Department of Electrical and Computer Engineering

ECE 457A Adaptive and Cooperative Algorithms

Instructor: Dr. Otman A. Basir

Assignment 2
Due Date: February 20, 2026 11:59PM

Instructions

- This assignment can be approached in groups. Each group consists of three students.
- If you are attempting this assignment as a group, then you are required to solve all questions and upload one submission on behalf of the group. Make sure that you list the names of the team members . If you are doing it solo, then you can ignore either Question 1 (re continuous SA), or question 4 (re combinatorial SA).
- You should upload your answers as a PDF file on learn before 11:59pm of the deadline date
- Attach any codes used for the assignment separately as a compressed file to the same dropbox
- You can use any programming language (Matlab and Python are preferred)
- Works with more than 30% similarity will not be marked
- Communicate any issues or concerns with the TAs.
- Deliverables: provided for each question
- The assignment has **five (5) problems**. Each problem is worth **20 points (20%)**.
- For each problem, the point breakdown is provided in the corresponding **Deliverables** box.

Problem 1

We want to minimize the Easom function of two variables:

$$\min_{\mathrm{x}} f(\mathrm{x}) = -\cos x_1 \cos x_2 \exp\left(-(x_1-\pi)^2-(x_2-\pi)^2\right), \quad \mathrm{x} \in [-100,100]^2 \tag{1}$$

The Easom function is plotted in Figure 1 for $x \in [-100, 100]$. The global minimum value is -1 at $x = (\pi, \pi)^T$.

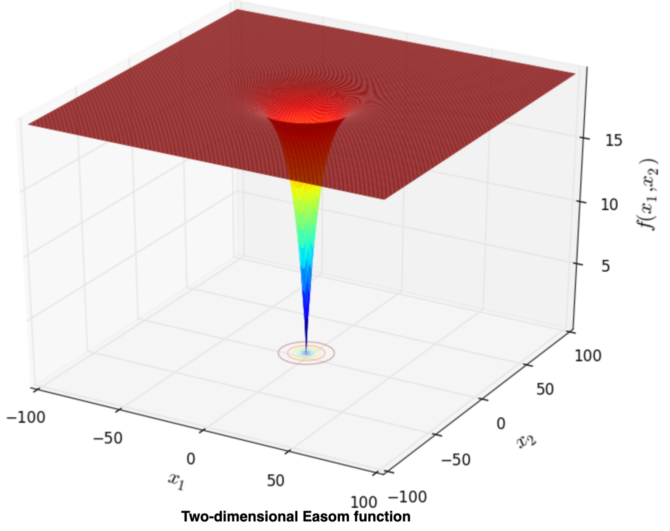


Figure 1: Easom function

This problem is hard since it has wide search space and the function rapidly decays to values very close to zero, and the function has numerous local minima with function value close to zero. This function is similar to a needle-in-a-hay function. The global optimum is restricted in a very small region.

Deliverables

1. Come up with a proper problem formulation, neighborhood function and cost function for this problem to apply SA algorithm to solve it. [15 *pts*]
2. Provide solution description: algorithm/pseudo code [10 *pts*]
3. Plot the solution profile as function of time for each of the following scenarios: [15 *pts*] ([5 *pts*] for each experiment)
   - Experiment 1: Selecting 10 different initial points randomly in $[-100, 100]$
   - Experiment 2: Selecting 10 different initial temperatures in a reasonable range
   - Experiment 3: Selecting 10 different annealing schedules
   - For each experiment, you can fix the other two parameters.
4. Report your observations on SA performance for solving this problem in the context of the three experiments. [15 *pts*]
5. What was the best solution you could find after conducting the three experiments? What was the setting of SA that achieved this solution? Why was that setting performing better than other settings? [10 *pts*].

   Sample output of your program, to demonstrate its functioning. [15 *pts*]

   Quality and completeness of the report you hand in. [10 *pts*]

   Discussion on time and memory complexity. [10 *pts*]
6. You must also upload the code you implemented to the assignment dropbox.

   ** You may use SA solver from MATLAB Global Optimization toolbox.

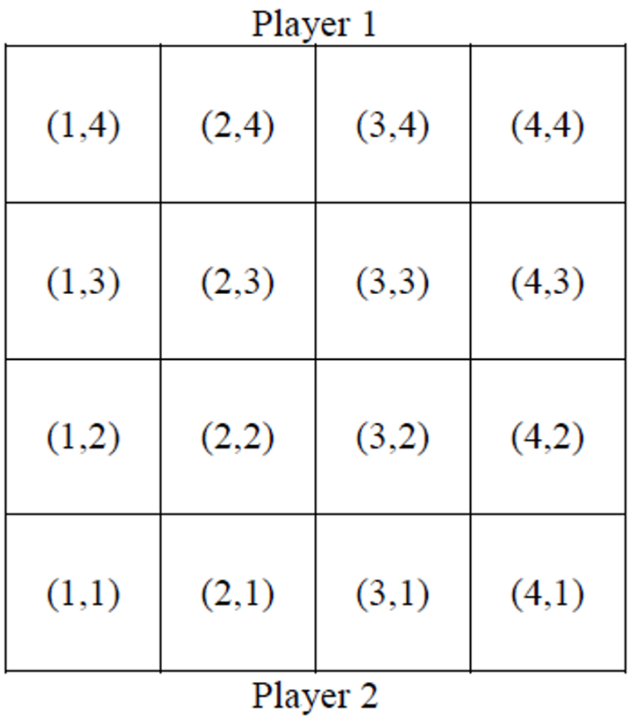


Figure 2: Cogna Board and 2 players

Problem 2

The game of Conga was developed by Matin Franke, and published by "Das Spiel Hamburg" in 1998. It is a two-player game played on a square 4X4 board, as shown in Figure 2. Initially, Player 1 has ten black stones in (1,4) and Player 2 has ten white stones in (4,1). The players alternate turns. On each turn, a player chooses a square with some of his stones in it, and picks a direction to move them, either horizontally, vertically or diagonally. The move is done by removing the stones from the square and placing one stone in the following square, two in the next one, and the others into the last one. The stones can only be moved in consecutive squares that are not occupied by the opponent; if a direction has less than three squares not occupied by the opponent in a row, then all remaining stones are placed in the last empty square. If a square has no neighbouring squares that are not occupied by the opponent, then the stones in that square cannot be moved. You can move stones from a square to a (series of) neighbouring square(s), provided the squares you are moving to are not occupied by the opponent. It makes no difference if the squares are free or occupied by yourself. You do not need any of the squares you are moving to be free; they could all already be occupied by your other stones. The only distinction is between squares that are occupied by the opponent (which block you) and squares that are not occupied by the opponent (which you can move to). For example, let's say you have a number of stones in square (1,4) and you want to move them right. There can be four possible cases:

① The opponent occupies square (2,4). You cannot move right. That is the example in Figure 5

② The opponent occupies square (3,4), but square (2,4) is either free or occupied by yourself. You move all the stones from (1,4) to (2,4).

③ The opponent occupies square (4,4), but squares (2,4) and (3,4) are either free or occupied by yourself. You move one stone from (1,4) to (2,4), and all other stones in (3,4). That is the example in Figure 3.

④ The opponent doesn't occupy any squares in that row, and all squares are either free or occupied by yourself. You move one stone from (1,4) to (2,4), two stones to (3,4), and all other stones in (4,4). That is the example in Figure 3.

The goal of the game is to block the opponent, so that he has no legal moves. In other words, all of the opponents' stones must be trapped in squares that are all surrounded by the player's stones.

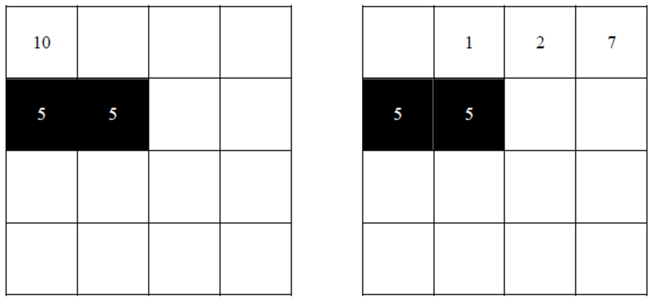


Figure 3: From the setup of the left board, white has only one legal move. The result that move is shown in the right board.

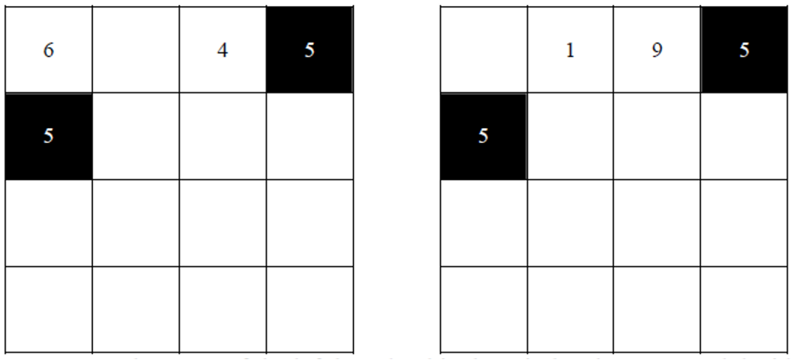


Figure 4: From the setup of the left board, white has six legal moves, and decides to move $(1, 4)$ to the right. The result of that move is shown in the right board.

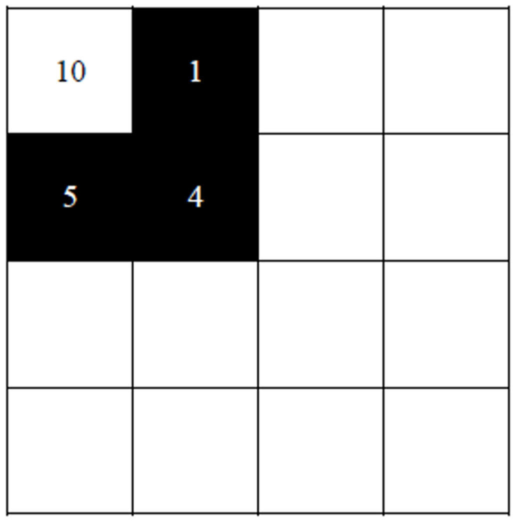


Figure 5: In the setup of this board, white has no legal moves. Black has won.

Deliverables

1. Design and implement a computer program to play the Conga game. The agents implemented should use the Minimax search algorithm and Alpha-Beta pruning, as well as some evaluation function to limit the search. They should also have a reasonable response time.

   Since it is unlikely that you will discover an optimal evaluation function on the first try, you will have to consider several evaluation functions and present them in your report.

   - Provide a problem representation. [10 *pts*]

   - Provide description of your solution in the form of an algorithm/pseudo code. [20 *pts*]

   - Provide a hand-worked example explaining the working of your solution/search strategy on a sample of the problem representation (similar to what we did in class for the tic-tac-toe) [15 *pts*]

2. You will also need to implement a Random Agent, or an agent that always plays a random legal move. Summarize the logic (algorithm) behind this agent. Report your observations on how well does this agent, using a performance index of your choice. [10 *pts*]

3. Submit a sample output of your program, to demonstrate its functioning. The output should include the depth and the number of nodes explored.[15 *pts*]

4. If you wish, you may implement improvements of the basic Minimax/Apha-Beta agent, such as an iterative deepening Minimax agent, or the Null-Window Alpha-Beta pruning, for **bonus points** [10 *pts*] of the question weight.

5. Provide a discussion on time and memory complexity [10 *pts*]

6. Quality and completeness of the report you hand in. [10 *pts*]

Notes on the game:

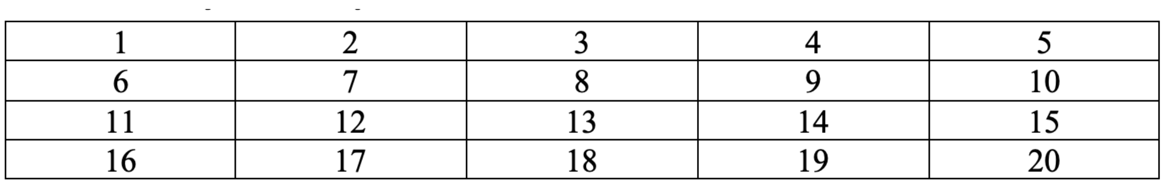
Ⓐ While researching the game of Conga, you may realize that there is a second victory condition. A player can win by making a line four squares long containing the same number of stones. This victory condition has been removed from the game for this project, in order to simplify the evaluation function.

Ⓑ A consequence of this simplification is that the game is now much harder to win. In fact, two players of equal strength can play for hours without being able to end the game. However, it is still possible for a player to defeat a player of lesser skill, or for an agent to defeat another agent with a shallower search and a less accurate evaluation function. And it is definitely possible for a rational agent to defeat the Random Agent, in as little as 30 moves.

Ⓒ Given the requirement of making your agent play against the Random Agent, you may come up with the idea of somehow optimizing their agent to play better against that opponent specifically, by exploiting its random, non-optimal play. That is not acceptable for this project. The goal of the project is to design an agent that can play rationally against any opponent, not one that is optimized to play only against the Random Agent.

Problem 3

This is one of the QAP (quadratic assignment problem) test problems of Nugent et al. 20 departments are to be placed in 20 locations with five in each row (see the table below). The objective is to minimize costs between the placed departments. The cost is (flow * rectilinear distance), where both flow and distance are symmetric between any given pair of departments. The flow and distance data can be found in Assignment 2 folder in two separate files (Assignment-2-flow and Assignment-2-distances) . The optimal solution is 1285 (or 2570 if you double the flows).



1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Figure 6: Layout of department locations

Deliverables

1. Derive an optimization formulation for this problem. [10 *pts*]
2. Develop a simple Tabu Search based program for solving this problem. To do this you need to encode the problem as a permutation, define a neighborhood and a move operator, set a Tabu list size and select a stopping criterion. Use only a recency based tabu list and no aspiration criteria at this point. Use less than the whole neighborhood to select the next solution. [25 *pts*]
3. Perform the following experiments on your TS code (one by one) and compare the results and summarize your conclusions. .

   - Experiment 1: Run your program with 20 different initial solutions. Report the best solution for each run and summarize your observations.[5 *pts*]

   - Experiment 2: Run your program with different tabu-list sizes: 2 larger sizes than your original choice; and two smaller sizes than your original choice. Report the best solution for each run and summarize your observations. [5 *pts*]

   - Experiment 3: Change the tabu list size to a dynamic one – an easy way to do this is to choose a range and generate a random uniform integer between this range every so often (i.e., only change the tabu list size infrequently). Report the best solution and summarize your observations.[5 *pts*]

   - Experiment 4: Add one or more aspiration criteria such as best solution so far, or best solution in the neighborhood, or in a number of iterations. Report your observation on the algorithm performance in the form of a summary on the experiment. [10 *pts*]

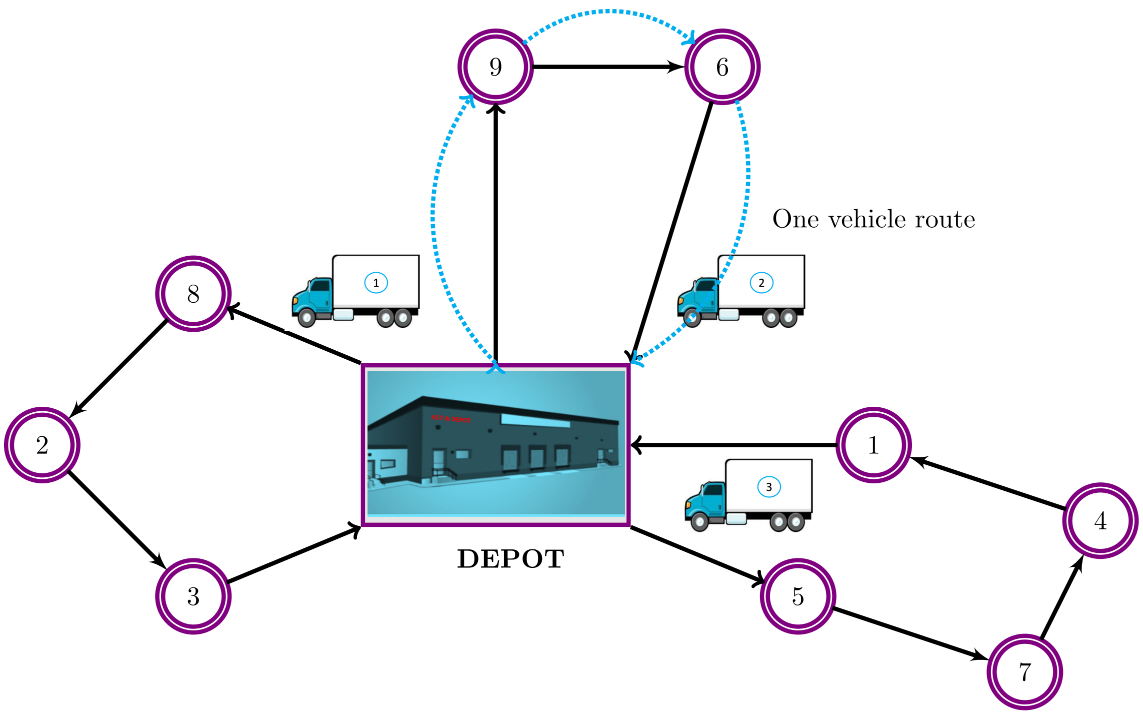
   - Experiment 5: Add a frequency based tabu list and/or aspiration criteria (designed to encourage the search to diversify). Report your observation on the algorithm performance in the form of a summary on the experiment. [10 *pts*]

   Sample output of your program for each experiment, to demonstrate its functioning. [10 *pts*]
4. Discussion on time and memory complexity. [10 *pts*]
5. Quality and completeness of the report you hand in. [10 *pts*]

Problem 4

Simulated Annealing can be used to solve The Vehicle Routing Problem (VPR). VRP is defined as having m vehicles at a depot that need to service customers in c cities. The travel distances between every two cities are defined by a matrix D with element d_{ij} denoting distance between cities i and j. The travel distances from the depot to each of the cities are known. Each customer j has a service time s_j. The objective of the VPR is determining the routes to be taken by the set of m vehicles while satisfying the following conditions:

- Ⓐ Starting and ending at the depot,
- Ⓑ Having minimum cost (travel time +service time),
- Ⓒ Each customer is visited exactly once by exactly one vehicle. The figure below illustrate possible routes for 9 customers and 3 vehicles problem



Deliverables

1. Assuming that the number of targeted routes is fixed and is equal to the number of vehicles. Develop a Simulated Annealing program for seeking an optimal solution for this problem. Submit a report that captures the following items:

   Develop an optimization problem formulation for this problem: optimization objective function and the optimization statement (eg, Minimize . . . subject to . . .). [10 *pts*]

   - Develop solution representation in the context of SA solution formulation. [15 *pts*]

   - Define a suitable neighborhood operator, [5 *pts*]

   - Define the objective function used to calculate the cost of a solution . [10 *pts*]

   - Show how your solution performs on instance **A-n39-k6.vrp** which can be found at the NEO site: http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances and compare your best solution to the reported optimal solution. [20 *pts*]

2. How would the problem formulation change if we add the constraint that the total duration of any route shouldn't exceed a preset bound T? [10 *pts*]

3. Provide a sample output of your program to demonstrate its functioning as it moves towards the optimal solution.. [10 *pts*]
4. Discussion on time and memory complexity. [10 *pts*]
5. Quality and completeness of the report you hand in. [10 *pts*]

Problem 5

Minimax and Alpha–Beta Game Search
Objective. Implement a two-player, Tic-Tac-Toe (3×3) game-playing agent using the **minimax** algorithm, and then extend it with **alpha–beta pruning**. Your implementation must be generic and clearly separate *game logic* from *search logic*.

Deliverables

1. Part A: Game formulation (20 points)

   Define a state representation.

   Implement:

   $$\texttt{legalMoves}(s), \quad \texttt{applyMove}(s,a), \quad \texttt{isTerminal}(s), \quad \texttt{utility}(s)$$

   Utilities must be from MAX's perspective: win $= +1$, loss $= -1$, draw $= 0$ (or an equivalent consistent scale).

2. Part B: Depth-limited minimax (35 points)

   Implement a recursive minimax procedure with a depth limit d.

   When the cutoff depth is reached, use an evaluation function $\hat{E}(s)$.

   Your function should return both:

   - the selected action
   - the backed-up minimax value

3. Part C: Alpha–beta pruning (35 points)

   Extend your minimax implementation with α–β pruning.

   Ensure correctness: pruning must not change the returned minimax value.

   Instrument your code to count the number of expanded nodes.

4. Part D: Experimental comparison (required). 10 points
   For at least three depth limits (e.g., $d = 2, 4, 6$):

   Report the chosen move,

   Report the minimax value,

   Report the number of nodes expanded by minimax vs. alpha–beta.

5. Bonus (up to 10 points). Implement one of the following:

   Move ordering to improve pruning effectiveness,

   Iterative deepening with alpha–beta,

   a stronger evaluation function and justify it.

6. Submission. Source code, a short report (1–2 pages), and clear instructions for running your program.