# Model Answer

## I. Graded Exercises

1. **[Programming Exercise - 5 Marks]** An industrial communication company is planning to lay cables to a new factory allowing all the factory's units to be linked for the interchange of data. If it is constrained to bury the cable only along certain paths, then there would be an undirected graph representing which points are connected by those paths. Some of those paths might be more expensive, because they are longer, or require the cable to be buried deeper; these paths would be represented by edges with larger weights. For example, Figure 1 shows 5 factory's units and the possible paths along which the cables can be buried.

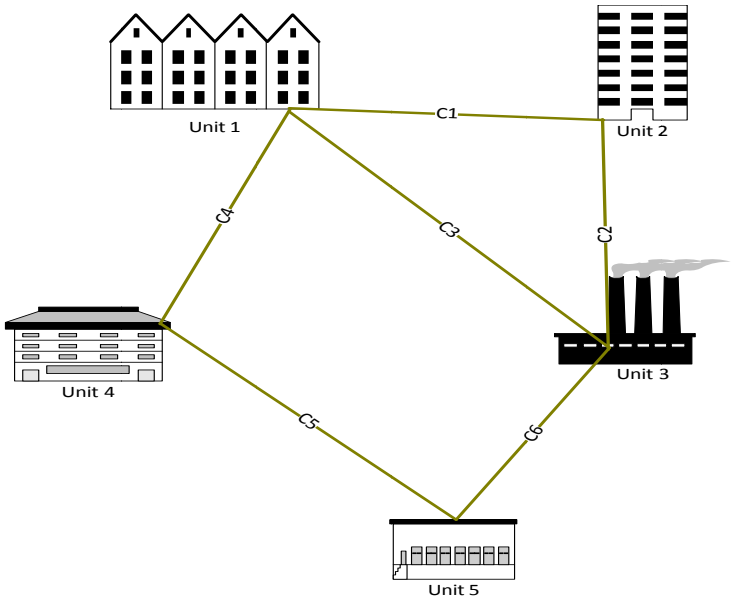



**Fig. 1** A Communication Cable Configuration

The minimum spanning tree (MST) of the graph that represents the factory's units and paths can be used to decide the most cost-effective paths for laying cables. MST is a tree that spans all the nodes of the graph with a minimum cost. MST is widely used in different applications such as communication network, transportation networks, scheduling and data clustering. Different greedy algorithms have been proposed for finding a MST such as Prim-Jarnik, Kruskal's and Baruvka's Algorithms.

In this problem, a MATLAB code that implements the Tabu Search algorithm for finding the MST of an undirected graph is given:

a) Use the given functions to find the most cost-effective paths for the graph given in **Units100.mat** and report the total cost of laying the cables**.**

b) Run the code with different tabu list lengths (5, 10, 15, 25, 50) and report the cost obtained in each case. What do you observe?

c) Modify **GetBestNeighbourST.m** and other necessary functions to include the use of an aspiration criterion. The aspiration criterion in this problem could be to allow solutions that are better than the currently known best solution. Repeat (a) and (b) with the modified code.

d) Suppose that splitters need be used at the factory's units if the number of cables going through the unit exceeds 2. The cost of the splitter increases as the number of cables connected to the splitter ***n*** increases and it can be calculated as

$$C_s = \frac{10}{1+e^{-n/10}}$$

Modify **GetBestNeighbourST.m** and other necessary functions to consider adding the cost of the splitters. Repeat (a) and (b) with the modified code.

e) Compare between Tabu Search and the greedy algorithm of Kruskal as an example for greedy algorithms.

**Notes:**

- You should run each experiment **several times** and report the average results.
- **Units100.mat** contains a sparse matrix **Graph** that represents the possible paths between 100 factory's units through which communication cables can be laid. The element Graph (*a*, *b*) represents that cost of the path between units *a* and *b*.

**Solution:**

a) Use the given functions to find the most cost-effective paths for the graph given in **Units100.mat** and report the total cost of laying the cables**.**

Call the TabuSearch function as follows:

```
% Solve the MST problem using tabu search
[MST MSTCost] = TabuSearch(Graph, TabuLength, NumIterations, ...
@GenInitialST, @GetBestNeighbourST);
```

You can select the tabu length to be equal to the square root of the number of edges or any other empirical value. You should run for large enough number of iterations (200-300).

The best cost obtained is 109. The algorithm gets this value after around 80-100 iterations.

Note that this value is the global minimum of the cost function and it is always obtained regardless of the starting solution because the cost function over the solution space is convex.

b) Run the code with different tabu list lengths (5, 10, 15, 25, 50) and report the cost obtained in each case. What do you observe?

The best cost obtained in all cases is 109 for the reasons mentioned in (b).

Note that after the tabu search gets this cost, it starts accepting non-improving solutions in the neighborhood of the current solution. The cost however does not increase during the next iterations. This is because the non-tabu solutions in the neighborhood contain other optimal solutions and the used tabu lengths are smaller than the number of these optimal solutions.

c) Modify **GetBestNeighbourST.m** and other necessary functions to include the use of an aspiration criterion. The aspiration criterion in this problem could be to allow solutions that are better than the currently known best solution. Repeat (a) and (b) with the modified code.

Modify GetBestNeighbourST prototype to pass the best cost obtained so far:

```
function [BestNeighbourST BestNeighbourSTCost TabuEdges] ...
= GetBestNeighbourST(Graph, ST, ...
TabuEdges, TabuLength, BestCost)
```

Modify the part of GetBestNeighbourST at which you check for tabu moves as follows:

```
% 2. Check if the edge to be removed (e1) is in the tabu list
if TabuEdges(NT1(e1), NT2(e1)) == 0
if NewSTCost < BestNeighbourSTCost
BestNeighbourST = NewST;
BestNeighbourSTCost = NewSTCost;
TabuEdgeN1 = N1(BestEdge);
TabuEdgeN2 = N2(BestEdge);
AddTabuMove = true;
end
% Check for aspiration criteria (accept tabu solution if it's better
% than the solution obtained so far)
elseif NewSTCost < BestCost
if NewSTCost < BestNeighbourSTCost
BestNeighbourST = NewST;
BestNeighbourSTCost = NewSTCost;
BestCost = NewSTCost;
end
end
```

Modify the function call in TabuSearch as follows:

```
[Soln SolnCost TabuList] = feval(GetBestNeighbourSolnFn, ProbData, ...
Soln, TabuList, TabuLength, BestSolnCost);
```

The best cost obtained in all cases is 109 for the reasons mentioned in (a).

d) Suppose that splitters need be used at the factory's units if the number of cables going through the unit exceeds 2. The cost of the splitter increases as the number of cables connected to the splitter ***n*** increases and it can be calculated as

$$C_s = \frac{10}{1+e^{-n/10}}$$

Modify **GetBestNeighbourST.m** and other necessary functions to consider adding the cost of the splitters. Repeat (a) and (b) with the modified code.

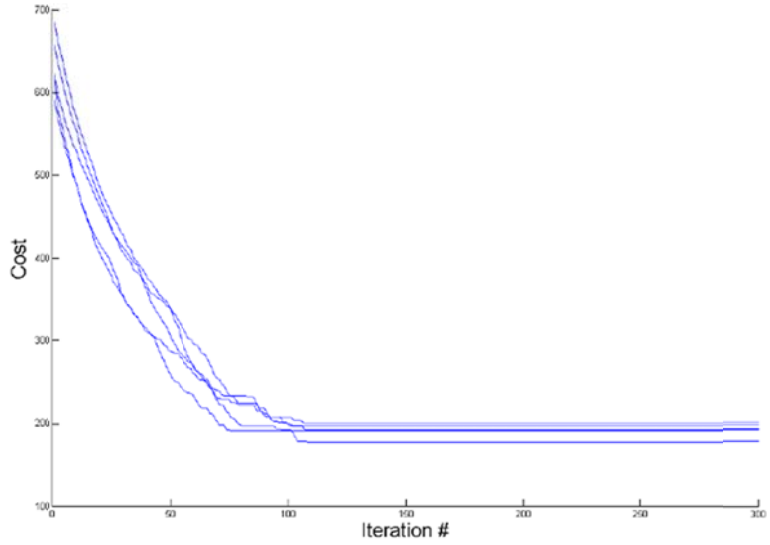
**Notes:** You should run each experiment **several times** and report the average results.

Modify the part at which you calculate the cost as follows

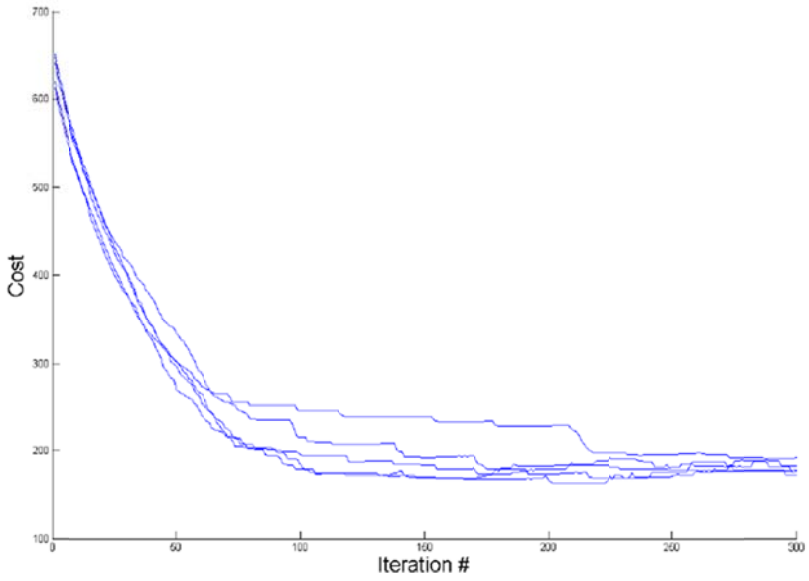
```
% Caclulate the cost of the new spanning tree
NewSTCost = sum(sum(Graph .* NewST)) / 2;
% Add the cost of the splitters
NumEdgesPerNode = sum(NewST, 2);
NumEdgesPerNode = NumEdgesPerNode(NumEdgesPerNode>2);
NewSTCost = NewSTCost + sum(10 ./ (1 + exp(-(NumEdgesPerNode)/10)));
```

The best costs obtained using the new cost function vary with the starting solution and the used tabu length. This is because the modified cost function is non-convex over the search space. This means that a local search algorithm might get trapped at local minimums. In this case, the tabu search should get out of these local minimums by allowing non-improving solutions.

We can observe that the best cost obtained highly depends on the starting solution. By observing the change of the cost during the iterations of the tabu search, we can see that using very small tabu lengths (5, 10) does not allow the algorithm to get out of the local minimum as shown in Figure 1s-a. The algorithm starts from the initial solution and keeps going towards the local minimum. The tabu length in this case is smaller than the number of neighboring solutions with the same local minimum cost so the algorithm will get trapped at the local minimum. On the other hand, using large tabu length (25, 50) allows the tabu search algorithm to get out of the local minimums and accordingly obtain better solutions than the local search in most of the cases as shown in Figure 1s-b.




(a) Tabu length = 5 with aspiration criteria




(b) Tabu length = 50 with aspiration criteria

**Fig. 1s** The change of the cost function with the iterations of tab search (different curves represent different starting solutions)

**2. [Programming Exercise- 5 Marks]** For Rosenbrock's banana function

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

we know that its global minimum $f^* = 0$ occurs at (1,l) (see Fig. 2). This is a standard test function and quite tough for most conventional algorithms. However, using the given MATLAB program, we can find this global minimum easily using simulated annealing.

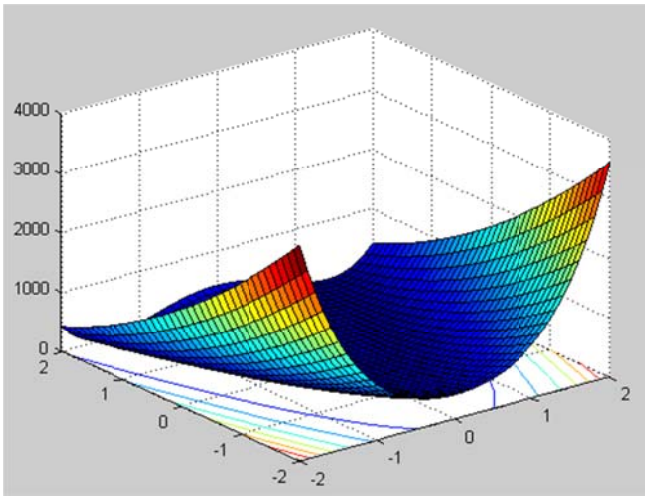



**Fig. 2** Rosenbrock's banana function

This banana function is still relatively simple as it has a curved narrow valley. Other functions such as the egg crate function are strongly multimodal and highly nonlinear. Extend the given MATLAB program to deal with egg crate function as an example for highly nonlinear multimodal functions. The egg crate function is:

$$f(x,y) = x^2 + y^2 + 25[\sin^2(x) + \sin^2(y)]$$

This function has the global minimum $f^* = 0$ at (0,0) in the domain $(x,y) \in [-5,5] \times [-5,5]$. It would takes about 2500 evaluations to get an optimal solution accurate to the third decimal place.

i. Modify the given MATLAB program so as to investigate the rate of convergence of simulated annealing for different cooling schedules such as:

$$T(t) = \frac{T_o}{1+\alpha t}, \quad \alpha > 0$$

ii. For standard SA, the cooling schedule is a monotonically decreasing function. There is no reason why we should not use other forms of cooling. For example, we can use:

$$T(t) = T_o \cos^2(t) \exp[-\alpha t], \quad \alpha > 0$$

Modify the given MATLAB program to study the behavior of various functions as a cooling schedule.

**Solution:** Matlab code is attached.

3. **[Written Exercise - 5 Marks]** Simulated annealing can be used to solve the Vehicle Routing Problem (VPR) defined by 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 given by a vector ***Depot***. Each customer *j* has a service time $s_j$ The VPR consists of determining the routes to be taken by a set of ***m*** vehicles satisfying the following conditions:

- Starting and ending at the depot,
- Having minimum cost (travel + service),
- Each customer is visited exactly once by exactly one vehicle.

Figure 3 illustrates possible routes for a 9 customers and 3 vehicles problem

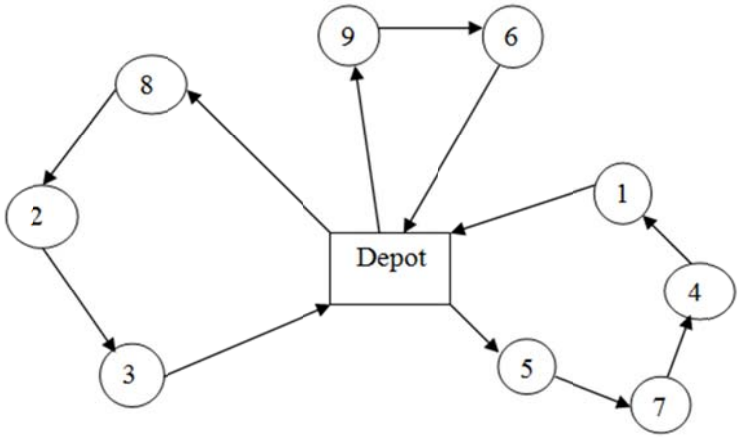



**Fig. 3** 9 Customers 3 Vehicles Routing Problem

a) Assuming that the number of required routes is fixed and that it's equal to the number of vehicles. Simulated Annealing could be applied in order to solve this problem, it's required to:
   i. Find a suitable solution representation,
   ii. Define a suitable neighborhood operator,
   iii. Define the objective function used to calculate the cost of a solution.

b) 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?

**Solution:**

a)

i. Find a suitable solution representation

The following vector solution representation can be used

$$Sol = [d, R_1, d, R_2, ..., d, R_m]$$

where
*Ri* denotes the list of customers serviced by vehicle *i* and these lists are separated by *d* denoting the depot.

*Example*

$Sol = [d,8,2,3,d,9,6,d,5,7,4,1]$

ii. Define a suitable neighbourhood operator

The neighbourhood can be defined by:

- Swapping two randomly selected cities in any two randomly selected routes

  *Example:* $[d,8,2,3,d,9,6,d,5,7,4,1] \rightarrow [d,9,2,3,d,8,6,d,5,7,4,1]$

- Removing a randomly selected city from one randomly selected route and inserting it into a randomly selected position in a different randomly selected route

  *Example:* $[d,8,2,3,d,9,6,d,5,7,4,1] \rightarrow [d,2,3,d,9,6,8,d,5,7,4,1]$

iii. Define the objective function used to calculate the cost of a solution.

The cost of a route $R_i$ is given by ($n$ is the number of cities in a route):

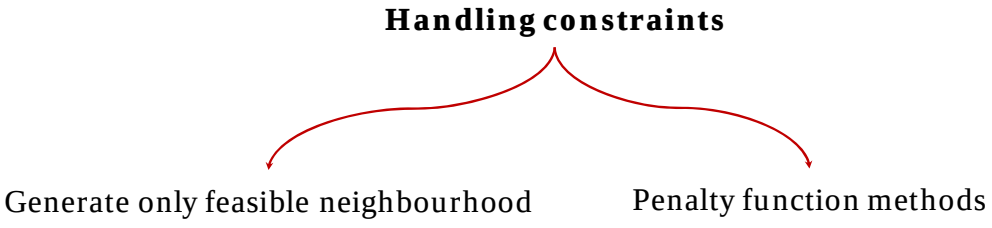
$$Cost(R_i) = Depot(R_{i1}) + \sum_{j=1}^{n-1} D(R_{ij}, R_{ij+1}) + Depot(R_{in})$$

and the total cost of a solution (for three vehicles) is given by:

$$f = Cost(Sol) = Cost(R_1) + Cost(R_2) + Cost(R_3)$$

b) 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?

Constraints can be handled by generating only feasible neighbourhood or by using penalty function methods as illustrated below.




[Slide 48 Lecture 9]

In the first method, no change will be required in the objective function as any solution that includes a route R whose duration > T will be considered as infeasible solution and will be rejected.

In the second method, a penalty $P$ can be added to the objective function as follows:

Assume $d_i$ is the total travel distance of the vehicle $i$ and $v_i$ is its constant velocity

$$f = \sum_{i=1}^{3} \left[ d_i + P\big|_{if\ \frac{d_i}{v_i} > T} \right]$$

## II. Non-Graded Extra Exercises

1. **[Programming Exercise]** Soccer playing robots competitions such as RoboCup and FIRA try to foster AI and intelligent robotics research by providing a standard problem where wide range of technologies can be integrated and examined. These technologies include: path planning, opponent detection, real-time reasoning, playing skills (ball-kicking, ball-passing, ball-keeping, etc…) and cooperative behaviors with teammates.

Path planning is the problem of finding a collision-free path for a robot from a start position to a given goal position, amidst a collection of obstacles. Path planning algorithms can be classified into cell decomposition-based approaches, roadmap methods and potential field-based approaches. In the artificial potential field (APF) approach, it is assumed that the obstacles exert repulsive forces onto the robot, while the target applies an attractive force to it as illustrated in Fig. 4. The sum of all forces determines the subsequent direction and travel speed of the robot.

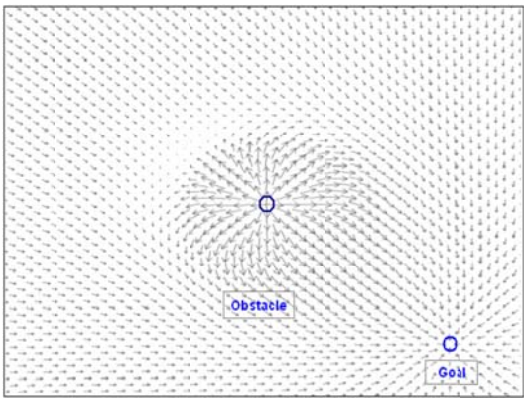



**Fig. 4** Potential Field Path Planning

In a simple soccer playing robot problem, you are given a small soccer field with a set of static circular obstacles spatially located as shown in Fig. 5. It is required to use the artificial potential field (APF) approach to plan a path for the robot from a starting location to the goal.

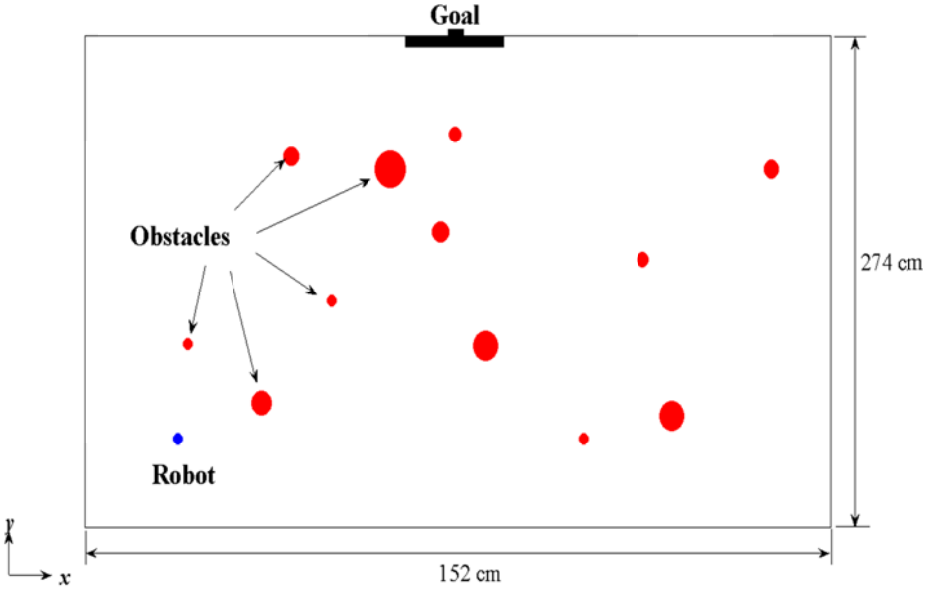



**Fig. 5** RoboCup Small-size League Soccer Field

You are given a MATLAB function (**SolvePathPlan_LS.m**) that solves the path planning problem using a local algorithm. The data of the sample soccer field can be loaded by calling the **LoadRoboCupField.m** function. The local search algorithm proceeds as follows:

1. Start from the given robot location (XYRobot).
2. Search the 8-neighbourhood of the current location
3. Calculate the artificial potential field (APF) at each neighboring location using the **CalcAPF.m** function.
4. Move the robot to the neighboring location with the minimum APF
5. Repeat 2-4 until the goal is reached or the potential field does not improve (local minimum).

One major problem with this local search algorithm is that the robot is usually trapped at local minima of the potential field and this prevents it from reaching the goal. Zhu et al[1] suggest the use of simulated annealing to get out of these local minima and continue moving towards the goal. The simulated annealing search is invoked when the local search algorithm gets stuck at a local minimum and it proceeds as follows:

1. Set $M$ to the location of the local minimum
2. Choose the annealing strategy; give the initial temperature $T_o$ enough high value,
3. Set the current temperature to the initial temperature $T = T_o$
4. Set the current location to the location of the local minimum $L = M$
5. While $T \geq T_f$ (the final temperature) and not escaped from local minimum, do:
   - 5.1 Repeat for a number of iterations $N$:
     - 5.1.1 Pick a random neighbor $L'$ of the current location $L$, Calculate the APF at $L'$: $U(L')$
     - 5.1.2 Set $\Delta U = U(L') - U(L)$
     - 5.1.3 If $\Delta U \leq 0$,
       - Set $L = L'$
       - If $U(L') \leq U(M)$, successful escape
     - 5.1.4 Else set $L = L'$ with probability $P = e^{-\Delta U/T}$ (Pick a random value $a$ within 0 and 1, if $P>a$, set $L = L'$)
   - 5.2 Decrease the temperature using the following geometric rule:

$$T(t) = \gamma T(t-1) \qquad 0.85 \prec \gamma \prec 0.98$$

Report failure if not escaped. Otherwise continue the local search.

Answer the following questions:

i. Use the **SolvePathPlan_LS.m** function to find a path to the goal starting from the following locations: (20, 50), (100, 140), (120, 10). Plot the APF values across the output path in each case. What do you observe?

ii. Implement the simulated annealing algorithm described above and modify the **SolvePathPlan_LS** function to invoke simulated annealing when the robot gets trapped at a local minimum.

[1] Q. Zhu, Y. Yan, and Z. Xing, "Robot Path Planning Based on Artificial Potential Field Approach with Simulated Annealing," Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06), 16-18 October 2006 in Jinan, China.

iii. Run the modified code for the robot starting locations given in (a). Use an initial temperature of 500, a final temperature of 10, a temperature decrease rate of 0.85, and 10 iterations for each temperature. Plot the APF values across the path to the goal in each case. What do you observe?

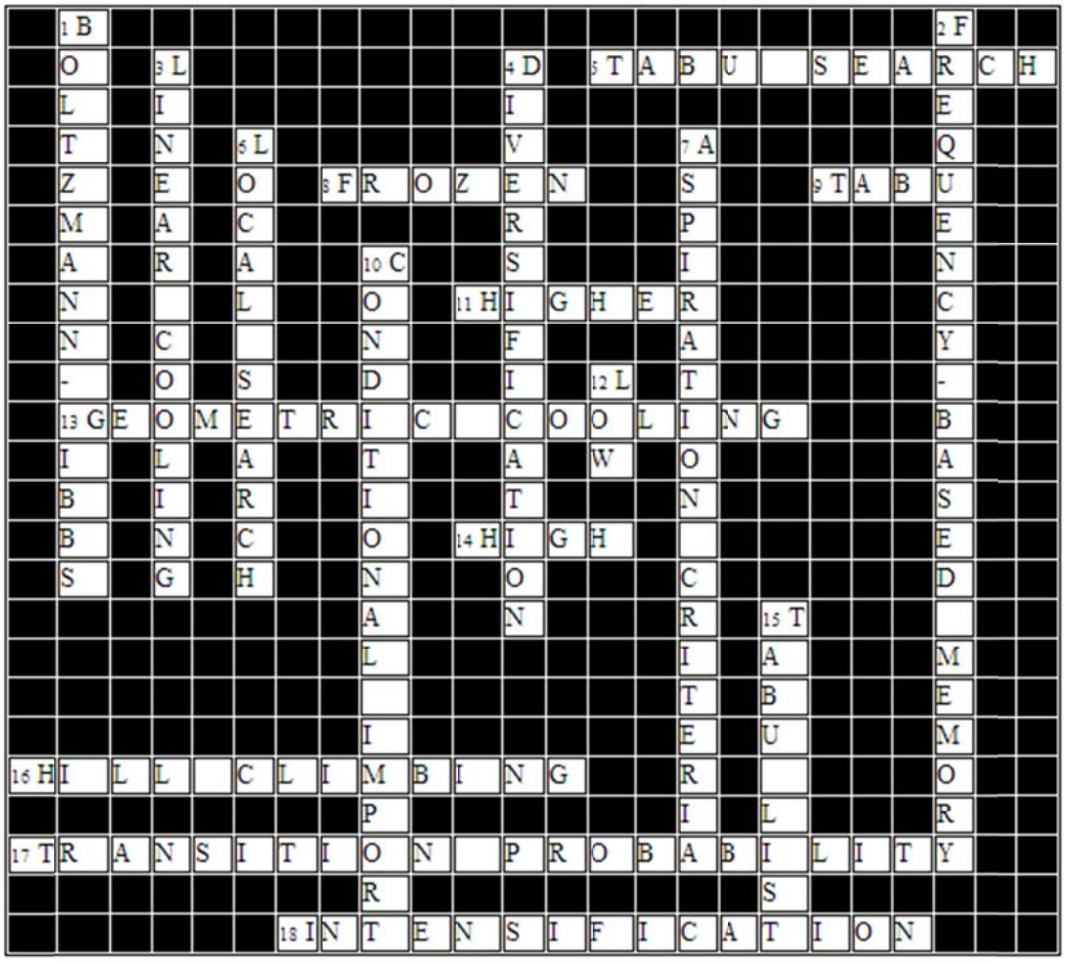
**Notes:**

The following MATLAB files are attached:

- **SolvePathPlan_LS.m**: This function solves the path planning problem using a local search algorithm.
- **CalcAPF.m:** This function calculates the artificial potential field (APF) at a given location in the soccer field.
- **LoadRoboCupField.m:** This function loads the data of the RoboCup soccer field.
- **DrawSoccerField.m, DrawRobot.m:** These functions draw the soccer field and the robot respectively.

**Solution:** Code is attached.

2. Solve the following crossword puzzle



**Across**

5. an iterative neighborhood search algorithm, where the neighborhood changes dynamically.
8. state of the system is at which no better or worse moves are being accepted.
9. a word comes from Tongan, a language of Polynesia, where it indicates things that cannot be touched because they are sacred.
11. the higher/lower the value of cooling factor, the longer it will take to reach the final temperature.
13. a cooling schedule that decreases the temperature by a cooling factor.
14. at high/low temperatures, SA explores parameter space.
16. compared to this search algroithm, the main difference is that SA probabilistically allows downwards steps controlled by current temperature and how bad move is.
17. the probability of acceptance or rejection of neighbouring solutions.
18. a form of exploitation to intensely explore known good areas.

**Down**

1. probability distribution used in transition probability of SA.
2. a long-term memory that maintains information about how often a search point has been visited (or how often a move has been made) during a specified time interval.
3. a cooling schedule in which maximum number of iterations needs to be specified.
4. a form of exploration to cause the search to consider new areas.
6. a search technique that uses only local information of the search space surrounding the current solution to produce a new solution.
7. an approach used to cancel the tabus in case of stagnation. Simulated annealing an optimization process based on the physical annealing process.
10. a way of handling the incoming solution in cooperative tabu searcher in which the own best-so-far solution of each tabu search is replaced by the incoming solution only if the incoming solution is better.
12. at high/low temperatures, SA restricts exploration.
15. a recency-based memory or short-term memory which maintains information about how recently a search point has been visited (or how recently a move has been made).

**Hint:** Spaces and dashes MUST be used if the answer consists of two or more words.