Part 1

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import math
import numpy
import random
import numpy as np
```

Generation Methods

In [ ]:

```
def generate_random_bidlist(num_bidders, num_rounds):
    bid_list = []
    for i in range(num_rounds):
        bidder_bids = [random.random() for i in range(num_bidders)]
        bid_list.append(bidder_bids)
    return bid_list

def generate_quadratic_bidlist(num_bidders, num_rounds):
    bid_list = []
    for i in range(num_rounds):
        bidder_bids = [math.sqrt(random.random()) for i in range(num_bidders)]
        bid_list.append(bidder_bids)
    return bid_list

def generate_exponential_bids(num_bidders, num_rounds):
    bids_uncapped = numpy.random.exponential(scale=1.0, size=num_bidders)
    for index in range(num_bidders):
        if bids_uncapped[index] > 10:
            bids_uncapped[index] = 10
    return bids_uncapped.tolist()

def generate_exponential_bidlist(num_bidders, num_rounds):
    return [generate_exponential_bids(num_bidders, num_rounds) for i in range(num_rounds)]

def generate_linear_discretization(val, epsilon):
    action_list = []
    k = val / epsilon

    #calculate number of incremented bids
    num_integ = math.ceil(k)

    #value of each linear increment
    increm = k / num_integ

    bid = val
    j = 0
    while bid >= 0:
        action_list.append(bid)
        j = j + 1
        bid = val - (epsilon * (increm * j))
    return action_list

def generate_geometric_discretization(val, epsilon):
    action_list = []
    k = 1/math.e * numpy.log(val)
    bid = val
    j = 0
    while bid >= 0:
        if j == 0:
            #add both the value bid and the bid of value - 1 according to formula
            action_list.append(bid)
            action_list.append(val - pow((1 + epsilon), j))
        else:
            if bid not in action_list:
                action_list.append(bid)
                bid = val - pow((1 + epsilon), j)
            else:
                bid = val - pow((1 + epsilon), j)
        j = j + 1

    #add a bid of 0 if not already done so
    if 0 not in action_list:
        action_list.append(0)


    return action_list

print(generate_linear_discretization(1, .05))

print(generate_geometric_discretization(1, 0.05))
```

Algorithm Classes

In [ ]:

```
class ExponentialWeights:

    def __init__(self, epsilon, num_actions=2):
        self.weights_vector = [1 for i in range(num_actions)]
        self.totals_by_round = []
        self.payoffs_by_round = []
        self.choices_by_round = []
        self.actions_list = [i for i in range(num_actions)]
```

```python
        self.epsilon = epsilon
        self.num_actions = num_actions

    def reset_instance(self, num_actions=2):
        self.weights_vector = [1 for i in range(num_actions)]
        self.totals_by_round = []
        self.payoffs_by_round = []
        self.choices_by_round = []
        self.actions_list = [i for i in range(num_actions)]
        self.num_actions = num_actions


    def choose_action(self, max_payoff):
        # find weights
        current_weights = [None for i in range(self.num_actions)]
        for action in range(self.num_actions):
            if self.totals_by_round == []:
                V_last = 0
            else:
                V_last = self.totals_by_round[-1][action]
            exp = V_last / max_payoff
            current_weights[action] = pow(1 + self.epsilon, exp)
        # randomly select from actions using weights as probabilities
        selected_action = random.choices(self.actions_list, weights=current_weights, k=1)[0]
        self.choices_by_round.append(selected_action)
        self.weights_vector.append(current_weights)
        #print('current weights', current_weights)
        return selected_action

    def process_payoff(self, selected_payoff, payoff_list):
        # add new payoffs to totals, add payoff choice this round to payoffs matrix
        self.payoffs_by_round.append(selected_payoff)
        if self.totals_by_round == []:
            self.totals_by_round.append([payoff_list[i] for i in range(self.num_actions)])
        else:
            last_round_totals = self.totals_by_round[-1]
            self.totals_by_round.append([last_round_totals[i] + payoff_list[i] for i in range(self.num_actions)])


    #NOTE: totals_by_round[-1] at the end of the simulation will help find 'OPT'
```

In [ ]:

```python
class AuctionCentricEW:

    def __init__(self, epsilon, num_actions=2):
        self.weights_vector = [1 for i in range(num_actions)]
        self.totals_by_round = []
        self.payoffs_by_round = []
        self.choices_by_round = []
        self.never_no_payoff = [True for i in range(num_actions)]
        self.actions_list = [i for i in range(num_actions)]
        self.epsilon = epsilon
        self.num_actions = num_actions

    def reset_instance(self, num_actions=2):
        self.weights_vector = [1 for i in range(num_actions)]
        self.totals_by_round = []
        self.payoffs_by_round = []
        self.choices_by_round = []
        self.never_no_payoff = [True for i in range(num_actions)]
        self.actions_list = [i for i in range(num_actions)]
        self.num_actions = num_actions


    def choose_action(self, max_payoff):
        # find weights
        current_weights = [None for i in range(self.num_actions)]
        for action in range(self.num_actions):
            if self.totals_by_round == []:
                V_last = 0
            else:
                V_last = self.totals_by_round[-1][action]
            exp = V_last / max_payoff
            current_weights[action] = pow(1 + self.epsilon, exp)
        # randomly select from actions using weights as probabilities
        selected_action = random.choices(self.actions_list, weights=current_weights, k=1)[0]

        # if selected action has never been 0, instead pick highest weight 'never_no_payoff'
        if self.never_no_payoff[selected_action]:
            max_weight = current_weights[selected_action]
            max_action = selected_action
            for action in range(self.num_actions):
                if self.never_no_payoff[action] and current_weights[action] > max_weight:
                    max_weight = current_weights[action]
                    max_action = action
            selected_action = max_action

        self.choices_by_round.append(selected_action)
        self.weights_vector.append(current_weights)
        #print('current weights', current_weights)
        return selected_action

    def process_payoff(self, selected_payoff, payoff_list):
        # add new payoffs to totals, track payoff this round
        self.payoffs_by_round.append(selected_payoff)
        if self.totals_by_round == []:
            self.totals_by_round.append([payoff_list[i] for i in range(self.num_actions)])
        else:
            last_round_totals = self.totals_by_round[-1]
            self.totals_by_round.append([last_round_totals[i] + payoff_list[i] for i in range(self.num_actions)])
```

```
        # update never_been_zero actions
        for index in range(len(payoff_list)):
            payoff = payoff_list[index]
            if payoff == 0 and self.never_no_payoff[index]:
                self.never_no_payoff[index] = False
```

In [ ]:

```
class FTL:

    def __init__(self, num_actions=2):
        self.totals_by_round = []
        self.payoffs_by_round = []
        self.choices_by_round = []
        self.actions_list = [i for i in range(num_actions)]
        self.num_actions = num_actions

    def reset_instance(self, num_actions=2):
        self.totals_by_round = []
        self.payoffs_by_round = []
        self.choices_by_round = []
        self.actions_list = [i for i in range(num_actions)]
        self.num_actions = num_actions

    def choose_action(self, max_payoff):
        # randomly select from actions using highest total payoff so far
        if self.totals_by_round != []:
            selected_action = self.totals_by_round[-1].index(max(self.totals_by_round[-1]))
            self.choices_by_round.append(selected_action)
            return selected_action
        else:
            selected_action = random.randrange(0, self.num_actions)
            return selected_action

    def process_payoff(self, selected_payoff, payoff_list):
        # add new payoffs to totals, add payoff choice this round to payoffs matrix
        self.payoffs_by_round.append(selected_payoff)
        if self.totals_by_round == []:
            self.totals_by_round.append([payoff_list[i] for i in range(self.num_actions)])
        else:
            last_round_totals = self.totals_by_round[-1]
            self.totals_by_round.append([last_round_totals[i] + payoff_list[i] for i in range(self.num_actions)])


    #NOTE: totals_by_round[-1] at the end of the simulation will help find 'OPT'
```

Auction Simulator

In [ ]:

```
# helpers to find regret of an algorithm
def sum_to_round_i(alg_payoffs, current_round):
    total = 0
    for i in range(current_round):
        total += alg_payoffs[i]
    return total

def individual_regrets(alg_payoffs, round_totals):
    final_payoffs = round_totals[-1]
    opt_action = final_payoffs.index(max(final_payoffs))
    individual_regrets = [0 for i in range(len(alg_payoffs))]
    for round_x in range((len(alg_payoffs))):
        individual_regrets[round_x] = (round_totals[round_x][opt_action] - sum_to_round_i(alg_payoffs, round_x)) / (round_x + 1)
    return individual_regrets

#returns winning bid/revenue
def find_payoff(r_price, bid_list, num_items=None):
    if num_items == None:
        sorted_bids = sorted(bid_list, reverse=True)
        max_bid = sorted_bids[0]
        second_bid = sorted_bids[1]
        if r_price > second_bid and r_price <= max_bid:
            return r_price
        elif r_price > second_bid and r_price > max_bid:
            return 0
        else:
            return second_bid
    else:
        sorted_bids = sorted(bid_list, reverse=True)
        first_bid = sorted_bids[num_items - 2]
        second_bid = sorted_bids[num_items - 1]
        if r_price > second_bid and r_price <= first_bid:
            return r_price
        elif r_price > second_bid and r_price > first_bid:
            return 0
        else:
            return second_bid

def auction_simulator(alg, bid_lists, num_rounds, max_bid, price_discretization, num_items=None):
    num_actions = len(price_discretization)
    for bid_list in bid_lists:
        # have the algorithm select a bid
        alg_action = alg.choose_action(max_bid)
        alg_price = price_discretization[alg_action]

        # calculate payoff list for each reserve price on the discretization
        payoff_list = []
        for reserve_price in price_discretization:
            payoff_list.append(find_payoff(reserve_price, bid_list, num_items))
```

```
        alg_payoff = payoff_list[alg_action]
        alg.process_payoff(alg_payoff, payoff_list)

    # calculate regrets and payoffs
    alg_regrets = individual_regrets(alg.payoffs_by_round, alg.totals_by_round)
    alg_payoffs = alg.payoffs_by_round

    return alg_regrets, alg_payoffs

#bid_lists = [[0, 1], [0, .75], [0, .5], [0, .25]]
#alg = ExponentialWeights(0.0, len(generate_linear_discretization(0, 1, 0.25)))
#alg_regrets, alg_payoffs = auction_simulator(alg, bid_lists, 4, 1, generate_linear_discretization(0, 1, 0.25))
#print(alg_regrets)
#print(alg_payoffs)
#print(alg.weights_vector[-1])
#print(alg.choices_by_round)
#print(alg.totals_by_round)
```

Auction Monte Carlo Trials

In [ ]:

```
## Auction Monte Carlo Trials
def auction_trial(alg, auction_list, num_rounds, max_bid, price_discretization, num_items=None):
    alg_avg_regret_per_round = None
    alg_avg_payoff_per_round = None

    for auction in auction_list:
        # find which trial number we are on
        n = auction_list.index(auction)

        # run matchup and find regret lists
        new_alg_regrets, new_alg_payoffs = auction_simulator(alg, auction, num_rounds, max_bid, price_discretization, num_items)

        # update average regrets
        if alg_avg_regret_per_round == None:
            alg_avg_regret_per_round = new_alg_regrets
        else:
            for i in range(len(alg_avg_regret_per_round)):
                alg_avg_regret_per_round[i] = ((n * alg_avg_regret_per_round[i]) + new_alg_regrets[i]) / (n + 1)

        # update average payoffs
        if alg_avg_payoff_per_round == None:
            alg_avg_payoff_per_round = new_alg_payoffs
        else:
            for i in range(len(alg_avg_regret_per_round)):
                alg_avg_payoff_per_round[i] = ((n * alg_avg_payoff_per_round[i]) + new_alg_payoffs[i]) / (n + 1)

        # reset alg internally stored values
        alg.reset_instance(num_actions=len(price_discretization))
        #print('final weights', alg.weights_vector)
    return alg_avg_regret_per_round, alg_avg_payoff_per_round

#bid_lists = [[0, 1], [0, .75], [0, .5], [0, .25]]
#alg = ExponentialWeights(1.0, len(generate_linear_discretization(0, 1, 0.25)))
#alg_regrets, alg_payoffs = auction_trial(alg, [bid_lists, bid_lists], 4, 1, generate_linear_discretization(0, 1, 0.25))
#print(alg_regrets)
#print(alg_payoffs)
```

Visualization of Regrets

In [ ]:

```
def visualize_rounds(alg_regrets, rounds, lr, plot_title, alg_name, trial_type, y_label):

    file_name = plot_title + '.png'

    x = numpy.array(list(range(0, rounds)))
    y_1 = numpy.array(alg_regrets)
    plt.plot(x, y_1, label='{alg_name}, learning rate = {lr}'.format(alg_name=alg_name, lr = lr), linewidth=1)
    plt.xlabel("Round")
    plt.ylabel(y_label)
    plt.title(plot_title)
    plt.legend(loc='best', prop={'size': 7})

    plt.savefig(file_name)

    plt.show()
```

Table Generation

In [ ]:

```
def generate_table(data, columns, rows, title):

    rcolors = plt.cm.BuPu(np.full(len(rows), 0.1))
    ccolors = plt.cm.BuPu(np.full(len(columns), 0.1))

    # Get some pastel shades for the colors
    #colors = plt.cm.BuPu(np.linspace(0, 0.5, len(rows)))
    #n_rows = len(data)

    the_table = plt.table(cellText=data,
                          rowLabels=rows,
                          rowColours=rcolors,
                            rowLoc='right',
                          colLabels=columns,
                          colColours = ccolors,
                          loc='center')
```

```
    the_table.scale(1, 1.5)

    #plt.title(title)
    plt.box(on=None)
    ax = plt.gca()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    fig = plt.gcf()

    file_name = title + '.png'
    plt.savefig(file_name,
                bbox_inches='tight',
                dpi=150
                )

    plt.show()
```

Trials on Different Bidder Generation Methods, Discretizations, Number of Bidders

In [ ]:

```
# PARAMETERS
NUM_TRIALS = 1000
NUM_ROUNDS = 500

# 2 bidder random values auction trial Epsilon = 1
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg1_regrets, alg1_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg1_regrets[-1])
print(alg1_payoffs[-1])
print('avg p1', sum(alg1_payoffs) / len(alg1_payoffs))
print('avg r1', sum(alg1_regrets) / len(alg1_regrets))

#visualize_rounds(alg1_regrets, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Randomized Auction: Regrets", "EW", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg1_payoffs, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Randomized Auction: Payoffs", "EW", "Randomized Auction", "Payoff per Round")

# 2 bidder random values auction trial with Epsilon = 0
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg2_regrets, alg2_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg2_regrets[-1])
print(alg2_payoffs[-1])
print('avg p2', sum(alg2_payoffs) / len(alg2_payoffs))
print('avg r2', sum(alg2_regrets) / len(alg2_regrets))

#visualize_rounds(alg2_regrets, NUM_ROUNDS, .5, "Epsilon 0.1 on Randomized Auction: Regrets", "EW", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg2_payoffs, NUM_ROUNDS, .5, "Epsilon 0.1 on Randomized Auction: Payoffs", "EW", "Randomized Auction", "Payoff per Round")

# 2 bidder random values auction trial Epsilon = 1, AuctionCentricEW
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = AuctionCentricEW(1.0, len(price_discretization))
alg3_regrets, alg3_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg3_regrets[-1])
print(alg3_payoffs[-1])
print('avg p3', sum(alg3_payoffs) / len(alg3_payoffs))
print('avg r3', sum(alg3_regrets) / len(alg3_regrets))

#visualize_rounds(alg3_regrets, NUM_ROUNDS, .5, "AuctionCentricEW Epsilon 1.0 on Randomized Auction: Regrets", "EW", "Randomized Auction", "Regret per Round
#visualize_rounds(alg3_payoffs, NUM_ROUNDS, .5, "AuctionCentricEW Epsilon 1.0 on Randomized Auction: Payoffs", "EW", "Randomized Auction", "Payoff per Round

# 2 bidder random values auction trial Epsilon = infinity
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = FTL(len(price_discretization))
alg4_regrets, alg4_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg4_regrets[-1])
print(alg4_payoffs[-1])
print('avg p4', sum(alg4_payoffs) / len(alg4_payoffs))
print('avg r4', sum(alg4_regrets) / len(alg4_regrets))

#visualize_rounds(alg4_regrets, NUM_ROUNDS, 0, "FTL on Randomized Auction: Regrets", "FTL", "FTL on Randomized Auction", "Regret per Round")
#visualize_rounds(alg4_payoffs, NUM_ROUNDS, 0, "FTL on Randomized Auction: Payoffs", "FTL", "FTL on Randomized Auction", "Payoff per Round")
```

```python
# 2 bidder random values auction trial Epsilon = 0
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(0, len(price_discretization))
alg5_regrets, alg5_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg5_regrets[-1])
print(alg5_payoffs[-1])
print('avg p5', sum(alg5_payoffs) / len(alg5_payoffs))
print('avg r5', sum(alg5_regrets) / len(alg5_regrets))

#visualize_rounds(alg5_regrets, NUM_ROUNDS, 0, "Random guessing on Randomized Auction: Regrets", "Random Guessing", "Randomized Auction", "Regret per Round"
#visualize_rounds(alg5_payoffs, NUM_ROUNDS, 0, "Random guessing on Randomized Auction: Payoffs", "Random Guessing", "Randomized Auction", "Payoff per Round"


file_name = 'All Algorithms Regret Comparison, 2 Bidders 1 Item' + '.png'

x_all = numpy.array(list(range(0, NUM_ROUNDS)))
y_1 = numpy.array(alg1_regrets)
y_2 = numpy.array(alg2_regrets)
y_3 = numpy.array(alg3_regrets)
y_4 = numpy.array(alg4_regrets)
y_5 = numpy.array(alg5_regrets)
plt.plot(x_all, y_1, label='EW Epsilon = 1.0'.format(alg_name="EW Epsilon = 1.0"), linewidth=1)
plt.plot(x_all, y_2, label='EW Epsilon = 0.1'.format(alg_name="EW Epsilon = 1.0"), linewidth=1)
plt.plot(x_all, y_3, label='AEW Epsilon = 1.0'.format(alg_name="AEW Epsilon = 1.0"), linewidth=1)
plt.plot(x_all, y_4, label='FTL Epsilon = infinity'.format(alg_name="FTL Epsilon = infinity"), linewidth=1)
plt.plot(x_all, y_5, label='Random Guessing Epsilon = 0'.format(alg_name="Random Guessing Epsilon = 0"), linewidth=1)

plt.xlabel("Round")
plt.ylabel("Average Regret")
plt.title('All Algorithms Regret Comparison, 2 Bidders 1 Item')
plt.legend(loc='best', prop={'size': 7})
plt.savefig(file_name)

plt.show()
```

In [ ]:

```python
## generate table with regrets and payoffs

columns = ['Average Revenue', 'Average Regret']
rows = ['EW Epsilon = 1.0', 'EW Epsilon = 0.1','AEW Epsilon = 1.0', 'FTL Epsilon = infinity',
       'Random Guessing Epsilon = 0']
title = 'Table of Average Regrets & Revenue With Different Learning Algorithms (2 bidders, 1 item)'

data = [[sum(alg1_payoffs) / len(alg1_payoffs), sum(alg1_regrets) / len(alg1_regrets)],
        [sum(alg2_payoffs) / len(alg2_payoffs), sum(alg2_regrets) / len(alg2_regrets)],
        [sum(alg3_payoffs) / len(alg3_payoffs), sum(alg3_regrets) / len(alg3_regrets)],
        [sum(alg4_payoffs) / len(alg4_payoffs), sum(alg4_regrets) / len(alg4_regrets)],
        [sum(alg5_payoffs) / len(alg5_payoffs), sum(alg5_regrets) / len(alg5_regrets)]]

data = [[np.round(float(i), 6) for i in nested] for nested in data]

generate_table(data, columns, rows, title)
```

Trials with different numbers of bidders

In [ ]:

```python
# PARAMETERS
NUM_TRIALS = 1000
NUM_ROUNDS = 500
# 3 bidders EW on random values

num_bidders = 3
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg1_regrets, alg1_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg1_regrets[-1])
print(alg1_payoffs[-1])
print('avg p1', sum(alg1_payoffs) / len(alg1_payoffs))
print('avg r1', sum(alg1_regrets) / len(alg1_regrets))
#visualize_rounds(alg1_regrets, NUM_ROUNDS, 0, "EW on Randomized Auction 3 Bidders: Regrets", "Random Guessing", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg1_payoffs, NUM_ROUNDS, 0, "EW on Randomized Auction 3 Bidders: Payoffs", "Random Guessing", "Randomized Auction", "Payoff per Round")


num_bidders = 5
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg2_regrets, alg2_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg2_regrets[-1])
print(alg2_payoffs[-1])
print('avg p2', sum(alg2_payoffs) / len(alg2_payoffs))
print('avg r2', sum(alg2_regrets) / len(alg2_regrets))
#visualize_rounds(alg2_regrets, NUM_ROUNDS, 0, "EW on Randomized Auction 5 Bidders: Regrets", "Random Guessing", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg2_payoffs, NUM_ROUNDS, 0, "EW on Randomized Auction 5 Bidders: Payoffs", "Random Guessing", "Randomized Auction", "Payoff per Round")
```

```
num_bidders = 20
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg3_regrets, alg3_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg3_regrets[-1])
print(alg3_payoffs[-1])
print('avg p4', sum(alg3_payoffs) / len(alg3_payoffs))
print('avg r4', sum(alg3_regrets) / len(alg3_regrets))
#visualize_rounds(alg3_regrets, NUM_ROUNDS, 0, "EW on Randomized Auction 20 Bidders: Regrets", "Random Guessing", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg3_payoffs, NUM_ROUNDS, 0, "EW on Randomized Auction 20 Bidders: Payoffs", "Random Guessing", "Randomized Auction", "Payoff per Round")


num_bidders = 50
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg4_regrets, alg4_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg4_regrets[-1])
print(alg4_payoffs[-1])
print('avg p4', sum(alg4_payoffs) / len(alg4_payoffs))
print('avg r4', sum(alg4_regrets) / len(alg4_regrets))
#visualize_rounds(alg4_regrets, NUM_ROUNDS, 0, "EW on Randomized Auction 20 Bidders: Regrets", "Random Guessing", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg4_payoffs, NUM_ROUNDS, 0, "EW on Randomized Auction 20 Bidders: Payoffs", "Random Guessing", "Randomized Auction", "Payoff per Round")


file_name = 'EW Epsilon=1.0 Regret Comparison, Varied Bidder Count' + '.png'

x_all = numpy.array(list(range(0, NUM_ROUNDS)))
y_1 = numpy.array(alg1_regrets)
y_2 = numpy.array(alg2_regrets)
y_3 = numpy.array(alg3_regrets)
y_4 = numpy.array(alg4_regrets)
plt.plot(x_all, y_1, label='EW Epsilon = 1.0, 3 bidders'.format(alg_name="3 bidders"), linewidth=1)
plt.plot(x_all, y_2, label='EW Epsilon = 1.0, 5 bidders'.format(alg_name="5 bidders"), linewidth=1)
plt.plot(x_all, y_3, label='EW Epsilon = 1.0, 20 bidders'.format(alg_name="20 bidders"), linewidth=1)
plt.plot(x_all, y_4, label='EW Epsilon = 1.0, 50 bidders'.format(alg_name="50 bidders"), linewidth=1)

plt.xlabel("Round")
plt.ylabel("Average Regret")
plt.title('EW Epsilon=1.0 Regret Comparison, Varied Bidder Count')
plt.legend(loc='best', prop={'size': 7})
plt.savefig(file_name)

plt.show()
```

In [ ]:

```
## generate table with regrets and payoffs

columns = ['Average Revenue', 'Average Regret']
rows = ['EW Epsilon = 1.0, 3 bidders', 'EW Epsilon = 1.0, 5 bidders',
        'EW Epsilon = 1.0, 20 bidders', 'EW Epsilon = 1.0, 50 bidders']
title = 'Table of Average Regrets & Revenue With Varied Bidder Count (EW, Epsilon = 1.0)'

data = [[sum(alg1_payoffs) / len(alg1_payoffs), sum(alg1_regrets) / len(alg1_regrets)],
        [sum(alg2_payoffs) / len(alg2_payoffs), sum(alg2_regrets) / len(alg2_regrets)],
        [sum(alg3_payoffs) / len(alg3_payoffs), sum(alg3_regrets) / len(alg3_regrets)],
        [sum(alg4_payoffs) / len(alg4_payoffs), sum(alg4_regrets) / len(alg4_regrets)]]

data = [[np.round(float(i), 6) for i in nested] for nested in data]

generate_table(data, columns, rows, title)
```

Trials with different numbers of auctioned items

In [ ]:

```
num_bidders = 5
num_items = 3
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg1_regrets, alg1_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization, num_items)
print(alg1_regrets[-1])
print(alg1_payoffs[-1])
print('avg p1', sum(alg1_payoffs) / len(alg1_payoffs))
print('avg r1', sum(alg1_regrets) / len(alg1_regrets))
#visualize_rounds(alg1_regrets, NUM_ROUNDS, 0, "EW on Randomized Auction 5 Bidders 1 Items: Regrets", "Random Guessing", "Randomized Auction", "Regret per R
#visualize_rounds(alg1_payoffs, NUM_ROUNDS, 0, "EW on Randomized Auction 5 Bidders 1 Items: Payoffs", "Random Guessing", "Randomized Auction", "Payoff per R


num_bidders = 5
num_items = 3
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
```

```
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg2_regrets, alg2_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization, num_items)
print(alg2_regrets[-1])
print(alg2_payoffs[-1])
print('avg p2', sum(alg2_payoffs) / len(alg2_payoffs))
print('avg r2', sum(alg2_regrets) / len(alg2_regrets))
#visualize_rounds(alg2_regrets, NUM_ROUNDS, 0, "EW on Randomized Auction 5 Bidders 3 Items: Regrets", "Random Guessing", "Randomized Auction", "Regret per R
#visualize_rounds(alg2_payoffs, NUM_ROUNDS, 0, "EW on Randomized Auction 5 Bidders 3 Items: Payoffs", "Random Guessing", "Randomized Auction", "Payoff per R

num_bidders = 5
num_items = 5
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg3_regrets, alg3_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization, num_items)
print(alg3_regrets[-1])
print(alg3_payoffs[-1])
print('avg p3', sum(alg3_payoffs) / len(alg3_payoffs))
print('avg r3', sum(alg3_regrets) / len(alg3_regrets))
#visualize_rounds(alg3_regrets, NUM_ROUNDS, 0, "EW on Randomized Auction 5 Bidders 5 Items: Regrets", "Random Guessing", "Randomized Auction", "Regret per R
#visualize_rounds(alg3_payoffs, NUM_ROUNDS, 0, "EW on Randomized Auction 5 Bidders 5 Items: Payoffs", "Random Guessing", "Randomized Auction", "Payoff per R

file_name = 'EW Epsilon=1.0 Regret Comparison, Varied Auctioned Items Count' + '.png'

x_all = numpy.array(list(range(0, NUM_ROUNDS)))
y_1 = numpy.array(alg1_regrets)
y_2 = numpy.array(alg2_regrets)
y_3 = numpy.array(alg3_regrets)
plt.plot(x_all, y_1, label='EW Epsilon = 1.0, 5 bidders 1 item'.format(alg_name="1 item"), linewidth=1)
plt.plot(x_all, y_2, label='EW Epsilon = 1.0, 5 bidders 3 items'.format(alg_name="3 items"), linewidth=1)
plt.plot(x_all, y_3, label='EW Epsilon = 1.0, 5 bidders 5 items'.format(alg_name="5 items"), linewidth=1)

plt.xlabel("Round")
plt.ylabel("Average Regret")
plt.title('EW Epsilon=1.0 Regret Comparison, Varied Auctioned Items Count')
plt.legend(loc='best', prop={'size': 7})
plt.savefig(file_name)

plt.show()
```

In [ ]:

```
## generate table with regrets and payoffs

columns = ['Average Revenue', 'Average Regret']
rows = ['EW Epsilon = 1.0, 5 bidders 1 item', 'EW Epsilon = 1.0, 5 bidders 3 items',
        'EW Epsilon = 1.0, 5 bidders 5 items']
title = 'Table of Average Regrets & Revenue With Varied Auction Items (EW, Epsilon = 1.0)'

data = [[sum(alg1_payoffs) / len(alg1_payoffs), sum(alg1_regrets) / len(alg1_regrets)],
        [sum(alg2_payoffs) / len(alg2_payoffs), sum(alg2_regrets) / len(alg2_regrets)],
        [sum(alg3_payoffs) / len(alg3_payoffs), sum(alg3_regrets) / len(alg3_regrets)]]

data = [[np.round(float(i), 6) for i in nested] for nested in data]

generate_table(data, columns, rows, title)
```

Trials on different distributions

In [ ]:

```
# 2 bidder random values auction trial Epsilon = 1
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_random_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg1_regrets, alg1_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg1_regrets[-1])
print(alg1_payoffs[-1])
print('p1', sum(alg1_payoffs) / len(alg1_payoffs))
print('r1', sum(alg1_regrets) / len(alg1_regrets))
#visualize_rounds(alg1_regrets, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Randomized Auction: Regrets", "EW", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg1_payoffs, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Randomized Auction: Payoffs", "EW", "Randomized Auction", "Payoff per Round")

# 2 bidder quadratic values auction trial Epsilon = 1
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_quadratic_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg2_regrets, alg2_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg2_regrets[-1])
print(alg2_payoffs[-1])
print('p2', sum(alg2_payoffs) / len(alg2_payoffs))
print('r2', sum(alg2_regrets) / len(alg2_regrets))
#visualize_rounds(alg2_regrets, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Quadratic Auction: Regrets", "EW", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg2_payoffs, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Quadratic Auction: Payoffs", "EW", "Randomized Auction", "Payoff per Round")
```

```
# 2 bidder exponential values auction trial Epsilon = 1
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction_list.append(generate_exponential_bidlist(num_bidders, NUM_ROUNDS))
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg3_regrets, alg3_payoffs = auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg3_regrets[-1])
print(alg3_payoffs[-1])
print('p3', sum(alg3_payoffs) / len(alg3_payoffs))
print('r', sum(alg3_regrets) / len(alg3_regrets))
#visualize_rounds(alg3_regrets, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Exponential Auction: Regrets", "EW", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg3_payoffs, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Exponential Auction: Payoffs", "EW", "Randomized Auction", "Payoff per Round")


file_name = 'EW Epsilon=1.0 Regret Comparison, Varied Bidder Value Distributions' + '.png'

x_all = numpy.array(list(range(0, NUM_ROUNDS)))
y_1 = numpy.array(alg1_regrets)
y_2 = numpy.array(alg2_regrets)
y_3 = numpy.array(alg3_regrets)
plt.plot(x_all, y_1, label='EW Epsilon = 1.0, 2 Bidders Random Distribution'.format(alg_name="2 Random Bidders"), linewidth=1)
plt.plot(x_all, y_2, label='EW Epsilon = 1.0, 2 Bidders Quadratic Distribution'.format(alg_name="2 Quadratic Bidders"), linewidth=1)
plt.plot(x_all, y_3, label='EW Epsilon = 1.0, 2 Bidders Exponential Distribution'.format(alg_name="2 Exponential Bidders"), linewidth=1)

plt.xlabel("Round")
plt.ylabel("Average Regret")
plt.title('EW Epsilon=1.0 Regret Comparison, Varied Bidder Value Distributions')
plt.legend(loc='best', prop={'size': 7})
plt.savefig(file_name)

plt.show()
```

In [ ]:

```
## generate table with regrets and payoffs

columns = ['Average Revenue', 'Average Regret']
rows = ['2 Bidders Random Distribution', '2 Bidders Quadratic Distribution',
        '2 Bidders Exponential Distribution']
title = 'Table of Average Regrets & Revenue With Varied Bidder Distributions (EW, Epsilon = 1.0)'

data = [[sum(alg1_payoffs) / len(alg1_payoffs), sum(alg1_regrets) / len(alg1_regrets)],
        [sum(alg2_payoffs) / len(alg2_payoffs), sum(alg2_regrets) / len(alg2_regrets)],
        [sum(alg3_payoffs) / len(alg3_payoffs), sum(alg3_regrets) / len(alg3_regrets)]]

data = [[np.round(float(i), 6) for i in nested] for nested in data]

generate_table(data, columns, rows, title)
```

Part 2 Selling Introductions

Part 2 Simulator

In [ ]:

```
def find_intro_payoff(price, employee_list, employer_list):
    total_payoff = 0
    for index in range(len(employee_list)):
        employee_val = employee_list[index]
        employer_val = employer_list[index]
        if employer_val + employee_val >= price:
            total_payoff += price
        else:
            total_payoff += 0
    return total_payoff

def introduction_auction_simulator(alg, employee_lists, employer_lists, num_rounds, max_bid, price_discretization, num_items=None):
    num_actions = len(price_discretization)
    for index in range(len(employee_lists)):
        employee_list = employee_lists[index]
        employer_list = employer_lists[index]
        # have the algorithm select a bid
        alg_action = alg.choose_action(max_bid)
        alg_price = price_discretization[alg_action]

        # calculate payoff list for each reserve price on the discretization
        payoff_list = []
        for introduction_price in price_discretization:
            payoff_list.append(find_intro_payoff(introduction_price, employee_list, employer_list))
        alg_payoff = payoff_list[alg_action]
        alg.process_payoff(alg_payoff, payoff_list)

    # calculate regrets and payoffs
    alg_regrets = individual_regrets(alg.payoffs_by_round, alg.totals_by_round)
    alg_payoffs = alg.payoffs_by_round

    return alg_regrets, alg_payoffs
```

Introduction Monte Carlo Trials

In [ ]:

```
## Auction Monte Carlo Trials
def introduction_auction_trial(alg, auction_list, num_rounds, max_bid, price_discretization, num_items=None):
    alg_avg_regret_per_round = None
```

```
    alg_avg_payoff_per_round = None

    for auction in auction_list:
        # find which trial number we are on
        n = auction_list.index(auction)

        # get employee and employer meet value lists
        employee_lists = auction[0]
        employer_lists = auction[1]

        # run matchup and find regret lists
        new_alg_regrets, new_alg_payoffs = introduction_auction_simulator(alg, employee_lists, employer_lists, num_rounds, max_bid, price_discretization, nu

        # update average regrets
        if alg_avg_regret_per_round == None:
            alg_avg_regret_per_round = new_alg_regrets
        else:
            for i in range(len(alg_avg_regret_per_round)):
                alg_avg_regret_per_round[i] = ((n * alg_avg_regret_per_round[i]) + new_alg_regrets[i]) / (n + 1)

        # update average payoffs
        if alg_avg_payoff_per_round == None:
            alg_avg_payoff_per_round = new_alg_payoffs
        else:
            for i in range(len(alg_avg_regret_per_round)):
                alg_avg_payoff_per_round[i] = ((n * alg_avg_payoff_per_round[i]) + new_alg_payoffs[i]) / (n + 1)

        # reset alg internally stored values
        alg.reset_instance(num_actions=len(price_discretization))
        #print('final weights', alg.weights_vector)
    return alg_avg_regret_per_round, alg_avg_payoff_per_round
```

Introduction Selling Tests

Different Algorithms

In [ ]:

```
# PARAMETERS
NUM_TRIALS = 1000
NUM_ROUNDS = 500
# 1 employees+employer pairs EW Epsilon = 1.0 on random values
num_pairs = 1

auction_list = []
min_payoff, max_payoff = 0, 1
# perform 1000 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg1_regrets, alg1_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg1_regrets[-1])
print(alg1_payoffs[-1])
print('avg p1', sum(alg1_payoffs) / len(alg1_payoffs))
print('avg r1', sum(alg1_regrets) / len(alg1_regrets))
#visualize_rounds(alg1_regrets, NUM_ROUNDS, 0, "EW on Introduction Offers 1 Employee+Employer Pair: Regrets", "EW Epsilon=1.0", "Randomized Auction", "Regre
#visualize_rounds(alg1_payoffs, NUM_ROUNDS, 0, "EW on Introduction Offers 1 Employee+Employer Pair: Payoffs", "EW Epsilon=1.0", "Randomized Auction", "Payof

# 1 employees+employer pairs EW Epsilon = 1.0 on random values
num_pairs = 1

auction_list = []
min_payoff, max_payoff = 0, 1
# perform 1000 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(0.1, len(price_discretization))
alg2_regrets, alg2_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg2_regrets[-1])
print(alg2_payoffs[-1])
print('avg p1', sum(alg2_payoffs) / len(alg2_payoffs))
print('avg r1', sum(alg2_regrets) / len(alg2_regrets))
#visualize_rounds(alg2_regrets, NUM_ROUNDS, 0, "EW on Introduction Offers 1 Employee+Employer Pair: Regrets", "EW Epsilon=0.1", "Randomized Auction", "Regre
#visualize_rounds(alg2_payoffs, NUM_ROUNDS, 0, "EW on Introduction Offers 1 Employee+Employer Pair: Payoffs", "EW Epsilon=0.1", "Randomized Auction", "Payof

# 1 employees+employer pairs FTL on random values
num_pairs = 1

auction_list = []
min_payoff, max_payoff = 0, 1
# perform 1000 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = FTL(len(price_discretization))
alg3_regrets, alg3_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg3_regrets[-1])
print(alg3_payoffs[-1])
```

```
print('avg p1', sum(alg3_payoffs) / len(alg3_payoffs))
print('avg r1', sum(alg3_regrets) / len(alg3_regrets))
#visualize_rounds(alg3_regrets, NUM_ROUNDS, 0, "FTL on Introduction Offers 1 Employee+Employer Pair: Regrets", "FTL", "Randomized Auction", "Regret per Roun
#visualize_rounds(alg3_payoffs, NUM_ROUNDS, 0, "FTL on Introduction Offers 1 Employee+Employer Pair: Payoffs", "FTL", "Randomized Auction", "Payoff per Roun

# 1 employees+employer pairs Random Guessing on random values
num_pairs = 1

auction_list = []
min_payoff, max_payoff = 0, 1
# perform 1000 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(0.0, len(price_discretization))
alg4_regrets, alg4_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg4_regrets[-1])
print(alg4_payoffs[-1])
print('avg p1', sum(alg4_payoffs) / len(alg4_payoffs))
print('avg r1', sum(alg4_regrets) / len(alg4_regrets))
#visualize_rounds(alg4_regrets, NUM_ROUNDS, 0, "RG on Introduction Offers 1 Employee+Employer Pair: Regrets", "RG", "Randomized Auction", "Regret per Round"
#visualize_rounds(alg4_payoffs, NUM_ROUNDS, 0, "RG on Introduction Offers 1 Employee+Employer Pair: Payoffs", "RG", "Randomized Auction", "Payoff per Round"

auction_list = []
min_payoff, max_payoff = 0, 1
# perform 1000 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = FTL(len(price_discretization))
alg5_regrets, alg5_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg5_regrets[-1])
print(alg5_payoffs[-1])
print('avg p1', sum(alg5_payoffs) / len(alg5_payoffs))
print('avg r1', sum(alg5_regrets) / len(alg5_regrets))
#visualize_rounds(alg4_regrets, NUM_ROUNDS, 0, "RG on Introduction Offers 1 Employee+Employer Pair: Regrets", "RG", "Randomized Auction", "Regret per Round"
#visualize_rounds(alg4_payoffs, NUM_ROUNDS, 0, "RG on Introduction Offers 1 Employee+Employer Pair: Payoffs", "RG", "Randomized Auction", "Payoff per Round"



file_name = 'All Algorithms Regret Comparison, 1 Employee+Employer Pair' + '.png'

x_all = numpy.array(list(range(0, NUM_ROUNDS)))
y_1 = numpy.array(alg1_regrets)
y_2 = numpy.array(alg2_regrets)
y_3 = numpy.array(alg3_regrets)
y_4 = numpy.array(alg4_regrets)
y_5 = numpy.array(alg5_regrets)
plt.plot(x_all, y_1, label='EW Epsilon = 1.0'.format(alg_name="EW Epsilon = 1.0"), linewidth=1)
plt.plot(x_all, y_2, label='EW Epsilon = 0.1'.format(alg_name="EW Epsilon = 1.0"), linewidth=1)
plt.plot(x_all, y_3, label='AEW Epsilon = 1.0'.format(alg_name="AEW Epsilon = 1.0"), linewidth=1)
plt.plot(x_all, y_4, label='Random Guessing Epsilon = 0'.format(alg_name="Random Guessing Epsilon = 0"), linewidth=1)
plt.plot(x_all, y_5, label='FTL Epsilon = infinity'.format(alg_name="FTL Epsilon = infinity"), linewidth=1)

plt.xlabel("Round")
plt.ylabel("Average Regret")
plt.title('All Algorithms Regret Comparison, 1 Employee+Employer Pair')
plt.legend(loc='best', prop={'size': 7})
plt.savefig(file_name)
```

In [ ]:

```
## generate table with regrets and payoffs

columns = ['Average Revenue', 'Average Regret']
rows = ['EW Epsilon = 1.0', 'EW Epsilon = 0.1','AEW Epsilon = 1.0', 'Random Guessing Epsilon = 0',
        'FTL Epsilon = infinity']
title = 'Table of Average Regrets & Revenue With Different Learning Algorithms (1 Employee+Employer Pair)'

data = [[sum(alg1_payoffs) / len(alg1_payoffs), sum(alg1_regrets) / len(alg1_regrets)],
        [sum(alg2_payoffs) / len(alg2_payoffs), sum(alg2_regrets) / len(alg2_regrets)],
        [sum(alg3_payoffs) / len(alg3_payoffs), sum(alg3_regrets) / len(alg3_regrets)],
        [sum(alg4_payoffs) / len(alg4_payoffs), sum(alg4_regrets) / len(alg4_regrets)],
        [sum(alg5_payoffs) / len(alg5_payoffs), sum(alg5_regrets) / len(alg5_regrets)]]

data = [[np.round(float(i), 6) for i in nested] for nested in data]

generate_table(data, columns, rows, title)
```

Different numbers of employee+employer pairs

In [ ]:

```
# PARAMETERS
NUM_TRIALS = 1000
NUM_ROUNDS = 100
# 3 bidders EW on random values

num_pairs = 3
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
```

```
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg1_regrets, alg1_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg1_regrets[-1])
print(alg1_payoffs[-1])
print('avg p1', sum(alg1_payoffs) / len(alg1_payoffs))
print('avg r1', sum(alg1_regrets) / len(alg1_regrets))
#visualize_rounds(alg1_regrets, NUM_ROUNDS, 0, "EW Introduction Selling, 3 Employee+Employer Pairs: Regrets", "Random Guessing", "Randomized Auction", "Regr
#visualize_rounds(alg1_payoffs, NUM_ROUNDS, 0, "EW Introduction Selling, 3 Employee+Employer Pairs: Payoffs", "Random Guessing", "Randomized Auction", "Payo


num_pairs = 5
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg2_regrets, alg2_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg2_regrets[-1])
print(alg2_payoffs[-1])
print('avg p2', sum(alg2_payoffs) / len(alg2_payoffs))
print('avg r2', sum(alg2_regrets) / len(alg2_regrets))
#visualize_rounds(alg2_regrets, NUM_ROUNDS, 0, "EW Introduction Selling, 5 Employee+Employer Pairs: Regrets", "Random Guessing", "Randomized Auction", "Regr
#visualize_rounds(alg2_payoffs, NUM_ROUNDS, 0, "EW Introduction Selling, 5 Employee+Employer Pairs: Payoffs", "Random Guessing", "Randomized Auction", "Payo

num_pairs = 10
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg3_regrets, alg3_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg3_regrets[-1])
print(alg3_payoffs[-1])
print('avg p3', sum(alg3_payoffs) / len(alg3_payoffs))
print('avg r3', sum(alg3_regrets) / len(alg3_regrets))
#visualize_rounds(alg3_regrets, NUM_ROUNDS, 0, "EW Introduction Selling, 10 Employee+Employer Pairs: Regrets", "Random Guessing", "Randomized Auction", "Reg
#visualize_rounds(alg3_payoffs, NUM_ROUNDS, 0, "EW Introduction Selling, 10 Employee+Employer Pairs: Payoffs", "Random Guessing", "Randomized Auction", "Pay

file_name = 'EW Epsilon=1.0 Regret Comparison, Varied Employee+Employer Pair Count' + '.png'

x_all = numpy.array(list(range(0, NUM_ROUNDS)))
y_1 = numpy.array(alg1_regrets)
y_2 = numpy.array(alg2_regrets)
y_3 = numpy.array(alg3_regrets)
plt.plot(x_all, y_1, label='EW Epsilon = 1.0, 3 pairs'.format(alg_name="3 pairs"), linewidth=1)
plt.plot(x_all, y_2, label='EW Epsilon = 1.0, 5 pairs'.format(alg_name="5 pairs"), linewidth=1)
plt.plot(x_all, y_3, label='EW Epsilon = 1.0, 10 pairs'.format(alg_name="10 pairs"), linewidth=1)

plt.xlabel("Round")
plt.ylabel("Average Regret")
plt.title('EW Epsilon=1.0 Regret Comparison, Varied Employee+Employer Pair Count')
plt.legend(loc='best', prop={'size': 7})
plt.savefig(file_name)

plt.show()
```

In [ ]:

```
## generate table with regrets and payoffs

columns = ['Average Revenue', 'Average Regret']
rows = ['EW Epsilon = 1.0, 3 pairs', 'EW Epsilon = 1.0, 5 pairs', 'EW Epsilon = 1.0, 10 pairs']
title = 'Table of Average Regrets & Revenue With Varying Employee+Employer Pair Counts (EW Epsilon = 1.0)'

data = [[sum(alg1_payoffs) / len(alg1_payoffs), sum(alg1_regrets) / len(alg1_regrets)],
        [sum(alg2_payoffs) / len(alg2_payoffs), sum(alg2_regrets) / len(alg2_regrets)],
        [sum(alg3_payoffs) / len(alg3_payoffs), sum(alg3_regrets) / len(alg3_regrets)]]

data = [[np.round(float(i), 6) for i in nested] for nested in data]

generate_table(data, columns, rows, title)
```

Different Distributions

In [ ]:

```
# 2 bidder random values auction trial Epsilon = 1
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
```

```python
alg1_regrets, alg1_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg1_regrets[-1])
print(alg1_payoffs[-1])
print('p1', sum(alg1_payoffs) / len(alg1_payoffs))
print('r1', sum(alg1_regrets) / len(alg1_regrets))
#visualize_rounds(alg1_regrets, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Randomized Introduction Selling: Regrets", "EW", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg1_payoffs, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Randomized Introduction Selling: Payoffs", "EW", "Randomized Auction", "Payoff per Round")

# 2 bidder quadratic values auction trial Epsilon = 1
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_quadratic_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_quadratic_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg2_regrets, alg2_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg2_regrets[-1])
print(alg2_payoffs[-1])
print('p2', sum(alg2_payoffs) / len(alg2_payoffs))
print('r2', sum(alg2_regrets) / len(alg2_regrets))
#visualize_rounds(alg2_regrets, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Quadratic Introduction Selling: Regrets", "EW", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg2_payoffs, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Quadratic Introduction Selling: Payoffs", "EW", "Randomized Auction", "Payoff per Round")

# 2 bidder exponential values auction trial Epsilon = 1
num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_exponential_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_exponential_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg3_regrets, alg3_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg3_regrets[-1])
print(alg3_payoffs[-1])
print('p3', sum(alg3_payoffs) / len(alg3_payoffs))
print('r3', sum(alg3_regrets) / len(alg3_regrets))
#visualize_rounds(alg3_regrets, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Exponential Introduction Selling: Regrets", "EW", "Randomized Auction", "Regret per Round")
#visualize_rounds(alg3_payoffs, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Exponential Introduction Selling: Payoffs", "EW", "Randomized Auction", "Payoff per Round")

num_bidders = 2
auction_list = []
min_payoff, max_payoff = 0, 1
# perform 500 trials each with 500 rounds and 2 bidders
for i in range(NUM_TRIALS):
    auction = []
    auction.append(generate_random_bidlist(num_pairs, NUM_ROUNDS))
    auction.append(generate_exponential_bidlist(num_pairs, NUM_ROUNDS))
    auction_list.append(auction)
price_discretization = generate_linear_discretization(1, 0.01)
alg = ExponentialWeights(1.0, len(price_discretization))
alg4_regrets, alg4_payoffs = introduction_auction_trial(alg, auction_list, NUM_ROUNDS, max_payoff, price_discretization)
print(alg4_regrets[-1])
print(alg4_payoffs[-1])
print('p4', sum(alg4_payoffs) / len(alg4_payoffs))
print('r4', sum(alg4_regrets) / len(alg4_regrets))
#visualize_rounds(alg4_regrets, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Exponential Employer Values vs. Random Employee Values: Regrets", "EW", "Randomized Auction
#visualize_rounds(alg4_payoffs, NUM_ROUNDS, 1.0, "Epsilon 1.0 on Exponential Employer Values vs. Random Employee Values: Payoffs", "EW", "Randomized Auction


file_name = 'EW Epsilon=1.0 Regret Comparison, Varied Employee+Employer Distributions' + '.png'

x_all = numpy.array(list(range(0, NUM_ROUNDS)))
y_1 = numpy.array(alg1_regrets)
y_2 = numpy.array(alg2_regrets)
y_3 = numpy.array(alg3_regrets)
y_4 = numpy.array(alg4_regrets)
plt.plot(x_all, y_1, label='EW Epsilon = 1.0, 1 Employee+Employer Pair, Random Values'.format(alg_name="Random Employee+Employer"), linewidth=1)
plt.plot(x_all, y_2, label='EW Epsilon = 1.0, 1 Employee+Employer Pair, Quadratic Values'.format(alg_name="Quadratic Employee+Employer"), linewidth=1)
plt.plot(x_all, y_3, label='EW Epsilon = 1.0, 1 Employee+Employer Pair, Exponential Values'.format(alg_name="Exponential Employee+Employer"), linewidth=1)
plt.plot(x_all, y_4, label='EW Epsilon = 1.0, 1 Employee+Employer Pair, Exponential Values vs. Random Values'.format(alg_name="Exponential Employer vs Rando

plt.xlabel("Round")
plt.ylabel("Average Regret")
plt.title('EW Epsilon=1.0 Regret Comparison, Varied Employee+Employer Distributions')
plt.legend(loc='best', prop={'size': 7})
plt.savefig(file_name)

plt.show()
```

In [ ]:

```python
## generate table with regrets and payoffs

columns = ['Average Revenue', 'Average Regret']
rows = ['1 Employee+Employer Pair, Random Values',
        '1 Employee+Employer Pair, Quadratic Values',
        '1 Employee+Employer Pair, Exponential Values',
       '1 Employee+Employer Pair, Exponential Values vs. Random Values']
title = 'Table of Average Regrets & Revenue With Varied Employee+Employer Distributions (EW, Epsilon = 1.0)'

data = [[sum(alg1_payoffs) / len(alg1_payoffs), sum(alg1_regrets) / len(alg1_regrets)],
        [sum(alg2_payoffs) / len(alg2_payoffs), sum(alg2_regrets) / len(alg2_regrets)],
```

```
        [sum(alg3_payoffs) / len(alg3_payoffs), sum(alg3_regrets) / len(alg3_regrets)],
       [sum(alg4_payoffs) / len(alg4_payoffs), sum(alg4_regrets) / len(alg4_regrets)]]

data = [[np.round(float(i), 6) for i in nested] for nested in data]

generate_table(data, columns, rows, title)
```