# Artificial Intelligence (CS 534)
**Assignment 1 (100 pts. total)**
**Due: October 06, ANYTIME**

**Note: For your submission each problem should include, as a separate file, a code file (*e.g.*, `Problem_1_1.py`) and a print out of each input state and final state (*e.g.*, a text file that has all inputs/outputs for all *n*). In addition, submit a single/combined Word doc file that includes notes for each algorithm as well as requested graphic plots.**

## Problem 1. (50 points)

a. Implement A* search by yourself (*i.e., not using an A* search library*)

b. Apply it to a K-puzzle search using heuristics $h_1$ (misplaced tiles) and $h_2$ (Manhattan distance) introduced in the class and run it 5 times for each of the following increasing K values by selecting 5 randomly selected initial states: K = 3 (2x2-1), K = 8 (3x3-1), K = 15 (4x4-1), …

c. Report two graphic plots:
   1. The size of the puzzle vs. the average of the total number of nodes generated;
   2. The effective branching factor vs. the total number of nodes generated. The effective branching factor can be approximated using the following formula:

$$b^* \approx \left(\frac{N}{d}\right)^{1/d}$$

**Important note**: Depending on your computer and the randomly selected initial state, your A* algorithm may get stuck at a looooong calculation. Thus, to avoid this, make sure that your algorithm has a defined "stoppage criterion" to abort after X steps or Y mins, where X and Y correspond to a long (but reasonable) time (*e.g.,* Y=10 mins). You may consider not running for the next K, if the previous K results in triggering the stoppage criterion for each of the 5 runs.

**Problem 2**. **(50 points)**

a. Implement a Genetic Algorithm (GA) for an n-Queen problem by yourself (*i.e., not importing a GA library*)

b. Try to apply your algorithm to the problem for $n = 5, 6, \ldots, 10$, where n is the board dimension and the corresponding number of queens (*i.e.,* for the standard 64-squares board $n = 8$). Note that the algorithm might not be able to find a solution (for instance, for n=2), so you also need to implement a stoppage criterion. Think about a good one! Run your algorithm 5 times for each $n$.

c. For each $n$, calculate the averages of total number of steps needed to find a solution and the final heuristic value. Output these data as a table.

d. Modify your GA, so that for every iteration, it does 3 mutations in a row, not just one (that is, one mutation for each of the 3 different genes). Apply it for $n = 5, 6, 7, \ldots 10$ and compare the results with the previous algorithm by plotting, in the same plot, the number of steps taken to find the solution vs. the board size for the original GA and the modified one.