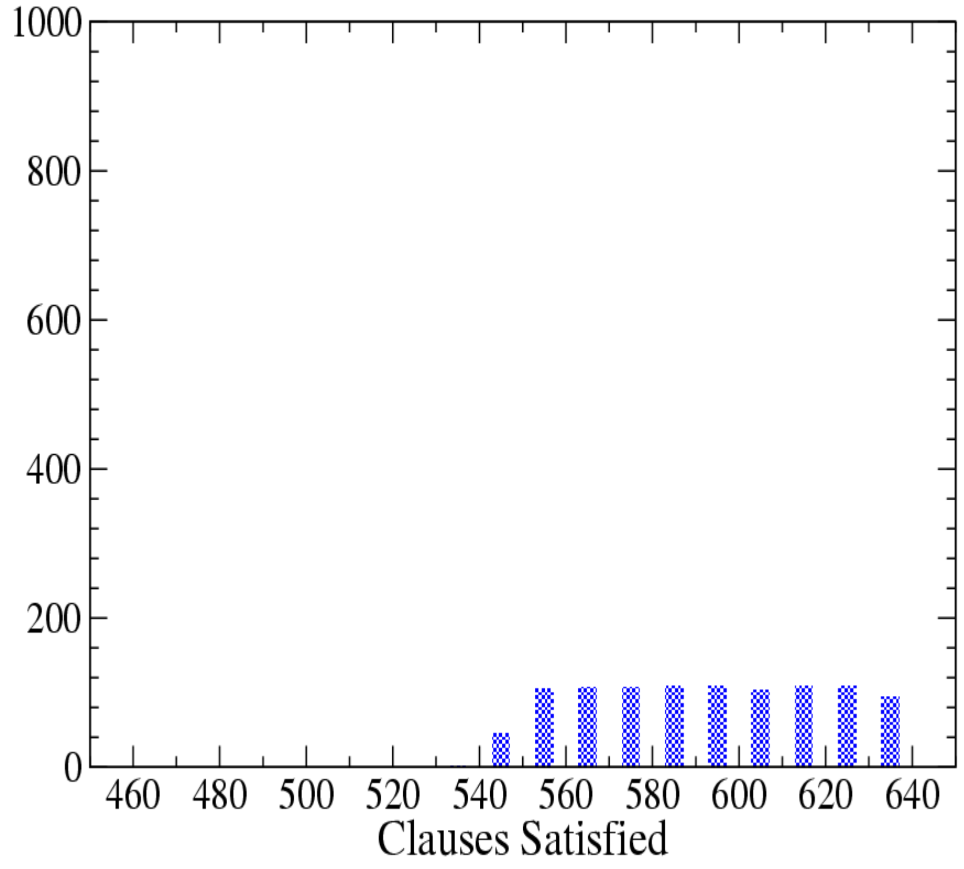


Example Population Distributions: CNF3 SAT with Tournament Selection

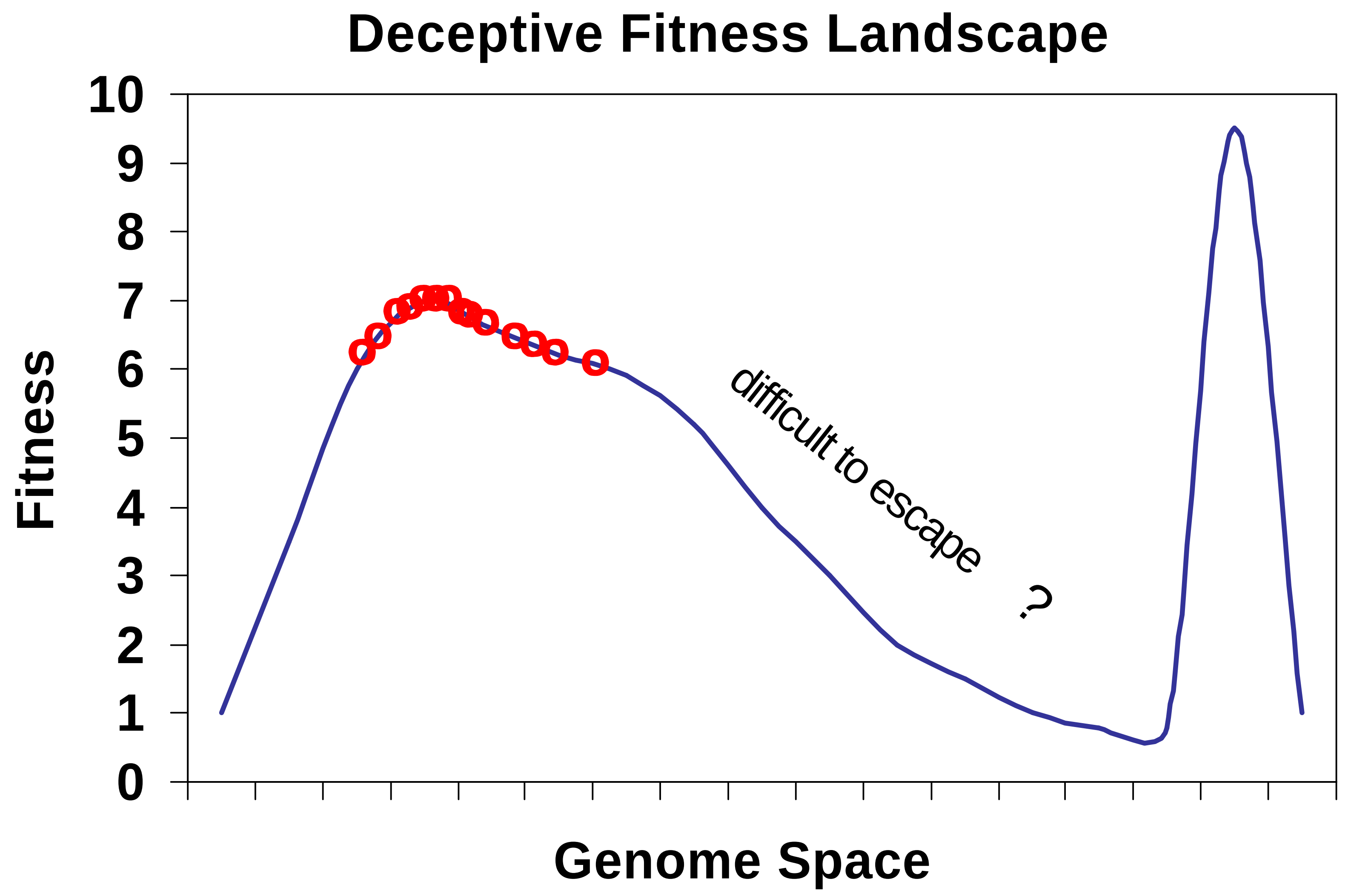
Random Deletion



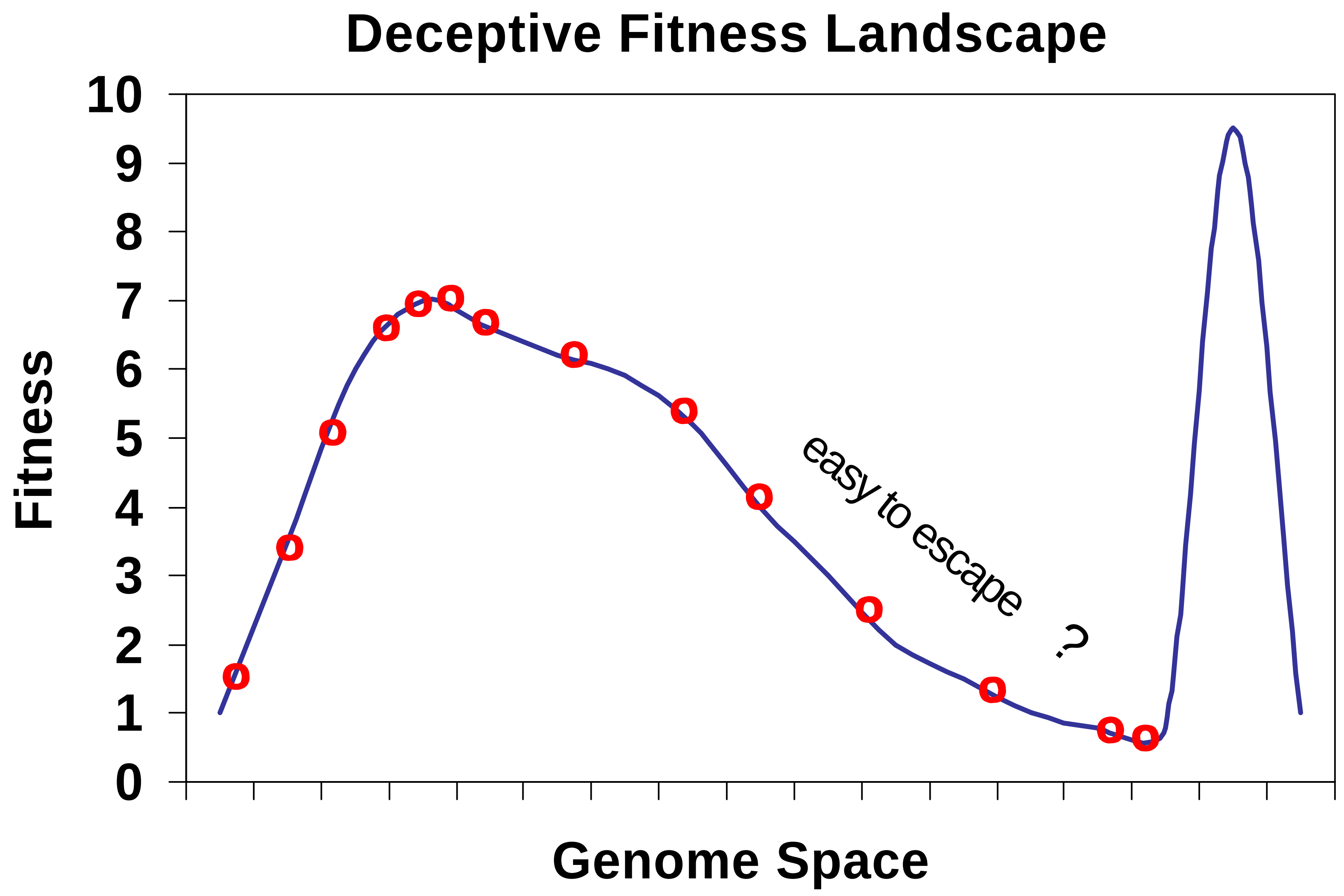
FUDS



Resulting Population Distribution when using Random Deletion

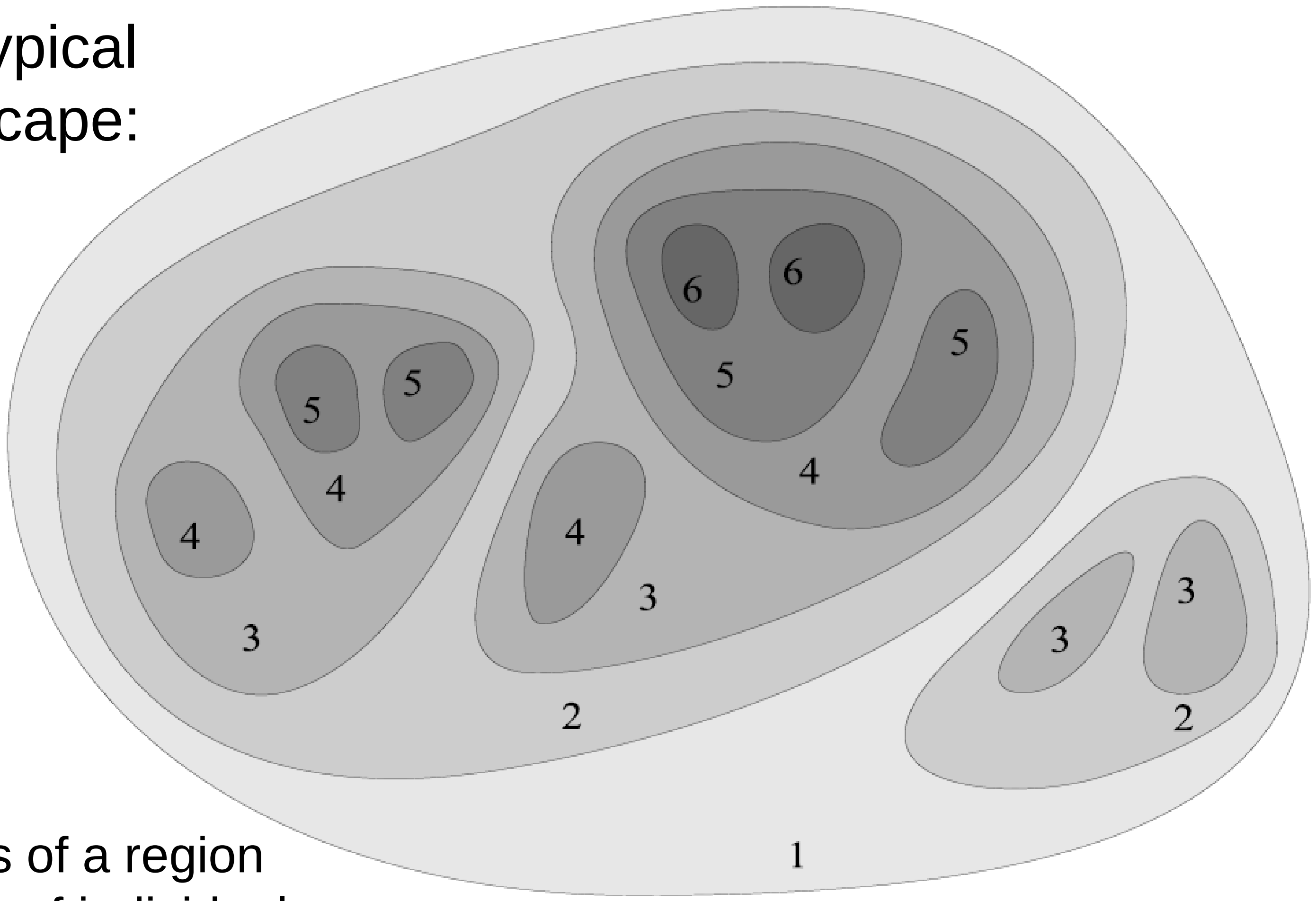


Resulting Population Distribution when using FUDS



FUDS is *not* uniform in the genome space!

Consider a typical fitness landscape:



numbers = fitness of a region
shading = density of individuals

With FUDS each fitness level tends to have the same number of individuals. Thus as high fitness regions are usually smaller, they will have a higher density of individuals.

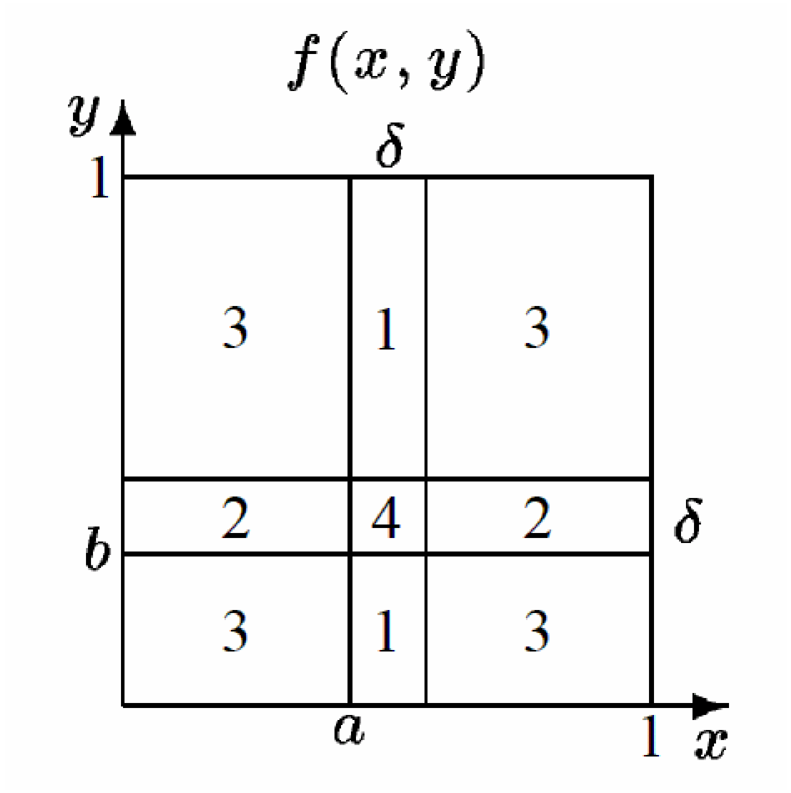
Properties of FUDS

- Uses fitness to control population diversity
- Doesn't require a similarity metric on the genome space to be constructed
- Problem and representation independent
- Trivial to implement
- Computationally efficient
- Should be good for deceptive problems with many local optima and deep valleys in the fitness landscape

Test System

- Implemented FUDS in Java
- Steady State GA rather than a Generational GA
- Used tournament selection for all tests
- Compared FUDS against random deletion
- Generations = number of cycles / population size
- Source code and test problems available on the internet (see paper)

Artificial Deceptive Problem

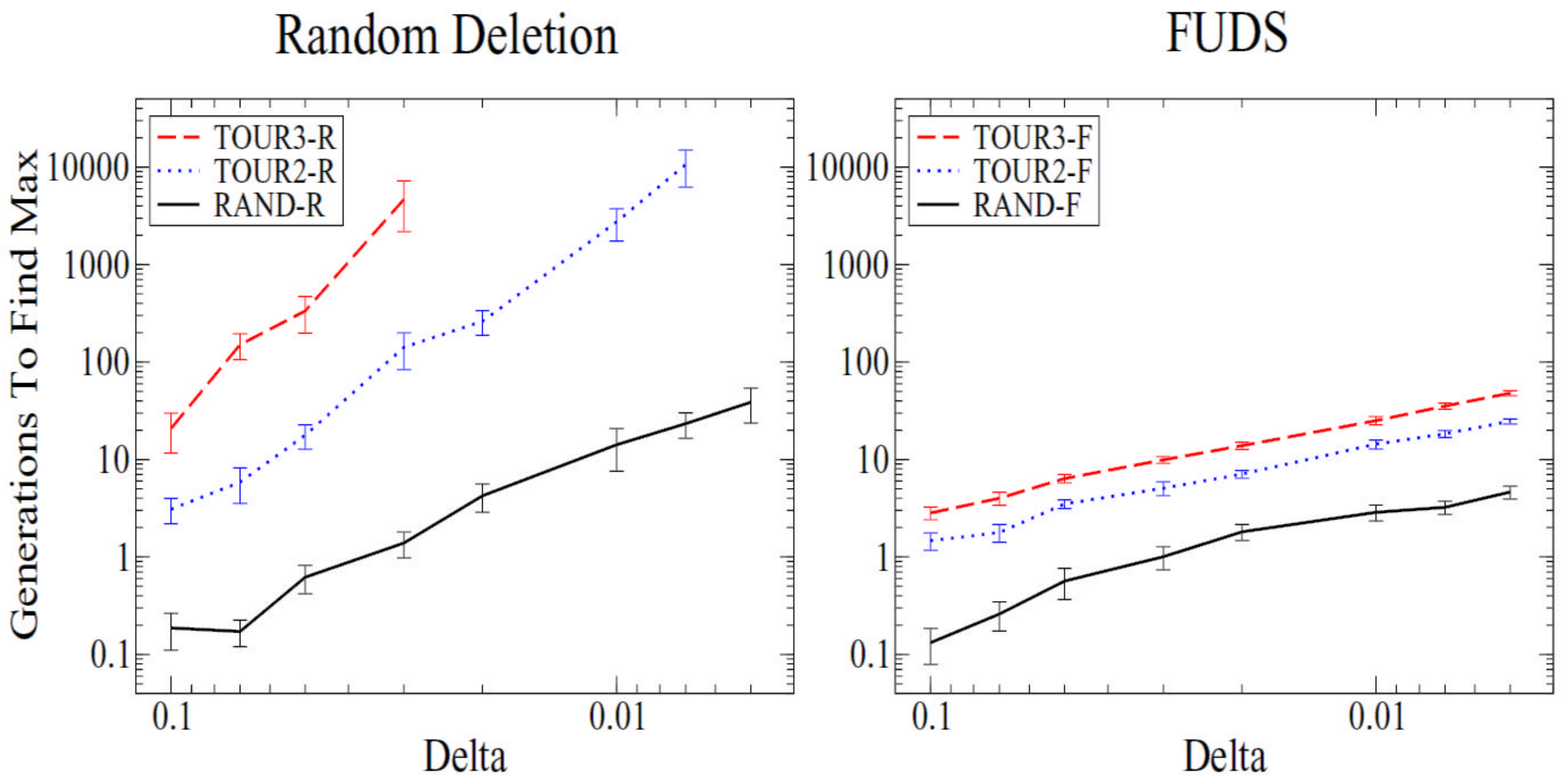


Mutation Operator = new random value for the *x* or *y* from [0,1]

Crossover Operator = *x* from one parent, *y* from the other

Default mutation and crossover probabilities of 0.5

Artificial Deceptive Problem



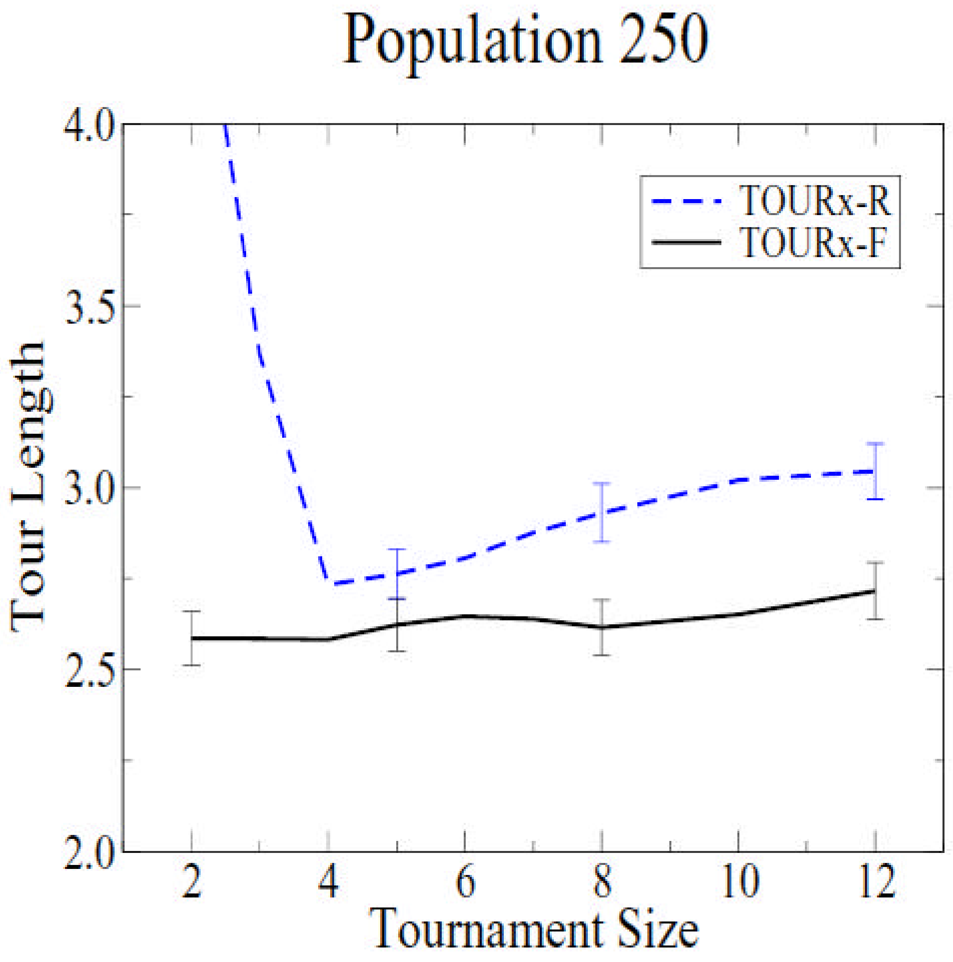
Error bars on graphs represent 95% confidence intervals

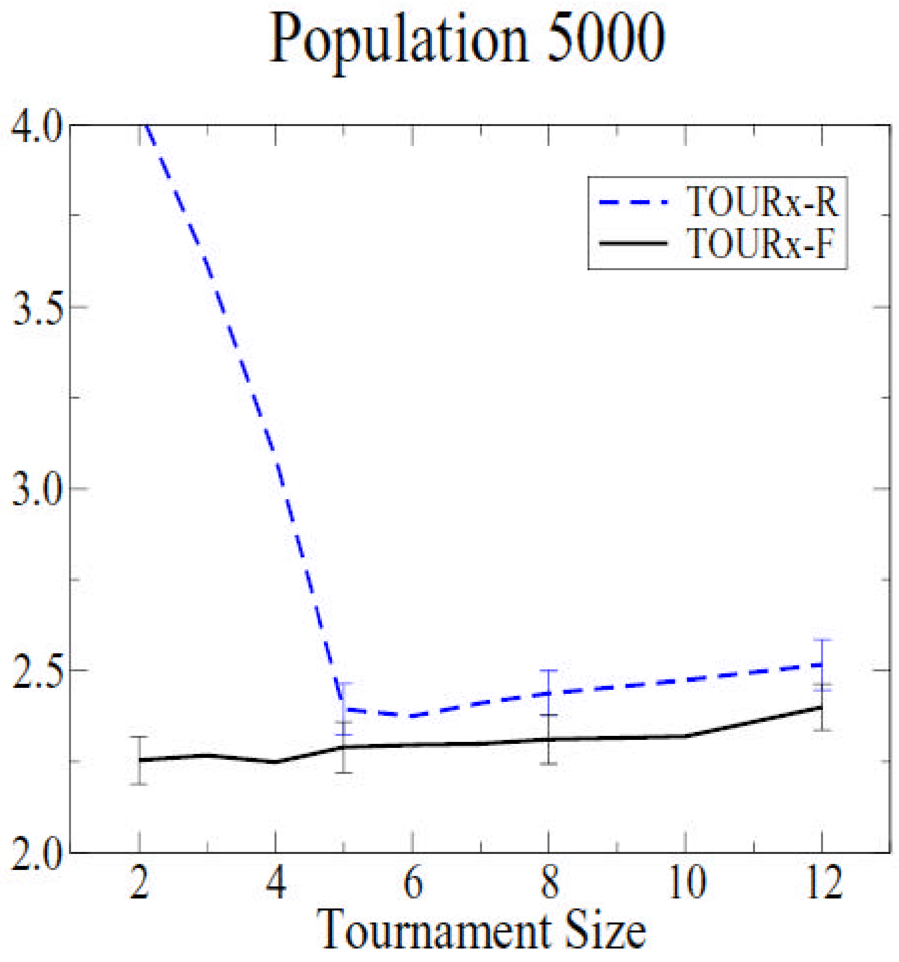
Random Distance TSP

- Distance between each pair of cities is random from [0,1]
- Triangle inequality does **not** hold in general
- Deceptive version of TSP but still has some structure

- Mutation operator = swap position of two cities in the tour
- Crossover operator = "Partial Match Crossover"
- 50 runs with 20 cities in each test

Random TSP


Population 250
Tour Length
4.0
3.5
3.0
2.5
2.0
2
4
6
8
10
12
Tournament Size
TOURx-R
TOURx-F

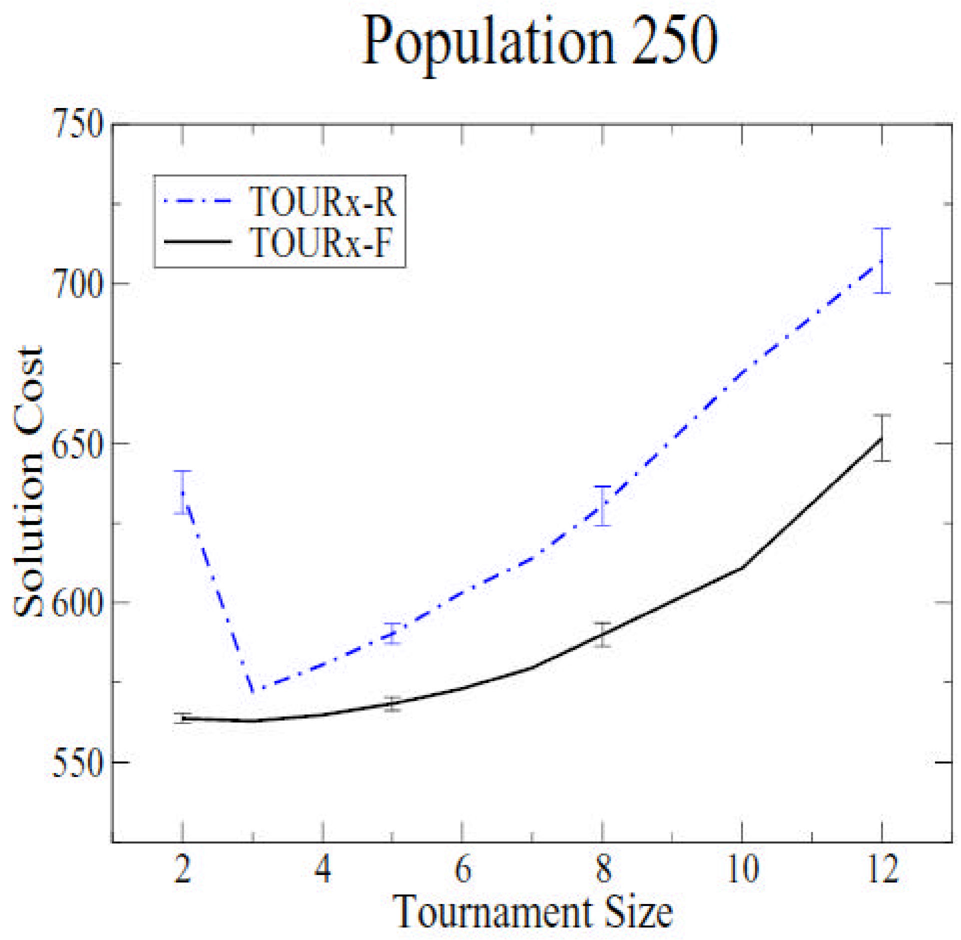

Population 5000
4.0
3.5
3.0
2.5
2.0
2
4
6
8
10
12
Tournament Size
TOURx-R
TOURx-F

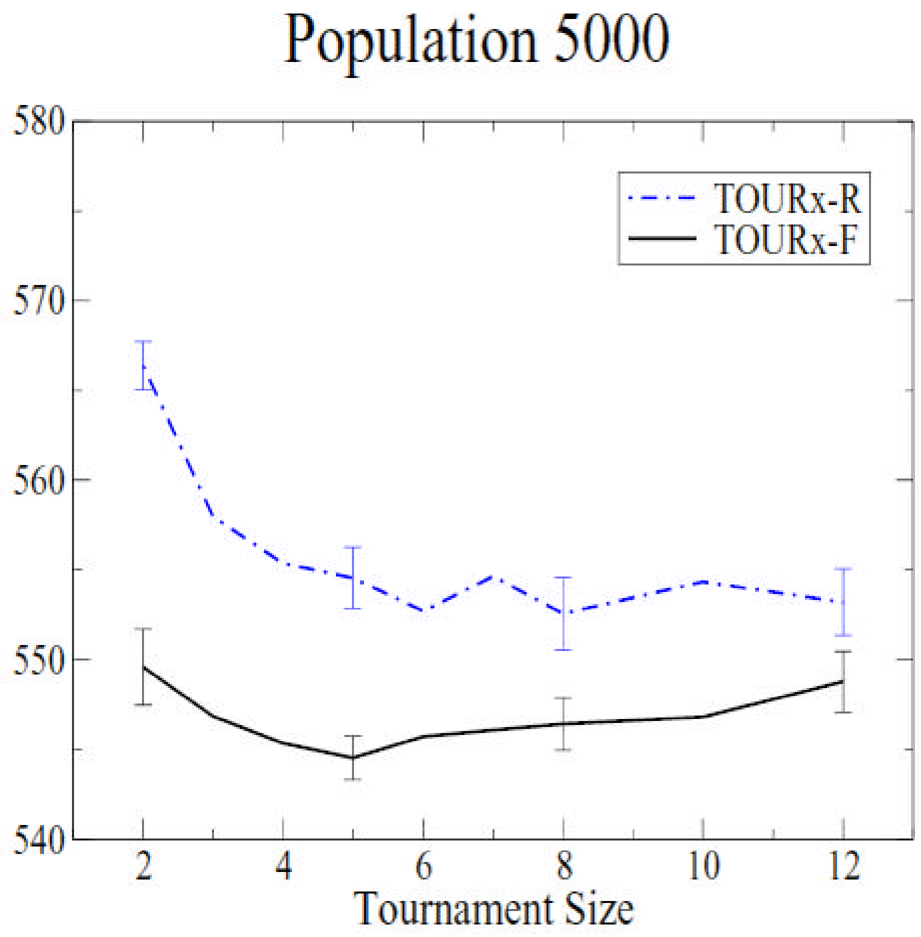
Set Covering Problem

1	0	0	1	
1	1	1	1	
0	0	1	1	
4	2	3	8	= 7 total cost

- NP-complete optimisation problem with real applications
- Low cost = high fitness
- Tested against standard benchmark set covering problems
- Crossover probability = 0.8, Mutation probability = 0.2
- Standard mutation and crossover operators

Set Covering Problem





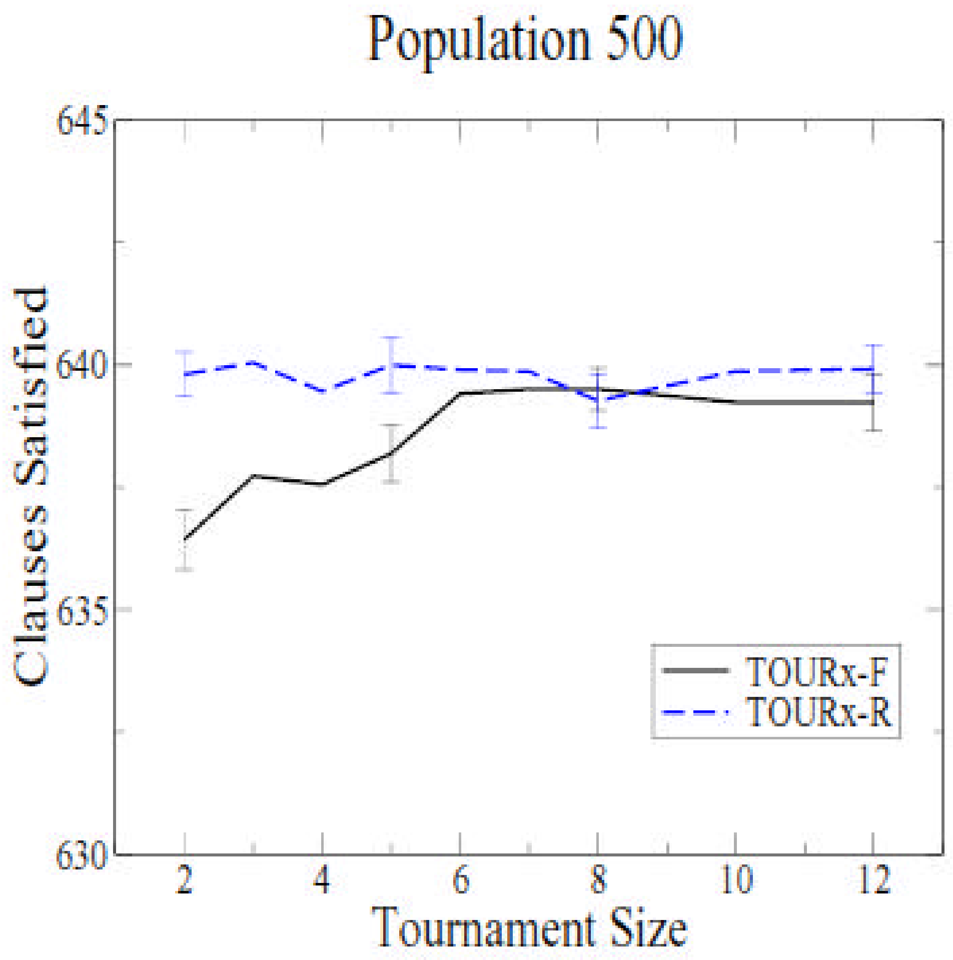
CNF3 SAT

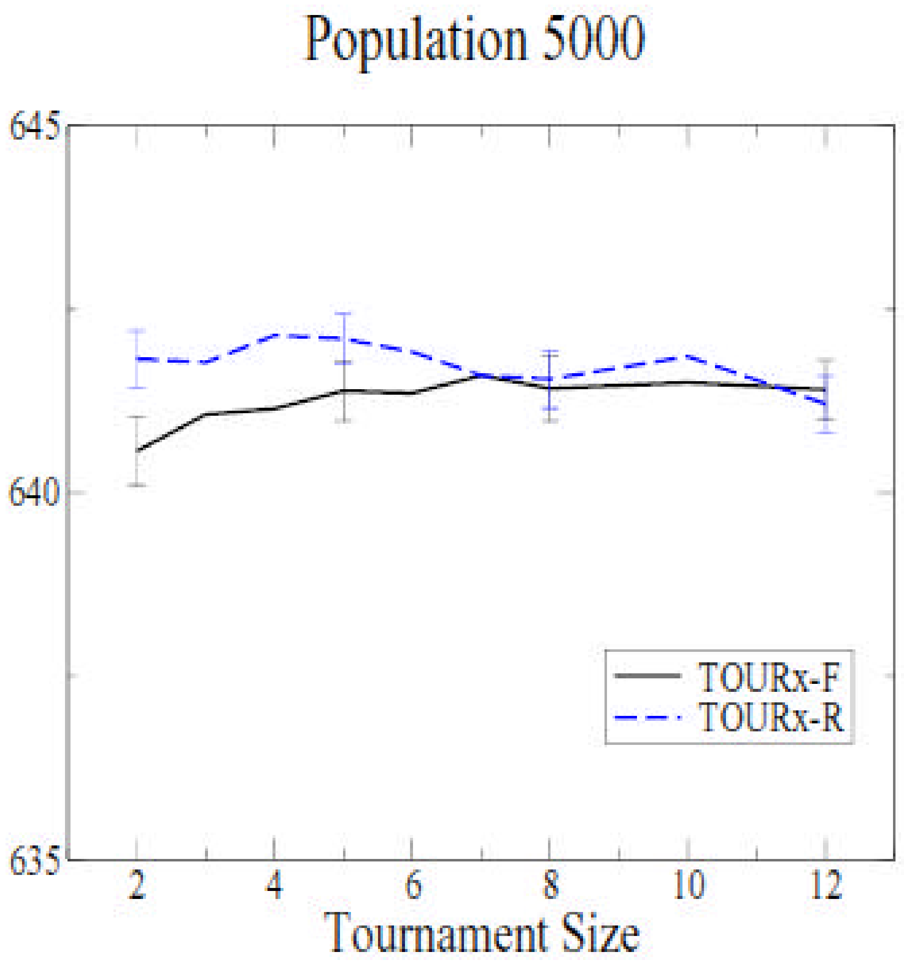
$$(a \vee b \vee \neg c) \wedge (a \vee \neg d \vee e) \wedge (a \vee c \vee e)$$

- An individual is a set of boolean values for the variables
- Fitness is the total number of clauses satisfied

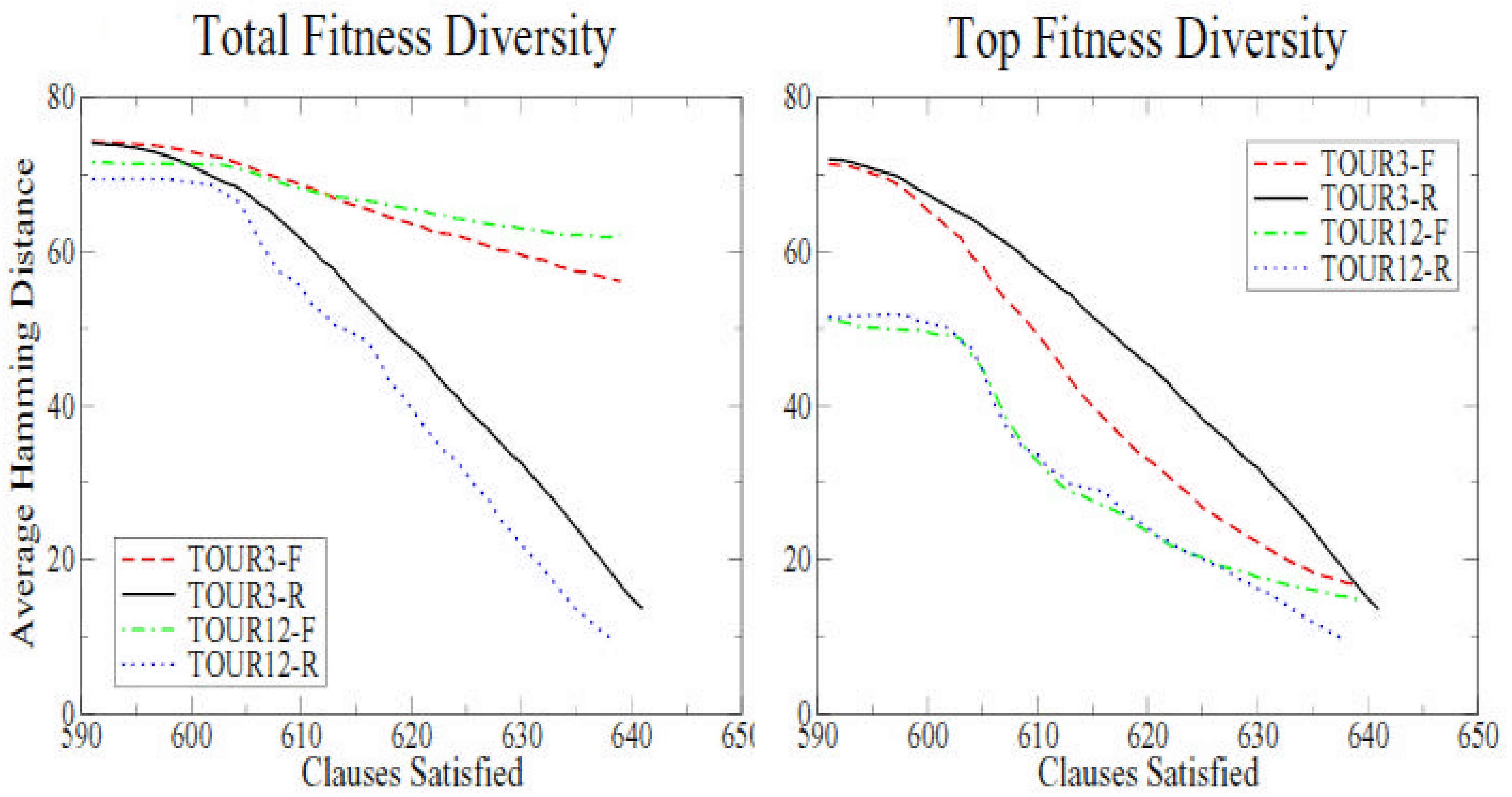
- Used standard benchmark problems for the tests
- 150 variables, 645 clauses in test problems
- Mutation = flip state of one boolean variable
- Crossover = uniform crossover
- Mutation probability & crossover probability = 0.5

CNF3 SAT with 150 clauses





CNF3 SAT Population Diversity



Summary

FUDS is:

- Problem and representation independent
- Easily implemented
- Computationally efficient
- Helps maintain total population diversity, however this isn’t always what matters
- Appears to work best for problems which are sensitive to the selection pressure parameter