# CSE257 Fall 2021: Assignment 2

Deadline: Nov-14 11:59pm. Submit a PDF of your answers on Gradescope. Upload a zip file of all your source code via `https://www.dropbox.com/request/cjWxmaOciQw6uu9jWdI6` by the same deadline. All of the code should be in Python and you can use standard libraries such as Numpy, Scipy, Matplotlib, etc. Finish all the answers and implementation completely by yourself. Feel free to search online and in textbooks for help with proofs, but you are not allowed to copy any code from other sources.

*Warning: More implementation is needed than the previous assignment. Make sure to start early.*

## 1 Stochastic Search (Implementation Involved)

**Question 1** (16 Points)**.** Consider the following functions:

- $f_1(x) = x_1^2 + x_2^2$
- $f_2(x) = -\dfrac{1 + \cos(12\sqrt{x_1^2 + x_2^2})}{0.5(x_1^2 + x_2^2) + 2}$ (this is called the drop-wave function)
- $f_3(x) = \sum_{i=1}^{50} x_i^2$ (yes it's a 50-dimensional function, don't freak out)

Implement the following algorithms (parameters will be specified below):

1. (GD) Gradient Descent with fixed step size $\alpha$.
2. (SA) Simulated Annealing with initial temperature $T$ and the annealing schedule $T_k = T/k$ for each $k$-th iteration (in the first iteration $k = 1$).
3. (CE) Cross-Entropy Methods with $k$ samples in each iteration, and the elite set is formed by the top $20\%$ of all samples in each iteration (whether "top" means higher or lower function value depends on whether you are minimizing or maximizing).
4. (SG) Search gradient with $k$ samples in each iteration, and a fixed learning rate of $\eta = 0.01$. (Also, first be clear about whether you are minimizing or maximizing a function, and then decide whether you should do gradient descent or ascent.)

For all algorithms, start at the initial point $x_i = 2$ for each variable $x_i$ used in the function. For (SA), where the search distribution for the next sample is needed, and (CE) and (SG), where an initial distribution is needed, use the Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ with $\Sigma$ being the identify matrix $I$ of the appropriate size (i.e., if the function uses $n$ variables, $I$ is of size $n \times n$.). You should know what $\mu$ should be in each of these cases.

Plot the following graphs. For each function, our goal is to **minimize the function** so make sure your updates are in the right direction. Also for each run of any algorithm, always **run 100 iterations from the initial point**.

1. (1 Point) Plot in 3D the drop-wave function $f_2$ defined above, over the domain $(x_1, x_2) \in [-5, 5] \times [-5, 5]$ (the vertical axis should show the value of $f_2(x_1, x_2)$ over this domain).
2. (2 Point) Perform GD with step size $\alpha = 0.01$ on the three functions. Plot how the function value changes with respect to the number of iterations (x-axis: number of iterations, y-axis: function value; same for all the following plots as well).

3. (3 Points) Perform SA with two different initial temperatures $T = 1000$ and $T = 10$, for each function. Plot how the function values changes over iterations. Overall 3 functions and 2 different temperatures, so 6 graphs in total. Because the algorithm is stochastic, for each function and each temperature, plot 5 different runs (i.e. **5 different random seeds** of your choice) in the same graph. That is, start at the same initial point, and since each step will be stochastic, you should plot 5 different trajectories for each function (i.e., 5 sequences of points in each of the 6 plots requested, use a different color for each sequence). This requirement is the same for the following two algorithms as well.

4. (4 Points) Perform CE with two sample sizes, $k = 10$ and $k = 50$, for each function. Perform $5$ runs for each function and each sample size. Plot the average of the function values of all samples in each iteration for each function. Overall 6 plots (each plot should show 5 random trajectories).

5. (5 Points) Perform SG with two sample sizes, $k = 10$ and $k = 50$, for each function. Perform $5$ runs for each function and each sample size. Plot the average of the function values of all in each iteration for each function. Overall 6 plots (each plot should show 5 random trajectories). Note: If you see the gradient update diverging (leading to very large values), you can use the normalized gradient $\nabla f(x)/\|\nabla f(x)\|_2$ in each update instead of just $\nabla f(x)$ (so that after multiplying with the learning rate, each update step is guaranteed to be small in magnitude), but you don't have to do this and you can just show the divergence behavior as well.

6. (1 Point) Based on the performance of these algorithms with different parameters on different types of functions, summarize some intuition about how to choose among these algorithms and the parameters. For instance, when the function dimension is large, is it better to sample or find gradient? What are the different trends of the algorithms for different sample sizes? Do some algorithms give more stable behavior than others? There is no standard answer here, just reflect a bit on how you should choose from the different strategies.

# 2 Classical Search on Graphs

**Question 2** (3 Points)**.** Prove the Separation Property for the Dijkstra algorithm. That is: after each iteration of the algorithm (i.e., after the frontier has been fully updated in an iteration), any *acyclic* path from any state in the explored set to any state in the unexplored set has to pass through some state in the frontier set. (The "acyclic" assumption here simplifies the proof a bit, but it is not important, because for any path that contains cycles we can always ignore the cycling part and only consider its acyclic part.) You will likely need to do mathematical induction over the number of iterations to prove this.

**Question 3** (2 Points)**.** Construct a small undirected graph (say, a start node, a goal node, and just a few intermediate nodes. You design the cost for each edge as well.) and design a heuristic function $h$ such that: $h$ itself is consistent, but multiplying it by $5$ (i.e. the heuristic value becomes $5h(s)$ for each node $s$) makes it inconsistent and misguides the search (i.e. such that it no longer returns the optimal path in the end). Show the path from start to goal in each case.

# 3 Adversarial Search (Implementation Involved)

*Warning: The minimax/expectimax algorithm is conceptually simple, but to make it work in an actual game it may take longer than you expected, especially if you are not very familiar with Python (Hint: "deep copy"). Start early.*

Using the starter code here `https://github.com/ucsdcsegaoclasses/expectimax` for the 2048 game. Implement two versions of Expectimax: one with Expectimax on just a depth-1 tree (we refer to it as "Exp-1") and one with a depth-3 tree ("Exp-3"). The depth-1 tree simply has the current game state as the root, and its children nodes that correspond to the four actions as the leaf nodes, with payoff defined by the game score (provided by the game engine) immediately annotated for these leaf nodes. The depth-3 tree is the game tree that contains all paths following the sequence of "player-move, computer-move, player-move." Use the score provide by the game engine (the number displayed on the upper left corner in the game interface) as the payoff value at any leaf node in these trees (i.e. the evaluation function here). If any action becomes infeasible (i.e. it becomes impossible to move the tiles in a

particular direction), you can annotate the corresponding child state as terminal with some very negative payoff such as -100 (you can feel free to design anything, as you'll only need to plot the actual game scores later).

**Question 4** (4 Points)**.** Plot the performance of Exp-1 and Exp-3 in the following way. For each version, plot 5 runs of the algorithm starting from the beginning of the game for 100 moves (i.e., call the algorithm 100 times), and show how the **actual** game score changes (i.e. what the game engines returns after you make each move. Annotate the y-axis in the plot with this value.) over the number of iterations (x-axis). In any run, if the agent fails the game early (can't move anywhere) then just stop the sequence there.

**Question 5** (3 Points)**.** Design a different evaluation function for Exp-3 to perform better than Exp-3 with the previous evaluation function that simply uses the original game score. You can use any information from the game state, such as highest tile, pattern of the existing tiles, etc. Design your plots to show that the new algorithm (i.e. Exp-3 with the evaluation function you designed) is better than the original Exp-3.

# 4 Basics of RL

Let's derive some basic concepts for RL in a simple setting. Suppose Figure 3 is a schematic for routes for driving from, say, UCSD to the SAN airport. Each state represents as a segment of the route. For instance $s_1$ is the part within campus. $s2$ is the segment that takes Highway 5 and $s_3$ is the other option of taking $52$ and $805$, etc. The time spent in each segment (i.e., each state), depending on traffic, is a random variable $C(s_i)$. Our goal is to find good policies, mapping states to actions ($a_{ij}$ is the action that takes you from $s_i$ to $s_j$), and **minimize** the expected total time that it takes to start from each state to arrival, which we write as $T_\pi(s_i)$ under some choice of $\pi$. Note that all these definitions are similar to but not the same as the usual MDPs: we are minimizing time, and the actions deterministically take you from states to states (assuming you are driving well enough and can always take exits successfully), but the cost of each state is a random variable, and there is no discount factor.

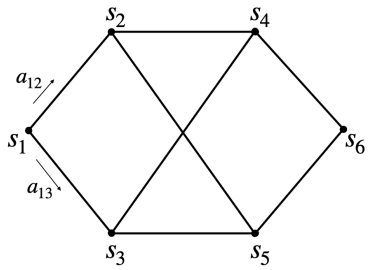


Figure 1: Graph for the RL questions.

**Question 6** (1 Point)**.** Consider the following deterministic policy $\pi$: $\pi(s_1) = a_{12}$, $\pi(s_2) = a_{24}$, $\pi(s_4) = a_{46}$, $\pi(s_3) = a_{35}$, $\pi(s_5) = a_{56}$, and $s_6$ is a terminal state with no action. How do you define the overall expected time $T_\pi(s_1)$ at state the $s_1$ under this policy $\pi$? Hint: The time cost at each state $s$ is a random variable $C(s)$, so use its expectation $\mathbb{E}[C(s)]$. Also, under this deterministic policy there is just one deterministic sequence of states that goes from $s_1$ to $s_6$, where it terminates. You do not need to involve any state that is not part of this sequence to estimate the expected time for starting from $s_1$.

**Question 7** (2 Points)**.** Suppose you have formed good estimates of the minimal total travel time for starting from $s_2$ and $s_3$, which we write as $T(s_2)$ and $T(s_3)$. How would you define the total travel time starting at $s_1$ by taking action $a_{12}$ (write that as $T(s_1, a_{12})$), and that of taking $a_{13}$ (write that as $T(s_1, a_{13})$)? Next, using $T(s_1, a_{12})$ and $T(s_1, a_{13})$, how do you define the optimal total travel time at $s_1$, written as $T(s_1)$? Write down these as three equations for defining $T(s_1, a_{12})$, $T(s_1, a_{13})$, and $T(s_1)$, in which you can use $T(s_2)$, $T(s_3)$, $C(s_1)$, etc. Hint: You need to take into account of the expected time cost of $\mathbb{E}[C(s_1)]$.

**Question 8** (2 Points)**.** Suppose you have lived here for long enough to form good estimates of $T(s_1)$. Today you have just driven past the $s_1$ segment and then took the action $a_{13}$. You realized that because of new construction, the time you spent at $s_1$ today, written as $C(s_1)$, is much longer than usual. You also make the assumption that after that there is no reason for $T(s_3)$ to change. Using this information, how would you update your estimated $T(s_1, a_{13})$? If you need a learning rate defined, write it as $\alpha$.

Reflect a bit on your answers in these questions. They correspond to the state values, state-action values (Q-values), and the temporal-difference update rule for Q-learning.