Assignment 6: Neuroevolution

In our last assignment, we explored the week's major idea (representation and genetic encoding) in a more realistic application in the Traveling Salesman Problem. This week we'll follow that trend and explore the ideas around mutation rates and measuring diversity in the setting of evoliving artificial neural networks (Neuroevolution). While neuroevolution really shines outside of standard machine learning benchmarks, for the sake of simplicity, we'll use one of the most basic benchmarks for neural networks, classification of handwritten digits in MNIST.

In [1]:

```
# imports
import numpy as np
import copy
import matplotlib.pyplot as plt
plt.style.use('seaborn')

import scikits.bootstrap as bootstrap
import warnings
warnings.filterwarnings('ignore') # Danger, Will Robinson! (not a scalable hack, and may s

import scipy.stats # for finding statistical significance

import time
```

I've taken the liberty to preprocess MNIST for you by deskewing (standard preprocessing step to straighten tilted images) and downscaling the images from `28x28` to `14x14` to try and keep out genome size down (at the cost of losing some resolution/information in the images), split out the labels (turning them into one-hot encodings), and separating the train and test sets. If you aren't familiar with machine learning practices like this, don't worry about it -- just load the datasets below.

Note: This dataset contains 60,000 training examples, and 10,000 testing examples. This is likely far overkill for what we need, so if your machine is struggling with the size of the dataset, feel free to use only a small portion of the training examples/labels provided (doing so didn't effect runtime much on my laptop, but your mileage may vary)

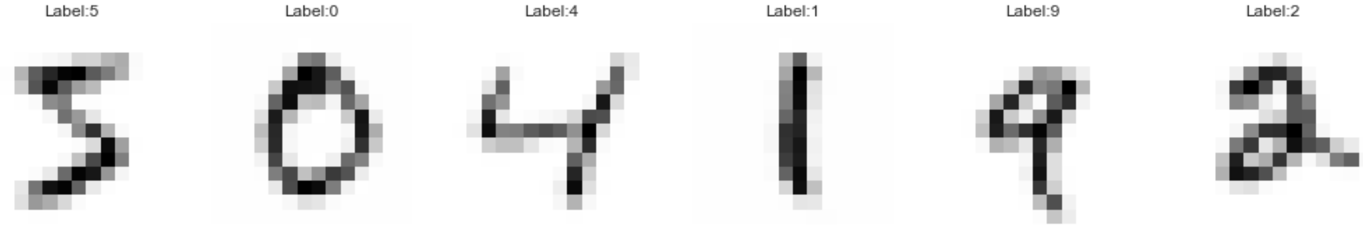
In [2]:

```
train_x = np.loadtxt("train_x.csv", delimiter=',')
test_x = np.loadtxt("test_x.csv", delimiter=',')
train_y = np.loadtxt("train_y.csv", delimiter=',')
test_y = np.loadtxt("test_y.csv", delimiter=',')
```

Let's take a look at the images!

In [3]:

```
# This is what the image looks like
num_images = 6
fig, axs = plt.subplots(1, num_images, figsize=(3*num_images, 3), sharey=True)
for i in range(num_images):
    axs[i].imshow(train_x[i].reshape(14,14)) # we will keep the images flat to easily feed
    axs[i].grid(False)
    axs[i].axis('off')
    axs[i].set_title("Label:"+str(np.argmax(train_y[i]))) # the argmax takes out one-hot e
```



Q1: Implementation

Our individual solutions this week will be (again keeping things overly simplistic) single-layer neural networks. These networks are defined by a single weight matrix with input dimenion of the size of the flattened image (`14*14=196`) and output dimension of the size of the number of possible classes (`10`). Feel free to implement the genome as the weight matrix, or simply as a flattened float vector of size `1960` .

In [4]:

```
class Individual:

    def __init__(self, fitness_function, genome_length):
        self.fitness_function = fitness_function
        self.genome = np.random.uniform(-1,1,genome_length)

    def eval_fitness(self):
        self.fitness = self.fitness_function(self.genome)
```

There are two main ways to measure the performance of a neural network, loss and accuracy. For the sake of intuition, let's use accuracy here, but I'm providing the implementaition of loss just in case you want to play around with it as well (thought returning the negative of the loss, as the smaller magnitudes are better so this allows us to continue going "uphill" if we do ever choose to optimize for loss).

As we haven't covered neural networks, I'm also providing the implementation of a single layer neural network (desite its apparent simplicity compared to mult-layer networks) in the fitness function below.

In [5]:

```
def accuracy(output, y):
    return np.sum(np.isclose(np.argmax(output,axis=1),np.argmax(y,axis=1)))/y.shape[0]

def loss (output, y):
    return -np.sum(np.square(output-y))/y.shape[0]

def neural_network_fitness(weights,x=train_x,y=train_y):
    weight_matrix = weights.reshape((14*14,10))
    output = x.dot(weight_matrix)
    return accuracy(output,y)
```

Q1b: Real-valued mutation

In class, we've only alluded indrectly to mutating vectors of floats as genomes (like neural network weights). Let's play around with the implmentations of these. For simplicity, we'll ignore crossover for now. Rather than flipping a given number of bits, let's try adding a small random value to each gene's value by adding `(np.random.rand(genome_length)*2-1)*mutation_size` to the genome. This takes a uniform distribution, normalizes it to be between -1 and 1, then scales it by some `mutation_size` scaling factor that you can pass into your `evolutionary_algorithm` function.

Q1c: Diversity Tracking

In addition to keeping track of the best genome, and fitness at each generation, let's also record the diversity of the population at each generation. The metric we talked about most in class was measuring genotypic diversity with the average standard deviation of the distribution across the population of the values for each gene.

In [6]:

```
def evolutionary_algorithm(fitness_function=None, total_generations=100, num_parents=10, r
    """ Evolutinary Algorithm (copied from the basic hillclimber in our last assignment)

        parameters:
        fitness_funciton: (callable function) that return the fitness of a genome
                           given the genome as an input parameter (e.g. as defined in Lanc
        total_generations: (int) number of total iterations for stopping condition
        num_parents: (int) the number of parents we downselect to at each generation (mu)
        num_childre: (int) the number of children (note: parents not included in this cour
        genome_length: (int) length of the genome to be evoloved
        num_elements_to_mutate: (int) number of alleles to modify during mutation (0 = no
        mutation_size: (float) scaling parameter of the magnitidue of mutations for floati
        crossover: (bool) whether to perform crossover when generating children
        tournament_size: (int) number of individuals competing in each tournament
        num_tournament_winners: (int) number of individuals selected as future parents frc

        returns:
        fitness_over_time: (numpy array) track record of the top fitness value at each ger
        solutions_over_time: (numpy array) track record of the top genome value at each ge
        diversity_over_time: (numpy array) track record of the population genetic diversit
    """

    # initialize record keeping
    solution = None # best genome so far
    solution_fitness = -99999 # fitness of best genome so far
    solution_generation = 0 # time (generations) when solution was found
    fitness_over_time = np.zeros(total_generations)
    solutions_over_time = np.zeros((total_generations,genome_length),dtype=int)
    diversity_over_time = np.zeros(total_generations)

    # the initialization proceedure
    population = [] # keep population of individuals in a list
    for i in range(num_parents): # only create parents for initialization (the mu in mu+la
        population.append(Individual(fitness_function,genome_length)) # generate new rando

    # get population fitness
    for i in range(len(population)):
        population[i].eval_fitness() # evaluate the fitness of each parent

    for generation_num in range(total_generations): # repeat

#         print(generation_num)

        # the modification procedure
        new_children = [] # keep children separate for now (lambda in mu+lambda)
        while len(new_children) < num_children:

            # inheretance
            [parent1, parent2] = np.random.choice(population, size=2) # pick 2 random pare
            child1 = copy.deepcopy(parent1) # initialize children as perfect copies of the
            child2 = copy.deepcopy(parent2)

#           # crossover
            if crossover:
                for child, this_parent, other_parent in [[child1, parent1, parent2],[chilc
                    child.genome = -1*np.ones(len(child.genome))
                    child.genome[0] = this_parent.genome[0]
                    next_index = np.where(other_parent.genome == this_parent.genome[0])
                    while next_index != 0:
                        child.genome[next_index] = this_parent.genome[next_index]
```

```
                    next_index = np.where(other_parent.genome == child.genome[next_inc
                child.genome[np.where(child.genome == -1)] = other_parent.genome[np.wh
                child.genome = child.genome.astype(int)

        # mutation
        for this_child in [child1,child2]:
            for _ in range(num_elements_to_mutate):
#                   [index_to_swap1, index_to_swap2] = np.random.randint(0,genome_lengt
#                   while index_to_swap1 == index_to_swap2: [index_to_swap1, index_to_sw
#                   orig_gene_1 = this_child.genome[index_to_swap1]
#                   this_child.genome = np.delete(this_child.genome,index_to_swap1)
#                   this_child.genome = np.insert(this_child.genome,index_to_swap2,orig_
                this_child.genome = this_child.genome + (np.random.rand(genome_length)

        new_children.extend((child1,child2)) # add children to the new_children list

    # the assessement procedure
    for i in range(len(new_children)):
        new_children[i].eval_fitness() # assign fitness to each child

    # selection procedure
    population += new_children # combine parents with new children (the + in mu+lambda
    population = sorted(population, key=lambda individual: individual.fitness, reverse

    # tournament selection
    new_population = []
    new_population.append(population[0])
    while len(new_population) < num_parents:
        tournament = np.random.choice(population, size = tournament_size)
        tournament = sorted(tournament, key=lambda individual: individual.fitness, rev
        new_population.extend(tournament[:num_tournament_winners])
    population = new_population

    # record keeping

    if population[0].fitness > solution_fitness: # if the new parent is the best founc
        solution = population[0].genome                 # update best solution records
        solution_fitness = population[0].fitness
        solution_generation = generation_num
    fitness_over_time[generation_num] = solution_fitness # record the fitness of the c
    solutions_over_time[generation_num,:] = solution

    all_gene_std = []
    for x in range(genome_length):
        this_gene_values=[]
        for y in range(len(population)):
            this_gene_values.append(population[y].genome[x])
            this_gene_std = np.std(this_gene_values)
        all_gene_std.append(np.std(this_gene_values))
    diversity_over_time[generation_num] = np.mean(all_gene_std)

return fitness_over_time, solutions_over_time, diversity_over_time
```

Q2: Experimentation

Due to the high dimensionality of this problem, the runs are a bit slower than before, so let's keep the scale small on this with just `50` generations and `5` repitions. Hopefully this keeps things managable from a runtime persepctive (runs in a little over 30 seconds for each repition, or a little under 3 minutes for all 5, on my machine). Let's use a mutation size of `1.0`, the same `50` parents and `50` children settings from last week, and a tournament size of `20`, choosing `10` winners.

Hint: If this still takes to long to run on your machine (especially while debugging/exploring code), feel free to run smaller test runs first by reducing the number of generations for the runs, plotting without bootstrapping, etc.

In [7]:

```
experiment_results = {}
solutions_results = {}
diversity_results = {}
```

In [8]:

```
num_runs = 5
total_generations = 50
genome_length = 14*14*10
num_elements_to_mutate = genome_length
mutation_size = 1.0
num_parents = 50
num_children = 50
tournament_size = 20
num_tournament_winners = 10
fitness_function = neural_network_fitness
crossover = False

for run_name in ["10_win", "5_win", "1_win"]:
    if "5" in run_name:
        num_tournament_winners = 5
    if "1" in run_name:
        num_tournament_winners = 1
    experiment_results[run_name] = np.zeros((num_runs, total_generations))
    solutions_results[run_name] = np.zeros((num_runs, total_generations, genome_length))
    diversity_results[run_name] = np.zeros((num_runs, total_generations))
    for run_num in range(num_runs):
        start_time = time.time()
        fitness_over_time, solutions_over_time, diversity_over_time = evolutionary_algorit
        experiment_results[run_name][run_num] = fitness_over_time
        solutions_results[run_name][run_num] = solutions_over_time
        diversity_results[run_name][run_num] = diversity_over_time
        print(run_name, run_num, time.time()-start_time,fitness_over_time[-1])
```

```
10_win 0 182.6263906955719 0.6169666666666667
10_win 1 181.6860806941986 0.5366166666666666
10_win 2 180.66820335388184 0.54855
10_win 3 181.08656358718872 0.6141833333333333
10_win 4 180.78980779647827 0.5763833333333334
5_win 0 182.05239629745483 0.5496666666666666
5_win 1 182.55733180046082 0.5667333333333333
5_win 2 182.56834053993225 0.5627833333333333
5_win 3 185.79662322998047 0.5459333333333334
5_win 4 192.97180700302124 0.4650166666666667
1_win 0 192.50090098381042 0.2594833333333334
1_win 1 191.59712195396423 0.6098333333333333
1_win 2 194.44707822799683 0.5144833333333333
1_win 3 193.61335945129395 0.5848666666666666
1_win 4 193.58283352851868 0.5764166666666667
```

Q2b: Visualization

Like last time, please plot the bootstrapped fitness values over time.

In [9]:

```
def plot_mean_and_bootstrapped_ci_over_time(input_data = None, name = "change me", x_label
    """

    parameters:
    input_data: (numpy array of shape (max_k, num_repitions)) solution metric to plot
```

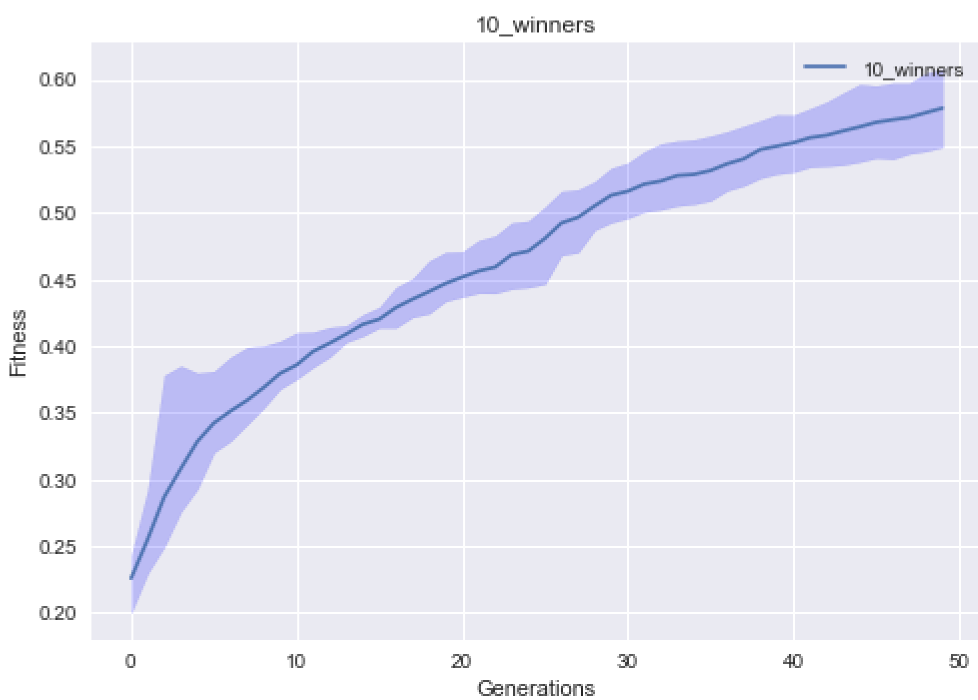
```
    name: (string) name for legend
    x_label: (string) x axis label
    y_label: (string) y axis label

    returns:
    None
    """

    print(input_data.shape)
    generations = input_data.shape[0]

    CIs = []
    mean_values = []
    for i in range(generations):
        mean_values.append(np.mean(input_data[i]))
        CIs.append(bootstrap.ci(input_data[i], statfunction=np.mean))
    mean_values=np.array(mean_values)

    print(CIs)
    high = []
    low = []
    for i in range(len(CIs)):
        low.append(CIs[i][0])
        high.append(CIs[i][1])

    low = np.array(low)
    high = np.array(high)
    fig, ax = plt.subplots()
    y = range(0, generations)
    ax.plot(y, mean_values, label=name)
    ax.fill_between(y, high, low, color='b', alpha=.2)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.legend()
    if (name) and len(name)>0:
        ax.set_title(name)
```

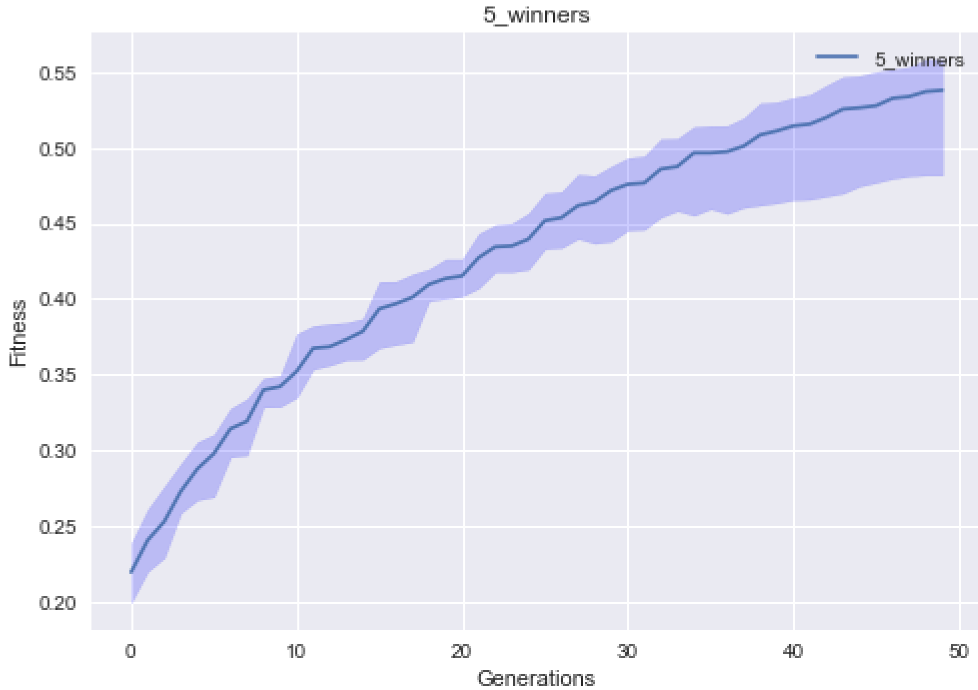
In [10]:

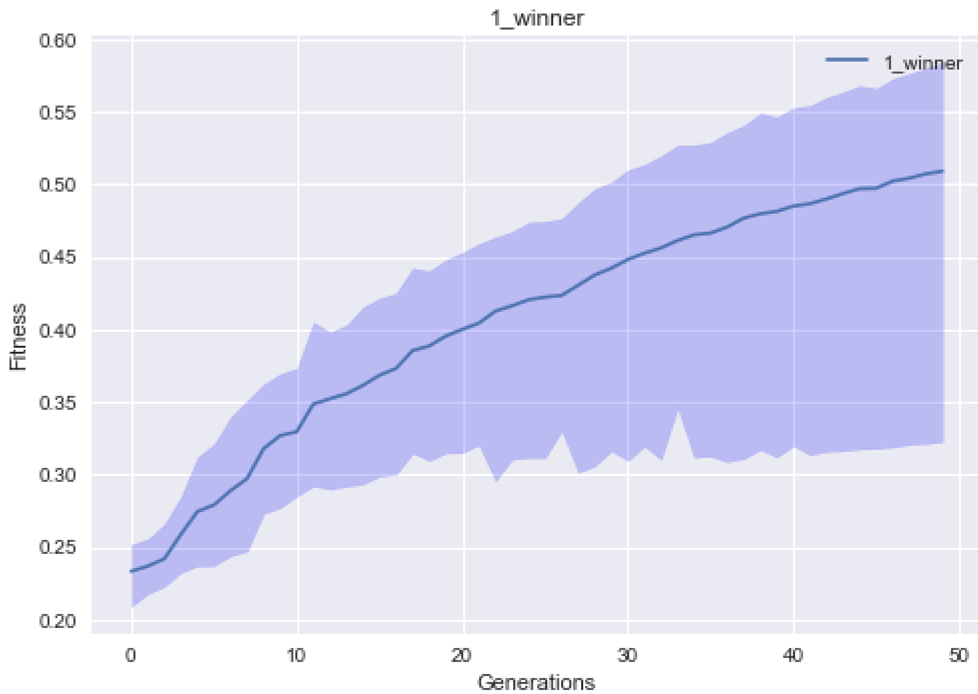
```
# plot fitness over time
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
```

```
(50, 5)
[array([0.20027667, 0.24294667]), array([0.22986333, 0.29225667]), array([0.24979333, 0.37
81    ]), array([0.27591667, 0.38506333]), array([0.29302667, 0.37974667]), array([0.32041
  , 0.38098667]), array([0.32882667, 0.39164667]), array([0.34113333, 0.39896   ]), array
([0.35363   , 0.40007333]), array([0.36789   , 0.40362667]), array([0.37536667, 0.4100266
7]), array([0.38424, 0.41049]), array([0.39196   , 0.41408667]), array([0.40303667, 0.4154
0333]), array([0.40757   , 0.42359333]), array([0.41388333, 0.42923   ]), array([0.4139
, 0.44405667]), array([0.42180667, 0.45057333]), array([0.42447667, 0.46385333]), array
([0.43379667, 0.47045333]), array([0.43704333, 0.47071   ]), array([0.43998   , 0.4793233
3]), array([0.43983   , 0.48275333]), array([0.44307   , 0.49252333]), array([0.44417   ,
0.49390667]), array([0.44668333, 0.50441333]), array([0.46832667, 0.51620333]), array([0.4
7036333, 0.51749333]), array([0.48716333, 0.52347   ]), array([0.49272333, 0.53358667]), a
rray([0.49607667, 0.53747667]), array([0.50092333, 0.54596333]), array([0.50253   , 0.5518
4333]), array([0.50539   , 0.55402667]), array([0.50664   , 0.55489667]), array([0.5092066
7, 0.55771333]), array([0.51676667, 0.56096333]), array([0.52040333, 0.56511   ]), array
([0.52612, 0.5692 ]), array([0.52923333, 0.57363667]), array([0.53069667, 0.57338   ]), ar
ray([0.53448, 0.57822]), array([0.53515333, 0.58320667]), array([0.53610333, 0.58996   ]),
array([0.53829667, 0.59642   ]), array([0.54125667, 0.59533667]), array([0.54045, 0.5974
]), array([0.54448667, 0.5974    ]), array([0.54636, 0.60486]), array([0.54934333, 0.60773
667])]
(50, 5)
[array([0.19933667, 0.2386    ]), array([0.21987667, 0.26096333]), array([0.22895667, 0.27
```

```
607667]), array([0.25858333, 0.29109   ]), array([0.2671    , 0.30534667]), array([0.26918
   , 0.31045667]), array([0.29552    , 0.32759333]), array([0.29663667, 0.33390667]), array
([0.32863333, 0.34756667]), array([0.32874333, 0.34908667]), array([0.33479667, 0.3768466
7]), array([0.35356667, 0.38218333]), array([0.35609333, 0.38349333]), array([0.3596    ,
0.38432333]), array([0.35991    , 0.38692333]), array([0.36753667, 0.41145   ]), array([0.3
6970667, 0.41161333]), array([0.37155333, 0.41635333]), array([0.39874, 0.41974]), array
([0.40023333, 0.42630333]), array([0.40186667, 0.4263    ]), array([0.40712667, 0.4432133
3]), array([0.41781    , 0.44887667]), array([0.41781    , 0.45000333]), array([0.4195 , 0.4
5662]), array([0.43314667, 0.46991333]), array([0.43385667, 0.47094   ]), array([0.43988,
0.4824 ]), array([0.43676667, 0.48149333]), array([0.43797333, 0.48774667]), array([0.4453
8333, 0.49326   ]), array([0.44595333, 0.49452667]), array([0.45409667, 0.50587   ]), arra
y([0.45822667, 0.50616667]), array([0.45527, 0.51391]), array([0.45954    , 0.51453333]), a
rray([0.45647333, 0.51462   ]), array([0.46044333, 0.51976333]), array([0.46199667, 0.5295
8   ]), array([0.46345    , 0.53027667]), array([0.46545333, 0.53318667]), array([0.4658733
3, 0.53522667]), array([0.46789333, 0.54138   ]), array([0.46985667, 0.54691667]), array
([0.47456667, 0.54761667]), array([0.47699667, 0.55019667]), array([0.47962667, 0.5518533
3]), array([0.48121333, 0.55346   ]), array([0.48194667, 0.55850667]), array([0.48194667,
0.55841333])]
(50, 5)
[array([0.20906, 0.25171]), array([0.21793333, 0.25559333]), array([0.22285667, 0.2657566
7]), array([0.23260333, 0.28463   ]), array([0.23691    , 0.31194333]), array([0.23715333,
0.32103667]), array([0.24394667, 0.33966667]), array([0.24725    , 0.35103333]), array([0.2
7328667, 0.36228667]), array([0.27707667, 0.36939667]), array([0.28521, 0.3731 ]), array
([0.29209667, 0.40487667]), array([0.28996    , 0.39814667]), array([0.29181667, 0.4029166
7]), array([0.29344333, 0.41535   ]), array([0.2989    , 0.42142667]), array([0.30027333,
0.42490667]), array([0.31474, 0.4423 ]), array([0.30959667, 0.44033333]), array([0.31487
, 0.44792333]), array([0.31511, 0.45309]), array([0.32044333, 0.45899   ]), array([0.29560
333, 0.46373333]), array([0.31056333, 0.46737333]), array([0.31166, 0.47381]), array([0.31
166   , 0.47448667]), array([0.33027    , 0.47626333]), array([0.30145333, 0.48731667]), ar
ray([0.30606667, 0.49698   ]), array([0.31638333, 0.50142667]), array([0.30969333, 0.50994
667]), array([0.31954333, 0.51365667]), array([0.31047333, 0.51967333]), array([0.34601
, 0.52692667]), array([0.31186333, 0.52699333]), array([0.31257333, 0.52890333]), array
([0.30886333, 0.53574333]), array([0.31099333, 0.54068667]), array([0.31714667, 0.5489233
3]), array([0.31210333, 0.54653667]), array([0.31991    , 0.55272333]), array([0.31365667,
0.5543    ]), array([0.31590667, 0.55991   ]), array([0.31631667, 0.56362667]), array([0.3
1763   , 0.56798333]), array([0.31787333, 0.56611333]), array([0.31897667, 0.57278333]), a
rray([0.32063667, 0.57639667]), array([0.32133667, 0.57995667]), array([0.32287, 0.5840
8])]
```







Q3: Visualizing Diversity

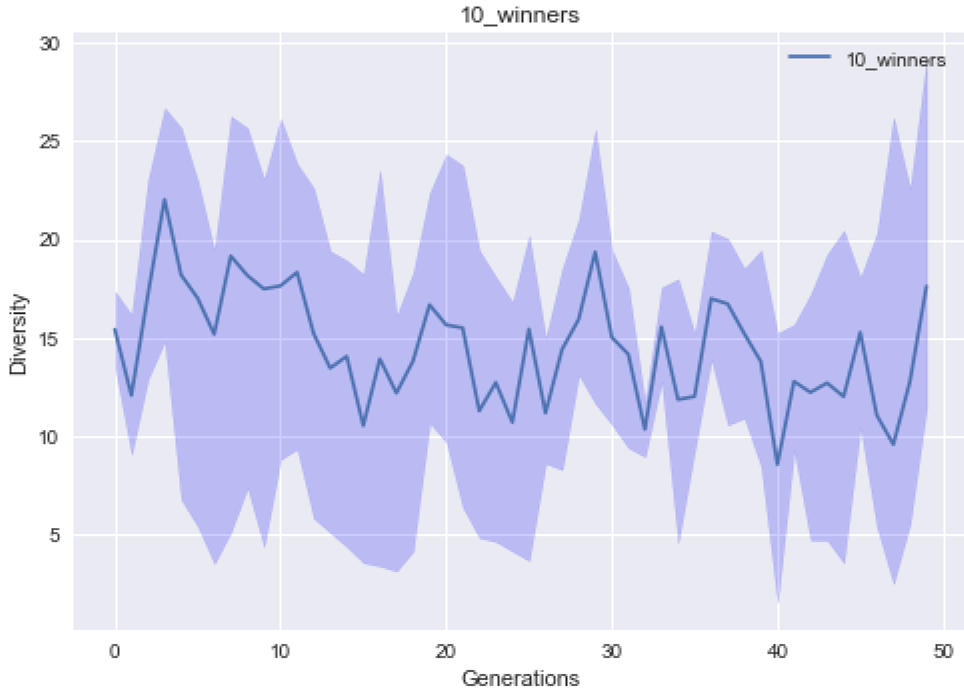
Please also plot the diveristy of our population over evolutionary time.

In [11]:

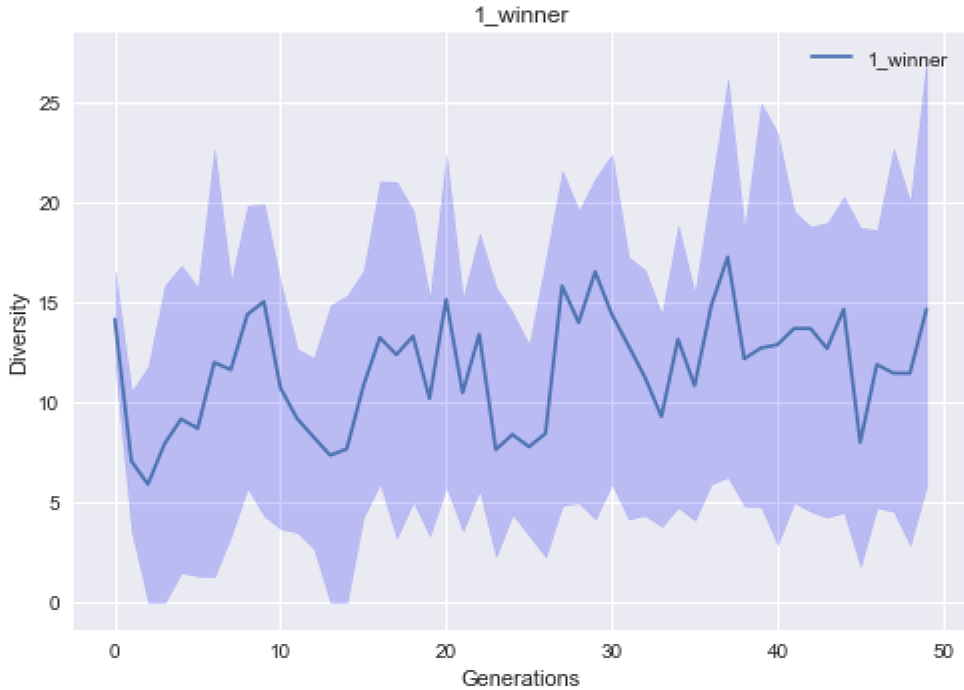
```
# plot diversity over time
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(diversity_results
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(diversity_results
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(diversity_results
```

```
(50, 5)
[array([13.47204487, 17.35889391]), array([ 9.08037103, 16.17404374]), array([12.90511683,
23.06047174]), array([14.82133693, 26.69720327]), array([ 6.76583402, 25.68171914]), array
([ 5.39552281, 22.98133549]), array([ 3.49866322, 19.45109669]), array([ 5.1146331 , 26.27
014827]), array([ 7.36738927, 25.67627717]), array([ 4.36870063, 23.05589225]), array([ 8.
80113232, 26.14677078]), array([ 9.33135815, 23.8506057 ]), array([ 5.80809961, 22.6075201
8]), array([ 5.08024855, 19.40719551]), array([ 4.36034001, 18.93418821]), array([ 3.54775
96 , 18.22988831]), array([ 3.37700934, 23.50705138]), array([ 3.1213812 , 16.11811005]),
array([ 4.13937887, 18.31769027]), array([10.67054392, 22.3669046 ]), array([ 9.68110406,
```

```
24.34031167]), array([ 6.36153599, 23.7684449 ]), array([ 4.81996208, 19.43438687]), array
([ 4.62427571, 18.09700525]), array([ 4.12909217, 16.81205007]), array([ 3.65780294, 20.19
665094]), array([ 8.60454409, 14.97728511]), array([ 8.28238375, 18.52352234]), array([13.
10893944, 21.02873991]), array([11.65601451, 25.6012048 ]), array([10.5787761 , 19.4594183
4]), array([ 9.39306437, 17.54319116]), array([ 8.94406945, 11.67443941]), array([12.78272
485, 17.57160072]), array([ 4.5589431, 17.9886921]), array([ 9.26836111, 15.20373449]), ar
ray([13.89955204, 20.41139644]), array([10.55923268, 20.04132666]), array([10.9075378 , 1
8.52326821]), array([ 8.46748866, 19.45357416]), array([ 1.52093229, 15.23094962]), array
([ 9.26730042, 15.66983111]), array([ 4.69395547, 17.23046883]), array([ 4.68540529, 19.25
824831]), array([ 3.54493365, 20.45597381]), array([10.45020623, 18.0636493 ]), array([ 5.
36081179, 20.31436263]), array([ 2.48143386, 26.2066129 ]), array([ 5.53839508, 22.6007892
7]), array([11.43410442, 29.16689592])]
(50, 5)
[array([13.81475315, 16.99010327]), array([21.47535726, 24.28155747]), array([24.55653708,
27.4534502 ]), array([25.01025037, 28.51323373]), array([25.00167791, 29.48950048]), array
([22.63281002, 28.92520959]), array([21.39888369, 32.15167793]), array([24.40933832, 33.79
195065]), array([28.69660142, 35.33314028]), array([29.97992325, 36.15271223]), array([25.
25843197, 38.62987852]), array([20.44637322, 37.61477232]), array([16.53428275, 36.8217139
6]), array([22.11509507, 33.66594351]), array([ 8.39963599, 32.12297137]), array([ 6.53206
001, 30.313762  ]), array([10.916983  , 26.23664533]), array([11.30887002, 25.63154566]),
array([10.93169401, 19.41364405]), array([15.25695275, 17.50528218]), array([15.14964792,
21.53085311]), array([16.69620474, 23.95377065]), array([20.45224186, 26.568512  ]), array
([21.97626254, 25.24994442]), array([17.73117793, 25.55588393]), array([19.5263595, 25.100
6735]), array([18.69637927, 29.44718762]), array([19.11063577, 30.12685226]), array([17.85
82689 , 27.60287421]), array([ 9.78621192, 25.971754  ]), array([10.66125836, 26.0945522
5]), array([16.5537571 , 25.79910724]), array([18.45734971, 25.5061665 ]), array([20.53722
5  , 29.70235094]), array([18.6206829 , 32.15694685]), array([17.68778462, 33.31148392]),
array([15.70395239, 26.26587164]), array([17.43543341, 24.64277104]), array([17.49503266,
28.36318757]), array([20.04074057, 29.54553281]), array([19.09411548, 23.75227253]), array
([18.22760863, 23.22359908]), array([16.0691427 , 23.41410183]), array([11.82282495, 23.14
036116]), array([16.29052837, 25.96380197]), array([ 6.77699999, 22.86323752]), array([ 6.
77112891, 22.17750228]), array([ 9.76791291, 25.32258183]), array([15.66164061, 26.6026013
7]), array([18.6226724 , 28.83385958])]
(50, 5)
[array([11.78811629, 16.60579837]), array([ 3.57376181, 10.57637677]), array([4.28795321e-
17, 1.18359529e+01]), array([4.29071461e-17, 1.58750709e+01]), array([ 1.49693659, 16.8621
9236]), array([ 1.32689247, 15.79145708]), array([ 1.29739163, 22.71874875]), array([ 3.32
939465, 16.09841377]), array([ 5.73332847, 19.87384716]), array([ 4.34491398, 19.9438800
6]), array([ 3.69203016, 15.99797864]), array([ 3.50731832, 12.6991027 ]), array([ 2.70711
683, 12.25382876]), array([2.06724094e-15, 1.48830054e+01]), array([2.37234835e-15, 1.5359
7945e+01]), array([ 4.28966132, 16.61368725]), array([ 5.95764271, 21.09106337]), array([
3.21688064, 21.05887111]), array([ 5.05278084, 19.62651554]), array([ 3.33709643, 15.26248
702]), array([ 5.79718369, 22.41306021]), array([ 3.57940776, 15.25606238]), array([ 5.625
70971, 18.46710692]), array([ 2.30085284, 15.80182526]), array([ 4.41572869, 14.5189485
7]), array([ 3.34245652, 12.90241815]), array([ 2.28715414, 17.17749164]), array([ 4.85446
146, 21.62088282]), array([ 4.98713403, 19.62559201]), array([ 4.18345568, 21.24430281]),
array([ 5.93777649, 22.43211612]), array([ 4.17631465, 17.30522657]), array([ 4.36478184,
16.65322496]), array([ 3.81176573, 14.41655762]), array([ 4.77350268, 18.87425902]), array
([ 4.11390532, 15.46853701]), array([ 5.93973573, 20.82138195]), array([ 6.27850396, 26.21
24363 ]), array([ 4.83378419, 18.6714951 ]), array([ 4.80908498, 24.98026514]), array([ 2.
89288423, 23.4909967 ]), array([ 5.01450616, 19.57977324]), array([ 4.55830604, 18.8207869
2]), array([ 4.26971266, 19.00471425]), array([ 4.51345105, 20.31524102]), array([ 1.79489
167, 18.76196693]), array([ 4.74084271, 18.64645131]), array([ 4.56280965, 22.72420619]),
array([ 2.86130993, 20.07485041]), array([ 5.85282657, 27.18194175])]
```


10_winners
10_winners
Diversity
Generations
0
10
20
30
40
50
5
10
15
20
25
30


5_winners
5_winners
Diversity
Generations
0
10
20
30
40
50
5
10
15
20
25
30
35
40



Q3b: Analysis

What do you notice about the diveristy over time? Is this what you expected to tradeoff exploration and exploitation -- and how it related to fitness?

It looks like diversity over time in basically all 3 cases jumps up and down across generations and stays within a fairly similar range. The diversity when selecting 5 winners seems to have some higher peaks that 10 or 1. I'm not sure if this is just due to some random chance since this only partially makes sense to me. Selecting more individuals in tournaments would decrease selection pressure, which should mean diversity is higher, but the runs with 10 winners have lower peaks than that of the runs with 5 winners. It looks like fitness varies much more greatly across runs when selection pressure is higher (the 1 winner case) and varies much less when selection pressure is lower (10 winners or 5 winners cases). It seems a bit odd that diversity is relatively similar across the 10 winner, 5 winner, and 1 winner cases since they have varying degrees of fitness as well as varying degrees of deviation across runs within each case.

Q4: Generalization to Test Datasets

Whenever doing classification, it's good to make sure that your algorithm isn't overfitting to the training data. Based on your intuition about diversity and overfitting, what do you expect this relationship to look like?

I think this would suggest that diversity is low but fitness is very high. It has basically memorized the training dataset and is only able to do well on what it has memorized. It won't have much diversity since it should only have individuals in the population with varying degrees of memorization of the training dataset.

Q5: Evaluating Test Accuracy

Since we already have test data loaded in above, let's evaluate your already trained algorithms (using your saved best-solution-so-far genomes at each generation) to see how test fitness tracks with the training fitness.

Please implement a script which calcualtes the test accuracy of the solutions over time below.

Hin: Look for where the training set is used during fitness evaluation during training for ideas of what functions/syntax to use

In [12]:

```
test_accuracy_results = {}

def calc_test_accuracy_over_time(name = None):

    test_accuracy_results = np.zeros((num_runs, total_generations))

    # loop through each run
    for run_num in range(num_runs):
        # loop through each solution over time
        for generation in range(total_generations):
            # calculate the fitness on the test set for this solution and add this to tes
            test_accuracy_results[run_num][generation] = neural_network_fitness(weights=sc



    return test_accuracy_results
test_accuracy_results['10_win'] = calc_test_accuracy_over_time(name="10_win")
test_accuracy_results['5_win'] = calc_test_accuracy_over_time(name="5_win")
test_accuracy_results['1_win'] = calc_test_accuracy_over_time(name="1_win")
# print(test_accuracy_results['mutation_only'])
```

Q5b: Running and Visualization

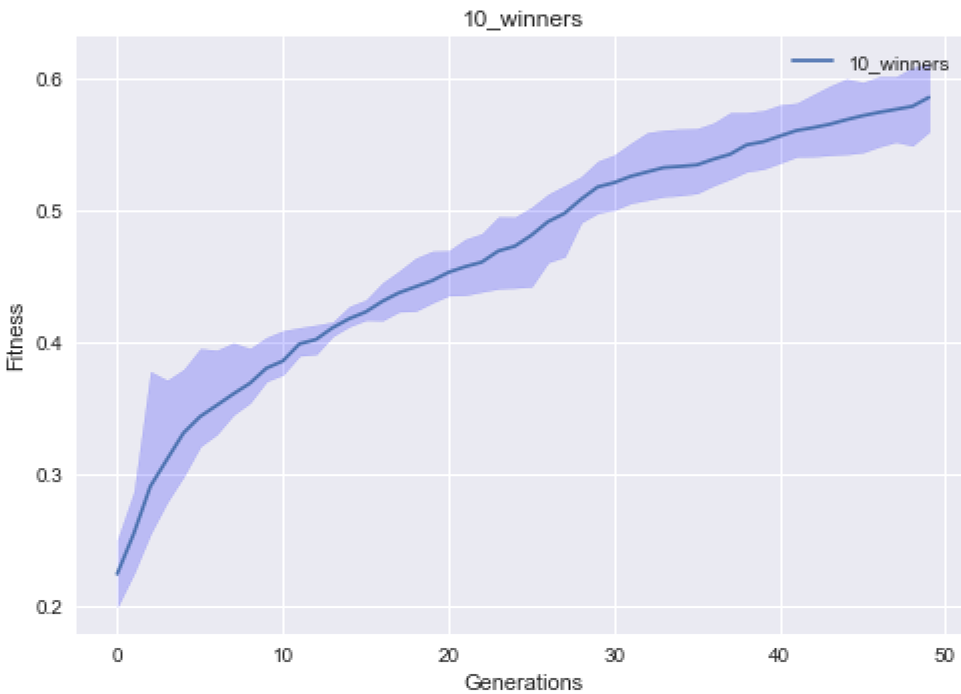
Run and plot the test accuracy over time of the runs you performed above (to reduce clutter, feel free to just do this for the tournaments of size `20`).

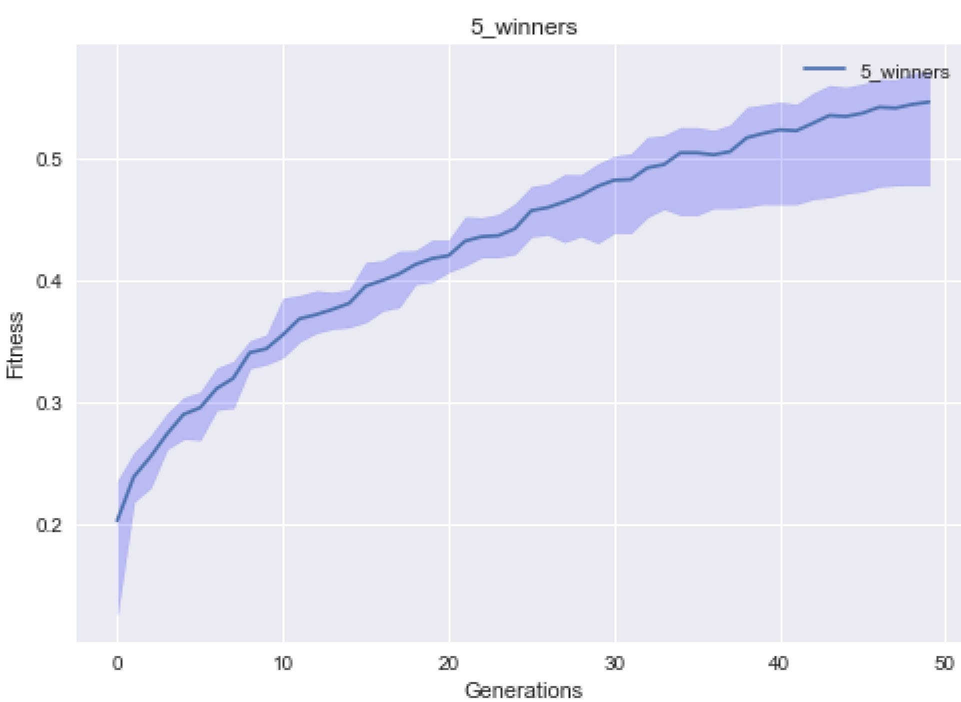
In [13]:

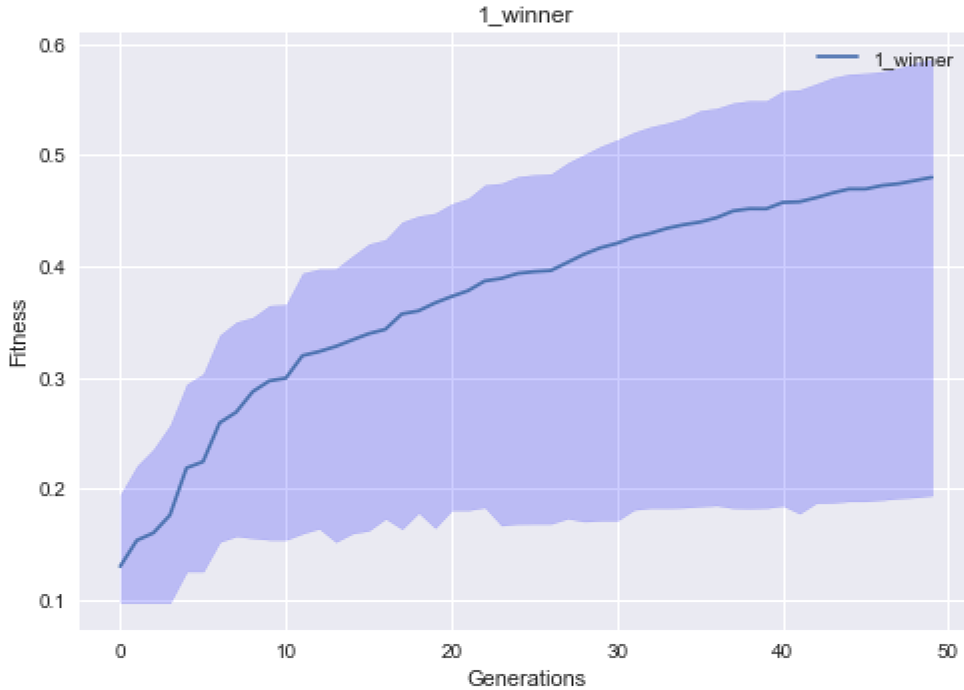
```
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(test_accuracy_res
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(test_accuracy_res
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(test_accuracy_res
```

```
(50, 5)
[array([0.19958, 0.25066]), array([0.22502, 0.28714]), array([0.25516, 0.37812]), array
([0.279  , 0.37154]), array([0.29864, 0.37978]), array([0.32136, 0.39552]), array([0.3305
8, 0.3943 ]), array([0.34534, 0.39986]), array([0.35444, 0.39562]), array([0.37078, 0.4041
]), array([0.37586, 0.40914]), array([0.39026, 0.41146]), array([0.39132, 0.41346]), array
([0.40512, 0.41574]), array([0.41248, 0.42762]), array([0.41732, 0.43256]), array([0.4170
2, 0.44554]), array([0.4237 , 0.45446]), array([0.4242 , 0.46424]), array([0.4304 , 0.4694
6]), array([0.43598, 0.46994]), array([0.43628, 0.47862]), array([0.4388 , 0.48288]), arra
y([0.44114, 0.49566]), array([0.4418, 0.4955]), array([0.44264, 0.5029 ]), array([0.46112,
0.5128 ]), array([0.4654 , 0.51932]), array([0.49144, 0.5262 ]), array([0.49854, 0.5378
8]), array([0.50096, 0.54272]), array([0.50614, 0.55158]), array([0.50854, 0.55968]), arra
y([0.5111 , 0.56142]), array([0.51198, 0.5623 ]), array([0.51362, 0.56268]), array([0.5195
2, 0.56698]), array([0.52448, 0.57482]), array([0.53014, 0.57486]), array([0.53202, 0.5763
]), array([0.53656, 0.58066]), array([0.54132, 0.58192]), array([0.54156, 0.58848]), array
([0.54252, 0.5949 ]), array([0.5432 , 0.60028]), array([0.54458, 0.59772]), array([0.5490
4, 0.6021 ]), array([0.55254, 0.6021 ]), array([0.54978, 0.60894]), array([0.56028, 0.6120
8])]
(50, 5)
[array([0.12534, 0.23502]), array([0.2179 , 0.25814]), array([0.22942, 0.27206]), array
([0.26124, 0.29026]), array([0.26946, 0.3034 ]), array([0.26876, 0.30796]), array([0.2933
8, 0.32756]), array([0.29488, 0.3331 ]), array([0.3275 , 0.34988]), array([0.33068, 0.3548
]), array([0.33642, 0.38492]), array([0.34936, 0.38722]), array([0.35642, 0.39088]), array
([0.3598 , 0.38978]), array([0.36112, 0.3918 ]), array([0.36534, 0.4143 ]), array([0.3746
8, 0.41566]), array([0.37748, 0.4234 ]), array([0.39648, 0.42396]), array([0.39848, 0.4326
]), array([0.40648, 0.43262]), array([0.41178, 0.45154]), array([0.4186 , 0.45108]), array
([0.4186, 0.4536]), array([0.42108, 0.46244]), array([0.43558, 0.4766 ]), array([0.437  ,
0.47856]), array([0.43102, 0.48636]), array([0.43586, 0.48616]), array([0.43018, 0.4950
4]), array([0.43856, 0.50144]), array([0.43848, 0.5034 ]), array([0.4515 , 0.51688]), arra
y([0.45818, 0.51836]), array([0.4532 , 0.52498]), array([0.4532 , 0.52498]), array([0.4588
4, 0.52258]), array([0.4588 , 0.52712]), array([0.45998, 0.54154]), array([0.4623 , 0.5435
6]), array([0.46206, 0.5458 ]), array([0.46216, 0.54394]), array([0.4663 , 0.55312]), arra
```

```
y([0.46782, 0.55948]), array([0.47086, 0.5579 ]), array([0.4727, 0.5607]), array([0.4765 ,
0.56452]), array([0.47754, 0.56418]), array([0.47774, 0.56946]), array([0.47774, 0.5709
6])]
(50, 5)
[array([0.098  , 0.19532]), array([0.098  , 0.22098]), array([0.098  , 0.23612]), array
([0.098  , 0.25762]), array([0.12612, 0.29436]), array([0.12612, 0.30362]), array([0.1528,
0.3385]), array([0.15784, 0.3502 ]), array([0.15604, 0.35444]), array([0.15464, 0.36506]),
array([0.15464, 0.36592]), array([0.1603 , 0.39418]), array([0.1647, 0.3978]), array([0.15
262, 0.398  ]), array([0.1603 , 0.40942]), array([0.16292, 0.42032]), array([0.17366, 0.42
422]), array([0.16428, 0.44004]), array([0.179  , 0.44544]), array([0.16518, 0.44798]), ar
ray([0.18126, 0.45648]), array([0.18124, 0.46138]), array([0.18376, 0.47346]), array([0.16
746, 0.4749 ]), array([0.1689 , 0.48086]), array([0.16908, 0.4825 ]), array([0.16908, 0.48
326]), array([0.17378, 0.49322]), array([0.17102, 0.50068]), array([0.17188, 0.5081 ]), ar
ray([0.17188, 0.51396]), array([0.18154, 0.52064]), array([0.18316, 0.52562]), array([0.18
316, 0.529  ]), array([0.18356, 0.53334]), array([0.18462, 0.54014]), array([0.1855 , 0.54
222]), array([0.18306, 0.54708]), array([0.18282, 0.54908]), array([0.18322, 0.549  ]), ar
ray([0.18518, 0.55792]), array([0.17818, 0.55892]), array([0.18772, 0.5642 ]), array([0.18
794, 0.56984]), array([0.18924, 0.57294]), array([0.18924, 0.57374]), array([0.19044, 0.57
534]), array([0.19188, 0.57856]), array([0.1928 , 0.58314]), array([0.19436, 0.5861 ])]
```







Q5c: Analysis

What did you find for a relationship between genetic diversity and overfitting to the training set? Was this what you expected?

It looks like in the case of 10 winners we are not overfitting, but that we might be slighly in the case of 5 winners and 1 winner. The graphs looks similar to those from the training accuracy case, but they have lower accuracy overall. When looking at the diversity for the 5 winner and 1 winner case, I don't notice anything that jumps out at me about them as being a relationship between genetic diversity and overfitting. I would expect there to be less diversity in a population with higher overfitting. The case with 10 winners has a ending diversity of ~14 with a similar accuracy on test and training. The charts with 5 winners show an ending diversity of ~22 and slight overfitting. The charts with 1 winner show an ending diversity of ~12 and more significant overfitting.

Q6: Modifying Mutation Rates

Next well modify the mutation rate for our algorithm. Based on the results you see above, and how you expect mutation rate to modify the genetic diveristy of a population, how might you think that increasing or decreasing the mutation rate might effect the different tournament size runs above?

I would think that this would help inject some diversity, but the tournament winner size is still going to have a significant impact on that diversity. I'm not sure that it will help much, but it could help to discover better solutions on other peaks if the mutation is great enough.

Q7: Experimentation

Let's find out! Let's do a mini grid search on the `mutation_size` and `num_tournament_winners` . To keep the number of runs down, let's just look at the exteme values of `num_tournament_winners` we had above (`1` and `10`), and run these for a `mutation_size` of `0.5` and `2.0` (in addition to the value of `1.0` we had before).

Hint: This is a good time to double check that your `mutation_size` parameter you implemented above is working correctly (i.e. your results for how it should effect diversity below make sense)

Note: This may take some time to run (if each condition is a couple minutes). Please try debugging code with smaller runs and make sure that if there are errors in along the way, what you've run already is saved and logged (so you don't have to rerun all 10 or 15 mins if you find a bug at the end of your script). And just use this time to go grab a coffee (or do some reading in your lovely evolutionary computation textbooks)!

In [14]:

```
num_runs = 5
total_generations = 50
genome_length = 14*14*10
num_elements_to_mutate = genome_length
mutation_size = [0.5,1,2]
num_parents = 50
num_children = 50
tournament_size = 20
num_tournament_winners = [1,10]

for i in range (len(mutation_size)):
    for j in range (len(num_tournament_winners)):
        name="m_{}_win_{}".format(mutation_size[i],num_tournament_winners[j])
        experiment_results[name] = np.zeros((num_runs, total_generations))
        solutions_results[name] = np.zeros((num_runs, total_generations, genome_length))
        diversity_results[name] = np.zeros((num_runs, total_generations))
        for k in range(num_runs):
            start_time = time.time()
            fitness_over_time, solutions_over_time, diversity_over_time = evolutionary_alg
            experiment_results[name][k] = fitness_over_time
            solutions_results[name][k] = solutions_over_time
            diversity_results[name][k] = diversity_over_time
            print(name, k, time.time()-start_time,fitness_over_time[-1])
```

```
m_0.5_win_1 0 191.85434341430664 0.5452666666666667
m_0.5_win_1 1 192.47287678718567 0.6410666666666667
m_0.5_win_1 2 193.1784851551056 0.5290166666666667
m_0.5_win_1 3 192.2902193069458 0.53955
m_0.5_win_1 4 190.12535333633423 0.6455
m_0.5_win_10 0 191.88787269592285 0.41886666666666666
m_0.5_win_10 1 193.72095251083374 0.49596666666666667
m_0.5_win_10 2 193.19750118255615 0.4971333333333333
m_0.5_win_10 3 193.49425649642944 0.41835
m_0.5_win_10 4 192.32875299453735 0.48606666666666665
m_1_win_1 0 192.42983961105347 0.5434166666666667
m_1_win_1 1 192.0730321407318 0.5587833333333333
m_1_win_1 2 194.0262153148651 0.58305
m_1_win_1 3 193.8860948085785 0.5547833333333333
m_1_win_1 4 192.88573217391968 0.5293166666666667
m_1_win_10 0 194.82690572738647 0.45816666666666667
m_1_win_10 1 195.75220274925232 0.49465
m_1_win_10 2 195.58105540275574 0.44366666666666665
m_1_win_10 3 194.6492531299591 0.46513333333333334
m_1_win_10 4 193.94714641571045 0.5089833333333333
m_2_win_1 0 190.08531951904297 0.64275
m_2_win_1 1 340.31679129600525 0.5524
m_2_win_1 2 385.9346058368683 0.5981666666666666
m_2_win_1 3 405.26026129722595 0.59265
m_2_win_1 4 406.72252082824707 0.5983
m_2_win_10 0 346.48510670661926 0.48586666666666667
m_2_win_10 1 345.7314577102661 0.48323333333333335
m_2_win_10 2 339.92145013809204 0.4513
m_2_win_10 3 344.2932186126709 0.42605
m_2_win_10 4 345.3746507167816 0.49361666666666665
```

Q8: Visualize

Please plot the results of these experiments (both fitness over time, and diveristy)

In [15]:

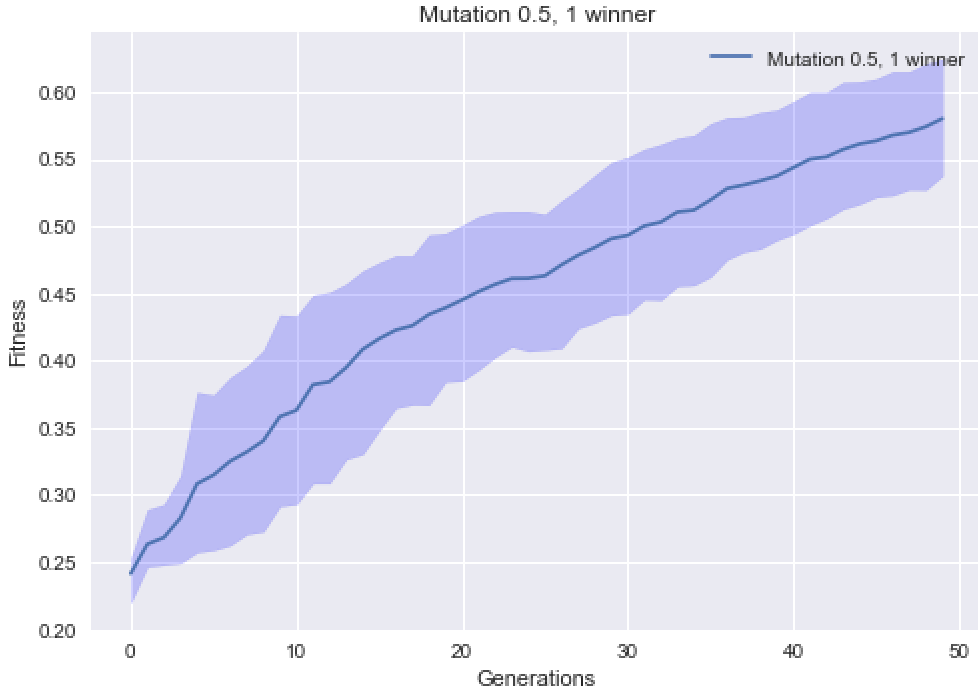
```
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(diversity_results
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(diversity_results
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(diversity_results
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(diversity_results
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(diversity_results
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(diversity_results
```

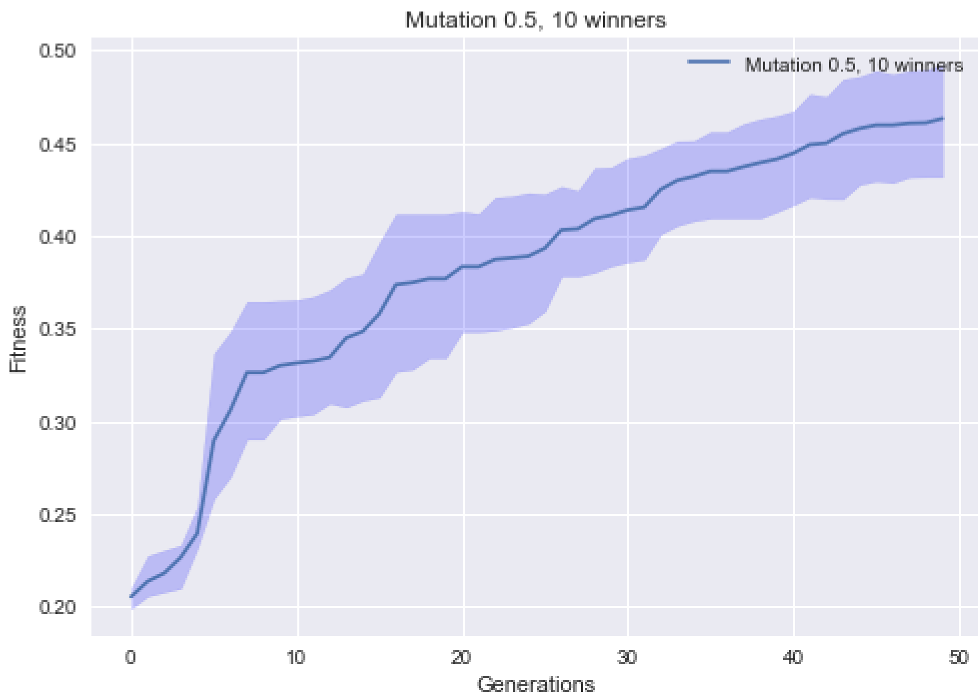
```
(50, 5)
[array([0.22003   , 0.25311333]), array([0.24671, 0.2892 ]), array([0.24809667, 0.2931
]), array([0.24943667, 0.31429   ]), array([0.25747667, 0.37644   ]), array([0.25905667,
0.37447667]), array([0.26266   , 0.38761333]), array([0.2709    , 0.39583333]), array([0.2
7289667, 0.40767   ]), array([0.29178333, 0.43401333]), array([0.29326667, 0.43333667]), a
rray([0.30896667, 0.44865   ]), array([0.30896667, 0.45083   ]), array([0.32676333, 0.4572
3333]), array([0.33051333, 0.46687667]), array([0.34836333, 0.47303333]), array([0.3648466
7, 0.47817667]), array([0.36744, 0.4781 ]), array([0.36722667, 0.49361333]), array([0.3844
1   , 0.49475333]), array([0.38509667, 0.50084667]), array([0.39330667, 0.50740667]), arra
y([0.40282667, 0.51067333]), array([0.41053   , 0.51101333]), array([0.40709   , 0.5110133
3]), array([0.40812   , 0.50895667]), array([0.40929667, 0.51896667]), array([0.42414333,
0.52776667]), array([0.42826667, 0.53790667]), array([0.43388667, 0.5475    ]), array([0.4
3495   , 0.55146333]), array([0.44564, 0.55741]), array([0.44511   , 0.56099667]), array
([0.45547333, 0.56564667]), array([0.45626, 0.56776]), array([0.46249333, 0.57648667]), ar
ray([0.47529667, 0.58089   ]), array([0.48084, 0.58135]), array([0.4833    , 0.58481667]),
array([0.48972333, 0.58657   ]), array([0.49431333, 0.59290667]), array([0.50076   , 0.599
69333]), array([0.50592   , 0.59969333]), array([0.51292333, 0.60725   ]), array([0.516533
33, 0.60759333]), array([0.52181333, 0.60979667]), array([0.52314333, 0.61513333]), array
([0.52737667, 0.61513333]), array([0.52717333, 0.62102667]), array([0.53762333, 0.6245666
7])]
(50, 5)
[array([0.19903333, 0.21016333]), array([0.20577333, 0.22756333]), array([0.20792667, 0.23
050667]), array([0.21001667, 0.23325667]), array([0.23165667, 0.25413   ]), array([0.25839
667, 0.33673667]), array([0.27015333, 0.34844   ]), array([0.29056, 0.36477]), array([0.29
056, 0.36477]), array([0.30157667, 0.36519667]), array([0.30293333, 0.36567   ]), array
([0.30401333, 0.36735667]), array([0.30979333, 0.37097667]), array([0.30780667, 0.3774933
3]), array([0.31129   , 0.37955333]), array([0.31309   , 0.39651667]), array([0.32693333,
0.41196   ]), array([0.3281 , 0.41196]), array([0.33422, 0.41196]), array([0.33422, 0.4119
6]), array([0.34835667, 0.4132    ]), array([0.34835667, 0.41206333]), array([0.34910667,
0.42086667]), array([0.35085333, 0.42152333]), array([0.35295333, 0.42325   ]), array([0.3
5956667, 0.42267   ]), array([0.37841333, 0.42672667]), array([0.37841667, 0.42450333]), a
rray([0.38040667, 0.43673   ]), array([0.38379333, 0.43709   ]), array([0.38599   , 0.4419
8333]), array([0.38739   , 0.44351333]), array([0.40118   , 0.44719333]), array([0.40564
, 0.45126333]), array([0.40819333, 0.45141667]), array([0.40963667, 0.45622   ]), array
([0.40963667, 0.45622   ]), array([0.40963667, 0.46052333]), array([0.40963667, 0.4630533
3]), array([0.41306   , 0.46469667]), array([0.41680333, 0.46726   ]), array([0.42093333,
0.47655   ]), array([0.42006333, 0.4752    ]), array([0.41999333, 0.48450333]), array([0.4
2773667, 0.48602667]), array([0.42957667, 0.48909   ]), array([0.42874, 0.48725]), array
([0.43177667, 0.48902667]), array([0.43199667, 0.48968667]), array([0.43199667, 0.4924733
3])]
(50, 5)
[array([0.22571, 0.24685]), array([0.23665333, 0.25874333]), array([0.26471333, 0.3000666
7]), array([0.28600333, 0.30338333]), array([0.28755333, 0.30652   ]), array([0.29437, 0.3
1926]), array([0.30447667, 0.34799   ]), array([0.32013333, 0.35515   ]), array([0.3306966
7, 0.38120667]), array([0.33905667, 0.38666667]), array([0.35395, 0.39201]), array([0.3592
6333, 0.39912333]), array([0.36592667, 0.41113333]), array([0.37705667, 0.41634333]), arra
y([0.38083333, 0.43304333]), array([0.40552, 0.44363]), array([0.40764667, 0.44908333]), a
```

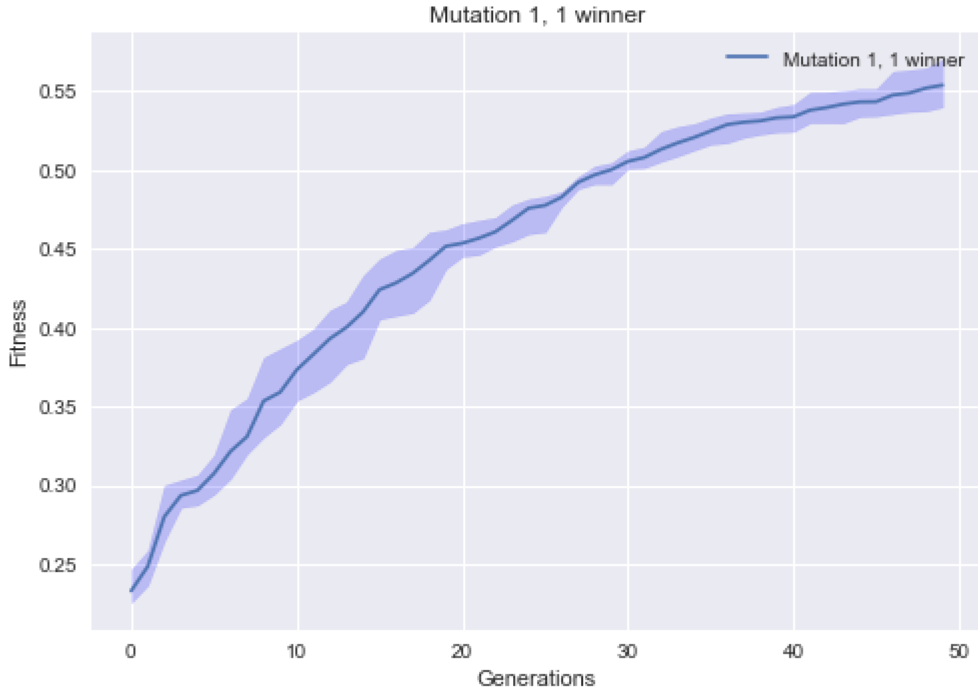
```
rray([0.40958667, 0.45082667]), array([0.41780667, 0.46062333]), array([0.43735667, 0.4620
8667]), array([0.44521   , 0.46607333]), array([0.44637   , 0.46807667]), array([0.4517733
3, 0.46992   ]), array([0.45491333, 0.47787667]), array([0.45954333, 0.48166333]), array
([0.46072667, 0.48339   ]), array([0.47691333, 0.48626333]), array([0.48817, 0.49586]), ar
ray([0.49127667, 0.50255667]), array([0.49117   , 0.50464667]), array([0.50099667, 0.51221
333]), array([0.50156667, 0.51463333]), array([0.50546333, 0.52444333]), array([0.5088466
7, 0.52753   ]), array([0.51266333, 0.52941667]), array([0.51631667, 0.53303   ]), array
([0.51729667, 0.53572333]), array([0.52068333, 0.53617667]), array([0.52265, 0.53679]), ar
ray([0.52407333, 0.53988333]), array([0.52449   , 0.54159333]), array([0.53003667, 0.54933
   ]), array([0.52999   , 0.54945333]), array([0.52996667, 0.55025667]), array([0.53389,
0.55155]), array([0.53432, 0.55155]), array([0.53578333, 0.56253333]), array([0.53710333,
0.56353667]), array([0.53751333, 0.56473   ]), array([0.54005, 0.57027])]
(50, 5)
[array([0.20698333, 0.25072667]), array([0.20698333, 0.26052667]), array([0.23727   , 0.26
449667]), array([0.25136667, 0.28102333]), array([0.26854667, 0.28268   ]), array([0.27504
333, 0.28867   ]), array([0.27933667, 0.29434   ]), array([0.29291333, 0.31432667]), array
([0.30515, 0.31691]), array([0.31172333, 0.31868   ]), array([0.31251333, 0.31926333]), ar
ray([0.31948, 0.36403]), array([0.32615   , 0.36153667]), array([0.33328, 0.36656]), array
([0.33401667, 0.36921667]), array([0.33582667, 0.37391667]), array([0.33402667, 0.3741
]), array([0.35361333, 0.40273333]), array([0.35751333, 0.40273333]), array([0.35789, 0.40
548]), array([0.36175667, 0.40603667]), array([0.36227667, 0.40603667]), array([0.3723733
3, 0.41047667]), array([0.37625333, 0.41649667]), array([0.37937333, 0.41570333]), array
([0.38210333, 0.42535667]), array([0.38401   , 0.42782333]), array([0.38595   , 0.4237133
3]), array([0.39467, 0.43928]), array([0.39866667, 0.44034   ]), array([0.39866667, 0.4428
6   ]), array([0.40506333, 0.45020333]), array([0.41438   , 0.45268667]), array([0.4202166
7, 0.45369667]), array([0.42683333, 0.46704   ]), array([0.42690667, 0.46633333]), array
([0.42897333, 0.46704   ]), array([0.43288, 0.46718]), array([0.43216   , 0.46636667]), ar
ray([0.43265333, 0.47438333]), array([0.43265333, 0.47438333]), array([0.4331 , 0.47464]),
array([0.43652667, 0.47542333]), array([0.43803   , 0.48709667]), array([0.44187, 0.4896
8]), array([0.44901   , 0.49075667]), array([0.45148333, 0.49280333]), array([0.45407   ,
0.49566667]), array([0.45508333, 0.49534333]), array([0.45386333, 0.49595333])]
(50, 5)
[array([0.20699667, 0.26309   ]), array([0.21621333, 0.2808    ]), array([0.23142   , 0.28
450333]), array([0.26931667, 0.30429333]), array([0.30018667, 0.32195   ]), array([0.31659
667, 0.3368    ]), array([0.33394333, 0.35437   ]), array([0.33646667, 0.37757333]), array
([0.34689333, 0.38317333]), array([0.3517    , 0.41139333]), array([0.3627    , 0.4124766
7]), array([0.36813333, 0.42810667]), array([0.37559333, 0.43200667]), array([0.38225   ,
0.43033333]), array([0.39331   , 0.45660333]), array([0.40923   , 0.46465667]), array([0.4
2457333, 0.47193   ]), array([0.42739333, 0.47583667]), array([0.44284333, 0.48201   ]), a
rray([0.45321667, 0.49416   ]), array([0.46153667, 0.49717   ]), array([0.46863667, 0.5051
1333]), array([0.46958   , 0.51296333]), array([0.47559667, 0.51707333]), array([0.4813533
3, 0.51953   ]), array([0.48537   , 0.52365667]), array([0.48749   , 0.52993333]), array
([0.49108667, 0.53431   ]), array([0.4925    , 0.54165333]), array([0.49875333, 0.5439
]), array([0.50016   , 0.55339333]), array([0.50452   , 0.55565667]), array([0.50892333,
0.56715333]), array([0.51434667, 0.56808   ]), array([0.52065667, 0.58089333]), array([0.5
2366333, 0.58287667]), array([0.52754333, 0.59453667]), array([0.53183667, 0.59604   ]), a
rray([0.53477667, 0.60127667]), array([0.53763667, 0.60207667]), array([0.54393333, 0.6091
9   ]), array([0.54512, 0.60919]), array([0.54432   , 0.60817667]), array([0.54913333, 0.6
0914333]), array([0.55480333, 0.61015   ]), array([0.55917   , 0.61416333]), array([0.5606
8667, 0.61648667]), array([0.56174333, 0.61798333]), array([0.56494333, 0.61859   ]), arra
y([0.56963   , 0.62381333])]
(50, 5)
[array([0.22054667, 0.29675667]), array([0.22205333, 0.29150333]), array([0.23177667, 0.29
874   ]), array([0.24475, 0.3001 ]), array([0.26887   , 0.32399667]), array([0.26781667,
0.32399667]), array([0.27279667, 0.32607667]), array([0.28894667, 0.33910333]), array([0.3
0458   , 0.35034667]), array([0.30531, 0.35135]), array([0.30581333, 0.35393333]), array
([0.31739, 0.35902]), array([0.32072667, 0.36227   ]), array([0.32577333, 0.36493333]), ar
ray([0.32933333, 0.36828   ]), array([0.33365333, 0.37005333]), array([0.35100667, 0.3756
   ]), array([0.35124667, 0.37602667]), array([0.35242667, 0.37852   ]), array([0.35344
, 0.39060667]), array([0.35445667, 0.38959   ]), array([0.35925333, 0.3984    ]), array
([0.37371   , 0.40557333]), array([0.37371   , 0.40704333]), array([0.37371   , 0.4078233
3]), array([0.37547333, 0.41216667]), array([0.38276333, 0.41216667]), array([0.38347333,
0.41428333]), array([0.38347333, 0.41428333]), array([0.39053333, 0.41906333]), array([0.3
9679333, 0.43915667]), array([0.40156333, 0.44029333]), array([0.40256333, 0.44006667]), a
rray([0.40197333, 0.43956667]), array([0.40774   , 0.44504667]), array([0.41116667, 0.4559
1667]), array([0.41116667, 0.45700667]), array([0.41167333, 0.45831   ]), array([0.4140266
```

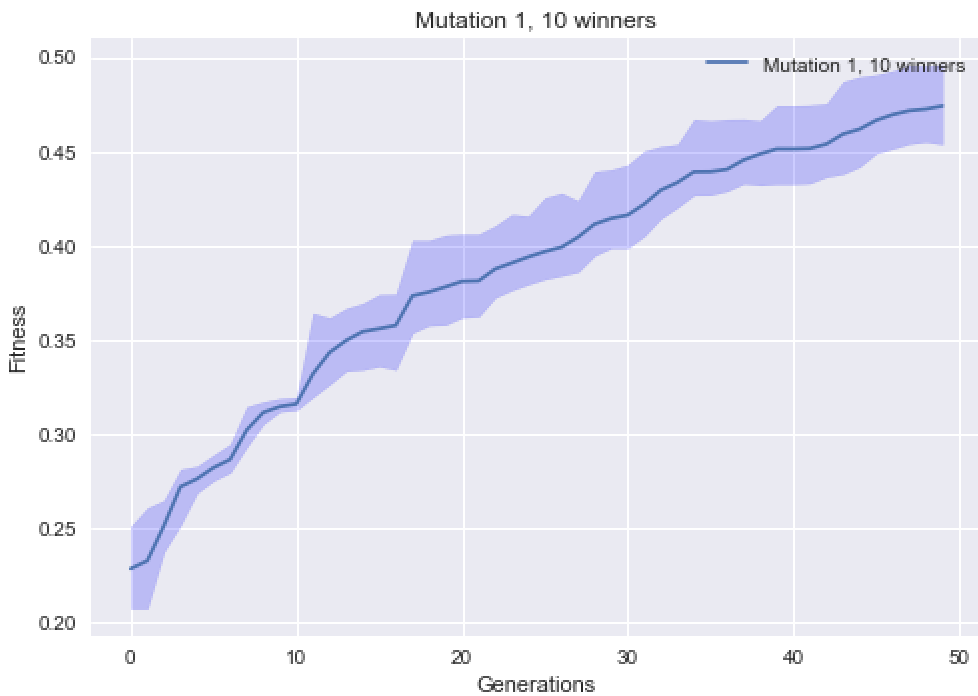
```
7, 0.45700667]), array([0.41432, 0.45866]), array([0.41914667, 0.46367333]), array([0.4219
1333, 0.46492667]), array([0.42494667, 0.46719333]), array([0.42433, 0.47144]), array([0.4
3025667, 0.47464   ]), array([0.43117   , 0.47787333]), array([0.43356667, 0.47918333]), a
rray([0.43488667, 0.48041333]), array([0.4343    , 0.47925333]), array([0.44253667, 0.4879
1333])]
(50, 5)
[array([6.38775002, 9.39557263]), array([1.89928936, 8.24684053]), array([1.04914466, 7.29
510778]), array([0.66872973, 7.21829554]), array([ 1.81825998, 10.93573154]), array([0.502
28713, 7.43742807]), array([0.81778245, 6.46695083]), array([1.89583802, 7.64577494]), arr
ay([1.66456371, 6.74198654]), array([2.68134501, 9.66255159]), array([2.72208693, 9.489898
89]), array([ 4.06592183, 11.19366236]), array([2.76866409, 8.29178423]), array([1.1666660
1, 6.53229628]), array([4.07236015, 9.51756261]), array([1.927689 , 7.9527484]), array([2.
10280661, 9.03911324]), array([ 5.97738477, 10.53756439]), array([2.01540824, 7.6308338
5]), array([1.63372818, 6.59430485]), array([1.66084671, 7.35276328]), array([4.023272  ,
9.46130109]), array([1.94014003, 8.91399765]), array([ 5.24177298, 11.96901863]), array
([0.53935142, 3.76486821]), array([0.82922401, 6.0643787 ]), array([3.30110574, 7.8492661
6]), array([ 4.12586044, 10.19131763]), array([4.08542054, 9.6961844 ]), array([5.2388675
2, 8.66531017]), array([4.34377746, 7.3054612 ]), array([6.2963714 , 8.63425436]), array
([3.89851132, 8.53250267]), array([5.32602093, 9.10333097]), array([ 4.63816227, 10.069910
86]), array([ 4.57301194, 10.40953613]), array([ 6.44719006, 11.65048364]), array([4.07865
727, 8.17987475]), array([1.20611427, 6.0876481 ]), array([2.03339193, 8.26910159]), array
([ 6.64362368, 11.14248583]), array([ 6.62552479, 11.90516111]), array([ 6.38784091, 14.23
0298  ]), array([5.07249646, 8.85947042]), array([5.60138507, 7.54346072]), array([1.32412
507, 5.4341434 ]), array([1.90473633, 7.73259988]), array([1.51681438, 5.84479475]), array
([4.12147639, 7.39005688]), array([5.39903261, 8.3244912 ])]
(50, 5)
[array([7.90422965, 9.11560852]), array([11.10163041, 12.01010802]), array([13.13769123, 1
4.63276644]), array([14.11596134, 16.23592898]), array([15.49548006, 17.52800669]), array
([16.33078527, 19.4913707 ]), array([17.12663703, 20.59338095]), array([17.37189801, 21.13
208855]), array([18.37639318, 21.6192244 ]), array([18.87991716, 21.89175617]), array([18.
48095146, 21.9149233 ]), array([17.21769273, 21.15982444]), array([16.78764675, 20.2982807
8]), array([16.18722006, 22.43517094]), array([16.66833715, 21.85134818]), array([17.25281
819, 20.63538859]), array([16.55776531, 20.0400403 ]), array([17.25812057, 19.58326034]),
array([18.04540901, 20.65419862]), array([18.40460992, 20.48636754]), array([16.53879187,
21.3718715 ]), array([16.49799615, 21.95839141]), array([16.84054456, 22.51253454]), array
([16.43632826, 22.065031  ]), array([17.24269276, 22.61230522]), array([16.7334038 , 22.55
067069]), array([16.80712398, 22.79971015]), array([15.56520195, 22.91560294]), array([16.
31488472, 22.28381155]), array([16.9323325 , 21.98451737]), array([16.37549656, 21.0606577
2]), array([17.20690166, 21.34180404]), array([17.73094624, 21.65961036]), array([18.10251
683, 21.58570983]), array([17.77346383, 22.12842333]), array([16.04589334, 21.6457299 ]),
array([16.19634612, 21.80718082]), array([17.17931876, 22.67165858]), array([17.23884638,
23.43321396]), array([17.70166655, 23.99699035]), array([18.25658892, 23.96662936]), array
([18.51660242, 24.18458083]), array([18.70151544, 24.1080343 ]), array([17.70234613, 22.81
039291]), array([18.75207606, 21.08372513]), array([18.57499211, 20.95472609]), array([18.
63930808, 21.66693996]), array([18.69489185, 22.57805498]), array([18.28878686, 21.7634863
]), array([18.31241596, 20.89480139])]
(50, 5)
[array([13.8327631 , 16.59937565]), array([15.47132183, 19.55540646]), array([11.18503871,
21.14039338]), array([ 4.20316576, 18.10602212]), array([ 2.39951374, 21.26305289]), array
([ 1.44522862, 24.33426642]), array([ 3.63251659, 18.54986334]), array([ 9.12767901, 17.38
513373]), array([10.46243625, 19.5880124 ]), array([11.01658027, 19.76234612]), array([10.
33324866, 22.53972385]), array([ 8.49259809, 16.90574012]), array([ 8.3332664 , 14.0621001
9]), array([ 8.72429251, 13.68614673]), array([12.03713414, 17.19348767]), array([ 9.00158
733, 13.40865731]), array([11.62275468, 17.24897028]), array([13.60727309, 19.63653001]),
array([10.56395036, 23.51823511]), array([12.90967482, 28.83900904]), array([ 1.5111513 ,
13.68990853]), array([5.77739245e-15, 1.33643677e+01]), array([ 4.48836485, 18.20757494]),
array([ 7.39038343, 15.40704895]), array([10.47124877, 19.91417975]), array([ 2.42665557,
12.04811446]), array([ 7.67855788, 11.15048609]), array([ 9.91348505, 16.8779441 ]), array
([11.35363559, 15.30833142]), array([ 9.8919682 , 18.32475879]), array([ 5.21432188, 20.02
904969]), array([ 5.29322513, 22.61759156]), array([ 6.06086915, 22.89881752]), array([15.
9190354 , 27.54079081]), array([10.99304769, 22.55719005]), array([ 3.18563977, 16.1093438
4]), array([10.1172648 , 15.75742737]), array([ 4.69443023, 19.93335242]), array([ 4.36695
779, 19.58381439]), array([ 5.56620136, 21.92380541]), array([ 1.94118354, 18.0030433 ]),
array([ 4.32880222, 20.69479585]), array([ 7.61270838, 11.77500813]), array([ 1.78860427,
10.15410458]), array([ 2.76864565, 13.11302519]), array([1.39983753, 8.17957903]), array([
3.79505521, 15.65794832]), array([ 5.95655804, 16.158844  ]), array([13.1406699, 20.918458
```

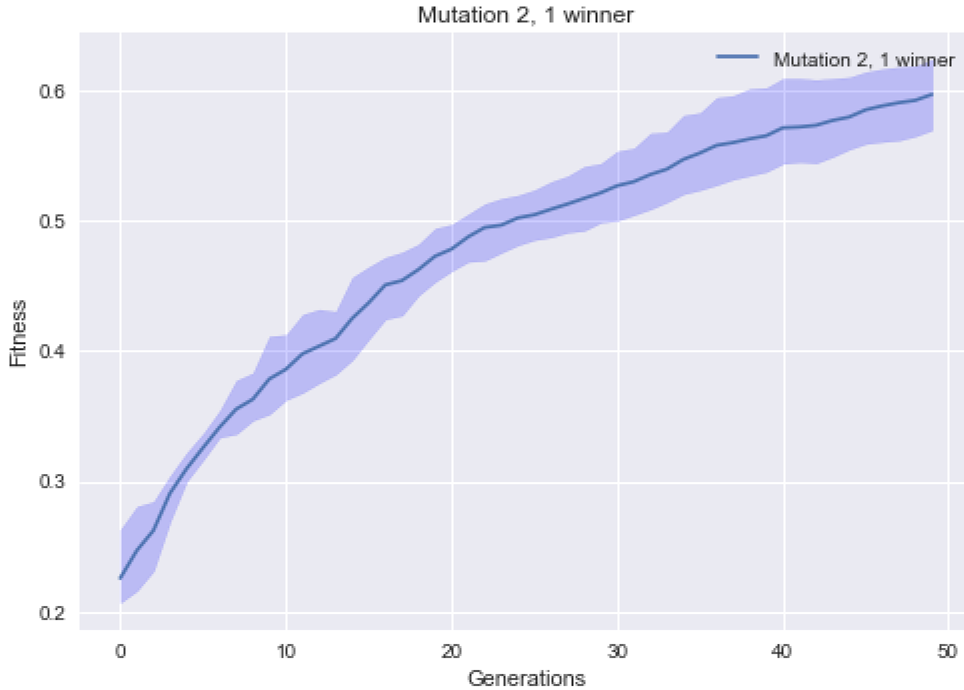
```
3]), array([12.98319106, 23.25486995])]
(50, 5)
[array([17.00846776, 19.01071887]), array([21.80689409, 25.61282578]), array([26.04255369,
29.11456671]), array([30.41973714, 33.93619928]), array([32.54835196, 37.53050625]), array
([36.05071598, 40.75457649]), array([37.58726256, 42.71180288]), array([38.30955908, 44.09
032284]), array([39.37756526, 45.10802773]), array([39.21971811, 44.36918903]), array([39.
10581125, 45.83551151]), array([36.77128798, 46.42573471]), array([36.04664382, 47.8498379
2]), array([37.37476611, 49.68465876]), array([37.44697097, 50.12631397]), array([38.82698
943, 50.0913618 ]), array([38.87556914, 46.9690376 ]), array([33.86530336, 40.75331016]),
array([32.98750401, 40.16212043]), array([34.62025524, 41.90784703]), array([37.24699971,
42.15438185]), array([36.76282625, 43.61658084]), array([37.50641112, 43.98754038]), array
([39.64877133, 45.48623328]), array([43.56159669, 48.41835654]), array([43.02199331, 49.38
823055]), array([42.26430275, 49.74275189]), array([42.56750221, 48.83883312]), array([42.
05091793, 50.5180762 ]), array([39.23891599, 48.07890364]), array([38.3249217 , 44.8906933
5]), array([36.17773844, 42.78739877]), array([35.39086897, 41.4864547 ]), array([36.67014
726, 47.49764205]), array([35.23209695, 48.17825671]), array([33.85442204, 44.40617257]),
array([34.51398014, 44.0365167 ]), array([35.28819347, 39.82900009]), array([34.05493453,
40.94984425]), array([34.94923205, 40.59189263]), array([36.36273734, 39.42820565]), array
([37.26004067, 40.41295393]), array([38.44094248, 41.44856065]), array([36.91642122, 43.23
503283]), array([36.08533504, 44.03480845]), array([38.23115406, 44.69297174]), array([38.
25060796, 44.74968683]), array([38.86354494, 46.33669988]), array([39.65532969, 46.6455708
4]), array([39.6810717 , 46.22961441])]
(50, 5)
[array([29.27416873, 34.20160145]), array([25.92611982, 32.40665711]), array([15.54298726,
37.78625955]), array([22.86040521, 42.07909613]), array([25.08648148, 46.06044523]), array
([34.88841385, 52.66983404]), array([31.24920246, 50.75830537]), array([17.84735917, 29.65
396943]), array([15.91375364, 34.67343384]), array([15.37984131, 42.23086348]), array([20.
85252096, 40.77273557]), array([21.31772197, 48.4345728 ]), array([23.97797564, 50.6683626
6]), array([10.3378438 , 45.72992905]), array([24.35921527, 41.37694001]), array([17.65130
637, 41.71973665]), array([ 9.37894089, 37.7585645 ]), array([ 7.62673772, 34.36262746]),
array([10.02816277, 42.05100289]), array([21.71655814, 49.08709989]), array([14.68790361,
45.66588412]), array([17.2959119 , 38.30000004]), array([20.04048287, 42.64772974]), array
([20.26882655, 35.78051612]), array([24.80579161, 36.0616366 ]), array([15.96427856, 36.97
147208]), array([25.30216068, 30.05198542]), array([16.3995107 , 29.06890386]), array([22.
43633979, 43.69943648]), array([19.45661358, 44.81545285]), array([27.89432118, 55.6455798
5]), array([ 8.67177954, 37.63559823]), array([16.31681055, 39.72656841]), array([ 9.01443
582, 40.95488023]), array([24.64269472, 40.71549532]), array([28.53793522, 43.14599167]),
array([28.83370617, 47.67690558]), array([17.84705208, 39.62469771]), array([ 7.63399722,
36.6294016 ]), array([24.04870819, 48.70038267]), array([28.82212667, 56.4038194 ]), array
([16.18598894, 23.20774796]), array([ 3.36689697, 24.70453528]), array([ 8.27464754, 41.20
922322]), array([ 7.18417081, 32.32355733]), array([21.35285511, 43.37890547]), array([18.
85439037, 36.89969262]), array([19.50105148, 34.38977622]), array([18.53172346, 39.6675235
9]), array([18.87346468, 38.05411548])]
(50, 5)
[array([34.66692715, 36.63180876]), array([42.06371923, 47.21684599]), array([53.81602647,
59.55452129]), array([59.18138355, 67.56054497]), array([66.6646903 , 72.56014485]), array
([67.44195969, 74.27989825]), array([66.91100922, 73.45970333]), array([68.61576804, 77.32
30518 ]), array([69.81141716, 80.99738304]), array([62.03142854, 80.16438313]), array([59.
5333823 , 78.35874891]), array([57.324598  , 76.27109082]), array([60.49882332, 79.2725813
7]), array([64.15767225, 87.70973141]), array([66.37260605, 88.70666635]), array([69.04994
855, 86.23371767]), array([69.73971432, 86.48308888]), array([69.92242007, 84.70925112]),
array([73.53556203, 87.67969196]), array([75.15173139, 90.8631068 ]), array([80.57321166,
99.42943879]), array([ 80.955368  , 101.54985387]), array([80.15339768, 98.91084274]), arr
ay([77.49169889, 91.37125488]), array([73.78670645, 89.34615398]), array([70.21524758, 93.
17480072]), array([66.11030109, 94.63854755]), array([68.62252366, 95.215479  ]), array([6
9.42934299, 91.6961654 ]), array([ 76.75759834, 103.48449992]), array([ 77.49223185, 106.8
0102506]), array([ 79.43541883, 104.77083558]), array([77.72383115, 97.16208234]), array
([74.76978416, 97.04269863]), array([ 80.07415243, 101.96395185]), array([ 81.43974757, 10
0.90654948]), array([ 80.50255415, 104.4427717 ]), array([ 77.8251244 , 104.11406954]), ar
ray([ 78.41916287, 114.83013909]), array([ 76.07524371, 115.58836405]), array([ 68.2546047
1, 113.26508188]), array([ 67.19530234, 109.92242266]), array([ 64.12529063, 116.5640493
3]), array([ 64.84053352, 124.06589942]), array([ 72.22941068, 117.94228031]), array([ 73.
39899305, 120.32270004]), array([ 71.19281479, 118.68032684]), array([ 70.87065156, 110.19
262867]), array([73.13118082, 99.61043458]), array([ 71.95982592, 101.5630572 ])]
```


Mutation 0.5, 1 winner
Mutation 0.5, 1 winner
Fitness
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0
10
20
30
40
50
Generations

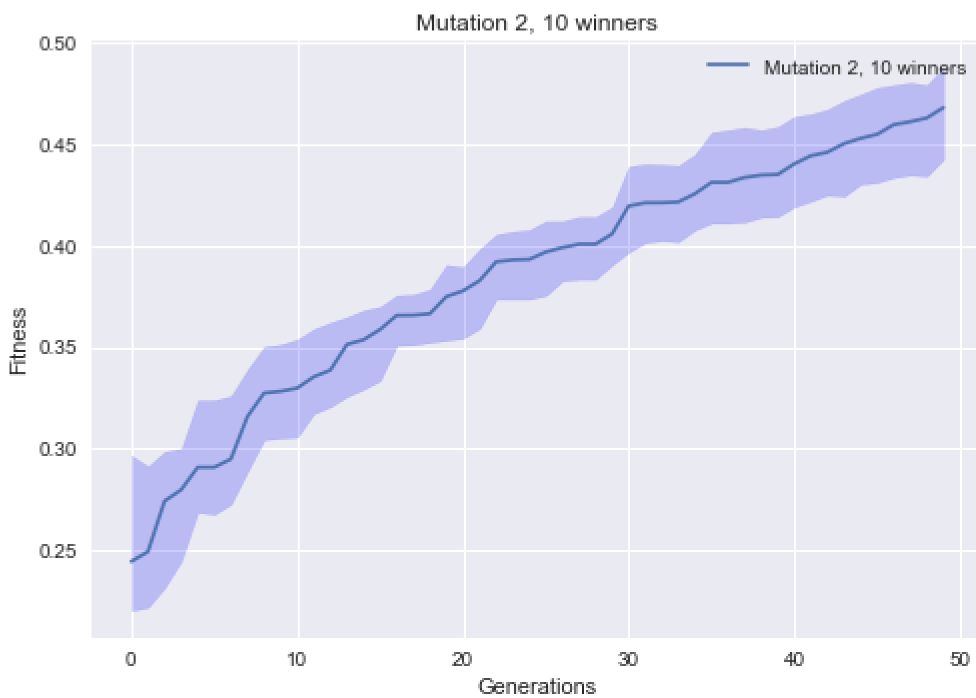

Mutation 0.5, 10 winners
Mutation 0.5, 10 winners
Fitness
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0
10
20
30
40
50
Generations


Mutation 1, 1 winner
Mutation 1, 1 winner
Fitness
Generations
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0
10
20
30
40
50

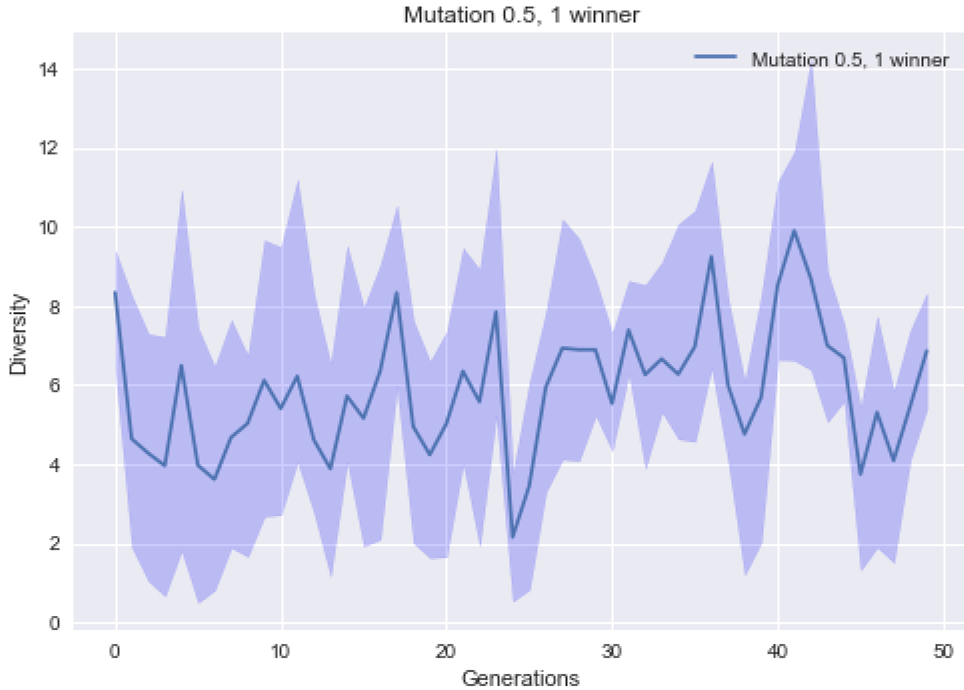

Mutation 1, 10 winners
Mutation 1, 10 winners
Fitness
Generations
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0
10
20
30
40
50



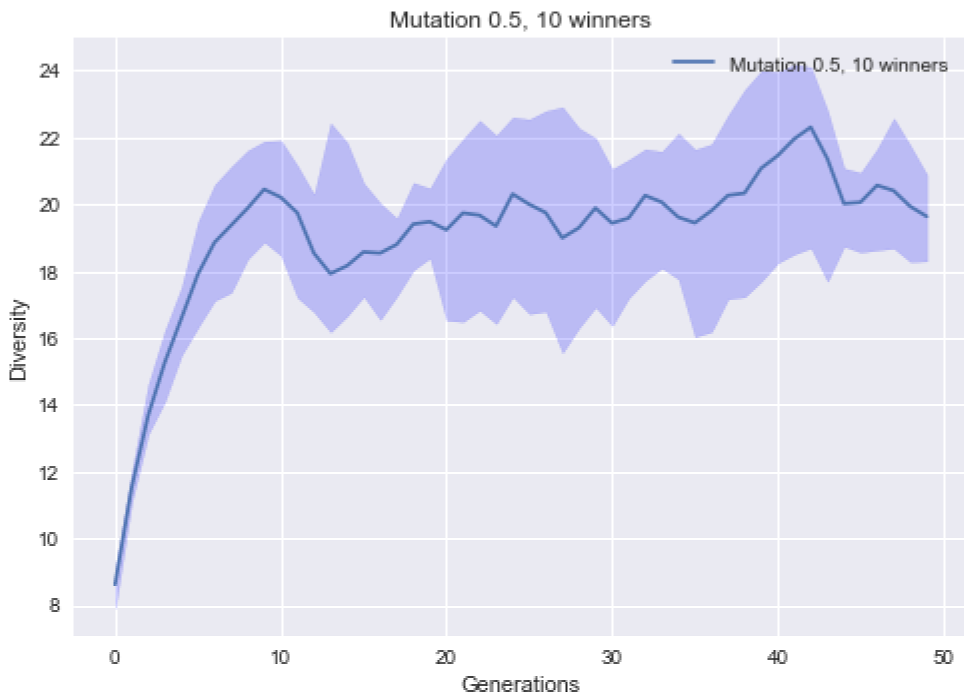
Mutation 2, 1 winner
Mutation 2, 1 winner
Fitness
Generations
0.2
0.3
0.4
0.5
0.6
0
10
20
30
40
50



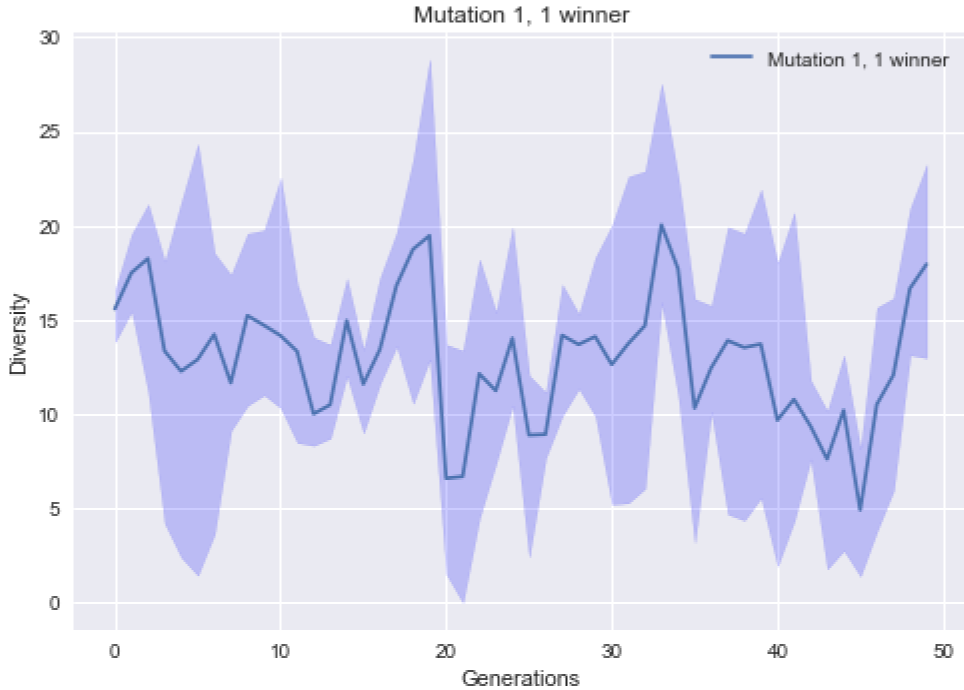
Mutation 2, 10 winners
Mutation 2, 10 winners
Fitness
Generations
0.25
0.30
0.35
0.40
0.45
0.50
0
10
20
30
40
50

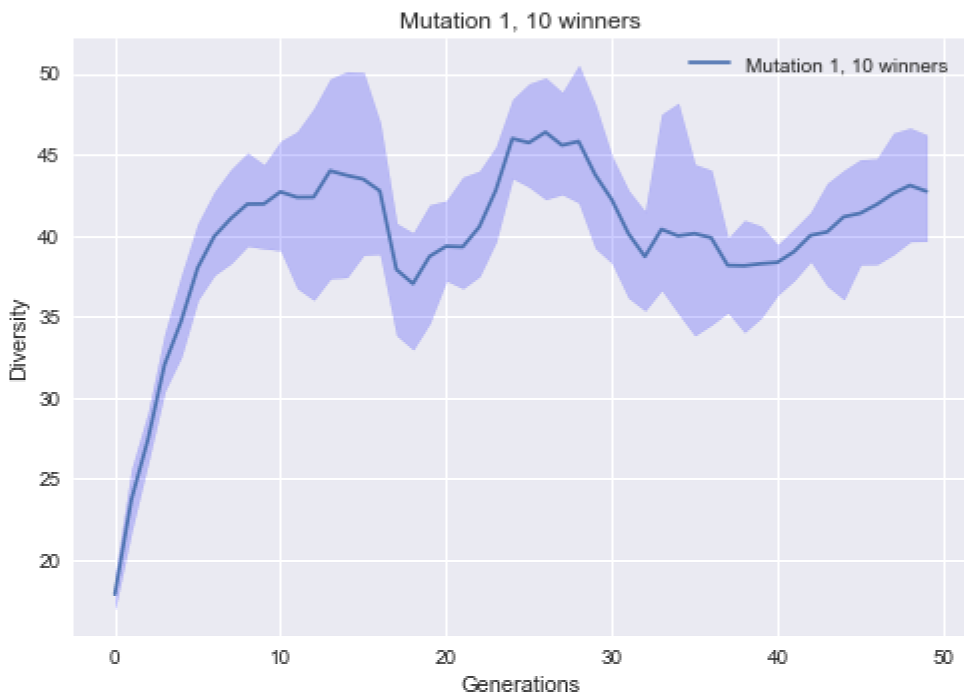


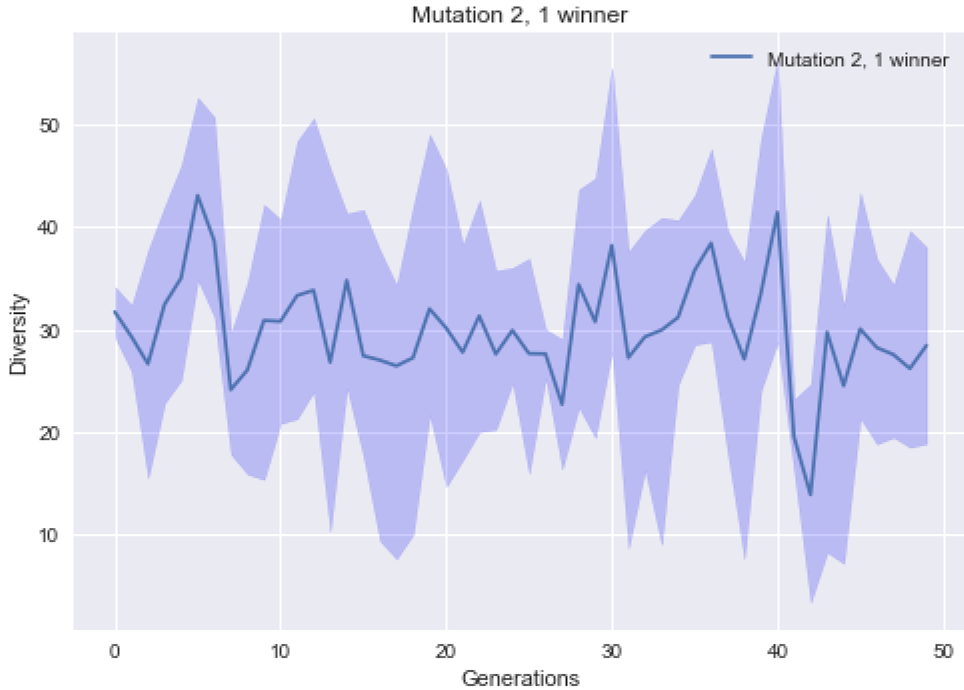
Mutation 0.5, 1 winner
Mutation 0.5, 1 winner
Diversity
Generations
0
2
4
6
8
10
12
14
0
10
20
30
40
50

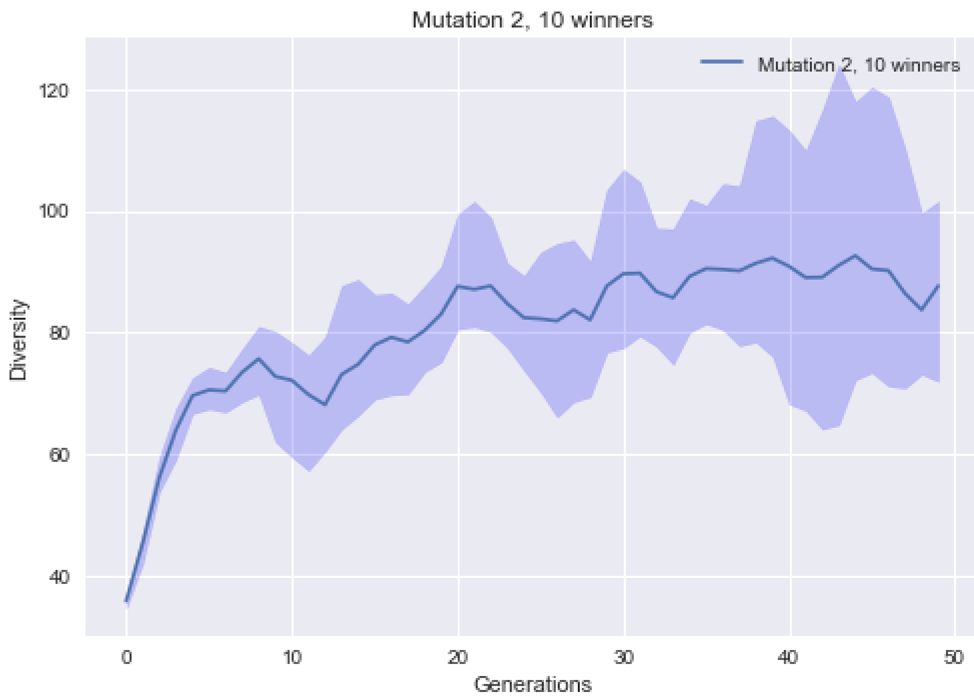


Mutation 0.5, 10 winners
Mutation 0.5, 10 winners
Diversity
Generations
8
10
12
14
16
18
20
22
24
0
10
20
30
40
50


Mutation 1, 1 winner
Mutation 1, 1 winner
Diversity
Generations
0
5
10
15
20
25
30
0
10
20
30
40
50


Mutation 1, 10 winners
Mutation 1, 10 winners
Diversity
Generations
20
25
30
35
40
45
50
0
10
20
30
40
50





Q8b: Analysis

What patterns do you see? Did you expect this given the implications of each independently? Does the level of diversity match your intuition about how well search should perform? Does this tell you anything about the role/interaction of variation (e.g. mutation rate) and selection (e.g. tournament parameters)?

These charts seem very difficult to interpret. It seems that there is a connection between mutation and diversity. Diveristy seems to stay higher when the tournament winners is higher (selection pressure is lower). It also appears that as mutation rates get higher the dropoff of diversity seems to get more likely. This is probably because the tournaments are starting to choose the better individuals which now have a larger gap between the worse individuals in the population. That being said the overall diversity rate of mutation 2, 10 winners is much higher than that of mutation 0.5, 10 winners even though the latter seems to have a more consistent rise in diversity. Mutation rate of 1 and 10 winners also has a higher overall diversity than that of 0.5 mutation rate, but it does see a dropoff from the selection pressure (I assume).

Q9: Dynamic Mutation Rate

We talked in class about many way to have dynamic or adaptive mutation rates. Let's experiment with the simplest form of this, a mutation rate that changes linearly over generational time, from some provided starting value to some provided ending value. Please modify your evolutionary algorithm code below to enable this.

In [16]:

```
def evolutionary_algorithm(fitness_function=None, total_generations=100, num_parents=10, r
    """ Evolutionary Algorithm (copied from the basic hillclimber in our last assignment)

        parameters:
        fitness_funciton: (callable function) that return the fitness of a genome
                           given the genome as an input parameter (e.g. as defined in Lanc
        total_generations: (int) number of total iterations for stopping condition
        num_parents: (int) the number of parents we downselect to at each generation (mu)
        num_childre: (int) the number of children (note: parents not included in this cour
        genome_length: (int) length of the genome to be evoloved
        num_elements_to_mutate: (int) number of alleles to modify during mutation (0 = no
        mutation_size_start: (float) scaling parameter of the magnitidue of mutations for
        mutation_size_end: (float) scaling parameter of the magnitidue of mutations for fi
        crossover: (bool) whether to perform crossover when generating children
        tournament_size: (int) number of individuals competing in each tournament
        num_tournament_winners: (int) number of individuals selected as future parents fro

        returns:
        fitness_over_time: (numpy array) track record of the top fitness value at each ger
        solutions_over_time: (numpy array) track record of the top genome value at each ge
        diversity_over_time: (numpy array) track record of the population genetic diversit
    """

       # initialize record keeping
    solution = None # best genome so far
    solution_fitness = -99999 # fitness of best genome so far
    solution_generation = 0 # time (generations) when solution was found
    fitness_over_time = np.zeros(total_generations)
    solutions_over_time = np.zeros((total_generations,genome_length),dtype=int)
    diversity_over_time = np.zeros(total_generations)

    # the initialization proceedure
    population = [] # keep population of individuals in a list
    for i in range(num_parents): # only create parents for initialization (the mu in mu+la
        population.append(Individual(fitness_function,genome_length)) # generate new rando

    # get population fitness
    for i in range(len(population)):
        population[i].eval_fitness() # evaluate the fitness of each parent

    for generation_num in range(total_generations): # repeat

#          print(generation_num)

        # the modification procedure
        new_children = [] # keep children separate for now (lambda in mu+lambda)
        while len(new_children) < num_children:

            # inheretance
            [parent1, parent2] = np.random.choice(population, size=2) # pick 2 random pare
            child1 = copy.deepcopy(parent1) # initialize children as perfect copies of the
            child2 = copy.deepcopy(parent2)

#           # crossover
            if crossover:
                for child, this_parent, other_parent in [[child1, parent1, parent2],[chilc
                    child.genome = -1*np.ones(len(child.genome))
```

```
                    child.genome[0] = this_parent.genome[0]
                    next_index = np.where(other_parent.genome == this_parent.genome[0])
                    while next_index != 0:
                        child.genome[next_index] = this_parent.genome[next_index]
                        next_index = np.where(other_parent.genome == child.genome[next_inc
                    child.genome[np.where(child.genome == -1)] = other_parent.genome[np.wh
                    child.genome = child.genome.astype(int)

            # mutation
            for this_child in [child1,child2]:
                for _ in range(num_elements_to_mutate):
#                   [index_to_swap1, index_to_swap2] = np.random.randint(0,genome_lengtl
#                   while index_to_swap1 == index_to_swap2: [index_to_swap1, index_to_sw
#                   orig_gene_1 = this_child.genome[index_to_swap1]
#                   this_child.genome = np.delete(this_child.genome,index_to_swap1)
#                   this_child.genome = np.insert(this_child.genome,index_to_swap2,orig_
#                   start - ((generation / total_generations) * (distance between start
                    this_child.genome = this_child.genome + (np.random.rand(genome_length)

            new_children.extend((child1,child2)) # add children to the new_children list

        # the assessement procedure
        for i in range(len(new_children)):
            new_children[i].eval_fitness() # assign fitness to each child

        # selection procedure
        population += new_children # combine parents with new children (the + in mu+lambda
        population = sorted(population, key=lambda individual: individual.fitness, reverse

        # tournament selection
        new_population = []
        new_population.append(population[0])
        while len(new_population) < num_parents:
            tournament = np.random.choice(population, size = tournament_size)
            tournament = sorted(tournament, key=lambda individual: individual.fitness, rev
            new_population.extend(tournament[:num_tournament_winners])
        population = new_population

        # record keeping

        if population[0].fitness > solution_fitness: # if the new parent is the best found
            solution = population[0].genome                # update best solution records
            solution_fitness = population[0].fitness
            solution_generation = generation_num
        fitness_over_time[generation_num] = solution_fitness # record the fitness of the c
        solutions_over_time[generation_num,:] = solution

        all_gene_std = []
        for x in range(genome_length):
            this_gene_values=[]
            for y in range(len(population)):
                this_gene_values.append(population[y].genome[x])
            all_gene_std.append(np.std(this_gene_values))
        diversity_over_time[generation_num] = np.mean(all_gene_std)

    return fitness_over_time, solutions_over_time, diversity_over_time
```

Q9b: Experimentation

Please peform a set of runs which decrease the mutation rate from `1.0` to `0.1` linearly over the 50 generations of search for a tournament of size `20` with `1` winner selected.

In [17]:

```
num_runs = 5
```

```
total_generations = 50
genome_length = 14*14*10
proportion_elements_to_mutate = 1.0
mutation_size_start = 1.0
mutation_size_end = 0.1
num_parents = 50
num_children = 50
tournament_size = 20
num_tournament_winners = 1

for run_name in ["decrease_mutation"]:
    experiment_results[run_name] = np.zeros((num_runs, total_generations))
    solutions_results[run_name] = np.zeros((num_runs, total_generations, genome_length))
    diversity_results[run_name] = np.zeros((num_runs, total_generations))
    for run_num in range(num_runs):
        start_time = time.time()
        fitness_over_time, solutions_over_time, diversity_over_time = evolutionary_algorit
        experiment_results[run_name][run_num] = fitness_over_time
        solutions_results[run_name][run_num] = solutions_over_time
        diversity_results[run_name][run_num] = diversity_over_time
        print(run_name, run_num, time.time()-start_time,fitness_over_time[-1])
```

```
decrease_mutation 0 219.97758054733276 0.5198666666666667
decrease_mutation 1 221.91374969482422 0.49115
decrease_mutation 2 222.3956642150879 0.5938833333333333
decrease_mutation 3 225.57640671730042 0.6075833333333334
decrease_mutation 4 220.59211015701294 0.5401666666666667
```

Q10: Visualize

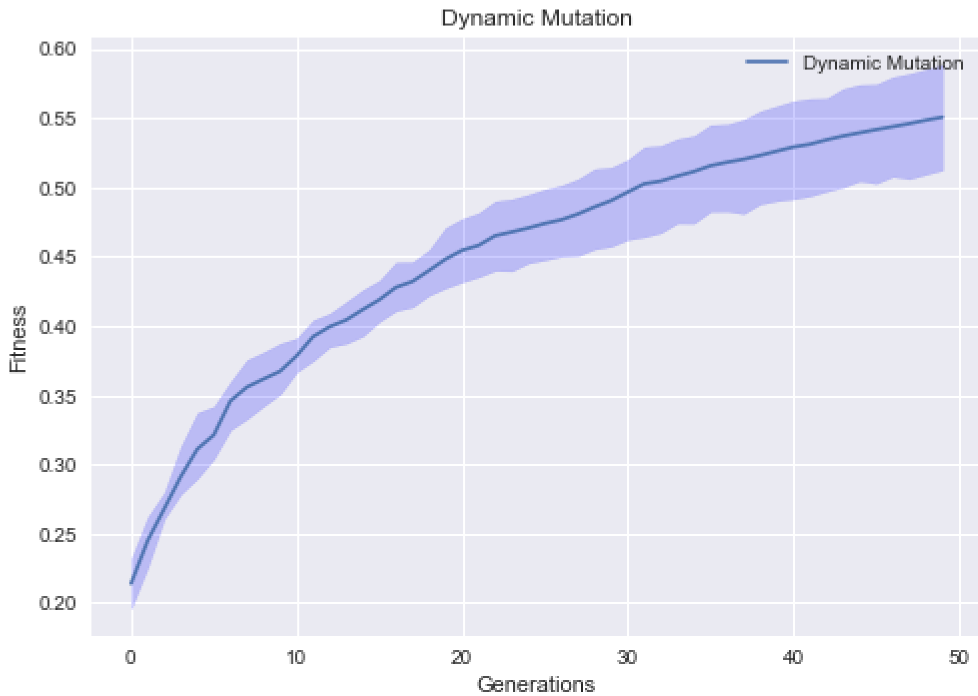
Please plot (fitness and diversity of) the dynamic mutation rate against fixed mutation rates of `1.0` and `0.5` for the same tournament parameters.

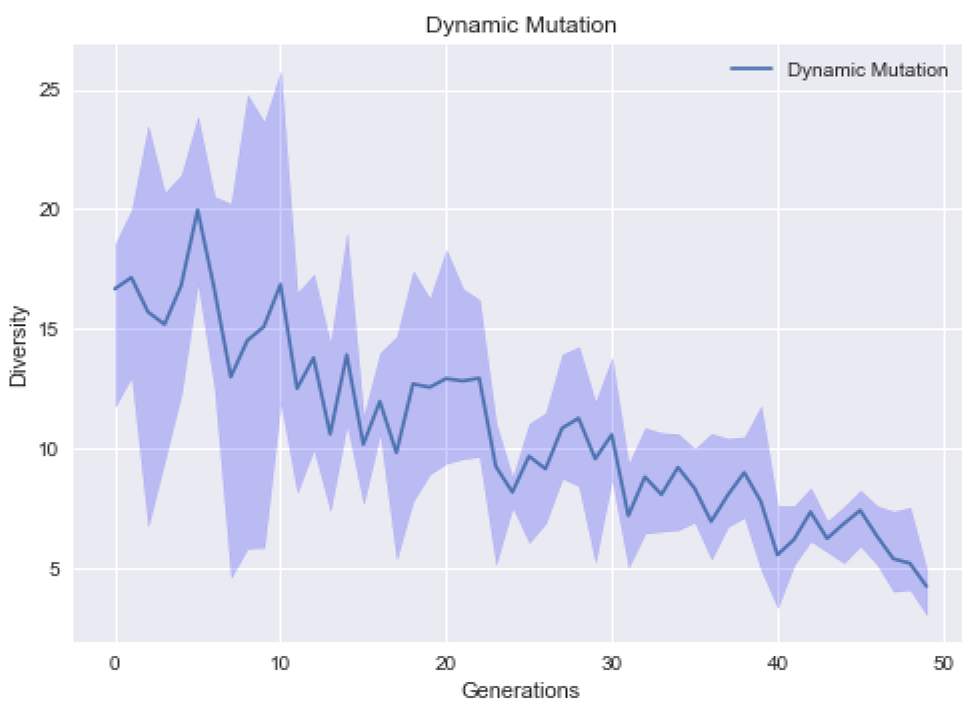
In [18]:

```
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(diversity_results
```

```
(50, 5)
[array([0.19597667, 0.23199667]), array([0.2257    , 0.26214333]), array([0.26086333, 0.27
982   ]), array([0.27861   , 0.31302667]), array([0.28995   , 0.33741667]), array([0.30420
667, 0.34194   ]), array([0.32502667, 0.35977333]), array([0.33293667, 0.37564   ]), array
([0.34218667, 0.38114667]), array([0.35083333, 0.38741   ]), array([0.36683   , 0.3911733
3]), array([0.37496333, 0.40428667]), array([0.38496, 0.40909]), array([0.38745, 0.4174
8]), array([0.39280667, 0.4261    ]), array([0.40325667, 0.43284667]), array([0.41098333,
0.44595667]), array([0.41367667, 0.44613667]), array([0.42211667, 0.45487667]), array([0.4
2740333, 0.47094333]), array([0.43171667, 0.47736333]), array([0.43542   , 0.48156667]), a
rray([0.44006   , 0.49003333]), array([0.4397    , 0.49147667]), array([0.44539667, 0.4947
9   ]), array([0.44768667, 0.49858   ]), array([0.45018   , 0.50145667]), array([0.45075,
0.50603]), array([0.45548667, 0.51331667]), array([0.45746   , 0.51437667]), array([0.4622
8667, 0.51995333]), array([0.46427333, 0.52914333]), array([0.46724333, 0.5301    ]), arra
y([0.47424   , 0.53494667]), array([0.47417   , 0.53720333]), array([0.48258, 0.54503]), a
rray([0.48275   , 0.54547667]), array([0.48106   , 0.54883667]), array([0.48796   , 0.5549
5667]), array([0.49028   , 0.55863667]), array([0.49163667, 0.56223667]), array([0.4937366
7, 0.56396333]), array([0.49712667, 0.56417667]), array([0.50013333, 0.57111667]), array
([0.50452   , 0.57405333]), array([0.50289667, 0.57434667]), array([0.50782   , 0.5795866
7]), array([0.50616333, 0.58201333]), array([0.5094    , 0.58497333]), array([0.51244, 0.5
8862])]
(50, 5)
[array([11.77555305, 18.5600672 ]), array([13.01779516, 19.9813138 ]), array([ 6.7817296 ,
23.48339291]), array([ 9.52235123, 20.69704631]), array([12.29861101, 21.44071442]), array
([16.93128977, 23.86236274]), array([12.56446317, 20.53545795]), array([ 4.63910295, 20.23
784464]), array([ 5.84800507, 24.77521488]), array([ 5.88217176, 23.6471406 ]), array([11.
9554559 , 25.76112685]), array([ 8.19074873, 16.51007985]), array([ 9.98623553, 17.2856093
8]), array([ 7.42479886, 14.38836517]), array([11.03956453, 18.992444  ]), array([ 7.72327
```

781, 11.33407037]), array([10.65603216, 14.01442092]), array([5.40533835, 14.68919547]), array([7.83880028, 17.41823645]), array([8.94838967, 16.27600097]), array([9.39304351, 18.31338607]), array([9.58662728, 16.68847921]), array([9.67446338, 16.21675876]), array([5.18497432, 11.06902889]), array([7.5754138 , 8.79473407]), array([6.08856635, 11.0564 0506]), array([6.90642267, 11.50535059]), array([8.77161907, 13.94913654]), array([8.45 697441, 14.26247974]), array([5.27917268, 11.94745581]), array([8.75408429, 13.8002468 4]), array([5.05602786, 9.36178788]), array([6.47810662, 10.88069888]), array([6.5568618 , 10.66293145]), array([6.61841081, 10.62830679]), array([6.93961942, 9.99548105]), array([5.39837445, 10.63848824]), array([6.75871944, 10.43101743]), array([7.13900022, 10.49 594751]), array([4.95542232, 11.78668832]), array([3.37446859, 7.61888744]), array([5.169 5838 , 7.61915332]), array([6.15261444, 8.36528943]), array([5.70397032, 6.98418853]), array([5.24589839, 7.57765319]), array([5.94610068, 8.2658313]), array([5.18151575, 7.618450 62]), array([4.04602292, 7.38557005]), array([4.11233916, 7.54937727]), array([3.07887288, 4.96481746])]





Q10b: Analysis

What do you see? Does the progress of the dynamic mutation rate track with what you expect given the fixed mutation rates? Why or why not? Talk especially about what happens near the end of search, realtive to what you might expect from that same time period in the case with a fixed mutation rate of `0.1` (feel free to run that experiment if you want, or just speculate based on those that you have run).

I see that the diversity drops off significantly as time moves forward. This seems to track well with the previous experiment where higher mutation rates had higher diversity. As mutation rate trends downward, so does the diversity of the population. The accuracy did not reach an extremely high level in this case. This is probably due to it not finding a great optima to climb toward and diversity then falling off preventing it from moving too far from the one it is on. It this was a fixed mutation rate, we would expect diversity to remain relatively consisten across the entire generational period whereas this sees a steeper decline.

Q11: Future Work

We've just begun to scratch the surface here. What other experiments would be intersting to run? What combinations of parameter interactions would be interesting? What other approaches to dynamic/adaptive learning rates would be fun to implement? Could you incorporate information about diversity in informing a dynamic learning rate -- what would that look like?

I think having something similar to multiple species (islands) evolving and being selected within their own niches would be extremely interesting to play around with. I think the interactions between those islands and within them would be fun to experiment on. It would be a way of maintaining diversity while also moving toward better and better solutions over time. We could incoporate a dynamic learning rate into something of that nature, but I might want to do it based on something other than generational time. Maybe we could do it based on each niches' fitness values or even the diversity among different niches.

Congratulations, you made it to the end!

Nice work -- and hopefully you're starting to get the hang of these!

Please save this file as a .ipynb, and also download it as a .pdf, uploading **both** to blackboard to complete this assignment.

For your submission, please make sure that you have renamed this file (and that the resulting pdf follows suit) to replce `[netid]` with your UVM netid. This will greatly simplify our grading pipeline, and make sure that you receive credit for your work.

Academic Integrity Attribution

During this assignment I collaborated with:

Alican for chart comparison

In [ ]:

```

```