COMP6202 Assignment2
How Do Coevolutionary Dynamics in a Minimal Substrate Change with an Increase in the Number of Populations?

Author: Rishin Amin
Student I.D: 26725479
ra8g14@soton.ac.uk
Personal Tutor: Alex Weddell
January 2018

Abstract

Three sets of results from a paper[1] are successfully reproduced to illustrate coevolutionary failures. The experiments are then repeated with more populations to confirm the hypothesis that increasing the number of populations from 2 to 10 solves the problem of 'focussing' or 'over specialization' within species containing multiple dimensions contributing to objective fitness.

1 The Original Paper

The original paper[1] explores and illustrates the following three cases pertinent to coevolutionary failure: Loss of gradient, over-specialisation and relativism. Three experiments are devised to aid this discussion, all of which make use of a minimal substrate. This makes it easier to attribute issues to coevolutionary techniques and diagnose that they are not a product of the complexity of the task

1.1 Experiment One: Loss of Gradient

Experiment one illustrates the concept of a 'loss of gradient' in a coevolutionary set-up. This is achieved using the coevolution of scalar values; where the scalar value being evolved is the unitation of a bit string i.e. the fitter the individual the more ones in its bit string. A loss of gradient causes a population to drift without improvement[1]. This coevolutionary pathology can be explained by the Red Queen Effect[2]. The subjective performance of individuals in a species does not improve despite improvements in their objective fitness as the competing species improves at the same rate.

The experiment is formalised using equation 1[1]. The value returned is with respect to a random sample of individuals S from the competing population.

$$f(a,S) = \sum_{i=1}^{|S|} score(a,S_i) \qquad \textbf{/eq.1}$$

where $score(a,b)=1$ if $a>b$, 0 otherwise.

This creates a distinction between the objective and subjective fitness of an individual. Individuals are evolved based on their performance against the other population as opposed to an objective fitness metric.

1.2 Distinction Between Objective and Subjective Fitness

An important distinction between the objective and subjective fitness of an individual is clarified in the original paper[1]. The objective fitness is defined as the metric that we are attempting to optimize, whilst the subjective fitness is the perceived performance of the co-evolving individual. This relates to the number of ones in an individuals bit string and the result of the defined subjective fitness functions (see equations) respectively.

1.3 Mutation Biases and Justification for Approach

The nature of the approach (representing individuals using bit strings) results in an inherent mutation bias. A mutation bias of 0.005 is used in the original paper. This value is derived from the string length, 100, and the natural bias towards half 0s and half 1s[5]. The chosen mutation bias results in, on average, 1 bit being assigned a new random value per individual selected for reproduction for each generation.

1.4 Experiment Two: Over Specialization

Experiment Two introduces multiple dimensions to an individuals definition. Instead of having one dimension 100 bits long. 10 dimensions each 10 bits long are used for representation.

A single dimension determines the outcome of a match between competing individuals. For Experiment Two, this is the dimension with the largest difference between the individuals. Equation 2[1] characterizes this approach:

$$f2(a,S) = \sum_{i=1}^{|S|} score2(a,S_i) \qquad \textbf{/eq.2}$$

where

$$score2((a_x,a_y),(b_x,b_y)) = \begin{cases} score(a_x,b_x), \text{ if } (|a_x-b_x|>|a_y-b_y|) \\ score(a_y,b_y), \text{ otherwise.} \end{cases}$$

This set-up is used to illustrate 'over specialization' or 'focussing'. A phenomenon where high performance cannot simultaneously be maintained in all dimensions. Whilst one dimension is focussed on and developed to improve against the other population the other nine drift towards their neutral "50:50" composition.

1.5 Experiment Three: Relativism

The third experiment demonstrates intransitive superiority[2] in a coevolutionary framework. The aim is to replicate a cyclical scenario where, for three members, "*a* beats *b* beats *c* beats *a*"[1]. A slight alteration to equation 2 so the dimension with the *smallest* difference is used to determine the score between two individuals, can introduce circular dominance relations. Equation 3[1] represents this altered approach.

$$f3(a,S) = \sum_{i=1}^{|S|} score3(a,S_i) \qquad \textbf{/eq.3}$$

where

$$score3((a_x,a_y),(b_x,b_y)) = \begin{cases} score(a_x,b_x), \text{ if } (|a_x-b_x|<|a_y-b_y|) \\ score(a_y,b_y), \text{ otherwise.} \end{cases}$$

and, as before, $score(a,b)$= 1 if $a>b$, 0 otherwise.

2 Reimplementing Results

2.1 Experimental Set Up

The experiments described in the previous section have been re-implemented, the parameters used can be found in table 1. The run function found in **APPENDIX** defines the general approach for each experiment. *Length* (of each dimension), *dimensions*, *eq* (equation used to calculate subjective fitness) and *generations* (number of generations to run the simulation) are the main variables between experiments (the sample size was altered on one occasion).

Parameter	Value
Evolutionary Algorithm	Generational
Selection Scheme	Roulette Wheel
Sample Size (S)	Variable
Representation	Bit string
Population Size	25
Dimensions	Variable
Dimension Length	Variable
Subjective Fitness Equation	Variable

Table 1: Parameters for reimplementing results

A Generational approach is taken; to produce a new generation fitness proportionate selection (roulette wheel selection) is used. The number of times the 'wheel' is spun is equal to the size of the population, each time the selected individual is passed through a mutate function, where each bit has a probability equal to the mutation bias of being assigned a new random value.

2.2 Reimplemented Results with Discussion

All of the reimplemented results were run for double the number of generations, to corroborate the expected behavior and in some cases showcase downward trends and separation between the populations multiple times (ruling out anomalous characteristics). This decision was substantiated by the fact that running the simulations was not very computationally expensive.

2.2.1 Experiment One: Loss of Gradient

The parameters shown in **Table 2** are used to replicate the results for the first part of Experiment One from the original paper. **Figure 1** shows the original and reimplemented results. Both sets of results show the same trend: The populations reach the optimal solution after approximately 400 generations, with a variation in the subjective fitness of one populations equal to around one minus the subjective fitness of the other. Due to the performance improvement of opponents at the same rate, the average subjective fitness of the population plateaus once the optimal objective fitness is reached. This is problematic in scenarios where an objective fitness of an individual is not measurable. However, the optimal solution could be inferred when the average subjective fitness has converged.

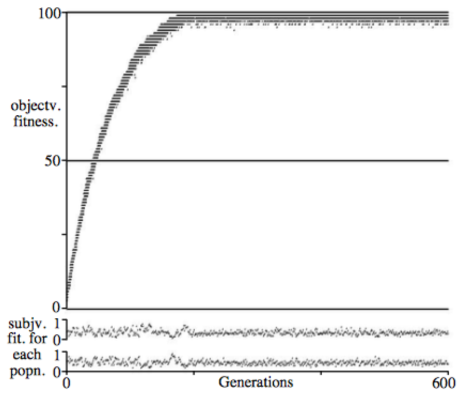
Parameter	Value
Sample Size (S)	15
Population Size	25
Dimensions	1
Dimension Length	100
Subjective Fitness Equation	eq1

Table 2: Parameters for reimplementing experiment one

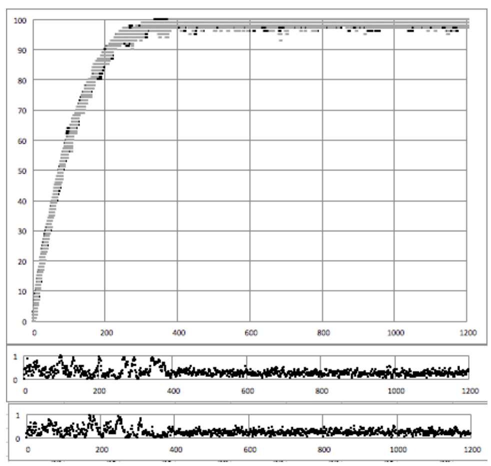
The parameter values used to replicate the second part of Experiment One are the same as those shown in Table 2, save the sample size, *S*, which is now 1. Now each member competes with a randomly selected member of the other population. Comparing the results shown in **Figure 2** it is clear that in both set there are multiple downward and upward trends, with periods of complete separation between the populations. These downward trends coincide with a period of polarisation in the subjective fitnesses of the populations for both graphs. Showing that all the individuals in one population beat all the individuals in the other.

2.2.2 Experiment Two: Over Specialization

Table 3 provides the parameters used to replicate the results for Experiment Two. **Figure 3** compares the reimplemented results with original set for this experiment and shows that the experiment was successfully reimplemented. Both results show the same

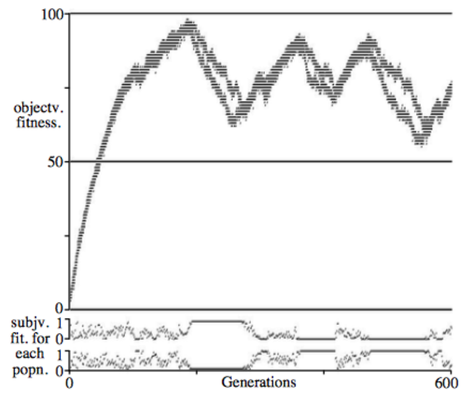


(a) Original results

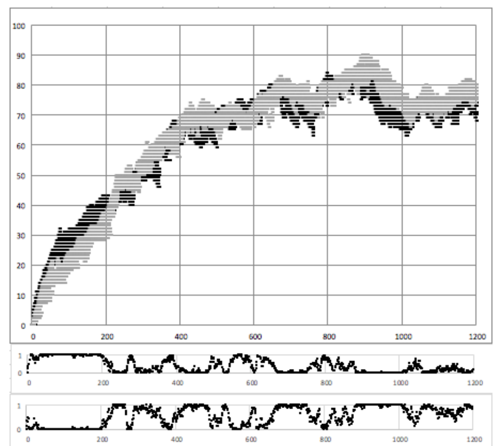


(b) Reimplemented Results

Figure 1: Reimplemented results for experiment one (original on top)



(a) Original results



(b) Reimplemented Results

Figure 2: Reimplemented results for experiment one (original on top)

trends in objective and subjective fitness. The objective performance never reaches 100 for an individual and the subjective performances continue to show an inversely proportional relationship.

In both cases the trends are explained by the fact that top performance across all dimensions cannot be maintained due to "focussing" within a single dimension. Here we can see the principles of the Red Queen Effect applied to dimensions within an individual. Although the performance across one dimension is improving, the performance across that dimension is improving at the same rate in the other population. Once maximal performance in this dimension is achieved, a change in performance in another dimension can cause a population to improve subjectively, leading to a focus on improving that dimension, subsequently the improvements in the original dimension are lost and the competing population is improving at the same rate in the new dimension - so the cycle continues.

2.2.3 Experiment Three: Relativism

Table 4 details the parameters used to replicate the results for Experiment Three. **Figure 4** provides a comparison between the original and reimplemented results for this experiment. Both sets of results show that the objective fitness is being driven down, below the natural bias towards 50. This is more apparent in the reimplemented version. A reduction in the objective fitness in a dimension for an individual can lead

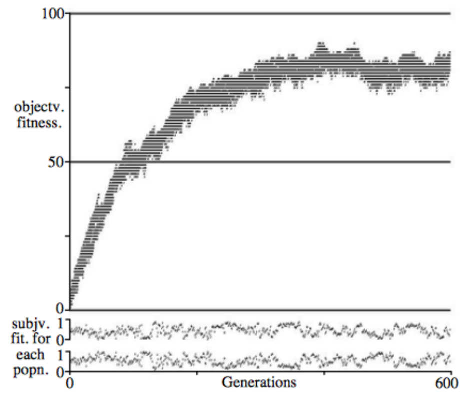
Parameter	Value
Sample Size (S)	15
Population Size	25
Dimensions	10
Dimension Length	10
Subjective Fitness Equation	eq2

Table 3: Parameters for reimplementing experiment two

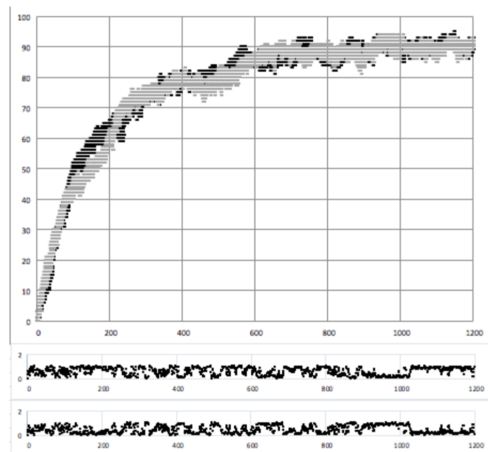
Parameter	Value
Sample Size (S)	15
Population Size	25
Dimensions	10
Dimension Length	10
Subjective Fitness Equation	eq3

Table 4: Parameters for reimplementing experiment two

to improvements in subjective performance, explaining this behavior. Reducing objective performance can increase the difference in that dimension between two competing individuals and change the applicable dimension when calculating the subjective fitness.



(a) Original results



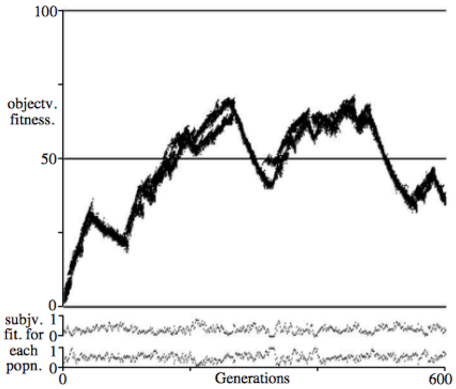
(b) Reimplemented Results

Figure 3: Reimplemented results for experiment two (original on top)

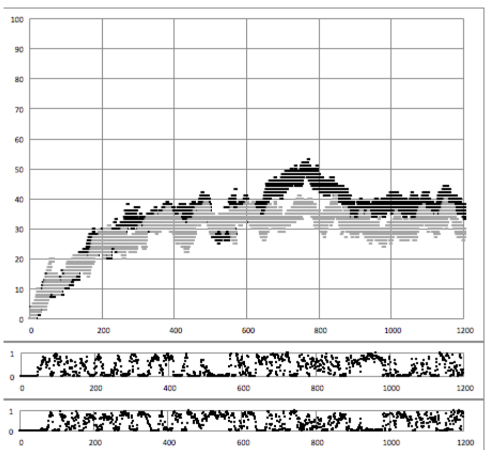
3 Coevolutionary Behavior With More Populations - Does this Fix Focussing?

The experiments from the original paper are extended to cover a coevolutionary set-up where more than 2 populations are used in the aim to fix the problem of over specialization illustrated in Experiment Two. The author hypothesizes that increasing the number of populations will lead to generalization in the case of Experiment Two. Drawing the sample, *S,* randomly from multiple populations (excluding the population that the individual is a part of) could lead to a situation where 'forgetting' objective improvement in a single dimension is detrimental to the individual's subjective performance.

No discernible improvement in the issues described in Experiment's One and Three is expected with an increase in the number of populations. For Experiment One (Figure 2), it is likely that there will still be periods of polarisation where some populations are dominant and others are not, leading to a downward drift towards the neutral performance position owing to a lack of selective pressure. The nature of the game defined for Experiment Three, means that intransitive superiority cannot be solved simply by increasing the number of populations. Objective performance levels can still be driven down to produce a gain in subjective performance.



(a) Original results



(b) Reimplemented Results

Figure 4: Reimplemented results for experiment three (original on top)

3.1 Justification

A similar coevolutionary hypothesis is proposed by Zhal et al.[4], where multiple populations are used for multiple objectives, so the fitness in a population is defined by a single objective. However, this extension does not assign each population a specialist 'dimension' or 'objective', but aims to produce a generalist across all populations; so improvements learnt in one dimension are not forgotten, when improvements are made in another.

3.2 Experimental Set Up

The exact same parameters described in section 2.2 are used to produce the extended results. The only difference in the experimental set-up is that 10 populations are used instead of 2. When calculating the subjective fitness of an individual, the sample, S, consists of members randomly drawn from the remaining 9 populations. This has two consequences: 1) The random sample contains a greater genetic diversity 2) The selective pressure is drawn from a larger pool of individuals, whilst maintaining the independence of reproduction and selection between populations.

3.3 Results

Figure 5 shows the coevolutionary dynamics for 10 competing populations. These results will be compared with **Figure 2**, hence, the parameters used are the same as those given in **Table 2** apart from the sample size, S, which equals 1.

Figure 6b shows the results for Experiment Two with multiple populations. The objective fitness took longer than 1200 generations to converge, hence, in **Figure 6a** the original experiment is repeated for the same number of generations to provide a fair comparison. The parameters given in **Table 3** are used to produce both figures.

The extended results for Experiment Three are illustrated in **Figure 7** for 1 and 10 populations. The same parameters in **Table 4** are used to produce this figure.

3.4 Discussion

Figure 5 shows an interesting dynamic. Unlike the case illustrated in **Figure 2**, the maximal objective fitness is reached by some populations. However, once this occurs, there is a general drift towards the natural bias, as expected. **Figure 6** shows the subjective fitnesses for the 10 populations. As in **Figure 2** some polarisation is observable. This is accompanied by the downward trend of some populations towards the natural bias. A particular matter of intrigue is

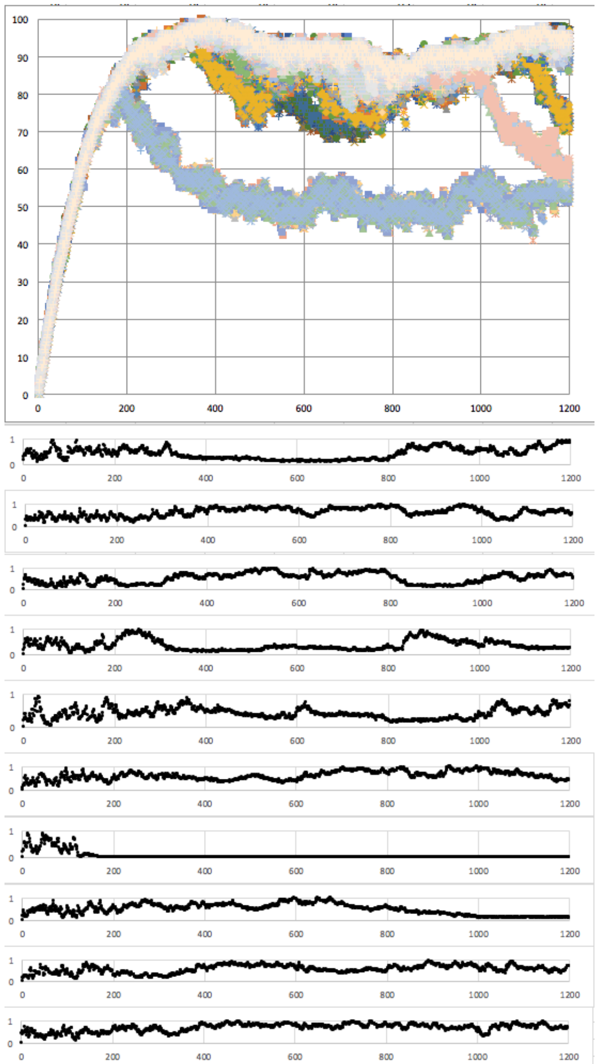
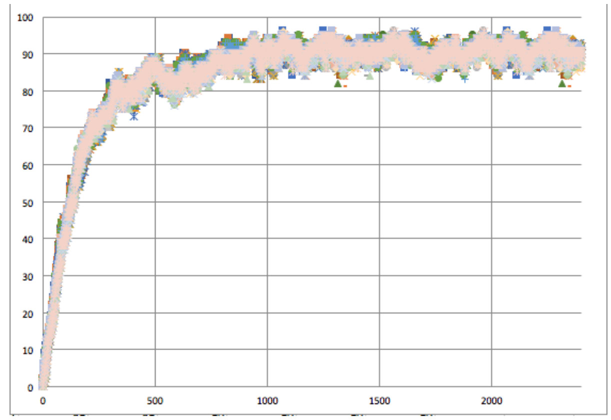


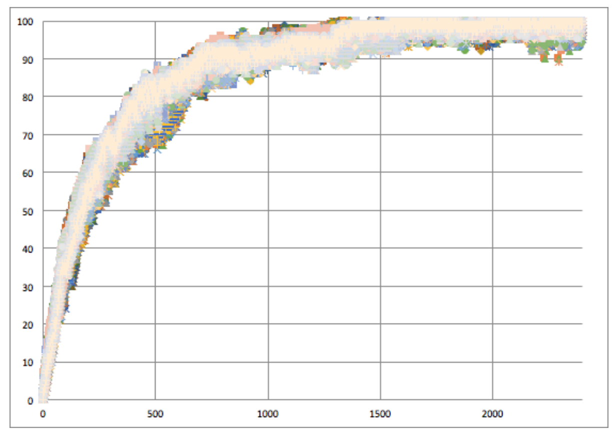
Figure 5: Extended Results for experiment one

the complete domination over population 7. A clear divergence at around 200 generations can be seen in **Figure 5**, corroborated by the polarization of the subjective fitness in **Figure 2.** This reveals that if a population does not improve at fast enough rate objectively, it can be 'left behind' by the other populations which continue to improve and not drift to the same level as they play against other competing populations that are improving objectively. In the case of two populations domination by one population will be resolved as both start to drift towards the natural bias.

Comparing the results in **Figure 6**, it is apparent



(a) 2 populations



(b) 10 populations

Figure 6: Extended results for experiment two (original on top)

that introducing more populations to the coevolutionary game has lead to the optimal fitness across all dimensions being found. The solution took a greater number of generations to converge, but this is expected, as it takes longer to maintain optimal performance across all dimensions.

Figure 7 illustrates that, as predicted, there is not much improvement in the objective fitness of the populations. Intransitive superiority still exists between multiple populations. The populations continue to repeatedly travel the same part of strategy space.

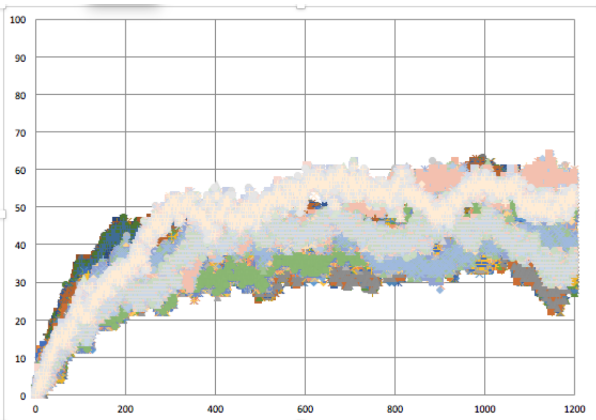


Figure 7: Extended Results for experiment three

4 Conclusion

The results from the original paper were successfully reimplemented and discussed. The parameters used for each experiment are clearly provided as well as the general approach taken. An extension to this paper was then proposed: Does increasing the number of populations in a coevolutionary set-up fix focussing? The experiments were repeated with 10 populations and compared against the results for two populations. As predicted, repeating Experiments One and Three illustrated the same issues of a loss of gradient and over specialization for more than two populations. Repeating Experiment Two, showed that increasing the number of populations does indeed solve the problem of over specialization or focussing in a coevolutionary set-up. Forgetting improvements in one dimension lead to poorer subjective performance as an individual could compete against a population containing specialists in that dimension in any generation.

References

[1] Watson, Richard A. and Pollack, Jordan B., Spector, Lee(ed.) (2001) Coevolutionary Dynamics in a Minimal Substrate Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001), pp. 702-709.

[2] Cliff, D, & Miller, GF, 1995, “Tracking the Red Queen: Measurements of adaptive progress in coevolutionary simulations”, Third European conference on Artificial Life, pp 200-218, Springer-Verlag, LNCS 929.

[3] Ficici, SG. & Pollack, JB., 1998, “Challenges in Coevolutionary Learning: Arms-Race Dynamics, Open- Endedness, and Mediocre Stable States.” Proceedings of the Sixth International Conference on Artificial Life. Adami, et al, eds. Cambridge: MIT, Press.

[4] Zhi-Hui Zhan, Jingjing Li, Jiannong Cao, Jun Zhang, H. Chung and Yu-Hui Shi, "Multiple Populations for Multiple Objectives: A Coevolutionary Technique for Solving Multiobjective Optimization Problems", IEEE Transactions on Cybernetics, vol. 43, no. 2, pp. 445-463, 2013.

[5] Fisher, RA, 1930, The genetical theory of natural selection, Clarendon press, oxford.

Appendix

Code

```python
from __future__ import division
import random, string
from random import randrange, sample
import time
import csv
from copy import deepcopy

# LENGTH = 10
POP_SIZE = 25
SAMPLE_SIZE = 15
# DIMENSIONS = 10
MUTATION_BIAS = 0.005

def create_population(length, size, dimensions):
        """
        Creates a population of size 'size'. Each member consists of a string of length
        'length' bits. All bits are initialised to 0.
        """
        pop = [[[0]*length for i in range(dimensions)] for j in range(size)]
        return pop

def obj_fitness(member):
        """
        Evaluates the objective fitness of a member by counting the number of 1s in the bitstring.
        """
        # consider using coutner if this is too slow
        fitness = 0
        for dim in member:
                fitness = fitness + dim.count(1)
        return fitness

def exp1_score(a,b):
        """
        score(a,b) = 1 if a>b, 0 otherwise
        """
        if a[0].count(1) > b[0].count(1):
                return 1
        else:
                return 0

def exp2_score(a,b):
        """
        use exp1_score(Ax,Bx) where x is the dimension with the largest difference in obj_fitness
        """
        # find dimension with largest differnce
        largest_diff = -1
        index = 0
        for i, dim in enumerate(a):
                diff = abs(dim.count(1) - b[i].count(1))
                if diff > largest_diff:
                        largest_diff = diff
                        index = i
        # use that dimension to determine the score
        if a[index].count(1) > b[index].count(1):
                return 1
        else:
                return 0

def exp3_score(a,b):
        """
        use exp1_score(Ax,Bx) where x is the dimension with the smallest difference in obj_fitness
        """
```

```
    # find dimension with smallest differnce
    smallest_diff = abs(a[0].count(1) - b[0].count(1))
    index = 0
    for i, dim in enumerate(a):
        diff = abs(dim.count(1) - b[i].count(1))
        if diff < smallest_diff:
            smallest_diff = diff
            index = i
    # use that dimension to determine the score
    if a[index].count(1) > b[index].count(1):
        return 1
    else:
        return 0

def exp2_subj_fitness(member, sample, eq):
    """
    Evaluates the subjective fitness of a member uisng the sum of exp1_score function
    against an input sample.
    """
    fitness = 0
    for member2 in sample:
        if eq == 0:
            fitness = fitness + exp1_score(member, member2[0])
        if eq == 1:
            fitness = fitness + exp2_score(member, member2[0])
        if eq == 2:
            fitness = fitness + exp3_score(member, member2[0])
        # fitness = fitness + exp1_score(member, member2[0])
    return fitness


def exp2_subj_fitness_pop(popfit1, popfit2, eq):
    """
    Evaluates the fitnesses of each member in an entire population, using samples from
    another population and the 'exp2_subj_fitness' function.
    Returns the fitnesses as a list.
    """
    fitnesses = []
    for member in popfit1:
        fitnesses.append(exp2_subj_fitness(member[0], random.sample(popfit2, SAMPLE_SIZE), eq))
    # update popfit1 with new subj fitnesses
    # popfit1 = zip(list(zip(*popfit1)), fitnesses)
    return fitnesses


def exp4_subj_fitness_pop(popfit_index, popfit, eq):
    """
    Evaluates the fitnesses of each member in an entire population, using samples from
```

```
        all other populations and the 'exp2_subj_fitness' function.
        Returns the fitnesses as a list.
        """
        fitnesses = []
        other_pops = []
        for i, pop in enumerate(popfit):
                if i == popfit_index:
                        continue
                for member in pop:
                        other_pops.append(member)
        for member in popfit[popfit_index]:
                fitnesses.append(exp2_subj_fitness(member[0],random.sample(other_pops, SAMPLE_SIZE), eq))
        return fitnesses

def mutate_member(member_original, mutation_rate):
        """
        Mutates a member in a popoulation with probability 'mutation_rate' that a bit is
        assigned a new random value (or is flipped).
        """
        member = []
        member = deepcopy((member_original))
        for dim in member:
                for i,bit in enumerate(dim):
                        # test = random.random()
                        # print test
                        if random.random() < mutation_rate:
                                # member[i] = !member[i]
                                dim[i] = 0 if random.random() < 0.5 else 1
        return member

def make_new_gen(pop_fit, mutation_rate):
        new_pop_fit = []
        for n in range(0,len(pop_fit)):
                # select member for mutation
                total = 0
                wheel = []
                # create wheel
                for member in pop_fit:
                        total = total + member[1] + 1
                        wheel.append(total)
                # pick from wheel
```

```
        pick = random.random() * total
        for i, wheel_val in enumerate(wheel):
            if wheel_val >= pick:
                index = i
                break
        new_pop_fit.append((mutate_member(pop_fit[index][0], mutation_rate), 0))
    return new_pop_fit

def run(length, dimensions, eq, generations, filename):
    obj_fitnesses =[[0 for i in range(0,POP_SIZE*2)]]
    # initialise populations and subj fitnesses
    pop1 = create_population(length, POP_SIZE, dimensions)
    pop2 = create_population(length, POP_SIZE, dimensions)
    subj_fitnesses1 = [[0 for i in range(0,POP_SIZE)]]
    subj_fitnesses2 = [[0 for i in range(0,POP_SIZE)]]
    # create a list of tuples containing members with their subjective fitnesses
    pop1_fit = zip(pop1, subj_fitnesses1[0])
    pop2_fit = zip(pop2, subj_fitnesses2[0])
    for i in range(0,generations):
        # make a new generation - use fitness proportionate selection to mutate a new parent
        pop1_fit = make_new_gen(pop1_fit, MUTATION_BIAS)
        pop2_fit = make_new_gen(pop2_fit, MUTATION_BIAS)
        # calculate new objective fitnesses for new populations
        obj_fitnesses_next = []
        for mem in pop1_fit:
            obj_fitnesses_next.append(obj_fitness(mem[0]))
        for mem in pop2_fit:
            obj_fitnesses_next.append(obj_fitness(mem[0]))
        obj_fitnesses.append(obj_fitnesses_next)
        # calc new subjective fitnesses for each member in both populations
        fitnesses1 = exp2_subj_fitness_pop(pop1_fit,pop2_fit,eq)
        pop1_fit = zip(list(zip(*pop1_fit)[0]), fitnesses1)
        fitnesses2 = exp2_subj_fitness_pop(pop2_fit,pop1_fit,eq)
```

```
                pop2_fit = zip(list(zip(*pop2_fit)[0]), fitnesses2)
                subj_fitnesses1.append(fitnesses1)
                subj_fitnesses2.append(fitnesses2)
        with open(filename + ".csv", "a") as fp:
                wr = csv.writer(fp, dialect='excel')
                wr.writerows(obj_fitnesses)
        with open(filename + "_subj_fitnesses" + ".csv", "a") as fp:
                wr = csv.writer(fp, dialect='excel')
                wr.writerows(subj_fitnesses1)
                wr.writerow("\n")
                wr.writerows(subj_fitnesses2)
        return [obj_fitnesses, subj_fitnesses1, subj_fitnesses2, pop1_fit, pop2_fit]

def run2(length, dimensions, eq, pops, generations, filename):
        obj_fitnesses =[[0 for i in range(0,POP_SIZE*pops)]]
        # initialise populations and subj fitnesses
        pop_fits = [0 for i in range(0,pops)]
        subj_fitnesses = []
        for i in range (0,pops):
                subj_fitnesses.append([[0 for j in range(0,POP_SIZE)]])
                # create a list of tuples containing members with their subjective fitnesses
                pop_fits[i] = zip(create_population(length, POP_SIZE, dimensions), subj_fitnesses[0][0])

        for i in range(0,generations):
                obj_fitnesses_next = []
                for k, pop_fit in enumerate(pop_fits):
                        # make a new generation - use fitness proportionate selection to mutate a new parent
                        pop_fits[k] = make_new_gen(pop_fit, MUTATION_BIAS)
                        # calculate new objective fitnesses for new populations
                        for mem in pop_fits[k]:
                                obj_fitnesses_next.append(obj_fitness
```

```
(
mem
[
)
            obj_fitnesses.append(
                obj_fitnesses_next)
            # calc new subjective
                fitnesses for each
                member in both
                populations
            for j in range(0, pops
                ):
                fitnesses =
                    exp4_subj_fitness_pop
                    (j,
                    pop_fits,
                    eq)
                subj_fitnesses
                    [j].append(
                    fitnesses)
                pop_fits[j] =
                    zip(list(
                    zip(*
                    pop_fits[j
                    ])[0]),
                    fitnesses)

        with open(filename + ".csv", "
            a") as fp:
            wr = csv.writer(fp,
                dialect='excel')
            wr.writerows(
                obj_fitnesses)
        with open(filename + "
            _subj_fitnesses" + ".csv",
            "a") as fp:
            wr = csv.writer(fp,
                dialect='excel')
            for fitness in
                subj_fitnesses:
                wr.writerows(
                    fitness)
                                    wr.writerow('
                                        new_pop')
        return [obj_fitnesses,
            subj_fitnesses]
```