Samuel Sicklick

Randomness

The populations in this implementation were generated randomly, that is to have random orderings of the strings given in the original list, by calling *Collections.shuffle* on copies of the list.

Fitness Criterion

The fitness criterion was determined by the returned value of *relativeImprovement()*, “which Returns the ratio of the compressed number of bytes associated with the original list (numerator) to the solution's compressed number of bytes (denominator).”[1]

Threshold

Being that our target improvement was “around 3% - 7%” improvement,[2] the threshold that I used was 1.07 as that would reflect 7% improvement as defined in the fitness criterion.

Selection Type

I found that the *Tournament* selection type provided the best results, which is logical since it was essentially proportional to the fitness of each “*chromosome*”.

Mutation

My mutation function swapped two of the *String*s in the given *Soluntion*’s list to provide a new ordering whose fitness had not yet been examined.

Crossover

My crossover function scanned the given “parent” *Soluntion*’s lists to find the common sequence of *String*s between them to be passed on to the next generation.

[1] As per the skeleton class comments.
[2] As per piazza post @145_f2.