Adaptive Complex Networks: Initial Ideas and Results

Overview:

We would like to extend the evolutionary approach to discover networks that are not only efficient, robust, inexpensive, but also adaptive. By an adaptive network we imply one that we can reconfigure to one of several desired configurations with low transition costs. For instance, one desired configuration may want a minimum guaranteed efficiency, and another may want guaranteed robustness.

We believe that such adaptive networks will be useful in applications like network centric warfare, adaptive supply chain systems, labor supply networks in information technology, disaster management, simulating terrorist cell networks, etc.

Initial Problems:

We now want to have two measures of cost: infrastructure cost and transition cost. The first is the number of edges in the network. The second in the number of edges that need to be reconfigured to move to a desired configuration. In general, all edges in the network may not be explicitly required for a given desired configuration. In other words, adaptive networks may have higher infrastructure costs than each of the desired configurations.

Problem 1:

We know that for infrastructure cost of n-1, a straight line has maximum robustness, and a star has maximum efficiency. What is the best adaptive network that has an infrastructure cost of n - 1, and the minimum transition cost of moving to either a circle or a star? Can we come up with an evolutionary mechanism that discovers this network?

Problem 2:

The same as Problem 1, but we allow higher infrastructure costs. For instance, if the infrastructure cost is (n+n/3) we can get a star in which each spoke is actually a cycle or circle of 4 nodes. There are n/3 such spoke-circles. All such spoke-circles share one common node---the center of the star. To transition to a regular star this takes n/3 edge re-configurations, and similarly to transition to a regular circle. Can we discover this configuration.

Problem 3:

Generalizes Problem 2. Given an infrastructure cost C(n), an efficiency requirement E(n), and an robustness requirement R(n), discover a network with infrastructure cost at most C(n) and which takes equal cost in moving to (1) a network with efficiency E(n) and (2) and network with robustness R(n).

Issues:

1. What are useful definitions of adaptability?
2. How do we handle possible service disruptions during transition? Are notions of overshooting from control theory helpful here?
3. Develop efficient genetic algorithms for discovering adaptive solutions.

Approach:

We decided to consider the Problem 1 described above and find some results. We decided that the star and the straight line as the two optimal target configurations. So, we need to find a tree topology that is optimally adaptive with respect to star and straight line. In other words, we need to find a topology that has the lowest transition cost to either star, C(star), or straight line., C(line) -- Minimize max(C(line), C(star)). It is also desirable that, |C(line) - C(star)| <= \epsilon. That is, the costs of moving either to a star and a straight line are comparable.

Computing transition costs:

We developed a simple method to compute the transition costs based on graph edit distance, which is a standard metric to measure the distance between two graphs. Edit distance is the number of edge additions, deletions and replacements needed to reach from one graph to another. In our case, since we are only dealing with trees, we only have edge replacements. So, the cost of transition is the number of edge replacements required in a tree to transform it into a straight line (or a star).

Intuitively, we can think of this as "redistributing" node degrees in a tree. Moving from a tree S to a tree T entails moving from a *degree set* S to a *degree set* T. Here, we are using the term *degree set* instead of *degree sequence* as the nodes are unlabeled, and their order does not matter. The total degree in the tree remains the same, which is 2(n - 1).

For example, a star with 8 nodes has the following degrees, S = {7, 1, 1, 1, 1, 1, 1, 1}, and a straight line with 8 has degrees, T = {2, 1, 1, 2, 2, 2, 2, 2}. So, a transformation here is basically a "redistribution of node degrees". In this example, we simply take 5 degrees from the node with a degree of 7 in S and add a degree 1 to 5 of the 7 remaining nodes. The minimal "amount" of degrees redistributed is the cost of transforming from S to T.

While this technique can be used to compute the transition cost between any pair or arbitrary trees, we are interested in computing C(star) and C(line). The above technique implies the following:

1. The cost of transforming an arbitrary tree S into a straight line is, C(line) = (no. of nodes with a degree 1 in S) - 2. Think of transforming an arbitrary tree into a straight line as "tying lose ends" together to form a long thread; you tie all but 2 lose ends together.
2. The cost of transforming an arbitrary tree S into a star is, C(star) = (n - 1) - (the highest node degree in S), where n is the number of nodes. Essentially, you move the edges that are not already attached to the biggest hub in the tree.

Implementation:

We use a genetic algorithm technique to find the tree topology that minimizes the transition costs to star and straight line. We use the following fitness function: f = 1/max(C(star), C(line)). Thus, f = 1.0, when the maximum transition cost to either star or a straight line is 1 i.e. only a single edge replacement; it reduces as the number of edge replacements required increases. We ignore cases where the denominator in the above expression is 0, because they occur only when n <= 3. Also, note that the above fitness function is used only to compare two trees of the same size (i.e. same n). We need to include a normalizing factor in case we want to compare trees of different sizes.

The genetic algorithm (GA) approach is briefly described below:

1. Initialization: We generate a large number of random trees to form the set of initial seed graphs. In the present experiments, we generate 100,000 random trees. While a significant percentage of these might be disconnected, this fact is not necessarily a setback.
2. Selection: We compute the fitness of each member of the initial population, and select 100 of them into the mating pool. Selection is done using roulette-wheel based selection, which is a standard GA technique.
3. Crossover: We do an all pairs crossover on the mating pool to generate about 20000 (~ 100 X 100 X 2) offsprings. We make suitable "corrections" such that the offsprings are not invalid (for example, they should not have more than n - 1 edges).
4. Mutation: Typically in a GA, a small percentage of the offsprings are subject to random mutations (edge addition, deletion, replacement). However, in the present experiments, we have not used mutations.

Once step 4 is done, the algorithm goes back to step 2 and repeats over a predefined number of generations or until it reaches "saturation". In the present experiments, we run the GA over 10 iterations. We observe that the crossovers don't make a major impact as we tend to find optimal topologies in the first few generations. However, this might be because of the small size of our networks -- we are using n = 4 through n = 30.

Results:

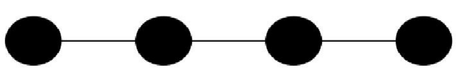
We can observe some patterns. However, we need more studies to understand if these patterns are interesting.

(1) Transition costs seems to follow the relation, C = n/2 - c, where c is a small positive constant. There are a few outliers; we need more studies to establish this.

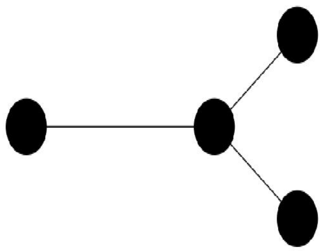
(2) C(star) and C(line) seem to be very close. In fact, we observe an interesting pattern for 4 < n <= 15. When n is odd, C(star) = C(line). But when n is even, we see the following three pairs of costs: (a) C(star) = C(line) (b) C(star) = C(line) + 1 (c) C(line) = C(star) + 1. We need more experiments to check if this holds for n > 15.

(3) We conjecture that the degree distribution of these topologies are interesting. They seem to follow some power law, though this needs to be confirmed.

(1) n = 4, Cost = 1

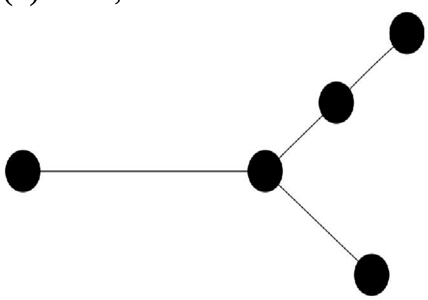


C(star) = 1, C(line) = 0



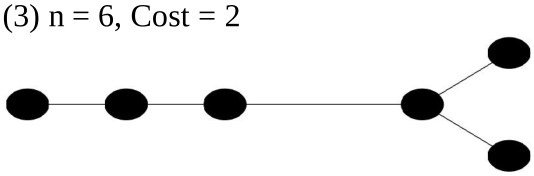
C(star) = 0, C(line) = 1

(2) n = 5, Cost = 1

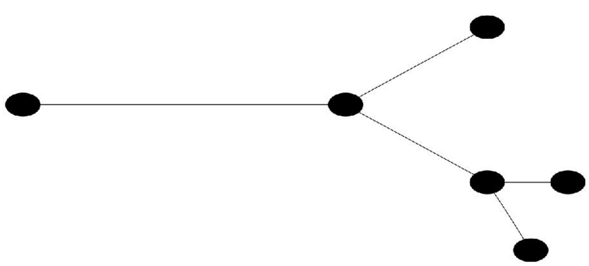


C(star) = 1, C(line) = 1

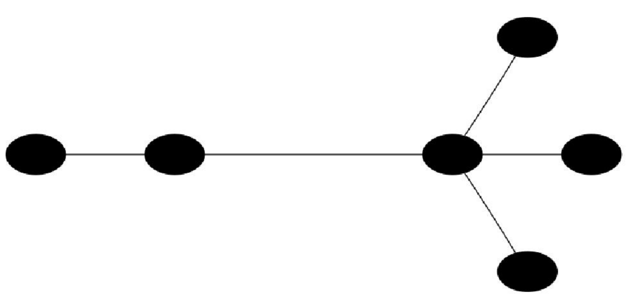
(3) n = 6, Cost = 2



C(star) = 2, C(line) = 1

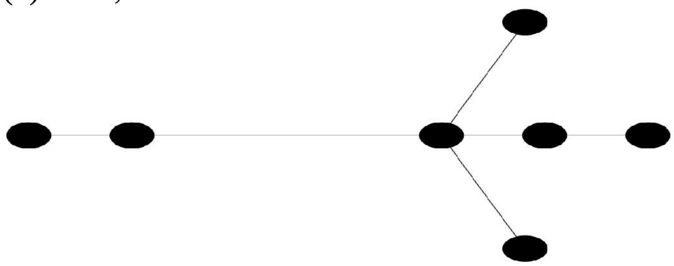


C(star) = 2, C(line) = 2



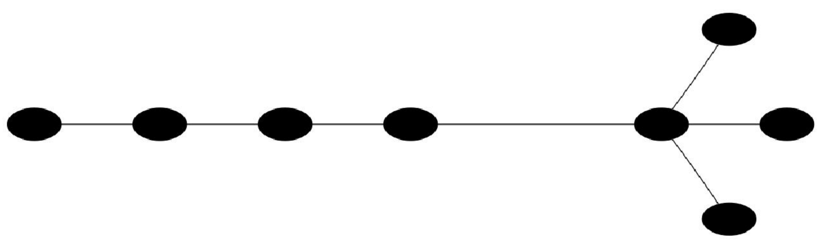
C(star) = 1, C(line) = 2

(4) n = 7, Cost = 2



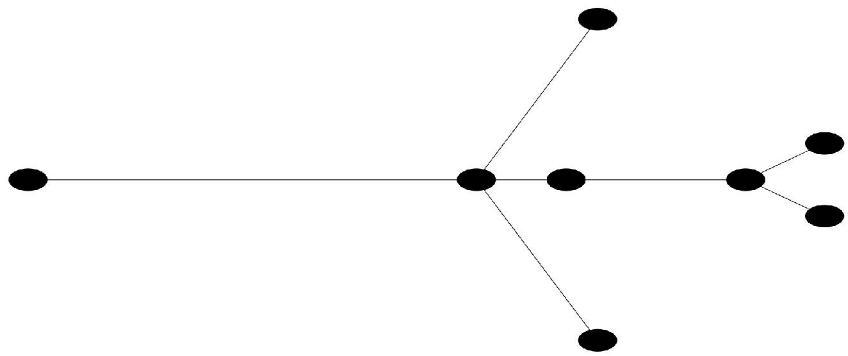
C(star) = 2, C(line) = 2

(5) n = 8, Cost = 3



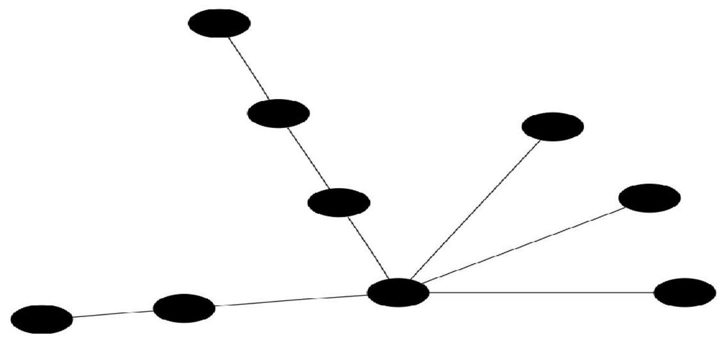
C(star) = 3, C(line) = 2

C(star) = 2, C(line) = 3



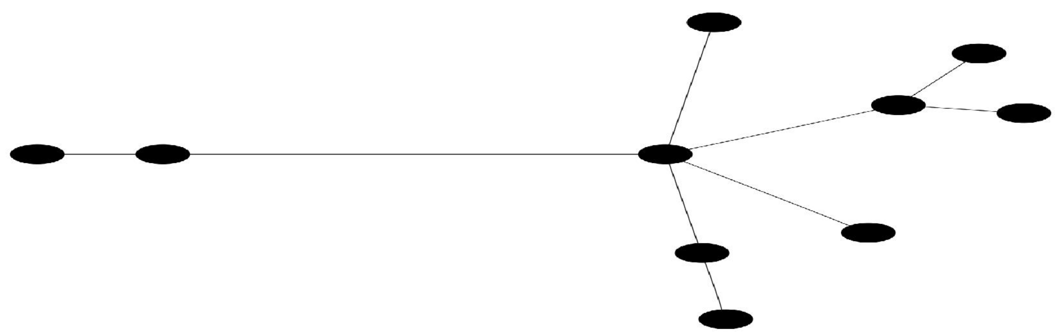
C(star) = 3, C(line) = 3

(6) n = 9, Cost = 3

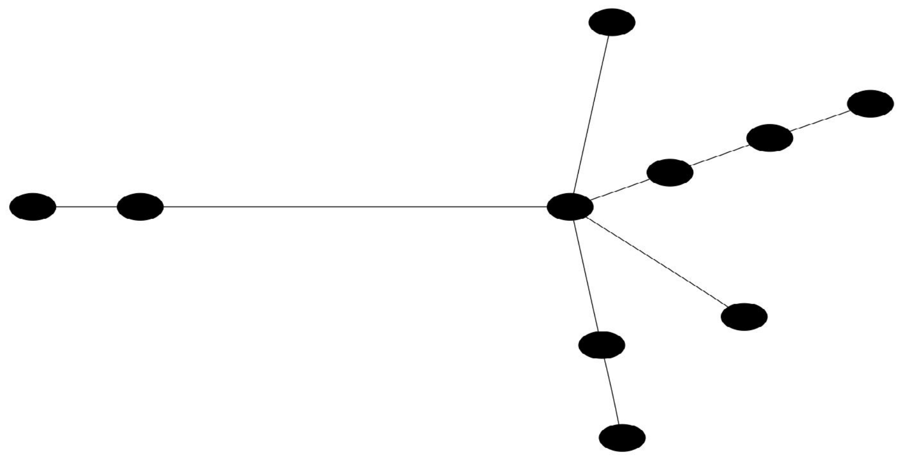


C(star) = 3, C(line) = 3

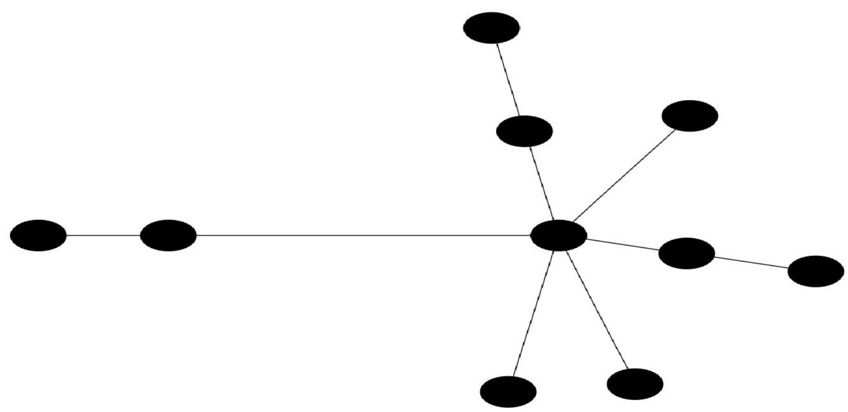
(7) n = 10, Cost = 4



C(star) = 4, C(line) = 4

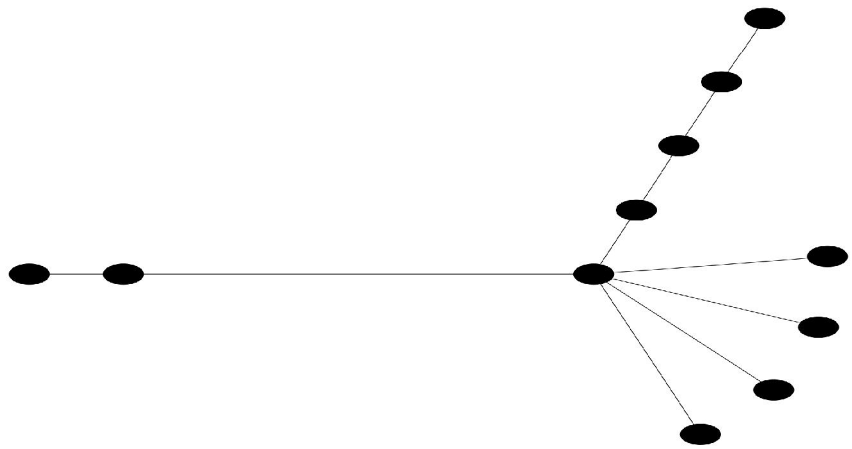


C(star) = 4, C(line) = 3



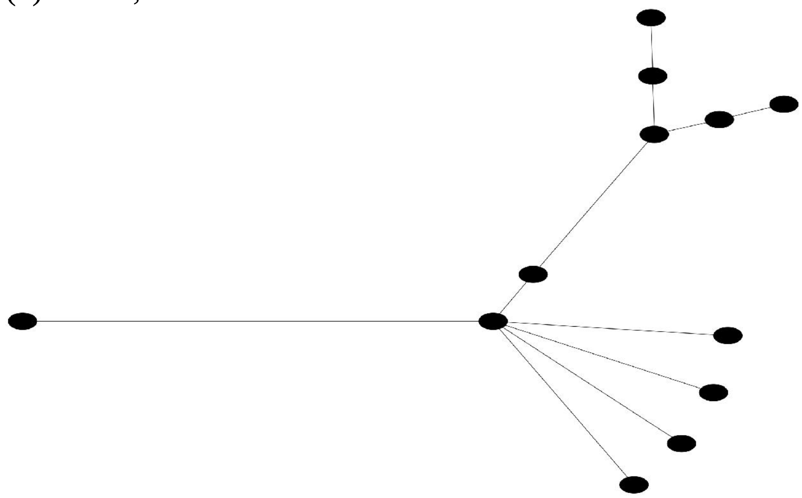
C(star) = 3, C(line) = 4

(8) n = 11, Cost = 4

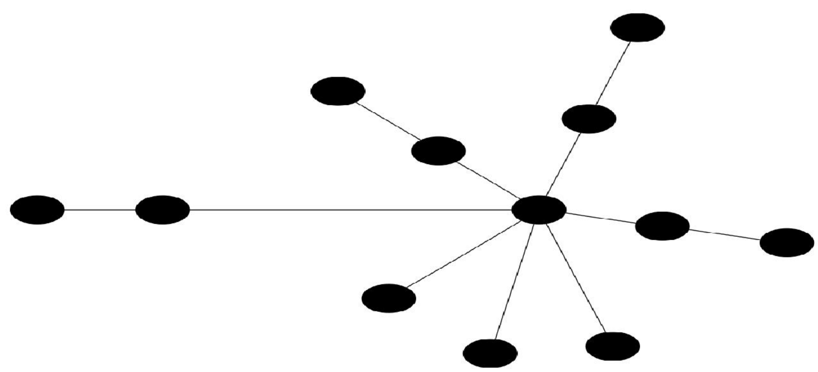


C(star) = 4, C(line) = 4

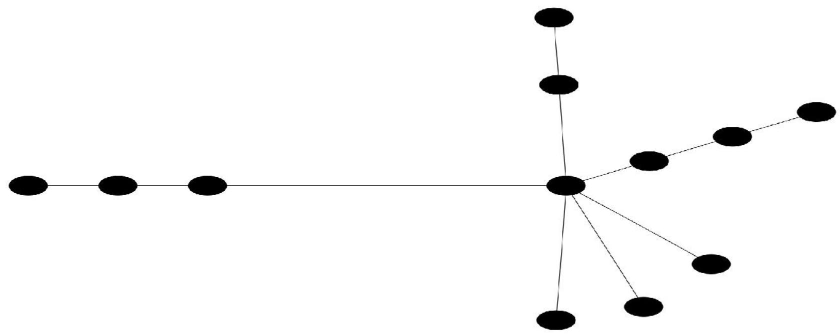
(9) n = 12, Cost = 5



C(star) = 5, C(line) = 5

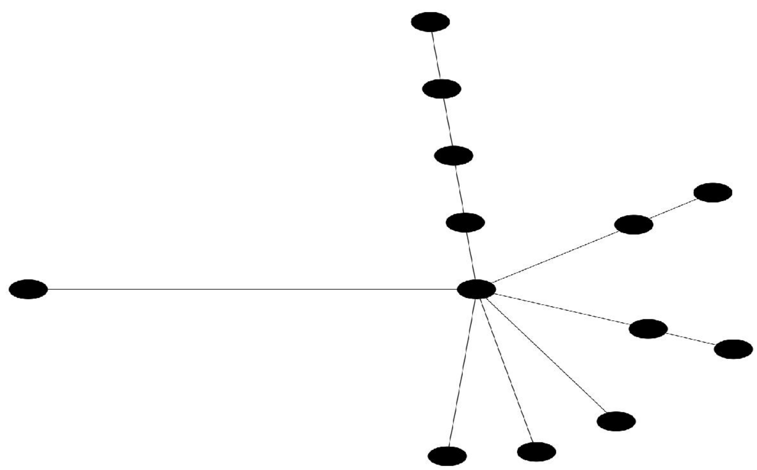


C(star) = 4, C(line) = 5



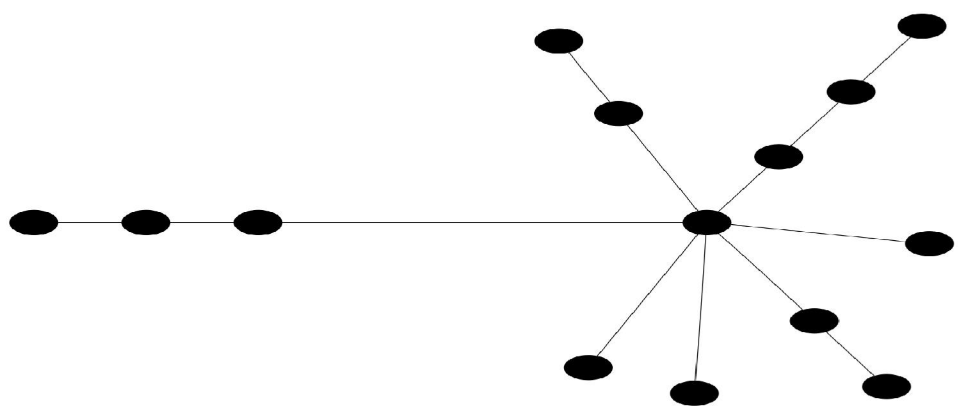
C(star) = 5, C(line) = 4

(10) n = 13, Cost = 5

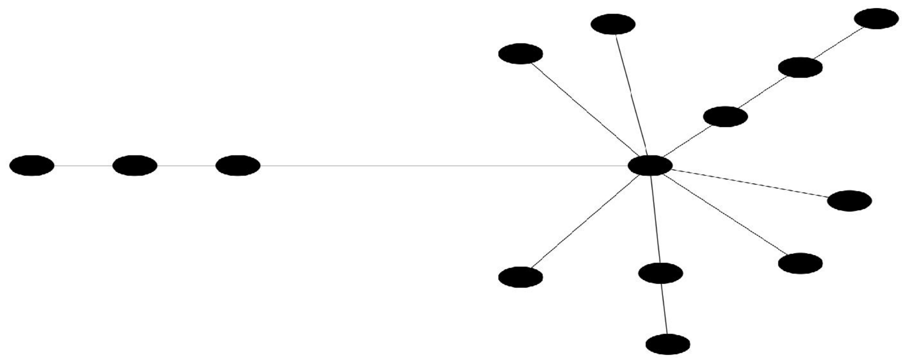


C(star) = 5, C(line) = 5

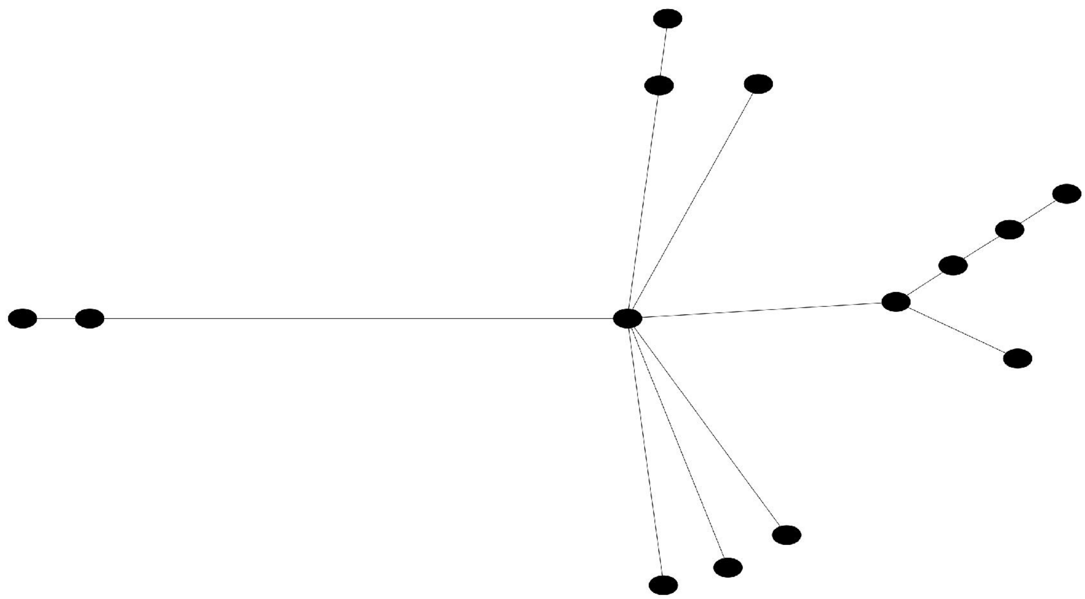
(11) n = 14, Cost = 6



C(star) = 6, C(line) = 5

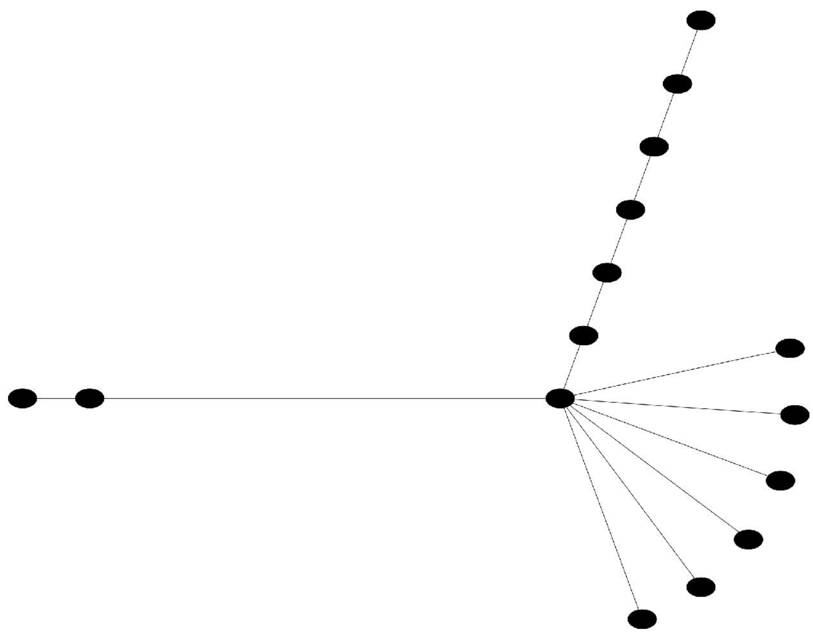


C(star) = 5, C(line) = 6



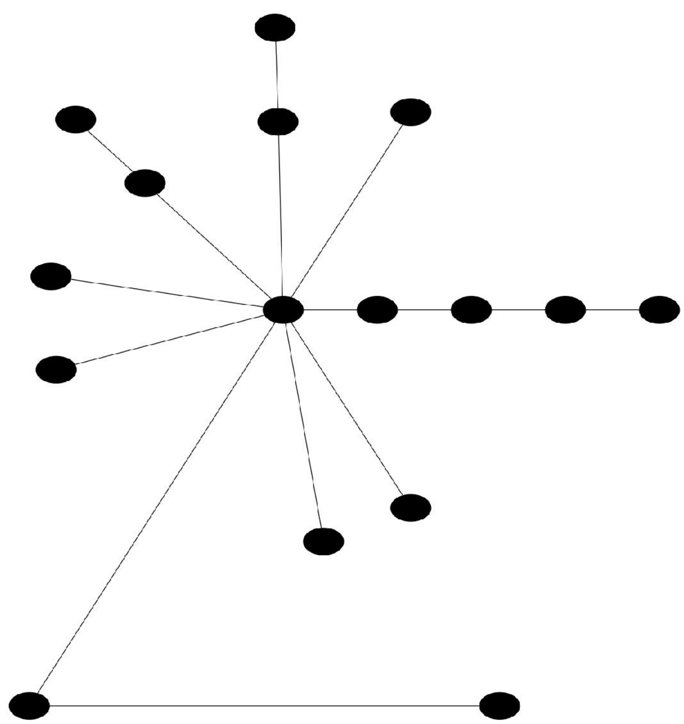
C(star) = 6, C(line) = 6

(12) n = 15, Cost = 6

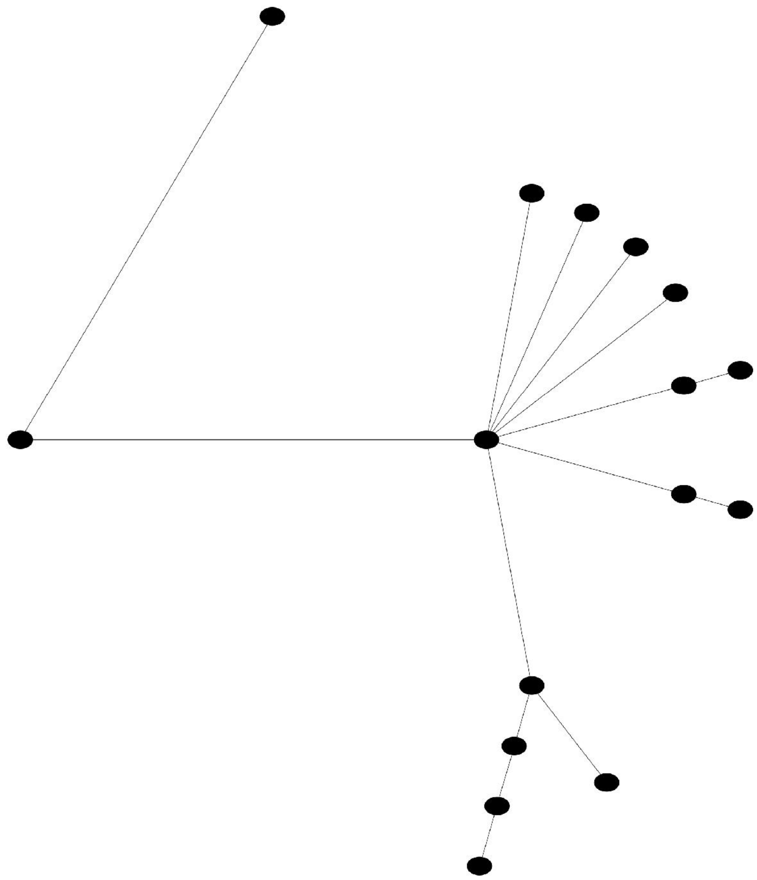


C(star) = 6, C(line) = 6

(13) n = 16, cost = 7

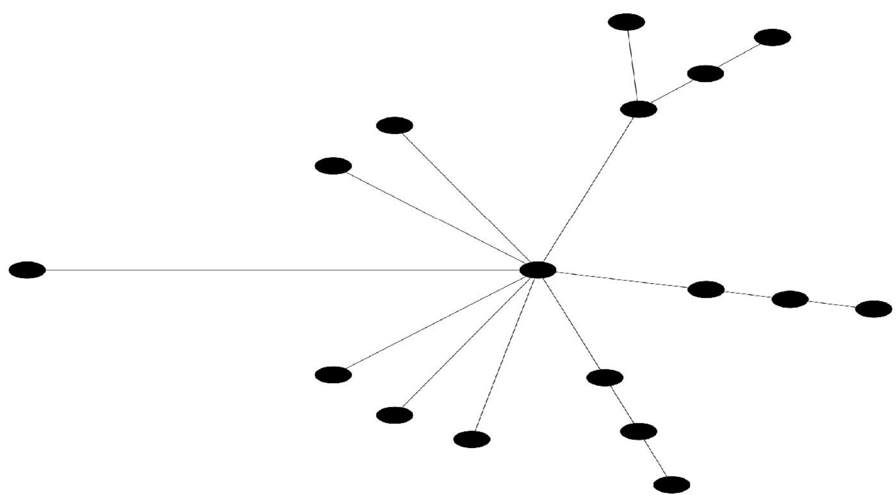


C(star) = 6, C(line) = 7



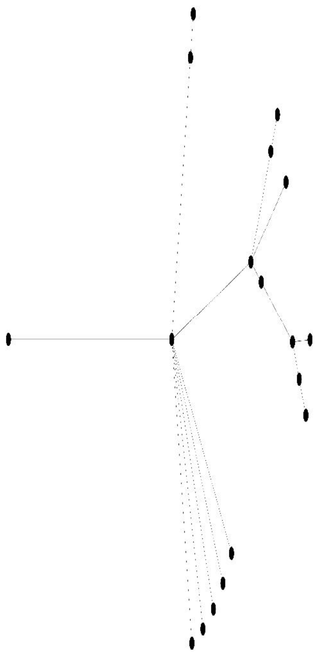
C(star) = 7, C(line) = 7

(14) n = 17, Cost = 8

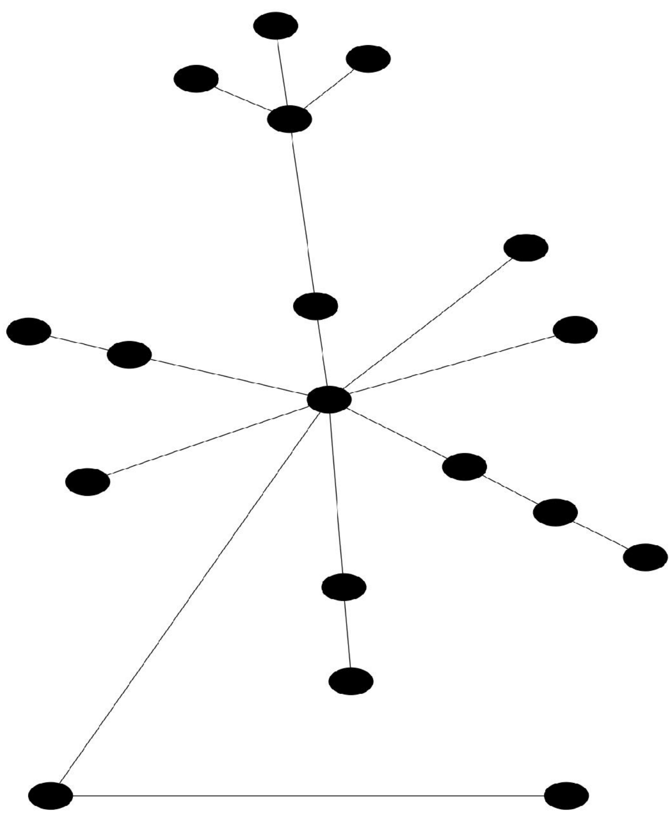


C(star) = 7, C(line) = 8

(15) n = 18, Cost = 9

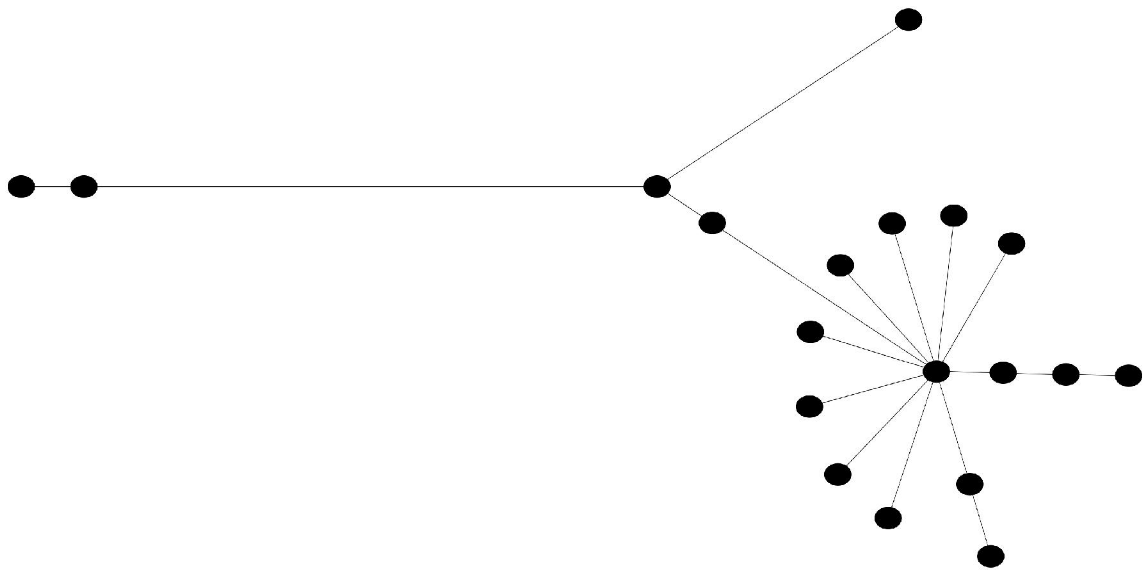


C(star) = 9, C(line) = 9



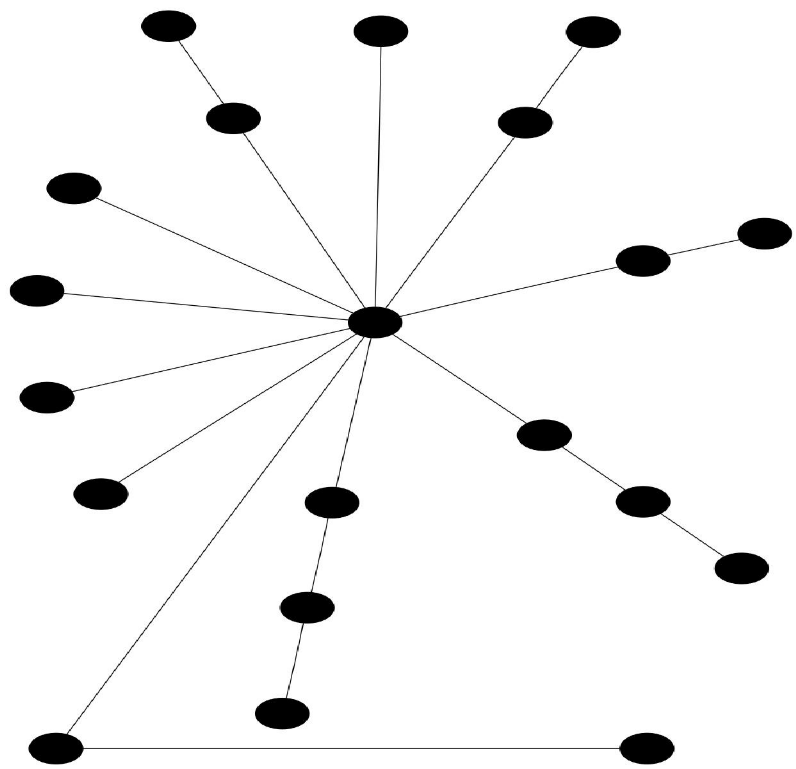
C(star) = 9, C(line) = 8

(16) n = 19, Cost = 10



C(star) = 7, C(star) = 10

(17) n = 20, Cost = 9



C(star) = 8, C(line) = 9