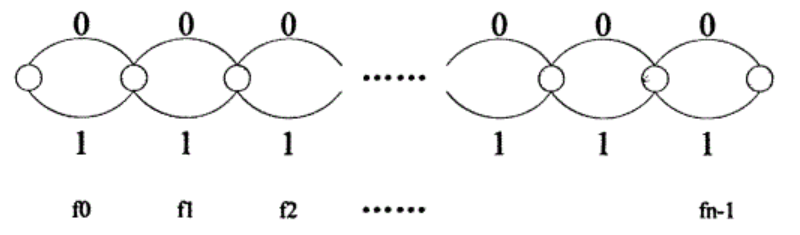
**University of Waterloo**

**Department of Electrical and Computer Engineering**

**ECE 457A: Cooperative and Adaptive Algorithms**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Project Title** | Feature Selection using Ant Colony Optimization | | |  |
| **Group Number** | 13 | | |
| **Team Member** | *Tian Zhu* | *20517833* | *t27zhu@edu.uwaterloo.ca* |
| *CongHao He* | *20589010* | *chhe@edu.uwaterloo.ca* | *10*  *points* |
| *Yiran Tian* | *20580120* | *y62tian@edu.uwaterloo.ca* |
| *YuCheng Liu* | *20558944* | *y698liu@edu.uwaterloo.ca* |

**[2 points] Algorithm and Problem:**

When performing feature selection ant colony optimization, each feature may be modeled as a node where an edge (in our case, the weight are all 1) that ants may take connect them. In this specific solution, the nodes are listed linearly, there are 2 edges connected between 2 neighbor nodes, indicating select or not select. Then, the subset that generates the optimal performance based on our objective cost function can be considered as the set of edge that the ant has taken. 

The probability of a path taken to a feature (i.e. feature that gets selected) depend on the probability parameter given. The evaporation function and its parameter are also defend prior starting the computation. Once the subset of features that optimizes the cost function is found or the maximum iteration has been reached, the algorithm will terminate.

The space complexity depend on the total number of feature. Depending on the variation of the algorithm, a tabu list may be used. Generally, the space complexity is O( features ).

For time complexity we can generalize ACO looping around 3 general steps at each iteration: constructing a solution, applying local search, update pheromone. The first step is O(features)) since all feature may be fully connected. The second step depend on the objective cost function. Pheromone should be updated to the entire set of edge that ant has traveled hence O((features)). The overall complexity is approximately O(iterations \* (features \* ants)).

**[2 points] Algorithm Scalability:**

Discuss what will happen if you increase your search space.

If feature increases, space complexity will linearly increase due to how we adapted the algorithm to our problem (see figure in first section). Hence to represent such graph, linear space complexity is needed. The same applies for storage space associated with data of phoremone. Our time complexity is also related to the number of ants that we decide to use, and this is a trade off between accuracy and time complexity, because with more ants for each iteration, we will be able to increase our explore space, which will increase our accuracy. However, for each feature that we add in, it will affect all the ants, so the time complexity will increase proportionally the number of ants.

As the number of features increase, there are more possible path for ants to explore at each iteration. Considering again the three steps within each iteration. Constructing the solutions will have its time increased linearly proportional to the increase of features. From the diagram in section 1, each subsequent node is only connected to the previous one via two edges, 0 and 1. The same can be said for applying local search, the complexity increase linearly depending on number of ants. Updating pheromone is also directly proportional to the number of features.

**[4 points] Results:**

Share the results achieved by the algorithm w.r.t. your problem and dataset. Comments on the solution(s) optimality.

Final solution of ACO:

['Best', ['radius\_mean', 'perimeter\_mean', 'area\_mean', 'smoothness\_mean', 'concave points\_mean', 'symmetry\_mean', 'fractal\_dimension\_mean', 'radius\_se', 'texture\_se', 'compactness\_se', 'concave points\_se', 'fractal\_dimension\_se', 'texture\_worst', 'concavity\_worst', 'symmetry\_worst', 'fractal\_dimension\_worst'], 0.986013986013986]

The searching processes of each ant are independent from each other, and the ants only use “pheromone” left to communicate. Thus, the ACO method can be seen as issuing distributed search agents, and the ants start searching at different aspects of searching space, thus the solutions have higher reliability, and the results are collected globally.

The optimality of the solution is dependent on several parameters: the evaporation rate, the pheromone deposit algorithm, the number of ants, and maximum iterations. The evaporation rate determines how quickly does ants globally forget about the past solutions. For the problem of feature selection, the evaporation rate is set to a low value because early convergence is not a concern. The pheromone deposit algorithm is set to reevaluate based on the final score of the solution reached. The higher the score, the greater amount of pheromone is added. The number of ants determines the probability of reaching an optimal solution because the greater the number of ants, the greater the exploration of solution space, thus better chance of reaching a global optimum. The maximum iteration determines the exploitation of the solution. For our problem, the iterations required is low.

The final solution reached by the ACO is better than any of the previous algorithms. It reached a solution of 0.986 accuracy, thus deeming itself a solution that can possibly reach the global optimum. Since the time complexity of ACO is low, the computation power required is relatively low, thus all the features can be run at same time. The space complexity of the problem is O(features\_numbers) because we only need to keep track of the best solution and the current solution and the current local best solution. Overall, the solution is more optimal than all other algorithms so far.

**[1 points] Performance Comparison [if applicable]:**

Comment on the algorithm performance and the optimality of the solution achieved compared to the results you previously achieved. This section should be used for all reports other than the Graph Search Algorithms.

Final solution of Tabu:

(['texture\_mean', 'perimeter\_mean', 'smoothness\_mean', 'compactness\_se', 'texture\_worst', 'perimeter\_worst', 'smoothness\_worst'], 0.972027972027972)

Final solution of BFS:

['area\_worst', 'compactness\_worst', 'concavity\_se', 'perimeter\_worst', 'radius\_worst', 'texture\_worst'] 97.2%

Final solution of SA:

('Best', ['radius\_mean', 'perimeter\_mean', 'area\_mean', 'perimeter\_se', 'area\_se', 'smoothness\_se', 'concavity\_se', 'concave points\_se', 'symmetry\_se', 'fractal\_dimension\_se', 'texture\_worst', 'compactness\_worst', 'concavity\_worst', 'concave points\_worst', 'symmetry\_worst'], 0.9790209790209791)]

Final solution of GA:

Features Selected: ['texture\_mean', 'radius\_worst', 'texture\_worst', 'area\_worst', 'compactness\_worst']

[('Current', ['texture\_mean', 'radius\_worst', 'texture\_worst', 'area\_worst', 'compactness\_worst'], 0.972027972027972), ('Best', ['texture\_mean', 'radius\_worst', 'texture\_worst', 'area\_worst', 'compactness\_worst'], 0.972027972027972)]

Final solution of ACO:

['Best', ['radius\_mean', 'perimeter\_mean', 'area\_mean', 'smoothness\_mean', 'concave points\_mean', 'symmetry\_mean', 'fractal\_dimension\_mean', 'radius\_se', 'texture\_se', 'compactness\_se', 'concave points\_se', 'fractal\_dimension\_se', 'texture\_worst', 'concavity\_worst', 'symmetry\_worst', 'fractal\_dimension\_worst'], 0.986013986013986]

ACO finds the best solution across all methods. ACO has better convergence than GA, and the searching efficiency is the highest across the methods. Since ACO reduced computation work drastically, different from other methods, we are able to run all the features at one time, therefore the search is more thorough.

The solutions of ACO is constructive building, which means the solution is built during program running. Other methods are generative building. Every feature in the solution is the node that the ants are finding, when the ants are deciding which path to go, they use heuristic information brought by existing nodes. While other methods, for example, GA and SA, perform searching with randomly generated solutions, and do not utilize information carried by problem itself. Especially GA, relevancy between features is not taken into account. Therefore, ACO increases searching efficiency by using “pheromone”, and irrelevant features can be prevented.