Research Article
Heuristics for Synthesizing Robust Networks with a Diameter Constraint

Harsha Nagarajan, 1 Peng Wei, 2 Sivakumar Rathinam, 1 and Dengfeng Sun 2

1 Department of Mechanical Engineering, Texas A&M University, College Station, TX 77845, USA
2 School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN 47907, USA

Correspondence should be addressed to Sivakumar Rathinam; srathinam@gmail.com

Received 28 May 2013; Revised 4 November 2013; Accepted 13 November 2013; Published 5 January 2014

Academic Editor: John Gunnar Carlsson

Copyright © 2014 Harsha Nagarajan et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Robustness of a network in the presence of node or link failures plays an important role in the design of the network. A key factor that quantifies this robustness is the algebraic connectivity of the network. In this paper, the authors address the problem of finding a network that maximizes the algebraic connectivity of the network while ensuring that the length of the shortest path joining any two nodes in the network is within a given bound. This problem is a simpler version of a system realization problem that arises in several applications including biomedicine, mechanical systems, and the design of Very Large Scale Integrated (VLSI) circuits [1]. We are mainly interested in addressing this problem as it arises in the coordination of Unmanned Aerial Vehicles (UAVs) and in the design of transportation systems.

Several UAV applications require a mobile ad hoc network (MANET) to be deployed to facilitate the exchange of information between the vehicles in the network. Each vehicle represents a node in the MANET and the communication links between the vehicles represent the edges in the MANET. The basic problem here is to synthesize a collection of communication links in the MANET so that all the vehicles can communicate with each other, the number of links on a communication path between any two vehicles is at most equal to a given number, and the MANET is well connected. Algebraic connectivity serves as a measure of well connectedness of a network and can be used to identify good network topologies.

The algebraic connectivity of a network is defined as the second, smallest eigenvalue of the Laplacian matrix associated with the network. In applications where a node or a link (an edge) can randomly fail, algebraic connectivity is specifically considered to be a superior measure of connectivity compared to some of the other well-known measures such as the node or edge connectivity. Node (edge) connectivity specifies the minimum number of nodes (edges) that must be removed for the network to be disconnected. The node and the edge connectivity of any spanning tree connecting all the nodes is equal to 1. Therefore, measures such as the node or edge connectivity cannot distinguish between a spanning tree, that is, a star versus a spanning tree, that is, a Hamiltonian path. However, star networks are considered to be more robust towards any random failures of nodes or edges than a Hamiltonian path. Algebraic connectivity serves as a better measure in these cases because the algebraic connectivity of a star network is much greater than the algebraic connectivity of a Hamiltonian path.

In addition to UAV applications, algebraic connectivity has also been used as a measure of well connectedness in...
the design of transportation networks. For example, an air
transportation network consists of airports and flight routes
between airport pairs [2]. Usually, an undirected graph is
used to describe an air transportation network [3–6] where
the set of nodes represents all the airports and the set of
edges represents all the flight routes. The topology or the
structure of this network plays a vital role in determining
the average time spent by the flights in the air and at airports,
the average number of flights waiting in queues over all
the airports, and the average number of hops or legs in a
passenger’s itinerary. The network also needs to be robust
towards unpredictable node or link failures that may happen
due to airline budget cuts, weather hazards, and economic
policies. There are several studies [4, 7–9] that have aimed
to relate the performance of an air transportation network in
terms of delays and failures (as mentioned above) with
the topology or the structure of the transportation network. The
overall approach here is that if a specific topology of structure
is better, then one can develop the required mathematical
tools to design a transportation network with the desired
topology. Specifically, the authors in [4, 7–9] have shown
that one of the factors that characterizes the performance
of a network is its algebraic connectivity. The authors in [4]
have demonstrated that an air transportation network for the
United States with a higher algebraic connectivity exhibited
superior performance in terms of average flight times, delays,
queue length, and the distance traveled by the passengers.
In order to restrict the number of legs or stops incurred by
the passengers as they travel, it is important to synthesize
a network with additional constraints that limit the number
of edges present in the path joining any two nodes in the
network.

In this paper, we restrict our attention to networks that are
spanning trees. We consider the basic mathematical problem
of synthesizing a spanning tree that maximizes the algebraic
connectivity of the spanning tree subject to a limit on the
diameter of the tree. The diameter of a tree is defined as
the number of edges present in the longest path connecting
any two nodes in the tree. Henceforth, we will refer to this
problem as the algebraic connectivity problem. It is well
known that this problem is NP-Hard [10]. Finding an optimal
solution to this problem seems to be very difficult even for
a network with 8 or 9 nodes [1, 11]. For n nodes, the number
of distinct spanning trees available to connect all the nodes
is \( n^{n-2} \); therefore, for 8 nodes, there are 262144 combinatorial
possibilities. As finding an optimal solution seems to be very
challenging, we focus on developing good heuristics in this
paper.

The algebraic connectivity problem has received scant
attention in the literature. In [12], the authors consider an
unweighted version of the algebraic connectivity problem
with no connectivity constraints. In [1], the authors explain
the importance of this problem in the context of several
applications and present an optimal algorithm to solve the
problem with no connectivity constraints. In [13], Wei and
Sun study a simpler variant of this optimization problem
which aims to add k additional edges to strengthen the
network. In [11], the authors solve the algebraic connectivity
problem in the context of UAV applications. The following
are the contributions of this paper.

(i) We develop heuristics based on k-opt and tabu search
methods to find good feasible solutions to the problem.

(ii) We present computational results to corroborate the
performance of the proposed algorithms. These
results indicate that the 3-opt heuristic performed the
best among the proposed heuristics while the 2-opt
heuristic provided a good trade-off between finding
good solutions and the required computation time.

The rest of the paper is organized as follows. In Section 2,
we formulate the problem. Local search heuristics like k-opt
and tabu search are presented in Sections 3 and 4, respectively.
In Section 5, we evaluate the performance of our algorithms
via simulations. Section 6 concludes this paper.

2. Problem Formulation

Consider an undirected graph \( G := (V; E) \) with \( V \) denoting
all the \( n \) nodes and \( E \) representing all the edges in the graph.
Each edge in the graph is associated with a weight which acts
as a proxy to the likelihood of the edge to fail [14]. An edge
is assigned a larger weight if the edge is a stronger link and
is not likely to fail; on the other hand, an edge is assigned
a relatively smaller weight if the edge is more likely to fail. Let
\( w_e \) represent the edge weight associated with the edge \( e \in E \).
For any edge \( e \in E \), let \( x_e \) represent a binary variable that
specifies if edge \( e \) is present in the chosen network or not; \( x_e =
1 \) if \( e \) is chosen and \( x_e = 0 \) otherwise. Let the incidence vector,
\( x \in \{0, 1\}^{|E|} \), denote the entire network with the component
\( x_e \) corresponding to \( e \in E \). Let \( c_i \) denote the \( i \)th column of
the identity matrix \( I_n \) of size \(|V| = n \). If the nodes \( i \) and \( j \)
are connected using edge \( e \), let \( L_e = w_e (c_i - c_j) \otimes (c_i - c_j) \)
where \( \otimes \) represents the tensor product. Using this notation,
the Laplacian [4, 12, 15] of the chosen network may be defined
as follows:

\[
L(x) = \sum_{e \in E} x_e L_e .
\]  

(1)

Let \( \lambda_2(L(x)) \) denote the algebraic connectivity of the
chosen network. To simplify the problem, we restrict our
search for optimal networks from the set of all the spanning
trees. A spanning tree connects all the nodes in the network
and contains exactly \(|V| - 1\) edges. As stated by Fiedler in [15],
a graph is connected if and only if the algebraic connectivity
of the graph is greater than zero. Therefore, the algebraic
connectivity of any spanning tree will be greater than zero
and the algebraic connectivity of any disconnected graph
will be equal to zero. Hence, the problem of choosing a spanning
tree while maximizing its algebraic connectivity can be posed
as follows:

\[
\max_x \lambda_2 (L(x)),
\]  

(2)
subject to
\[ \sum_{e \in E} x_e = |V| - 1, \quad (3a) \]
\[ x \in \{0, 1\}^{|E|}, \quad (3b) \]
\[ \delta_{ij}(x) \leq D \quad \forall i, j \in V, \quad (3c) \]
where \( \delta_{ij}(x) \) denotes the number of edges in the shortest path joining any two nodes \( i \) and \( j \) in the network represented by \( x \) and \( D \) is the limit on the diameter of the network. Using the results in [1, 4, 12], the above problem can be formulated as a non-linear program with the diameter constraint as follows:
\[ y^* = \max_{x \in \mathcal{X}} y, \quad (4) \]
subject to
\[ L(x) \geq y \left( I_n - e_0 \otimes e_0 \right), \quad (5a) \]
\[ \sum_{e \in E} x_e = |V| - 1, \quad (5b) \]
\[ x \in \{0, 1\}^{|E|}, \quad (5c) \]
\[ \delta_{ij}(x) \leq D \quad \forall i, j \in V, \quad (5d) \]
where \( e_0 \) is the normalized eigenvector corresponding to the first eigenvalue of \( x \), and for any two square matrices \( A, B \) with the same dimensions, \( A \succeq B \) if and only if \( A - B \) is a positive semidefinite matrix.

3. **k-opt Heuristic**

\( k \)-opt heuristics were initially proposed for solving routing problems in [16]. In this heuristic, a feasible spanning tree is first generated by choosing any star graph with one of the vertices having a degree equal to \( |V| - 1 \). Then, a new spanning tree in a local neighborhood of the current, feasible solution is chosen such that the new solution is feasible and has a larger algebraic connectivity. This procedure is iterated until a better spanning tree cannot be found.

Suppose \( \mathcal{T}_1 \) and \( \mathcal{T}_2 \) are two feasible solutions for the algebraic connectivity problem. Then, \( \mathcal{T}_2 \) is said to be in the \( k \)-exchange neighborhood of \( \mathcal{T}_1 \) if \( \mathcal{T}_2 \) can be obtained by replacing \( k \) edges in \( \mathcal{T}_1 \). In a \( k \)-opt exchange, we find a (new) feasible solution in the \( k \)-exchange neighborhood of the current solution and replace the current solution with this new solution if the algebraic connectivity of the new solution is larger than the algebraic connectivity of the current solution. In summary, a \( k \)-opt heuristic starts with an initial feasible solution and iteratively applies a \( k \)-opt exchange until no improvements can be made.

There are two main steps in this \( k \)-opt exchange: choosing a collection of \( k \) edges to remove from the current solution and then reconnecting the resulting, disjoint components with a new collection of \( k \) edges. Clearly, there are several combinations of \( k \) edges that can be removed from (or added to) a given solution. Therefore, choosing an efficient procedure for the deletion and addition of edges is critical for developing a relatively fast algorithm. In the following subsections, we provide procedures for implementing these steps.

3.1. **Selecting a Collection of \( k \)-Edge Combinations to Delete.**

The basic idea here is to list all the possible combinations of \( k \)-edges that can be deleted from the current feasible solution, assign a value for each combination, rank the combinations based on these values, and then choose a subset of these combinations for further processing. We assign a value to a combination of edges by first asking the following basic question: which are the \( k \) edges that need to be deleted from the current solution \( \mathcal{T}_0 \) so that \( \mathcal{T}_0 \) (possibly) incurs the smallest reduction in the algebraic connectivity? To answer this question, let \( \mathcal{T}_0 \setminus \{e\} \) denote the graph obtained by deleting an edge \( e \) from the graph \( \mathcal{T}_0 \) and let the Laplacian of the graph, \( \mathcal{T}_0 \setminus \{e\} \), be denoted by \( L(\mathcal{T}_0 - e) \). By variational characterization, we have the following inequality:
\[ \lambda_2(L(\mathcal{T}_0 - e)) \leq v^T L(\mathcal{T}_0 - e) v \quad \forall v \in \mathcal{V} \]
\[ = v^T L(\mathcal{T}_0 - e) v - v^T L(\mathcal{T}_0) v \quad \forall v \in \mathcal{V} \]
\[ = v^T L(\mathcal{T}_0) v - w_e (v_i - v_j)^2 \quad \forall v \in \mathcal{V}, \]
where \( \mathcal{V} := \{ v : \sum_i v_i = 1, \|v\| = 1 \} \), \( w_e \) is the weight of edge \( e = (i, j) \), and \( v_i \) represents the \( i \)-th component of the vector \( v \). One may observe from the above inequality that by choosing an edge with a minimum value of \( w_e (v_i - v_j)^2 \), the upper bound on the algebraic connectivity of the graph, \( \mathcal{T}_0 \setminus \{e\} \), is kept as high as possible. Also, we numerically observed that \( w_e (v_i - v_j)^2 \) was kept to a minimum by choosing \( v \) as the eigenvector corresponding to the maximum eigenvalue of \( L(\mathcal{T}_0) \). Hence, for any combination of \( k \) edges denoted by \( S \), we assign a value given by \( \sum_{e \in S} w_e (v_i - v_j)^2 \). Then, we rank all the combinations based on the increasing values and choose a subset of these combinations that corresponds to the lowest values. In this work, the fraction of combinations that is considered for deletion is specified through a parameter called the edge deletion factor. The edge deletion factor is defined as the ratio of the number of \( k \)-edge combinations considered for deletion to the maximum number of possible \( k \)-edge combinations (i.e., \( \binom{\gamma}{k-1} \)). We will discuss more about this factor later in the simulations section.

3.2. **Selecting a Collection of \( k \)-Edge Combinations to Add.**

In the case of spanning trees, after removing \( k \) edges, we are guaranteed to have a graph \( \mathcal{T}_0 \) with exactly \( k + 1 \) connected components \( \{C_1, C_2, C_3, \ldots, C_{k+1}\} \); therefore, by suitably adding a collection of \( k \) edges connecting all the \( k + 1 \) components in \( \mathcal{T}_0 \), one is guaranteed to obtain a spanning tree, \( \mathcal{T}_1 \). Also, we add these edges while ensuring that the resulting tree satisfies the diameter constraints. The new feasible solution \( \mathcal{T}_1 \) is considered for replacing \( \mathcal{T}_0 \) if it has a larger algebraic connectivity than \( \mathcal{T}_0 \).
4 Mathematical Problems in Engineering

Input: \( D \) (positive integer)

Initial feasible solution satisfying diameter constraints

\( \lambda_2(\mathcal{L}(\mathcal{F}_0)) \) \( \lambda_0 \) \( \lambda_2(\mathcal{L}(\mathcal{F}_0)) \)

Subset of \( k \)-edge combinations considered for possible deletion as obtained by the edge ranking procedure in Section 3.1

for each edge combination in \( E_{dd} \) do

Delete the \( k \) edges present in the edge combination to obtain connected components \( \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \ldots, \mathcal{C}_{k+1} \)

Subset of \( k \)-edge combinations considered for possible addition as obtained by the edge ranking procedure in Section 3.2

Let \( \mathcal{F}_1 \) be the spanning tree which is feasible and has the largest connectivity obtained from adding the edges in an edge combination from \( E_{dd} \)

if \( \lambda_2(\mathcal{L}(\mathcal{F}_1)) > \lambda_0 \) then

\( \mathcal{F}_0 \leftarrow \mathcal{F}_1 \)

\( \lambda_0 \leftarrow \lambda_2(\mathcal{L}(\mathcal{F}_1)) \)

end if

end for

Output \( \mathcal{F}_0 \) as the (new) current solution

Algorithm 1: \( k \)-opt exchange.

As in the edge-deletion procedure, checking for every addition of \( k \) edges may be computationally intensive for large instances. Therefore, we develop another edge ranking procedure for adding edges as follows. Let \( \mathcal{F}_0 \cup \{ e \} \) denote the graph obtained by adding an edge \( e = (i, j) \) to the graph \( \mathcal{F}_0 \) and let the Laplacian of the graph, \( \mathcal{F}_0 \cup \{ e \} \), be denoted by \( \mathcal{L}(\mathcal{F}_0 + e) \). By variational characterization, we have the following inequality:

\[
\lambda_2(\mathcal{L}(\mathcal{F}_0 + e)) \leq \lambda_2(\mathcal{L}(\mathcal{F}_0)) + \lambda_0 \left( \mathcal{L}(\mathcal{F}_0) + e \right) \forall \nu \in \mathcal{X} \quad (7a)
\]

\[
\lambda_2(\mathcal{L}(\mathcal{F}_0 + e)) = \lambda_2(\mathcal{L}(\mathcal{F}_0)) + \lambda_0 \left( \mathcal{L}(\mathcal{F}_0) + e \right) + \lambda_0 \lambda_2(\mathcal{L}(\mathcal{F}_0)) \forall \nu \in \mathcal{X} \quad (7b)
\]

One may observe from the aforementioned inequality that by choosing an edge with a maximum value of \( \sum_{e=(i,j)\in\mathcal{E}} w_e(v_i - v_j)^2 \), the upper bound on the algebraic connectivity of the graph \( \mathcal{F}_0 \cup \{ e \} \) is kept as high as possible. Just like the edge deletion step, let \( \nu \) be the eigenvector corresponding to the maximum eigenvalue of \( \mathcal{L}(\mathcal{F}_0) \). Hence, for any combination of \( k \) edges denoted by \( S \), we assign a value given by \( \sum_{e=(i,j)\in\mathcal{S}} w_e(v_i - v_j)^2 \). Then, we rank all the combinations based on decreasing values and choose a subset of these combinations that corresponds to the highest values. The number of combinations that are considered for addition is another parameter and can be specified based on the problem instances.

A pseudocode of the \( k \)-opt exchange is outlined in Algorithm 1. An illustration of such a procedure on one such pair \( k = 2 \) of edges for a spanning tree with 4 nodes is shown in Figure 1. This exchange is iteratively applied on the current solution until no improvements can be made.

Figure 1: An example illustrating a 2-opt exchange.

4. Tabu Search Heuristic

In this section, we present a tabu search heuristic (Algorithm 2) for finding good feasible solutions for the algebraic connectivity problem. Similar to the \( k \)-opt heuristic, the tabu search heuristic first starts with a feasible solution \( \mathcal{F} \) and attempts to construct a candidate feasible solution \( \mathcal{F}' \) in the neighborhood of \( \mathcal{F} \) with a larger algebraic connectivity.
Let $s^*$ be the best optimal solution found by the algorithm and let $\lambda^*_2$ be the algebraic connectivity of $s^*$. Let $m$ denote the number of edges in the initial feasible solution (i.e., $m := |V| - 1$). Also, let $\text{MaxIterations}$ represent the maximum number of iterations allowed in the tabu heuristic.

(1) $\mathcal{F}_0 \leftarrow$ Initial feasible solution satisfying diameter constraints
(2) $s := \mathcal{F}_0, \lambda^*_2 := \lambda_2(\mathcal{F}_0), s^* := \mathcal{F}_0, T$ is an empty tabu list which can store at most $m$ solutions.
(3) for $i = 1$ to $\text{MaxIterations}$ do
(4) for edge $e_i$ in $s$ do
(5) Construct the set of edges $N(s, e_i)$
(6) end for
(7) for any spanning tree $s'$ such that $s'$ contains exactly one edge from each of the sets in $\{N(s, e_1), \ldots, N(s, e_k)\}$ do
(8) if $s'$ is feasible then
(9) $\lambda_2(s') > \lambda^*_2$ then
(10) $s = s'$
(11) $T \leftarrow T \cup \{s\}; T$ only stores at most $m$ solutions from iterations $i - m + 1, \ldots, i - 1, i$
(12) $\lambda^*_2 = \lambda_2(s), s^* = s$
(13) break
(14) else
(15) if $s'$ is not in $T$ then
(16) $s = s'$
(17) $T \leftarrow T \cup \{s\}; T$ only stores at most $m$ solutions from iterations $i - m + 1, \ldots, i - 1, i$
(18) break
(19) end if
(20) end if
(21) end if
(22) end for
(23) if $s \neq s'$ then
(24) break //The heuristic terminates here if there is no feasible tree $s'$ in the neighborhood of $s$ or if every feasible $s'$ with a lower algebraic connectivity ($< \lambda^*_2$) is already in the tabu list.
(25) end if
(26) end for
(27) output $\lambda^*_2$ and $s^*$

Algorithm 2: Tabu search heuristic.

(we will later explain how we choose the neighborhoods for the tabu search). If the algebraic connectivity of $s'$ ($\lambda_2(s')$) is larger than the best known value ($\lambda^*_2$), $s'$ is set as the current feasible solution (i.e., $s := s'$) and $\lambda^*_2$ is updated (steps 9–13 in the algorithm). If the algebraic connectivity of $s'$ is at most equal to $\lambda^*_2$, $s'$ is set as the current feasible solution but no updates are made to $\lambda^*_2$ (steps 15–19 in the algorithm). The above procedure is then repeated again until the number of iterations reaches a maximum limit.

There are two features that makes the tabu search heuristic different from the $k$-opt heuristic. Firstly, unlike the $k$-opt heuristic which aims to make an exchange only if a better solution is found, the tabu search allows for the heuristic to move to solutions that may have a smaller algebraic connectivity than the current best solution. Even though there may not be any improvement in the short run, the tabu search heuristic can end up finding better solutions as the number of iterations increases. Secondly, in any iteration, $s'$ is set as the current feasible solution only if $s'$ has not been encountered in any of the previous $m$ iterations. This is accomplished by creating a tabu list and storing all the recent $m$ feasible solutions that have been previously considered by the algorithm in this list. Therefore, when the algebraic connectivity of $s'$ is found to be at most equal to $\lambda^*_2$, $s'$ is set as the current feasible solution only if $s'$ is not present in the tabu list.

The overall tabu search heuristic is presented in Algorithm 2. In the remainder of this subsection, we explain how the local neighborhoods are chosen for the current feasible solution $s$. Suppose $e_i$ is the $i$th edge in $s$. Let the edge $e_i$ join vertices $v_1$ and $v_2$ in the solution $s$. Also, define $N(s, e_i)$ as the set of all the edges that are incident on $v_1$ or $v_2$ but not present in $s$. Then, the neighborhood of the current solution $s$ consists of any solution $s'$ such that there is exactly one edge present in $s'$ from $N(s, e_i)$ for $i = 1, \ldots, |V| - 1$.

5. Computational Results

The $k$-opt and tabu search heuristics were implemented in MATLAB. The bound on the diameter of a feasible tree was set to four ($D = 4$). The diameter of a tree is found by
implementing the well-known Floyd-Warshall algorithm [17].

To check for feasibility, we find the diameter of the given tree using the Floyd-Warshall algorithm and verify if this diameter is at most equal to the prescribed bound. For the tabu search heuristic, the size of the tabu search list was set to 20 and the parameter MaxIterations was set to 100.

For the \( k \)-opt heuristic, we set the edge deletion factor (Section 3.1) to be equal to 0.15 in all the simulations. We chose this value based on the simulation results shown in Figure 2. This figure shows the average algebraic connectivity of the final solution (and the computation time) obtained using the 3-opt heuristic as a function of the edge deletion factor. We observed that there was not much improvement in the quality of the feasible solutions beyond a value of 0.15 (for the edge deletion factor) even for large instances \( (n = 30, 40) \). Hence, we chose 0.15 as the edge deletion factor. Also, we set the number of combinations of edges to be added (Section 3.2) to be at most equal to \( 5^k \). For 3-opt, this parameter was set to 125. We chose this value for 3-opt based on the simulation results shown in Figure 3. For 2-opt, this parameter was set to 25 after performing similar simulations.

The semidefinite programming toolboxes (Yalmip and Sedumi) in MATLAB could not solve the proposed formulation (B.1) with the semidefinite and diameter constraints even for instances with 5 nodes primarily due to inefficient memory management. Therefore, Algorithm 3 was implemented to find an optimal solution for instances up to 8 nodes. For more than 8 nodes, the algorithm in the appendix could not find an optimal solution in a reasonable amount of run time. This algorithm was implemented in C++ programming language and the resulting Integer Linear Programs (ILPs) were solved using CPLEX 12.2 with all the solver options set to default. All the developed algorithms were run on a Dell Precision T5500 workstation (Intel Xeon E5630 processor @ 2.53 GHz, 12 GB RAM).

Let us now compare the deviation of the solutions found by the algorithms with respect to the optimal solution. We define deviation (in %) as \( ((\lambda^*_2 - \lambda^*_2)^{\text{heuristic}})/(\lambda^*_2) \times 100 \) where \( \lambda^*_2 \) denotes the algebraic connectivity of the solution found by the heuristic and \( \lambda^*_2 \) represents the optimum. The results shown in Table 1 are for 10 random instances generated for networks with 8 nodes. Based on the results in Table 1, we observed that the average run time to obtain an optimal solution was around 371 seconds. Also, it can be seen that the \( k \)-opt \( (k = 2, 3) \) and tabu search heuristic found optimal solutions for all problem instances with 8 nodes. Each of the heuristics ran within a fraction of a second for each of the instances with 8 nodes. Instance 1 of Table 1 is pictorially shown in Figure 4.

For more than 8 nodes, we used the heuristics to generate feasible solutions for the algebraic connectivity problem. In the remainder of this section, we compare the results from the \( k \)-opt and tabu search heuristics. As previously mentioned in the paper, choosing an initial solution for the heuristics is relatively straightforward; we choose a star graph with the largest algebraic connectivity \( (\lambda^\text{initial}_2) \) as an initial feasible solution. A sample initial feasible solution is shown in part (b) of Figure 4.

For more than 8 nodes, we inferred the deviation of a solution for an instance by comparing it with the solution obtained by using the 3-opt heuristic for the same instance. Specifically, the deviation of a solution is defined as \( ((\lambda^\text{3opt}_2 - \lambda^\text{heuristic}_2)/(\lambda^\text{3opt}_2) \times 100 \) where \( \lambda^\text{3opt}_2 \) is the algebraic connectivity of a solution obtained by using the 3-opt heuristic. For each
problem size, the average deviation and computation time obtained for the 10 instances is shown in Table 2. For each of the instances, the solution obtained by the 3-opt heuristic had the largest algebraic connectivity.

From the results in Table 2, it is clear that the 2-opt heuristic performed better than the tabu search for medium sized instances (up to 30 nodes). For larger problems (40, 45, 55, and 60 nodes), we observed that the quality of the solutions found by the tabu search was better compared to the 2-opt heuristic. In summary, we observe that the 2-opt heuristic provides a good trade-off between obtaining good feasible solutions and the required computation time. Figure 5 illustrates the solutions obtained using the $k$-opt and tabu search heuristic for a 40 nodes problem. Also, as shown in Table 3, we observed that the maximum deviation (out of 10 instances) of the 2-opt solutions remained lower than that of tabu search for instances up to 30 nodes. For large instances (35, 40, 45, 55, and 60 nodes), the maximum deviation for tabu search remained lower than that of 2-opt solutions. Overall, the tabu search heuristic performed better than the 2-opt search heuristic for most of the large instances.

6. Conclusions

We studied design methods to maximize the robustness of a network in the presence of diameter constraints. Algebraic connectivity is used to measure the robustness of the network. Our design methods mainly involve the development of heuristics. Our first method, the $k$-opt heuristic, aims to iteratively search for better solutions by performing an exchange
Figure 4: A network with all possible edges connecting 8 nodes including edge weights is shown in (a). (b) represents the initial feasible solution which is a star graph. (c) represents an optimal network which also happens to be the solution found by the 2-opt, 3-opt, and tabu search methods.

Table 2: Comparison of 3-opt, 2-opt, and tabu search heuristic solutions for various problem sizes. Here, the deviation was averaged over ten random instances for each $n$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>3-opt Deviation (%)</th>
<th>3-opt Time (sec)</th>
<th>2-opt Deviation (%)</th>
<th>2-opt Time (sec)</th>
<th>Tabu Deviation (%)</th>
<th>Tabu Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>0.29</td>
<td>0</td>
<td>0.10</td>
<td>0.00</td>
<td>0.77</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>3.06</td>
<td>0.01</td>
<td>0.52</td>
<td>1.52</td>
<td>9.36</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>16.32</td>
<td>0.27</td>
<td>1.65</td>
<td>1.78</td>
<td>25.58</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>60.38</td>
<td>0.60</td>
<td>3.59</td>
<td>1.88</td>
<td>60.19</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>274.93</td>
<td>2.07</td>
<td>12.13</td>
<td>3.12</td>
<td>144.92</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
<td>480.15</td>
<td>2.02</td>
<td>37.85</td>
<td>2.77</td>
<td>333.20</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>1016.99</td>
<td>5.62</td>
<td>57.28</td>
<td>2.48</td>
<td>3470.71</td>
</tr>
<tr>
<td>45</td>
<td>0</td>
<td>2309.60</td>
<td>7.10</td>
<td>116.09</td>
<td>5.10</td>
<td>4272.58</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
<td>4219.17</td>
<td>1.38</td>
<td>139.99</td>
<td>3.55</td>
<td>6675.32</td>
</tr>
<tr>
<td>55</td>
<td>0</td>
<td>6798.34</td>
<td>6.98</td>
<td>350.83</td>
<td>4.90</td>
<td>8654.29</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>8974.46</td>
<td>7.97</td>
<td>505.36</td>
<td>6.06</td>
<td>10049.10</td>
</tr>
</tbody>
</table>

Table 3: Comparison of maximum deviations computed over ten random instances for each $n$ for 2-opt and tabu search heuristic.

<table>
<thead>
<tr>
<th>$n$</th>
<th>Maximum deviation (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00</td>
</tr>
<tr>
<td>15</td>
<td>0.08</td>
</tr>
<tr>
<td>20</td>
<td>2.73</td>
</tr>
<tr>
<td>25</td>
<td>4.92</td>
</tr>
<tr>
<td>30</td>
<td>7.56</td>
</tr>
<tr>
<td>35</td>
<td>12.59</td>
</tr>
<tr>
<td>40</td>
<td>17.89</td>
</tr>
<tr>
<td>45</td>
<td>15.41</td>
</tr>
<tr>
<td>50</td>
<td>5.10</td>
</tr>
<tr>
<td>55</td>
<td>17.56</td>
</tr>
<tr>
<td>60</td>
<td>16.92</td>
</tr>
</tbody>
</table>

of edges in each iteration. Unlike the $k$-opt heuristic, our second method (the tabu search heuristic) iteratively moves to solutions that may have a lower algebraic connectivity in the short run but could find better solutions as the number of iterations increases. Computational results suggested that the 3-opt heuristic performed the best while the 2-opt heuristic provided a good trade-off between finding good solutions and the required computation time. There are several future directions for this work. One research direction would be to generalize the network synthesis problem to include more connectivity constraints. Another research direction would be to compare the performance of the developed algorithms in a simulation setting similar to the one in [4] and test the performance in the presence of node or edge failures.

Appendices

For completeness, we summarize the algorithm in [11] for finding an optimal solution to the algebraic connectivity
problem. There are two issues that need to be addressed before one can solve the formulation in (5a), (5b), (5c), and (5d). First, the diameter constraint, \( \delta_{ij}(x) \leq D \), must be converted to a more tractable form so that standard optimization tools can be used to solve it. Second, one must develop an algorithm to find an optimal solution to solve the resulting algebraic connectivity problem.

A. Handling the Diameter Constraint

We assume that \( D \) is even. A similar approach can be used when \( D \) is odd. To impose the the diameter constraint, we add a source node \( (r) \) to the graph \((V, E)\) and add an edge joining \( r \) to each vertex in \( V \), that is, \( \tilde{V} = V \cup \{r\} \) and \( \tilde{E} = E \cup (r, j) \) for all \( j \in V \). We then construct a tree.
Algorithm 3: Synthesizing robust networks with maximum algebraic connectivity subject to diameter constraints.

spanning all the nodes in \( \overline{V} \) while restricting the length of the path from \( r \) to any other node in \( \overline{V} \). The additional edges emanating from the root vertex are used only to formulate the diameter constraints \([11]\), and they do not play any role in determining the algebraic connectivity of the original graph. Hence, the diameter constraints in (5d) can be equivalently formulated using the multicommodity flow constraints as follows:

\[
\sum_{j \in \overline{V} \setminus r} (f_{ij}^k - f_{ji}^k) = 1 \quad \forall k \in V, \ i = r, \quad (A.1a)
\]

\[
\sum_{j \in \overline{V}} (f_{ij}^k - f_{ji}^k) = 0 \quad \forall i, k \in V, \ i \neq k, \quad (A.1b)
\]

\[
\sum_{j \in \overline{V}} (f_{ij}^k - f_{ji}^k) = -1 \quad \forall i, k \in V, \ i = k, \quad (A.1c)
\]

\[
f_{ij}^k + f_{ji}^k \leq x_e \quad \forall e := (i, j) \in \overline{E}, \ \forall k \in V, \quad (A.1d)
\]

where \( f_{ij}^k \) is the \( k \)th commodity flowing from node \( i \) to node \( j \). Constraints (A.1a) through (A.1c) state that each commodity must originate at the root node and terminate at its corresponding vertex. Equation (A.1d) states that the flow of commodities between two vertices is possible only if there is an edge joining the two vertices. Constraint (A.1e) ensures that the number of edges in the chosen network corresponds to that of a spanning tree. An advantage of using this formulation is that one now has access directly to the number of edges on the path joining the source node to any vertex in the graph. That is, \( \sum_{(i,j) \in \overline{E}} f_{ij}^k \) denotes the length of the path from \( r \) to \( k \) and hence (A.1h) represents the diameter constraint. Therefore, the constraints in (A.1a), (A.1b), (A.1c), (A.1d), (A.1e), (A.1f), (A.1g), (A.1h), and (A.1i) can be used to replace the diameter (5d) and spanning tree constraints (5b) in the formulation.

\[
B. Optimal Algorithm
\]

The second issue is to reduce the problem in (5a), (5b), (5c), and (5d) to a pure binary semidefinite programming problem (BSDP) as the tools associated with the construction of valid inequalities are more abundant for BSDPs as compared to mixed-integer programs. Also, with the further relaxation of the semidefinite constraint, it can be solved using CPLEX [18], a high performance solver for Integer Linear Programs (ILP). Therefore, we adopt a different approach for finding an optimal solution by casting the algebraic connectivity problem as the following decision problem: is there an augmented network such that the algebraic connectivity of the network is at least equal to a prespecified value \( (\overline{\lambda}_2) \) and the diameter of the graph is at most equal to \( D \)? This problem
can be posed as a BSDP by choosing to find a spanning tree that minimizes the degree of the root vertex $r$. This can be mathematically written as follows:

$$\min \sum_{e \in \delta(r)} x_e$$

s.t. $L(x) \geq \lambda_2(I_n)$.

Diameter and spanning tree constraints as in (A.1a), (A.1b), (A.1c), (A.1d),

(A.1e), (A.1f), (A.1g), (A.1h), and (A.1i).

(B.1)

In this formulation, $\delta(r)$ denotes a cutset defined as $\delta(r) = \{e = (r, j) : j \in V \setminus r\}$. The above BSDP can be efficiently solved by a cutting plane procedure and then a bisection algorithm can be used to maximize the algebraic connectivity. In the cutting plane procedure, the semidefinite constraint is replaced by a finite subset of the infinite number of linear constraints and successively tighter polyhedral approximations are constructed by augmenting valid inequalities until a feasible solution is obtained by satisfying a desired level of connectivity. Based on the notation defined in this paper, a detailed pseudocode of this procedure is outlined in Algorithm 3. Steps 4 through 16 of Algorithm 3 employ the cutting plane procedure until the semidefinite constraint is satisfied and step 17 is the bisection step where $\lambda$ is incremented until the optimization problem becomes infeasible.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References


