* 1. Since determining a max cut is equivalent to partitioning the network into two parts, with every vertex in the network belonging to one and only one partition, the size of the solution space is O(2N)
  2. The solution will be represented as a vector with entries corresponding to each of the vertices appearing in one of the partitions of the network. Since the network is being partitioned into only two parts, all vertices not included in this vector are implicitly in the other partition. This representation is easy to keep track of, takes up a small amount of memory, and is always a solution to the problem at hand, so there is no need to undergo costly analysis to determine if a newly generated vector is a solution or not.
  3. The initial solution is generated by randomly selecting some subset of the entire vertex set to be in a given partition.
  4. The neighborhood of a given solution is a single removal OR a single addition of a vertex to the solution vector.
  5. Since the above defined neighborhood makes no assumption about which vertex may be selected, there are N other vectors that are neighbors of the current solution vector.
  6. The initial Temperature is O(Total Edge Weights). This value is large enough to allow almost any transition to occur, which seems reasonable at the beginning since the initial solution vector was generated pseudo-randomly, but not too large to interfere with the proper functioning of the Temperature Reduction Function.
  7. The algorithm I’ve constructed runs for O(N2) iterations, O(N) for each temperature and O(N) temperatures. This allows for, on average, almost every vertex to attempt a move at every temperature, which seems like a reasonable decision.
  8. I’ve chosen a linear Temperature Reduction Function for my algorithm, as it is easy to implement and seems like a reasonable choice. Another potential function would be a logarithmically decreasing function, as this would quickly move from very high temperatures to much lower temperatures, but then spend many more iterations at slowly decreasing, lower temperatures.
  9. I have not introduced a stopping condition into my algorithm, as it seems reasonable to allow the algorithm to explore the space for the preset number of iterations and for all temperatures.

1. See MATLab file.
   1. Same as in the SA case, as this is an easy and convenient representation for the solution. Just as easy to modify in the Tabu setting as in the SA setting.
   2. Same as in the SA case.
   3. Same as in the SA case, only generate a list of neighbors and not just a single neighbor. Generated neighbor set can be referenced by simply a subset of all vertices in the graph due the nature of the neighborhood.
   4. Once a move is made, the particular vertex that changed which partition it was in is added to the Tabu List, which is simply a list of Tabued Vertices and the number of iterations it has left in the Tabu List.
   5. I selected the Tabu tenure to be O(N), as it seems reasonable to restrict a second movement of a vertex until at least almost all other vertices have been moved once. However, this tenure should not be too long, or too many neighbors will be restricted, which could potentially drive the solution far away from the optimal solution.
   6. The only reasonable aspiration criteria for this problem is allowing a Tabued move to take place only if it generates a solution that is better than any previously seen solution.
   7. The stopping condition I’ve implemented stops the algorithm if N iterations are completed without a new global best solution being found. This seems reasonable as it allows for almost every vertex to be considered for a move without finding a move that creates a solution that is better than the best one found so far.
2. See MATLab file
3. The Tabu Algorithm seems to work better for this problem, likely because each iteration explores a larger portion of the solution neighborhood.