# A Consistent Labeling Algorithm for the Frequency/Code Assignments in a Rapidly Deployable Radio Network (RDRN)

*Technical Report*

*Written by*
Shane M. Haas

*Supervisor*
Dr. David Petr

*Telecommunications and Information Sciences Laboratory*
*Department of Electrical Engineering and Computer Science*
*University of Kansas,*
*Lawrence, Kansas, USA 66045*
*Tel.: (913) 864-7757*
*email: shaas@tisl.ukans.edu*

January 1994

**Abstract**

The Rapidly Deployable Radio Network (RDRN) is an Advanced Research Projects Agency (ARPA) grant awarded to the University of Kansas's Telecommunications and Information Sciences Laboratory (TISL). The goal of this project is to develop a portable high speed wireless communications network that could easily be deployed in a time of battle or emergency.

One of the major obstacles facing a decentralized wireless network is its topological configuration. This network consists of a collection of asynchronous transfer mode (ATM) switch nodes that are capable of forming multiple radio links with one another. Since these beams can be unintentionally received by another node, interference may occur. To provide effective communication throughout the network, an algorithm must carefully establish links between nodes. The configuration algorithm must also ensure that the network is fully connected.

By modeling RDRN as a consistent labeling $(U, L, T, R)$ model [HAR79] a tree search is used to find the non-interfering network configurations. This does not, however, guarantee full network communication. A connectivity matrix is created showing direct node communications. Another tree search is then used to find communication classes of this matrix. In this manner, network connectivity may be tested.

This report documents the ideas and methods that contributed to the above algorithms.

# Contents

# 1 Summary

The Rapidly Deployable Radio Network (RDRN) is an Advanced Research Projects Agency (ARPA) grant awarded to the University of Kansas's Telecommunications and Information Sciences Laboratory (TISL). The goal of this project is to develop a portable high speed wireless communications network that could easily be deployed in a time of battle or emergency.

One of the major obstacles facing a decentralized wireless network is its topological configuration. This network consists of a collection of asynchronous transfer mode (ATM) switch nodes that are capable of forming multiple radio links with one another. Since these beams can be unintentionally received by another node, interference may occur. To provide effective communication throughout the network, an algorithm must carefully establish links between nodes. The configuration algorithm must also ensure that the network is fully connected.

By modeling RDRN as a consistent labeling $(U, L, T, R)$ model [HAR79] a tree search is used to find the non-interfering network configurations. This does not, however, guarantee full network communication. A connectivity matrix is created showing direct node communications. Another tree search is then used to find communication classes of this matrix. In this manner, network connectivity may be tested.

This report documents the ideas and methods that contributed to the above algorithms.

# 2 The Consistent Labeling Problem

## 2.1 An Overview

Let $U$ be a set of units and $L$ be a set of possible labels for these units. Units are objects requiring identification by names or labels. They can be anything from boolean expressions to radio links, whose corresponding labels would be $\{0, 1\}$ and frequency pairs.

Consider a matrix of $N$ columns that provides a relationship between $N$ units that restrict each other. This is often called the unit constraint relation $T$, where $T \subseteq U^N$. For example, let $U = \{u_1, u_2, u_3\}$ and $L = \{l_1, l_2\}$. If the units $u_1$ and $u_2$ constrain each other and $u_1$ also constrains $u_3$, then

$$T = \begin{pmatrix} u_1 & u_2 \\ u_1 & u_3 \end{pmatrix}$$

Another matrix, $R$, is formed by combining the matrix $T$ and the vector

$L$. This matrix contains all of the labels permitted for each $N$-tuple in $T$, $R \subseteq (UxL)^N$. It is called the unit-label constraint relation because it consists of all allowable $2N$-tuples of unit label pairs. For our example, if $u_1$ cannot have the same label as $u_2$, and $u_1$ must have the same label as $u_3$, then

$$R = \begin{pmatrix} u_1 & l_1 & u_2 & l_2 \\ u_1 & l_2 & u_2 & l_1 \\ u_1 & l_1 & u_3 & l_1 \\ u_1 & l_2 & u_3 & l_2 \end{pmatrix}$$

Taken collectively, these four variables are called the real world model or $(U, L, T, R)$ model. A labeling of all units in $U$ is called *consistent* to the model if for all $N$-tuples in $T$ when the labeling is applied, the $2N$-tuple formed is a member of $R$. In other words, the labeling $\{l_1, \cdots, l_P\}$ is a consistent labeling of units $\{u_1, \cdots, u_P\}$ if for all $\{i_1, \cdots, i_N\} \subseteq \{1, \cdots, P\}$ such that $(u_{i1}, \cdots, u_{iN})$ is a member of $T$ creates a $2N$-tuple $(u_{i1}, l_{i1}, \cdots, u_{iN}, l_{iN})$ that is a member of $R$. [HAR79]

The consistent labeling problem consists of finding all labelings for units in $U$ that are permitted by the $(U, L, T, R)$ model. In the previous example, one consistent labeling of units $\{u_1, u_2, u_3\}$ would be $\{l_1, l_2, l_1\}$. To verify this, apply the definition of a consistent labeling. By labeling the units in the first 2-tuple of $T$, $(u_1, u_2)$, with their proposed labels, the 4-tuple $(u_1, l_1, u_2, l_2)$ is formed. Since this is a member of $R$, the next 2-tuple in $T$ is examined. By labeling $(u_1, u_3)$ with their proposed labeling $(l_1, l_1)$, the 4-tuple $(u_1, l_1, u_3, l_1)$ is produced. This too is a member of $R$. Because all $N$-tuples in $T$ were checked, the labeling $\{l_1, l_2, l_1\}$ is concluded to be a consistent labeling of units $\{u_1, u_2, u_3\}$ with respect to the $(T, R)$ model (the $U$ and $L$ are not noted to emphasize the constraining relations). The other consistent labeling solution for units $\{u_1, u_2, u_3\}$ would be $\{l_2, l_1, l_2\}$.

The consistent labeling problem is just another way of representing direct and indirect relationships among a group of objects. The $(U, L, T, R)$ model is a nice representation of a complicated problem. Once in this form, several methods are capable of finding solutions.

## 2.2 Solving the Consistent Labeling Problem

### 2.2.1 The Tree Search

A variety of techniques exist to find all of these labelings. One such method is the tree search. This algorithm takes the first unit appearing in $T$ and assigns to it the first label permitted. It then restricts $R$ to those $2N$-tuples that do

not conflict with this unit-label pair. If the restricted $R$, $R'$, contains a single-valued consistent labeling of all units in $T$, then a solution has been found. Otherwise, the procedure continues with the next unit in $T$ and assigns to it its first permitted labeling. If at any point $R'$ is empty, the routine backtracks and chooses the next label permitted.

For solving the above example, the tree search would first assign $u_1$ with its first labeling of $l_1$. Then it would restrict $R$ to those $2N$-tuples that either had $u_1$ paired with $l_1$ or did not involve $u_1$. The matrix formed would be

$$R' = \begin{pmatrix} u_1 & l_1 & u_2 & l_2 \\ u_1 & l_1 & u_3 & l_1 \end{pmatrix}$$

The resulting $R'$ is a single-valued consistent labeling since each $N$-tuple in $T$ has a labeling in $R'$ and each unit appearing in $R'$ has only one label associated with it. The particular solution found within this $R'$ for $\{u_1, u_2, u_3\}$ is $\{l_1, l_2, l_1\}$.

To find all solutions, the procedure then assigns $l_2$ to $u_1$. The resulting $R'$ is also a single-valued consistent labeling.

$$R' = \begin{pmatrix} u_1 & l_2 & u_2 & l_1 \\ u_1 & l_2 & u_3 & l_2 \end{pmatrix}$$

The solution from this $R'$ is $\{l_2, l_1, l_2\}$. Since there are no more labels for the first unit,$u_1$, the tree search stops.

Notice this simple example required only one unit-label assignment. For more complicated problems, further unit-label pairs must be fixed before either a single-valued consistent labeling occurs or $R'$ is the empty set. When either occurs the last unit that was assigned a label is given the next label that it can receive. If there are no more labels for this unit then the previous unit is given its next label. This continues until the first unit has no more possible labels left.

### 2.2.2 Computational Time

The time required to find consistent labeling solutions from a real world model is exponential. Consider the general case when each unit appearing in $R$ is allowed each label in $L$. For the worst case scenario, the tree search would eventually assign each label in $L$ to each individual unit in $R$. Thus the total number of permutations created is the number of labels in $L$ raised to the number of units in $R$

$$t_{max} = n(L)^{n(R)}$$

Thus, increasing the number of units in $R$ exponentially increases the computational time of the tree search.

# 3 Preliminary Investigations

## 3.1 Preliminary Network Constraints

The first configuration of RDRN followed four rules. First, if two communication links are formed between three nodes, then the angle could not be less than a specified acute angle. Second, near linear consecutive links should also be avoided. Consecutive links are those links that share a common node. If either of these rules are broken, interference has the potential to occur. The third rule defines the concept of a linking radius. A node can only form a communication link with another node within a given distance or transmission radius. Fourth, above all else, full network communication must exist. In other words, a node must have a path to every other node.

These rules started as the basis for the configuration problem. Soon, additional constraints were added. The first two rules were applied to non-consecutive links. A maximum number of frequencies was then imposed. Other desirable constraints are to minimize the number of links, to decentralize the network, to limit total links per node, and to minimize the number of frequencies used.

## 3.2 Early Connecting Algorithms

The first attempt at satisfying the above constraints was a forward searching algorithm. The connecting procedure started on any node and detected the nearest neighboring node within the linking radius. A link was formed with this node using the first send/receive frequency pair. The procedure then repeated itself until all nodes within each others link radii were connected. While connecting the network, if the link just formed interfered with the preceding link then the next send/receive frequency pair was used. Likewise, if the new link was formed with a node that had no other unconnected switches within its linking radius, the procedure moved to the previous node after making the connection.

Further revisions of this algorithm included a desirability factor that determined the next node to be connected. Distance from the node and from the geographical center of the network were the main criteria for this factor. In addition, the idea of changing previously created links was explored.

Even though this was not implemented into code, it has the same idea of the tree search. The first link is chosen and assigned the first frequency pair. If the second link does not interfere with the first link then it also receives the first frequency pair, otherwise it receives the second pair. Furthermore, previously created links would be changed when all frequency pairs were exhausted for the current link. This is another way of solving the network interference problem; however, it might not be as efficient as the consistent labeling approach because it must continually keep track of interfering links. The consistent labeling approach separates the two ideas of interference and the tree search.

## 3.3 The Consistent Labeling Problem

### 3.3.1 Representing a Network

The application of the consistent labeling problem to a network was puzzling at first. Basic questions arose regarding what was to be defined as units and labels. Initially, the nodes and links were units and labels, respectively. This was based on the edge-labeling problem presented by Haralick. Later, however, links were considered units and the frequencies that composed them were labels. Even this representation needed further refining since each label consisted of two frequencies.

Constraining these units was the next concern. At first, all consecutive links were members of $T$. Based on this, $R$ then was the collection of all permitted frequency labelings of links that shared a common node. Though this representation included the maximum links per node constraint, it was discarded since it did not take into consideration non-consecutive links and because each central node could have a different number of possible links radiating from it. To accommodate this, interference was considered in terms of pairs of links. Thus, $T$ consisted of all pairs of links that could interfere with each other. Similarly, $R$ contained all the permutations of labelings that did not cause interference between the pairs.

The next problem resolved was assigning frequency labelings to potentially interfering links. This in itself is a challenge since a variety of topological arrangements of nodes can exist. When creating $R$, the matrix that takes frequencies into consideration, it was first assumed that there was a general assigning rule that applied to all situations. This assumption was later found to be false since each pair of potentially interfering links is arranged in a different manner. More on the consistent labeling application to RDRN will be discussed in a later section.

### 3.3.2 Finding Configurations

Haralick introduces a class of look-ahead operators used to decrease the tree search time for consistent labeling problems. The $\phi_{KP}$ operator is one such operator. It was not found to significantly improve tree search performance due to the complexity of network constraints; therefore, a tree search that did not use the $\phi_{KP}$ operator was developed to solve the consistent labeling problem.

# 4 Network Connectivity

The most important criteria of an acceptable network configuration is full network connectivity. This means that each node must have a path to every other node. By converting the network into a pseudo-stochastic matrix, called a connectivity matrix and then applying techniques to classify states of a stochastic matrix, network connectivity may be tested.[HAA94]

## 4.1 The Connectivity Matrix

The connectivity matrix is based on the idea of a stochastic matrix. A stochastic matrix ($M$) contains a process's probabilities of transitioning between states. The probability that the system will move from state $i$ to state $j$ is defined by the matrix entry $m_{ij}$. Similarly, a connectivity matrix ($C$) shows which nodes can directly communicate with each other. If node $i$ has a direct communication link with node $j$ then matrix entries $c_{ij}$ and $c_{ji}$ both receive the value of one. All non-communicating pairs receive a value of zero.

## 4.2 Communication Classes

A recursive tree search is used to determine which nodes can communicate both directly and indirectly. This procedure starts by creating a list of nodes that are directly accessible from a given node. For each node in the list, it determines which nodes can be directly reached from it. This continues until all the nodes that can be reached from the first node have been found. This set of nodes forms a communication class. The network is fully connected if it contains only one communication class. This guarantees that every node can be reached from an arbitrary node.

# 5 Applying the Consistency Labeling Problem to RDRN

## 5.1 The Network Application

For a network application, $U$ consists of all links that could exist between nodes. For each node, all nodes within a specified maximum connecting distance, known as the linking radius, are found. These combinations of pairs are placed in the column vector $U$. All frequency/code permutation pairs and the

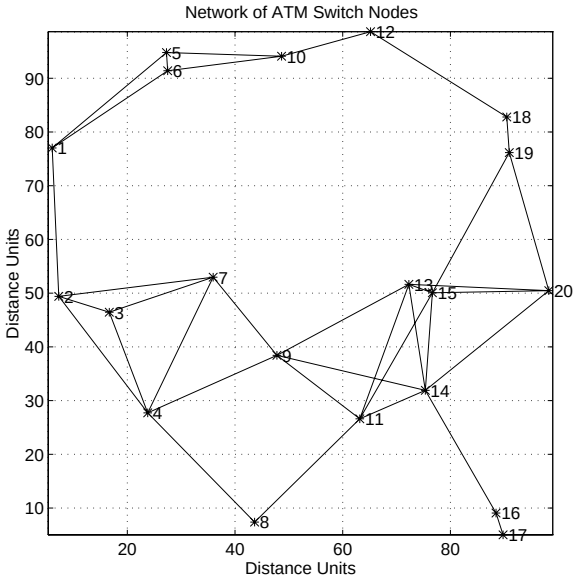



Figure 1: A plot of all the possible links of a 30 node network with a linking radius of 30 distance units. These links are the units of a $(U, L, T, R)$ model.

option of zero, or no link, constitute $L$. The unit constraint relation, $T \subseteq U^N$, consists of those pairs of units that mutually constrain one another. If two links have the potential of interfering with each other, then they are a pairs in $T$. The unit-label constraint relation, $R \subseteq (UxL)^N$, contains all permitted labelings of the pairs within $T$. For RDRN, $R$ contains all frequency/code labelings for the pairs of nodes in $T$ that would not cause interference between them. The consistent labeling problem is to find all permitted labelings of the units in $U$ that are allowable by $T$ and $R$.

## 5.2 Conventions and Notations

To implement this algorithm, certain conventions must be adopted. First, each node receives an integer index value dependent upon its x-coordinate. Numbering ascends from one to positive infinity.

Second, a link is represented by the notation, $n_1000n_2$, where $n_1$ and $n_2$ are the index values of the connected nodes. Also note that this is an ordered pairing with $n_1 < n_2$. If $n_1$ sends data to $n_2$ on frequency/code $f_1$ and receives information from $n_2$ on frequency/code $f_2$, then the link $n_1000n_2$ has the frequency/code label $f_1777f_2$. If a link does not exist, a frequency/code label of 0 is assigned. These conventions are used to allow easy manipulation.

## 5.3 Constraints in RDRN

Creating the consistent labeling model for RDRN requires the examination of several rules or constraints. The first restriction placed on the network is nodes can only form links within their given transmission radius. This limitation influences the creation of the unit vector $U$. Second, two links will constrain each other if the transmission pattern from a node in the first link can be received by a node in the second link or vice versa. An example of this is shown in Figure 2. The unit constraint relation $T$ is created on this principle. Third, if two links constrain each other, frequency/code labels must

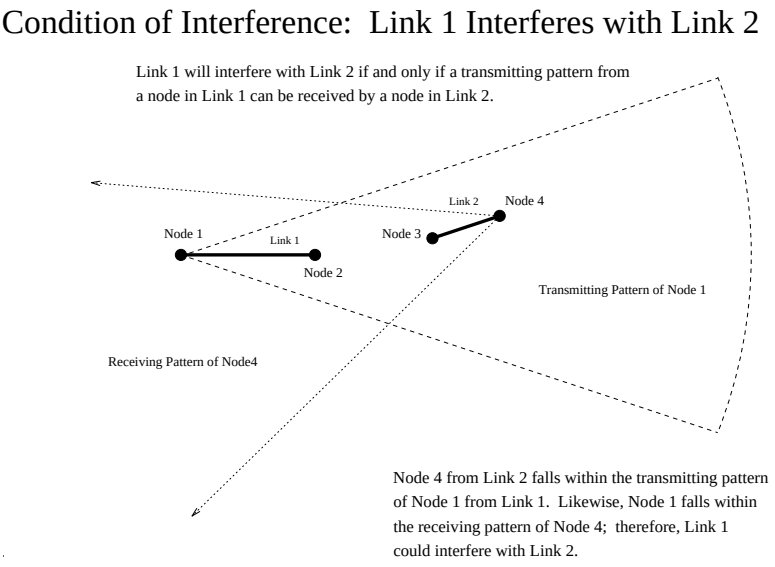


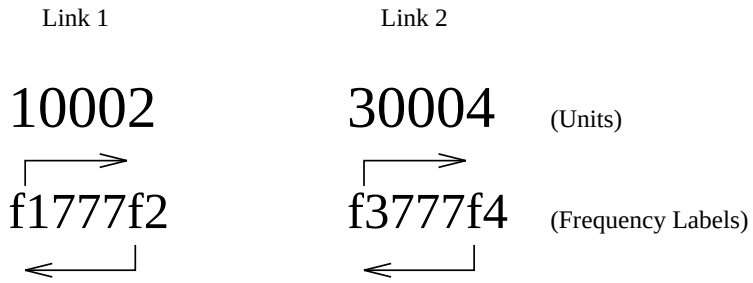

Figure 2: Link 1 Could Interfere With Link 2

be assigned in such a way to avoid interference. The unit label constraint

relation $R$ contains all of these permitted link/labeling 4-tuples. The rules for these assignments are in Figure 3. Finally, network connectivity must be ensured in all configurations.

# The Rule for Assigning Non-Interfering Frequency Labels

In the below diagram two links consisting of four nodes are shown. From this configuration eight different rules are developed. Even though two non-consecutive links are shown, the rules apply for a links with a common node. In this case, node 2 is identical to node 3.




If node 1 could interfere with node 3 then the node 1's transmitting frequency, f1, must not equal node 3's receiving frequency, f4. Through similar arguments the following constraints are developed.

| | | | |
|---|---|---|---|
| f1 ~= f4 | if node 1 could interfere with node 3 | f3 ~= f2 | if node 3 could interfere with node 1 |
| f1 ~= f3 | if node 1 could interfere with node 4 | f3 ~= f1 | if node 3 could interfere with node 2 |
| f2 ~= f4 | if node 2 could interfere with node 3 | f4 ~= f2 | if node 4 could interfere with node 1 |
| f2 ~= f3 | if node 2 could interfere with node 4 | f4 ~= f1 | if node 4 could interfere with node 2 |

Figure 3: The Rule for Assinging Non-Interfering Frequency Pairs

## 5.4 The RDRN Example

Consider the following example. A network consists of 5 nodes whose $(x, y)$ coordinates are stored in the matrix $GPS$.

```
GPS =
```

```
2.0558    1.8853
3.3753    3.2836
5.0886    4.9969
6.1870    7.2534
6.5066    7.0450
```

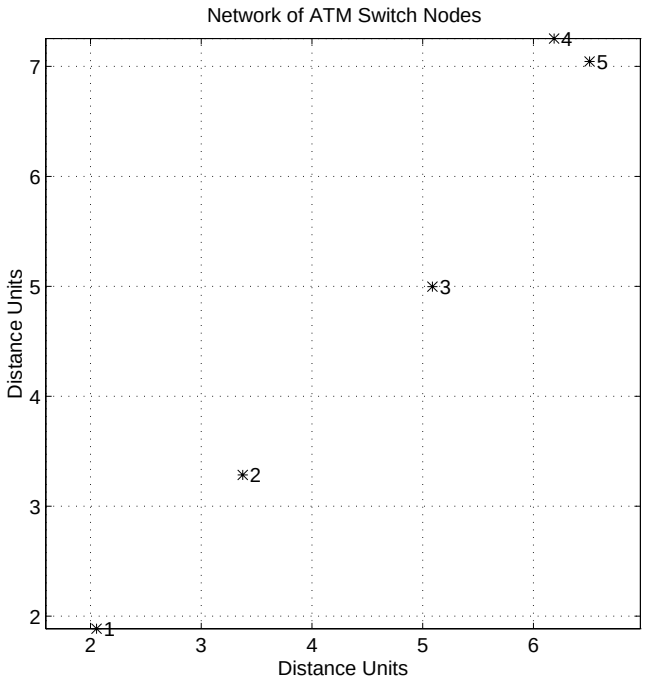



Figure 4: Plot of a Sample RDRN Network

If nodes are allowed to make connections within 2.9 distance units, the following $U$ is created.

```
U =

    10002
    20003
    30004
    30005
    40005
```

If a maximum of three frequencies is available, then seven frequency/code pairs can be created including the value of no link or 0.

```
L =

        0
    17772
    17773
    27771
    27773
    37771
    37772
```

The unit constraint relation $T$ contains pairs of interfering links. This occurs six times in this network. The links 10002 and 30005 are one pair that could interfere if frequency labelings were assigned arbitrarily. Because

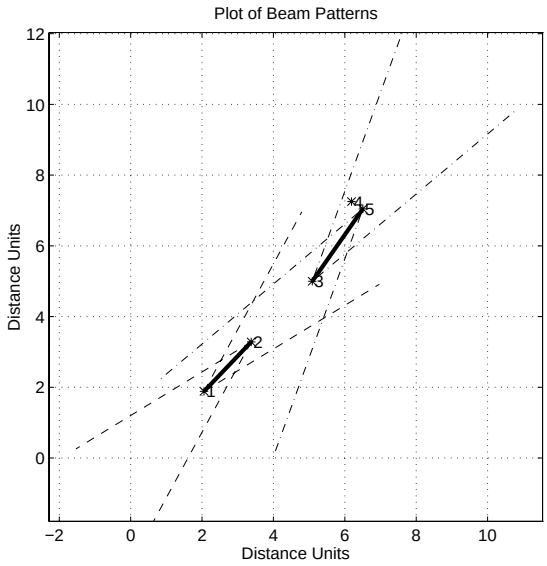



Figure 5: Link 30005 will interference with Link 10002

the transmitting pattern from node 5 can be received by node 1, link 30005 interferes with link 10002. In Figure 5 a pair of dashed lines emanates from each node. The triangle defined by these lines (the third side not shown) defines the "receivable area for the nodes transmission. Notice, however, that link 10002 does not interfere with 30005 since the beam patterns from its nodes cannot reach node 5. The pair 30004 and 30005 in Figure 6 provides another example of constraining links. Node 4's transmitting pattern is received by the antenna intended to capture node 5's beam.

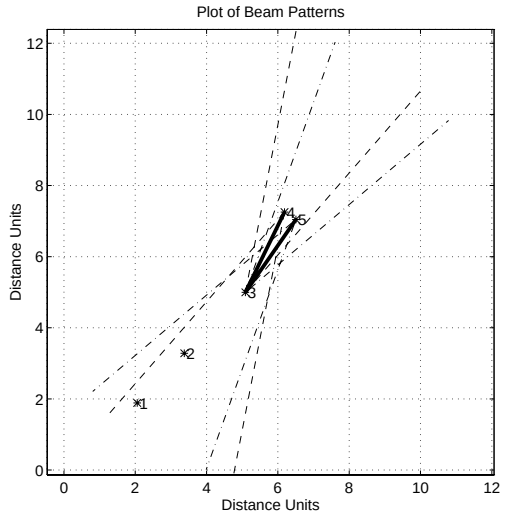



Figure 6: Link 30004 and Link 30005 Mutually Interfere

Several parameters provide a wide range of control over the network. For this example, an interference multiplier (distance multiplier defining distance of interference relative to the distance between nodes) of 3, a beamwidth angle of $30^{o}$ will create a $T$ containing all the pairs of links that could interfere.

```
T =

      10002        20003
      10002        30004
      10002        30005
      20003        30004
      20003        30005
      30004        30005
```

In addition to $T$, two more matrices must be introduced to show how each links' nodes interfere with each other. The matrices $X_1$ and $X_2$ accomplish this. Each row in $X_1$ describes exactly which transmitting beam pattern from the first link (from the left column of $T$) can be received by a node from the second link (from the right column of $T$) for each pairing in $T$. The second matrix, $X_2$, shows which transmitting patterns from nodes in the second link can be received by nodes from the first link. The four columns of $X_i$, where $i$ is the primary link, mean the following: the first column shows if the pattern

from the first node of the primary link can be received by the first node of the secondary link, the second column shows if the pattern from the first node of the primary link can be received by the second node of the secondary link, the third column shows if the pattern from the second node of the primary link can be received by the first node of the secondary link, and the fourth column shows if the pattern from the second node of the primary link can be received by the the second node the the secondary link. So to generate $X_1$, the link in the left column of $T$ is considered the primary link, and the link in the right column is the secondary link. Likewise, to generate $X_2$, the link in the second column of $T$ is considered the primary link. In short, if a row of $T$ was 10002 and 30004, then the resulting $X_1$ has the format

$$X_1 = \begin{pmatrix} 1w3 & 1w4 & 2w3 & 2w4 \end{pmatrix}$$

where $1w3, 1w4$, etc., are boolean values indicating interference. Likewise, $X_2$ has the format

$$X_2 = \begin{pmatrix} 3w1 & 3w2 & 4w1 & 4w2 \end{pmatrix}$$

In this network example, $X_1$ and $X_2$ are

```
X1 =

     0     1     0     0
     0     0     0     0
     0     0     0     0
     0     1     0     0
     0     1     0     0
     0     1     1     0

X2 =

     0     0     1     0
     0     0     1     0
     0     0     1     0
     0     0     1     0
     0     0     1     0
     0     1     1     0
```

For example, if the pair 30004 and 30005 is examined, $X_1$ shows that the pattern from node 3 intended for node 4 can be received by node 5 of the link 30005. In addition, the beam from node 4 intended for node 3 of the link 30004 can be received by node 3 of the link 30005. This last statement means

that node 3 will not be able to distinguish signals coming from nodes 4 and 5 if they are both sending on the same frequency. $X_2$ shows a similar picture from the second link's point of view.

The unit-label constraint relation $R$ contains those permitted labelings for each pair in $T$. Determining these labelings requires $X_1$ and $X_2$. The rules for assigning non-interfering frequency pairs are given in Figure 3. In this example, $R$ is a matrix of 198 rows and 4 columns. A portion of $R$ is shown below.

```
R =

      10002          0      20003          0
      10002          0      20003      17772
      10002          0      20003      17773
      10002          0      20003      27771
      10002          0      20003      27773
      10002          0      20003      37771
      10002          0      20003      37772
      10002      17772      20003          0
      10002      17772      20003      27771
      10002      17772      20003      27773
      10002      17772      20003      37771
      10002      17773      20003          0
      10002      17773      20003      27771
      10002      17773      20003      37771
      10002      17773      20003      37772
      10002      27771      20003          0
      10002      27771      20003      17772
      10002      27771      20003      17773
      10002      27771      20003      37772
      10002      27773      20003          0
      10002      27773      20003      17772
      10002      27773      20003      37771
      10002      27773      20003      37772
      10002      37771      20003          0
      10002      37771      20003      17772
      10002      37771      20003      17773
      10002      37771      20003      27773
      10002      37772      20003          0
      10002      37772      20003      17773
```

```
10002        37772        20003        27771
10002        37772        20003        27773
```

With these matrices, a $(U, L, T, R)$ model of a network is created. The next step is to find all the consistent labelings of all units in $U$ with respect to $T$ and $R$. In addition, each of the non-interfering configurations must be checked for connectivity.

### 5.5 Finding Non-Interfering Network Configurations

Using the tree search and the connectivity test, 48 different network configurations were found for those links in $T$. If a link in $U$ does not appear in $T$, then it is unrestricted; therefore, it may have any frequency/code pair label as long as it contributes to a connected network. One of the solutions is displayed below.

```
CL =

       10002        27771
       20003        37772
       30004        17773
       30005            0
```

In this example, link 10002 has the frequency/code label 27771, and 20003 operates on the send-receive pair of 37772. Notice that link 30005 has a label of 0. This means that it does not exist.

## 6 The Beamform Algorithms

### 6.1 An Overview

Over seventy MATLAB functions and script files constitute the beamform algorithms. They have been tested using Version 4.2a MATLAB on a DEC Alpha 3000. The functions' purposes range from geometric utilities to specialized tree searches. The following subsections provide a summary of the main procedures needed to find minimally interfering network configurations. A brief description of all the function and scripts are presented in the appendix.

Three basic steps are required to solve this interference topology problem:

1. Defining node coordinates
2. Creating the $(U, L, T, R)$ Model

3. Finding acceptable network configurations

These steps can be entered as text commands, selected from a graphic user interface, or a combination of both. All commands have online help accessed by typing **help** then the command name.

## 6.2 Command Line Interface

### 6.2.1 Defining node coordinates

The $GPS$ matrix can be created in several ways. The coordinates can be entered into the matrix directly with the left column containing the x-coordinate and the right column contains the y-coordinate. Random node locations may also be incorporated into the $GPS$ matrix through the following procedures:

**MakeGPS** Creates, sorts, graphs, and labels random $GPS$ coordinates

**AddRandPt** Adds a bounded uniform random point to $GPS$

Other useful $GPS$ manipulating tools are

**DeleteNode** Removes a node from the $GPS$ matrix

**SortGPS** Sorts $GPS$ matrix by ascending x-coordinates

In addition, several graphical utilities allow easy viewing and manipulation of the $GPS$ matrix.

**LabelGPS** Displays index labels on nodes

**PlotGPS** Plots and labels the nodes in $GPS$

**gAddGPS** Graphically adds a node to the $GPS$ matrix

### 6.2.2 Creating the $(U, L, T, R)$ Model

Many functions exist to create and manipulate the $(U, L, T, R)$ model; however, the main function, **MakeULTR2**, is the most important. This function requires two inputs: the $GPS$ coordinate matrix and a configuration vector ($Config$). The configuration is a row vector that contains five parameters. The first is the linking radius. This is the maximum distance for which a communication link may be formed. The second is the maximum number of frequencies allowed for the network. The third is the interference multiplier. This value multiplied by a link's actual length will determine the range of interference created by the link. Basing the interference multiplier on actual link

length implicitly assumes adaptive power control in the transmitter. The last two parameters are the beam and receiving beamwidth. These are the entire angles across the transmitting patterns. In summary, the $Config$ variable has the following format:

$$Config = \begin{pmatrix} R_{link} & F_{max} & I_{mult} & T_{width} & R_{width} \end{pmatrix}$$

The full syntax of the command is

```
[U,L,T,R,X1,X2] = MakeULTR2(GPS,Config)
```

The $(U, L, T, R)$ model is critical for the tree search procedure.

## 6.3 Finding Acceptable Network Configurations

The main procedure to find all the fully connected non-interfering network is **SolveModel**. This is a recursive tree search procedure that will return all acceptable configurations. The syntax for this command is the following:

```
NetTot = SolveModel(NetTot,u,U,L,T,R,NetSize)
```

The variables $U$,$L$,$T$, and $R$ are the real world model. $NetSize$ is the total number of nodes in the network. This variable is required for the connectivity test. The input $u$ is a unit that can receive the labels in $L$. When starting the algorithm use $T(1,1)$ as the starting unit. $NetTot$ are all previously found acceptable network configurations.

Once the real world model has been created the following command will find all acceptable network configurations.

```
NetTot = SolveModel([],T(1,1),U,L,T,R,NetSize)
```

The output variable $NetTot$ contains all the acceptable configurations among the constrained links appearing in $T$. A portion of $NetTot$ from the previous RDRN example is shown below.

```
NetTot =

  Columns 1 through 6

      10002       17772       10002       17772       10002       17772
      20003       27771       20003       27771       20003       27773
      30004           0       30004       17773       30004           0
```

```
     30005       17773       30005           0       30005       37771

  Columns 7 through 12

     10002       17772       10002       17772       10002       17772
     20003       27773       20003       37771       20003       37771
     30004       37771       30004           0       30004           0
     30005           0       30005       17773       30005       27773

  Columns 13 through 18

     10002       17772       10002       17772       10002       17773
     20003       37771       20003       37771       20003       27771
     30004       17773       30004       27773       30004           0
     30005           0       30005           0       30005       17772
```

Each unit in $T$,receives its own row. A useful utility to select a consistent labeling from $NetTot$ is **SelectCL**. For example, to select the fourth configuration, type

```
CL = SelectCL(NetTot,4)
```

at the MATLAB prompt.

The function **DrawNet2** will graphically display a network configuration in a figure window. To display the selected configuration $CL$ type

```
DrawNet2(GPS,U,CL,FMap2)
```

at the command prompt. The last variable is a color map that contains the decimal percentages of red, green, and blue colors for each individual frequency. For example, if $L$ is composed of three frequencies then an appropriate color map that assigns black to no link, red to $f_1$, green to $f_2$, and blue to $f_3$, would be

```
FMap2 =

         0         0         0
    1.0000         0         0
         0    1.0000         0
         0         0    1.0000
```

## 6.4 The Graphic User Interface

In addition to the typing commands at the prompt, the beamform algorithms may be operated from a GUI. To initiate the GUI type **RDRNControl** at

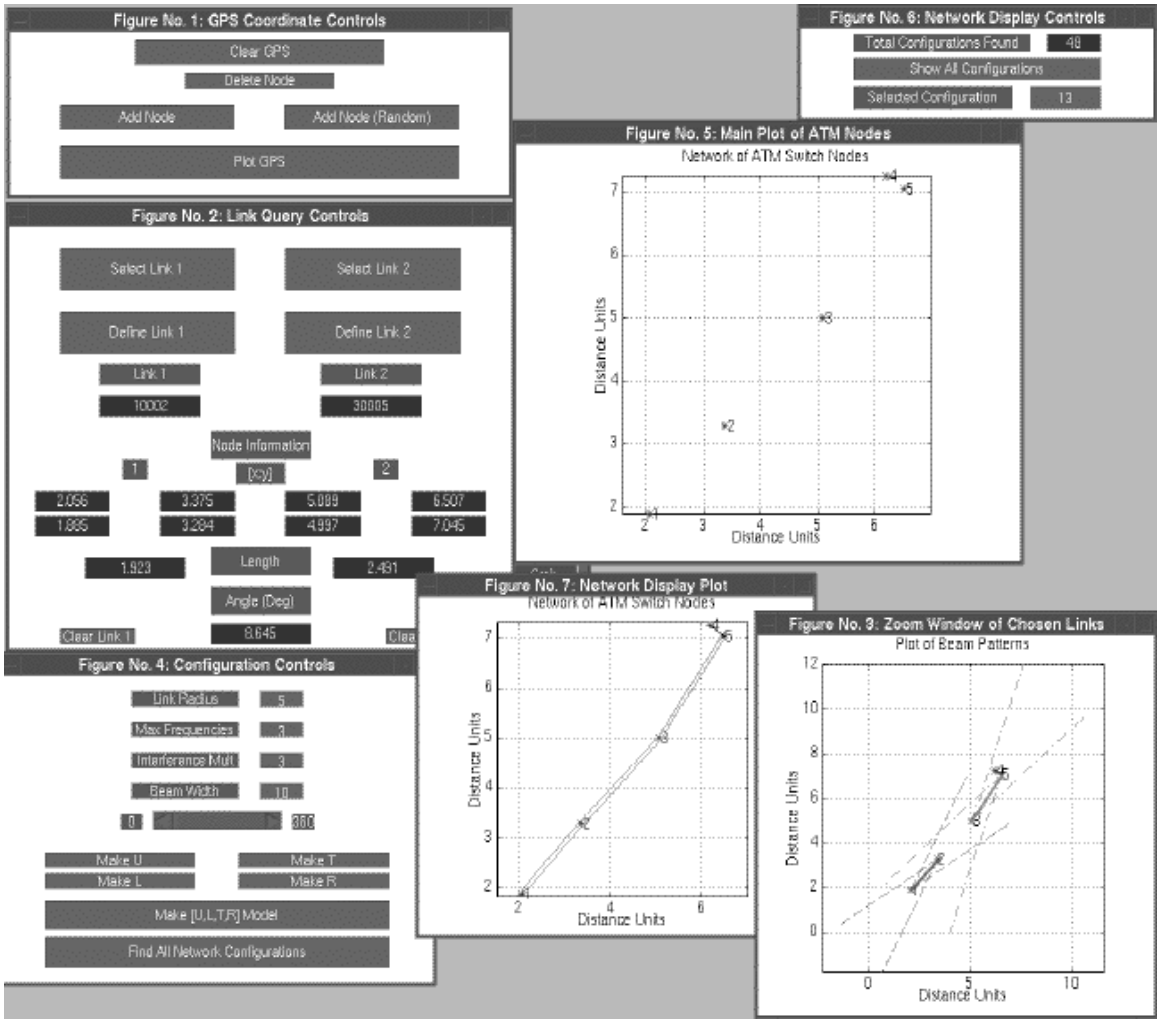



Figure 7: The Graphic User Interface for the Beamform Algorithms

the prompt. This will bring up seven "figure" windows (see Figure 7. Figure window one is the $GPS$ coordinate controls. Nodes may be added or deleted from $GPS$ graphically by using these controls and the main plot of the ATM nodes in window number 5. Window number two contains the link query controls. This allows the user to select a link from $U$ by clicking on its midpoint, or defining a link by clicking on two nodes. To use the *Select Link* option of the query window $U$ must have already been created. Since links are constrained

in pairs, two links may be examined simultaneously. Statistics such as length, node position, and angle between links are shown in this window. When a link is selected or defined, a plot appears in window number three showing its transmitting patterns. The length of the pattern is determined by the link length multiplied by the interference multiplier. The receiving pattern is identical to the transmitting pattern except it has infinite length. By selecting or defining two links, potential interference may be observed visually. The fourth window contains the configuration and model controls. Each component of the real world model may be selected individually or consecutively. This window also starts the tree search to find all acceptable network configurations. Figure window five is the main plot of the ATM nodes. Whenever graphical input is required, it must be selected from this window. Window six contains the network display controls. Once configurations have been found, they may be displayed individually or all at once. To display all configurations select the button in the middle of the window. Starting with the first, configurations will be displayed in window seven by pressing any key in the MATLAB command window. Note: to cancel any command accessed through the GUI, press **control-c** at the MATLAB command prompt.

# 7 Improving Efficiency

The following are ideas that could increase the efficiency of creating the $(U, L, T, R)$ model and the tree search used to find network configurations.

## 7.1 Direct and Indirect Constraints

Using an idea similar to the connectivity matrix, a network consisting of a large number of constraints may be subdivided into several smaller problems. Each one of these smaller consistent labeling problems then can be solved individually. By combining the resulting label solutions, a configuration for the full network is found.

After $T$ is created, each row or constraint may be represented as an element in a square matrix. The size of this matrix is determined by the number of constrained links in $T$. Each link in $T$ may be given a positive index. If link $i$ constrains link $j$, as determined in $T$, then the $(i, j)$ and $(j, i)$ elements of this relation matrix will contain a one; otherwise, they will be zero. The same tree search that finds communication classes can then determine which links will constrain each other indirectly. These smaller collections of constraints can then be used to create a smaller $(U, L, T, R)$ model. After finding labelings

for each of the smaller models, the solutions are combined to form consistent labeling solutions for the original larger problem. These final solutions must then be checked for connectivity, since connectivity is a constraint placed on the entire network.

This method can increase the efficiency since the computational time of a tree search is exponential. By solving smaller models the speed is greatly increased; however, for complicated networks where each link indirectly constrains every other link, this method will not decrease the tree search time. At this point in time, this idea has not been implemented into code.

## 7.2 Critical Links

It is sometimes necessary for certain links to exist to ensure full network connectivity. These critical links can be identified using the connectivity test. With this information, 4-tuples from $R$ can be removed that have critical links labeled as zero. This will increase the performance of the tree search by eliminating bad link configurations from $R$.

## 7.3 Another Language

Speed is one of the disadvantages of using MATLAB to execute the beamform code. Since it must interpret every script and function file before it is executed, processing time is slow. By converting the algorithms into a faster language or using more of MATLAB's built in functions, the computational time may be greatly decreased.

# A Summary of the Beamform Algorithms

Each command has online help that gives a brief description of the function and its syntax. For more detailed information about a procedure, read the comments at the beginning of each file.

## A.1 *GPS* Manipulating Algorithms

The following commands add, delete, label, sort, and plot nodes of the RDRN network.

**AddRandPt** Adds a uniform random point to *GPS*

**DeleteNode** Removes a node from *GPS*

**GPSBox** Loads the GUI Box for $GPS$ control

**LabelGPS** Graphically labels nodes with their index value

**MakeGPS** Creates, sorts, and graphs a random $GPS$ matrix

**PlotGPS** Plots and labels $GPS$ coordinates.

**SortGPS** Sorts nodes based on x-coordinate

**gAddGPS** Graphically adds a node to $GPS$

## A.2 GUI Algorithms

These functions and scripts provide a graphical interface and allow graphical inputs and displays.

**ClearLinkGUI** Clears a link from the query control box

**ConfigBox** Loads configuration control box

**DefineLinkGUI** Defines a link by selecting two nodes

**DisplayBox** Loads the display control box

**DrawAllU** Displays all the links contained in $U$

**DrawLink** Displays a link with one color

**DrawLink2** Displays a link using a color for each frequency

**DrawNet** Displays a network configuration using $DrawLink$

**DrawNet2** Displays a network configuration using $DrawLink2$

**DrawPat** Displays the beam pattern of a node

**GPSBox** Loads the $GPS$ control box

**LabelGPS** Graphically labels nodes with their index value

**PlotGPS** Plots and labels $GPS$ coordinates

**QueryBox** Loads the link query box

**RDRNControl** Loads and sets defaults for GUI controls

**SelectLinkGUI** Selects a link in $U$ by its midpoint

**SetConfigGUI** Sets $Config$ variable based on the the configuration controls

**SetTotGUI** Sets the *Total Configurations Found* on the display controls

**ShowConfigGUI** Displays selected or all network configurations

**ZoomBox** Loads box that shows link patterns

**gAddGPS** Graphically adds a node to $GPS$

**gSelectLink** Graphically finds the nearest link

**gSelectNode** Graphically finds the nearest node

## A.3 Network Geometry Algorithms

These functions determine the geometric relationships between links, nodes, points, and lines.

**Ang** Calculates the angle between three pts

**Dist** Calculates the distance between two points

**FarDist** Calculates the distance between the farthest nodes in two links

**IntRange2** Determines if two links are in each other's interference range

**LinkAngle** Determines the angle between two links

**LinkInter** Determines the point where two links intersect

**LinkLength** Determines the length of a link

**LinkLine** Calculates the line representation of a link

**LinkMid** Determines the midpoint of a link

**MidDist** Calculates the distance between midpoints of two links

**PDist** Calculates the perpendicular distance from a line to a point

## A.4 Conversion Algorithms

These functions convert one format or unit to another.

**Deg2Rad** Converts degrees to radians

**Freq2Color** Converts a frequency pair into their color representation

**Freq2Label** Converts a frequency pair into their proper label in $L$

**Label2Freq** Converts a label into the frequencies that it represents

**Link2Node** Converts a link back to its corresponding nodes

**Node2Coord** Converts a node back to its coordinates

**Node2Link** Creates a unit or link given two nodes

**Rad2Deg** Converts radians to degrees

## A.5 Connectivity Algorithms

These functions convert a network configuration into a connectivity matrix. A tree search can then determine communication classes of this matrix.

**AddToClass** Adds a group of states to a class

**AdjState** Determines direct communications

**CL2Mat** Converts a labeling into a connectivity matrix

**CanAccess** Determines which nodes are accessible from a given node

**CanComm** Finds communicating states of a stochastic matrix

**FindClass** Finds communication classes of a stochastic matrix

**IsOneClass** Determines if the network is fully connected

**RemoveOneState** Auxiliary procedure used to direct the classification tree search

## A.6 Consistent Labeling Algorithms

These functions create, manipulate, and solve the model for the RDRN consistent labeling problem.

**AreCL** Determines if a proposed group of labelings is consistent to $T$ and $R$

**CLAdd** Adds a single labeling to a cumulative group of labelings

**CLabel** Converts a single valued $R$ to a consistent labeling solution

**FindV** Used to find next unit for consideration in the tree search

**IS** Determines if the labeling is single valued

**IsCL** Determines if one proposed labeling is consistent with $T$ and $R$

**IsTInR** Determines if all members of $T$ are in $R$

**IsULPair** Determines if a unit-label pair occurs in R

**LabelLeft** Determines how many labels are left for a unit in $R$

**MakeL2** Creates a set of frequency pair labels ($L$)

**MakeR2** Creates the unit-label constraint relation ($R$)

**MakeT2** Creates the unit constraint relation ($T$)

**MakeU** Creates all possible links ($U$)

**MakeULTR2** Creates a Haralick $(U, L, T, R)$ model of a network

**RemoveOneLabel** Auxiliary procedure used to direct the tree search

**RestrictR** Creates $R'$

**SelectCL** Selects a single consistent labeling from a collection of solutions

**SolveModel** Finds all acceptable network configurations from a $(U, L, T, R)$ model of the network

**UnitOfR** Finds the units remaining in $R$

**UintOfT** Finds the units remaining in $T$

### A.7 Miscellaneous

These are functions that did not fit into any other category.

**IsElement** Searches a matrix for a specified element

**IsMember** Searches a matrix for a particular row

# References

[HAR79] Haralick, Robert M. and Shapiro, Linda G.,"The Consistent Labeling Problem", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, April 1979.

[HAA94] Haas, Shane M.,"Simulations and Applications of Markov Chains", August 1994.