MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

SUMMER PRACTICE REPORT CENG 300

Student Name & Number: Ahmet Eren Çolak - 2587921

Organization Name: TÜBİTAK BİLGEM İLTAREN

Start-End Dates: 24.07.2023 - 04.09.2023

Total Working Days: 30

Address: Ümit Mh. Şehit Mu. Kur. Yzb. İlhan TAN Kışlası, ANKARA

Student's Signature **Organizational Approval**

Contents

List of Figures

List of Tables

1. Description of the Company

1.1 Company Name

Scientific and Technological Research Council of Türkiye, Advanced Technologies Research Institute
(TÜBİTAK İLTAREN)

1.2 Company Location

Ümitköy, 2432. Cad. 2489. Sok. Şehit Mu. Kur. Yzb. İlhan TAN Kışlası, Çankaya, Ankara 06800, Turkey.

1.3 Organizational Structure of the Company

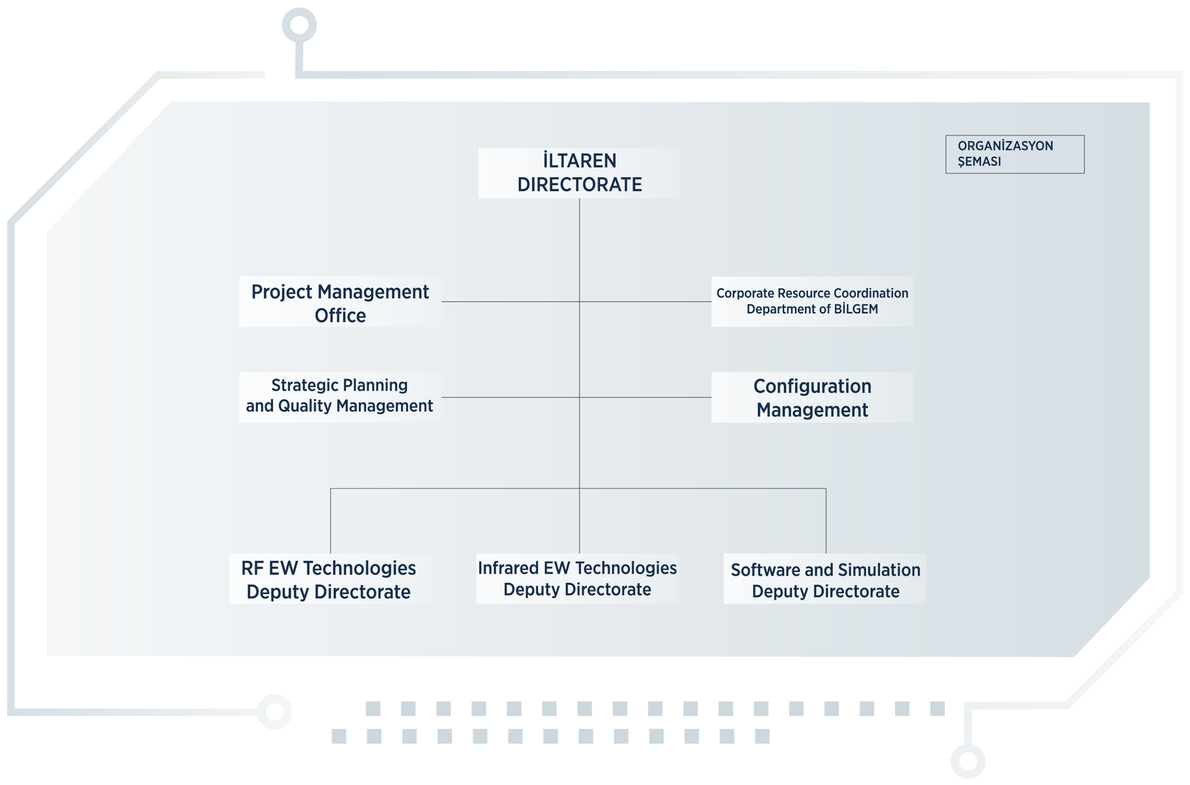

ORGANİZASYON ŞEMASI
İLTAREN DIRECTORATE
Project Management Office
Corporate Resource Coordination Department of BİLGEM
Strategic Planning and Quality Management
Configuration Management
RF EW Technologies Deputy Directorate
Infrared EW Technologies Deputy Directorate
Software and Simulation Deputy Directorate

Figure 1.1: Organizational structure

1.4 Number and Duties of Engineers Employed

Duties and number of employees can not be shared due to confidentiality policies of the company.

1.5 Main Area of Business

Advanced Technologies Research Institute (İLTAREN) performs the tasks of conducting research on all kinds of software, hardware and equipment in the field of Electromagnetic Warfare, developing technologies and prototypes, creating methods and processes for developing and evaluating EW techniques and tactics, creating infrastructures for testing, measurement, analysis and evaluation activities for EW systems, and conducting scientific research in related fields of activity.

1.6 Brief History of the Company

The Advanced Technologies Research Institute (İLTAREN) is an organization based in Turkey that specializes in electronic warfare and advanced technology research. Here is a brief history of İLTAREN:

1999 - Establishment: İLTAREN was established as a result of a protocol signed between the General Staff Communications Electronics and Information Systems Directorate and TÜBİTAK (The Scientific and Technological Research Council of Turkey). This protocol laid out the framework for cooperation in establishing the institute.

2000 - Inception: İLTAREN officially began its activities under the TÜBİTAK Presidency in the year 2000. Its primary mission was to enhance the electronic warfare capabilities of the Turkish Armed Forces (TAF) through advanced research and technology development.

2003 - Restructuring: İLTAREN underwent restructuring, with the signing of a second protocol. It continued its operations as a unit within the National Research Institute of Electronics and Cryptology (UEKAE).

2007 - National Frequency Management System: İLTAREN achieved a significant milestone with the development of the first version of the National Frequency Management System. This system is crucial for managing and allocating radio frequencies efficiently.

2010 - International Contributions: İLTAREN developed the ARCADE product to meet NATO Spectrum Management needs and actively contributed to NATO Electronic Warfare exercises.

2011 - Infrared Trace Analysis Software: İLTAREN developed Infrared Trace Analysis Software, which is likely a significant asset in the field of electronic warfare.

2012 - Engineering Support Contracts: İLTAREN signed contracts with the Turkish Land Forces Command and NATO for engineering support in the field of electronic warfare and Spectrum Management needs, respectively.

2013 - İLTAREN Institute: İLTAREN began to serve as an institute affiliated with BİLGEM (Informatics and Information Security Advanced Technologies Research Center).

2014 - Hardware Development: İLTAREN continued its hardware development efforts, including the development of the Digital Tactical Simulator for Electronic Warfare self-defense.

2016 - Hardware In-Loop Laboratories: İLTAREN expanded its capabilities with the development of Hardware In-Loop Laboratories for self-protection methods against both radio frequency (RF) and infrared systems.

2017 - Multi-Purpose Phased Array Radar Project: İLTAREN contributed to radar technology development within the scope of the Multi-Purpose Phased Array Radar Project.

2018 - Instrumented Infrared Seeker Test System: İLTAREN developed the Instrumented Infrared Seeker Test System, further advancing its expertise in infrared technology.

2019 - Engineering Support Agreement: İLTAREN signed an agreement with the Turkish Air Forces Command to provide engineering support in the field of electronic warfare.

2020 - ESEMOD Software: İLTAREN expanded the use of its ESEMOD software to work on MİLGEM class ships.

2021 - Jamming Code Development: İLTAREN developed laboratory infrastructure for the development of jamming codes used in Directed Infrared Countermeasure Systems.

2022 - National Combat Aircraft Project: İLTAREN played a role in the development of the integrated processor unit and avionic interface platform hardware for the National Combat Aircraft Project.

2023 - Radar Analysis Software: İLTAREN delivered Radar Analysis Software for the National Joint Electronic Warfare Data Bank (MMEHBB) and celebrated the first functional flight of the Electronic Warfare Pod on the F16.

Throughout its history, İLTAREN has consistently contributed to the advancement of electronic warfare technologies, Spectrum Management, and related fields, serving as a vital asset for the Turkish Armed Forces and international partners.

2. Introduction

This report provides a summary of the work I have done and experiences I have gained during my 6-week summer practice at the Advanced Technologies Research Institute (İLTAREN).

Throughout my internship at the Software and Simulation Deputy Directorate, I mainly worked on development of algorithms for specific problems and implementation of data structures required for the implementation of algorithms.

Tasks I have accomplished can be inspected in three parts:

- Implementation of statically allocated vector and linked list
- Approximation of the longest path in a graph
- De-interleaving radar pulses based on time of arrivals.

3. Project

3.1 Longest Path in a Graph

3.1.1 Analysis Phase

Problem Definition: There exists a CSV file which contains distances from 81 cities of Turkey to all other 81 cities in Turkey. Using this distance matrix read from that CSV file, you are expected to compute the maximum amount of city that can be traveled (without visiting a city twice) only using the roads which its distances are in the interval of $[x - y, x + y]$. The values x (distance) and y (tolerance) are given by the user.

Example CSV file format:

```
ADANA;ADIYAMAN;AFYONKARAHISAR;AGRI;AMASYA;ANKARA;  ...
01;ADANA;0;338;579;978;604;492;548;1005;881;903;78 ...
02;ADIYAMAN;338;0;917;646;628;738;886;717;1219;124 ...
03;AFYONKARAHISAR;579;917;0;1320;596;255;287;1225; ...
04;AGRI;978;646;1320;0;740;1056;1426;365;1635;1576 ...
05;AMASYA;604;628;596;740;0;331;824;679;936;842;62 ...
06;ANKARA;492;738;255;1056;331;0;543;960;596;536;3 ...
07;ANTALYA;548;886;287;1426;824;543;0;1433;346;506 ...
08;ARTVIN;1005;717;1225;365;679;960;1433;0;1565;14 ...
09;AYDIN;881;1219;340;1635;936;596;346;1565;0;291; ...
10;BALIKESIR;903;1241;324;1576;842;536;506;1452;29 ...
11;BILECIK;786;1039;207;1361;626;314;473;1236;525; ...
12;BINGOL;633;348;1099;352;642;889;1191;373;1401;1 ...
13;BITLIS;746;414;1290;232;833;1080;1294;547;1592; ...
14;BOLU;689;927;417;1146;411;190;683;1021;735;431; ...
...
```

When the distances which does not fit in the interval of $[x - y, x + y]$ are replaced with the zero (0) and the rest is set to one (1), distance matrix becomes into a adjacency matrix of an unweighted graph. Then the goal becomes into computing the longest path in that graph. How dense or how sparse the graph is controlled by the tolerance variable (y), and combinations of edges in the graph is adjusted by the distance variable (x).

Finding an exact solution to this problem requires comparing all different permutations of cities and checking whether they form a path. This process requires comparison of $n!$ different orderings. Therefore it is not possible to find an exact solution to this problem in a polynomial time. When the case with our problem is considered where $n = 81$ and

$n! \approx 5.8 \cdot 10^{120}$, computation of an exact solution may take many months even for a modern computer.

Therefore, computing an approximate solution in polynomial time rather than an exact solution is much more feasible. To compute an approximation of the longest path, I decided to follow a heuristic approach where the locally optimum neighbouring city is chosen to continue with at each iteration. This therefore raises the question of what kind of heuristics should be used for choosing an ideal local optimum at each step ?

3.1.2 Design Phase

Assume that we are looking for a long path in the unweighted and undirected graph $G = (V, E)$, starting from the vertex $s \in V$. Let N be the set of all vertices which have an edge with vertex s, in other words N is the set of neighbours of s.

Then the process of approximating a long path in graph G follows as:

1. Compute a score value for each vertex in N.
2. Choose the vertex with the maximum score.
3. Update N as the set of neighbours of the chosen vertex.
4. Go to 1 until there is no vertex left to visit.

Computing the score for a vertex is the most critical task which entirely determines the performance of the algortihm and the length of the approximated path. Heuristic functions should be chosen such that even when the graph is sparse (which means tolerance variable y, is low) algorithm should not perform poorly.

First thing I thought was the direct neighbour count of the vertices. Idea of choosing the vertex with the most neighbours leads to longer paths, seems reasonable at first sight however it is not. Because choosing the vertices with most neighbours repetitevely, leaves out the vertices with few neighbours. That makes it hard for visiting vertices with few connections in the future. Although this heuristic does not seem useful right now, it can be helpful for the solution of the problem as I am going to explain later.

Since count of direct neighbours is known to be a poor choice of heuristic for finding long paths, it can be thought that maximum count of second order neighbours may perform better. Maximum count of second order neighbours, is the maximum of count of direct neighbours which a vertex's direct neighbour has. On average, it actually performs better than the count of direct neighbours.

Observation of the increase in the performance with looking deeper in the graph for the count of neighbours, leads to trial of maximum count of the third and forth order neighbour counts. Maximum count of third order neighbours, is the maximum of count of second order neighbours which a vertex's direct neighbour has. Same idea applies for the definition of the forth order neighbours. Although choosing the heuristic as the maximum count of third order neighbours performs better than the heuristic of maximum count of the second order neighbours, when heuristic is chosen to be the maximum count of the forth order neighbours it performs poorer than the third order one. Decrease in the performance continues as the order of neighbours are increased which is used in the heuristic.

Since heuristics related to neigbour counts can not be improved any more, I looked for different kinds of metrics of graphs. Centrality metrics which gives information about the positions of vertices in the graph, may be useful for finding long paths. Motivation of using indicators of centrality originates from the idea that long paths tend to contain more vertices which are positioned at the central regions of the graph.

Closeness centrality is one of the methods to measure centrality of a vertex in a graph. Closeness centrality or closeness of a vertex is the average length of the shortest path between the vertex and all other vertices in the graph. Thus the more central a vertex is, the closer it is to all other nodes. Closeness centrality of a vertex v is computed as below where $d(u, v)$ is the shortest distance between vertices u and v:

$$C(v) = \frac{1}{\sum\limits_{u \in V} d(u, v)} \tag{3.1}$$

Another measure of centrality is the betweenness centrality. Betweenness centrality is the measurement of how many times a vertex acts as a bridge between shortest path of any two vertices in the graph. Vertices which are at the central regions tend to be contained in many paths in the graph, because of this number of times a vertex being in any paths in graph is a measurement of centrality. Betwenness centrality of vertex v is computed as below, where σ_{st} indicates the total number of shortest paths from node s to t and $\sigma_{st}(v)$ is the number of those paths which contains vertex v.

$$C_B(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{3.2}$$

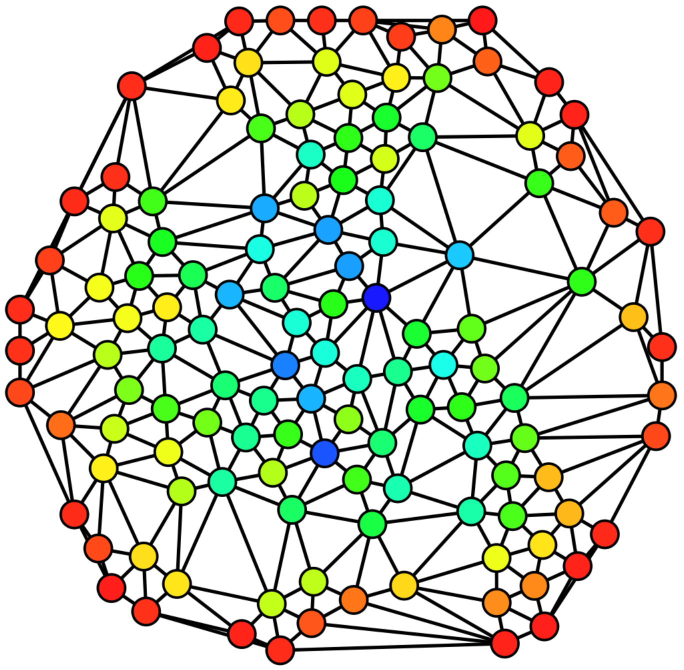


Figure 3.1: Betweenness centrality of vertices in a graph

Apart from the centrality metrics, another heuristic approach for searching long paths could be considering the total length up to vertices which are available to be chosen. Rationale for this method is that as algorithm iterates further, it may get harder to visit closer vertices to the starting vertex of the travel. Therefore choosing the vertex which is closest to starting vertex may lead to longer paths.

An alternative of distance to starting vertex heuristic is considering the distance to previously visited vertex. Aim is to choosing the vertex which is closest to previously visited vertex, thus leading a long path in graph. Downside of this heuristic compared to distance to previous heuristic is that there are only two score values for a vertex. The score is either 1 or 2 for any vertex. Because, score of a vertex is 1 if and only if a vertex has a direct edge with the previously visited vertex. Otherwise it's score is always 2, since shortest path from that vertex to previously visited vertes has to contain only the current vertex of the iteration. Therefore this approach may fail to distinguish multiple neighbours of a vertex, when compared with the distance to previous heuristic.

To benefit from different aspects of all heuristics mentioned above, linear combination of heuristic functions can be used for computing a score for vertices. Since, it is known that visiting vertices with the most direct neigbour counts performs poorly, weight of this function may be set to a negative value. As a result of trial and error, best performing weights and functions are found to be as below where H_{LC} corresponds to heuristic function of the linear combination, H_{N1} corresponds to heuristic function of first order neigbours, H_{CC} corresponds to heuristic function of closeness centrality, H_{N3} corresponds to heuristic function of third order neighbours and H_{DS} corresponds to heuristic function of distance to start.

$$H_{LC}(v) = -0.1 \cdot H_{N1} + 0.2 \cdot H_{N3} + 0.5 \cdot H_{CC} + 0.1 \cdot H_{DS} \tag{3.3}$$

3.1.3 Implementation Phase

Algorithm iterates as choosing a predicted locally optimum vertex among the neighbouring vertices of the starting vertex, then repeating the same process with the chosen vertex. Therefore, a recursive function can be written to accomplish entire process as in the example below.

```
int find_max_travel(int start, bool[] visited, int[][] adjmatrix) {
    visited[start] = true; //Mark currently visited vertex

    int max_vertex = -1; double max_score = 0;
    //Iterate all neighbours of starting vertex
    for (int i = 0; adj_matrix[start][i] && i < SIZE; ++i) {
        if (visited[i]) continue; // Skip visited vertices

        //Compute a score for i'th vertex
        double score = score(i, visited, adjmatrix);
        if (score > max_score) { //Set the vertex with the max score
            max_score = score;
            max_vertex = i;
        }
    }
    if (max_vertex != -1) //If a vertex found, continue with it
        return 1 + find_max_travel(max_vertex,visited, adjmatrix);
    return 1; //If not found return 1
}
```

The most crucial part of the algorithm is the behaviour of the `score()` function which solely determines the city to be selected. Therefore different choice of heuristics for the score function are going to yield different outputs.

I implemented an abstract class representing the behaviour of the score function called `LocalOptima` which only has a single abstract function which is the `score()` function.

```
class LocalOptima {
public:
    virtual double score(int node, bool[] visited, int[][] adjmatrix) = 0;
};
```

All of the heuristic functions are implemented as the child class of the `LocalOptima` class.

```
class FirstOrderNeighbors : public LocalOptima {
public:
    double score(int node, bool[] visited, int[][] adjmatrix) {
        int neighbours = 0;
        for (int i = 0; i < SIZE; ++i) {
            if (adjmatrix[node][i] && i != node && !visited[i]) neighbours++;
        }

        return neighbours;
    }

};

/*  Implementation of the second and third order neighbours
    are similar to the first orders */

class SecondOrderNeighbors : public LocalOptima {
public:
    double score(int node, bool[] visited, int[][] adjmatrix) {
        /*  Return the maximum amount of 'neighbur_count(v, ...)'
            which a direct neighbour v of the node has */
    }
private:
    //Returns the count of the first order neighbours of the node
    int neighbour_count(int node, bool[] visited, int[][] adjmatrix) {}
};

class ThirdOrderNeighbors : public LocalOptima {
public:
    double score(int node, bool[] visited, int[][] adjmatrix) {
        /*  Return the maximum amount of 'neighbour_count2(v, ...)'
            which a direct neighbour v of the node has */
    }
private:
    //Returns the count of the second order neighbours of the node
    int neighbour_count2(int node, bool[] visited, int[][] adjmatrix) {}
};
```

To compute closeness, betweenness centralities and other heuristics, shortest path between any two nodes must be computed. Since our graph is unweighted, shortest paths can be computed with breadth first search.

```
class ClosenessCentrality : public LocalOptima {
public:
    double score(int node, bool[] visited, int[][] adjmatrix) {
        int* dists = compute_distances(node, adjmatrix); int sum = 0;
        for (int length : dists) sum += length;
        return 1.0 / sum;

    }
private:
    int* compute_distances(int start, int[][] adjmatrix) {
        static int[] distances; //Array of distances to each city

        for (int i : distances) {
            //Start breadth first search from 'start' stop when node 'i' encountered
            //Store the path length in the i'th element in the array
            distances[i] = path_length;
        }
        return distances;
    }
};

class BetweennessCentrality : public LocalOptima {
public:
    double score(int node, bool[] visited, int[][] adjmatrix) {
        int total = 0, containing = 0;
        for (int i = 0; i < SIZE; ++i) {
            for (int j = 0; j < SIZE; ++j) {
                // Loop through every city pair (i, j)
                if (path_contains_node(node, i, j, adjmatrix)) containing++;
                total++;
            }
        }
        return containing / total;
    }
private:
    int path_contains_node(int node, int start, int end, int[][] adjmatrix) {
        /*  Start breadth first search from 'start' and stop when 'end'
```

```
            node is encountered. If 'node' is encountered during this search
            return 1 else 0;
        */
    }
};

class DistanceToStart : public LocalOptima {
public:
    DistanceToStart(int start = 25) : start(start) {}
    double score(int node, bool[] visited, int[][] adjmatrix) {
        int length = //find length with breadth search from node to start
        return 1.0 / length;
    }

    int start;
};
```

Linear combination is the weighted sum of the four heuristics, therefore it takes an array of coefficients as parameter in its constructor.

```
class LinearCombination : public LocalOptima {
public:
    LinearCombination(double weights[4], int start);
    double score(int node, bool[] visited, int[][] adjmatrix) {
        double sum = 0;
        for (int i = 0; i < 4; ++i)
            sum += weights[i] * hs[i]->score(node, graph, visited);
        return sum;
    }

    double weights[4];
private:
    FirstOrderNeighbors h1; ThirdOrderNeighbors h2;
    ClosenessCentrality h3; DistanceToStart h4;
    LocalOptima* hs[4] = { &h1, &h2, &h3, &h4 };
};
```

3.1.4 Testing Phase

Heuristic Function	**Worst Case Complexity**	**City Count From Ankara**	**City Count From Istanbul**	**City Count From Izmir**
First Order Neighbours	$O(n^3)$	21	16	25
Second Order Neighbours	$O(n^4)$	38	23	38
Third Order Neighbours	$O(n^5)$	55	53	46
Closeness Centrality	$O(n^4)$	60	48	39
Betweenness Centrality	$O(n^5)$	44	46	44
Distance to Previous	$O(n^4)$	14	18	22
Distance to Start	$O(n^3)$	14	23	22
Linear Combination	$O(n^5)$	76	55	50

Table 3.1: Longest path results with distance = 250, tolerance = 50

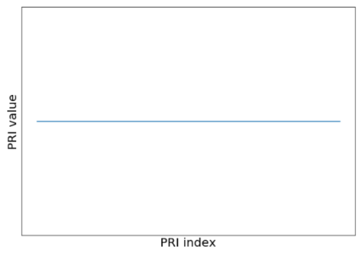
Heuristic Function	**Worst Case Complexity**	**City Count From Ankara**	**City Count From Istanbul**	**City Count From Izmir**
First Order Neighbours	$O(n^3)$	34	37	36
Second Order Neighbours	$O(n^4)$	50	50	40
Third Order Neighbours	$O(n^5)$	51	58	58
Closeness Centrality	$O(n^4)$	26	55	53
Betweenness Centrality	$O(n^5)$	54	46	56
Distance to Previous	$O(n^4)$	32	32	32
Distance to Start	$O(n^3)$	37	46	47
Linear Combination	$O(n^5)$	77	74	51

Table 3.2: Longest path results with distance = 350, tolerance = 50

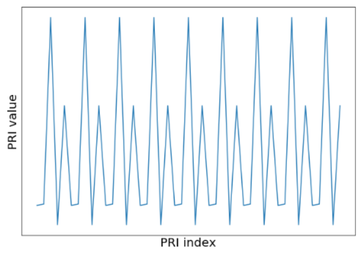
3.2 Radar Pulse De-interleaving

3.2.1 Analysis Phase

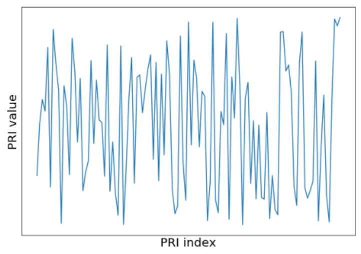
Problem Definition: There exists an array of numbers where each number represents the arrival time of an individual radar pulse received from several sources. The task is to determine different radars and which pulses belong to them. Radars may have different modulations such as stable, jittered, sliding, dwell and switch and staggered.



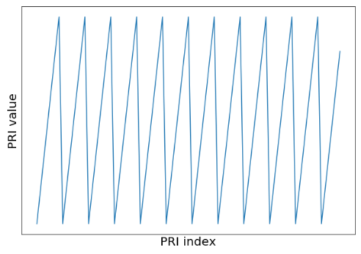
(a) Stable PRI.



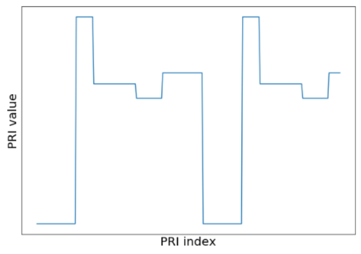
(b) Stagger PRI.



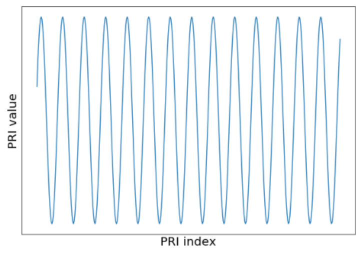
(c) Jitter PRI.



(d) Sliding PRI.



(e) Dwell and switch PRI.



(f) Sin PRI.

Figure 3.2: Radar modulation types

Pulse repetition interval (PRI) in radar is the time between consecutive radar pulse transmissions and is a critical parameter for determining a radar system's ability to distinguish between different targets.

Stable modulated pulses have a fixed pulse repetition interval (PRI) among all consecutive pulses. Staggered radar pulses have varying PRIs which the variation is periodic. Jittered modulation introduces random deviations to the PRI. Sliding radar pulses have increasing or decreasing PRI periodically. Lastly, dwell and switch pulses have alternating PRIs.

An example input with two sources of radar pulses with a stable modulation is given below:

```
10, 40, 50, 60, 80, 90, 100, 120, 130, 140, 160, 170
```

One of the sources produces the radar pulses with period 20, the other source produces with period 40. My task is to determine the different sources and their modulations.

3.2.2 Design Phase

To extract stable pulses, assuming that arrival times are stored in a sorted array, we begin by traversing the array and examining each arrival time individually. Our aim is to find a recurring pattern starting from the current element of the traversal. If a recurring pattern is found which starts from the current element of the traversal, then all elements of the array belonging to that pattern is marked to be ignored in future. Then the same process is applied to the next unmarked arrival time in the array until all elements are marked.

To handle cases with jittered pulses, an error threshold is determined to limit maximum and minimum deviations from the original PRI. In my implementation this error threshold is set to 10 percent of the original PRI value. Previous example with jittered pulses could be like below:

```
10, 41, 52, 62, 79, 87, 100, 121, 130, 141, 160, 173
```

For cases with sliding and dwell & switch modulated pulses, another approach is followed. All possible recurring groups are gathered in a list that meet the criteria for the sliding or dwell & switch modulations. Then, groups that belong the same source are combined together. An example of an increasing sliding modulated pulse array is given below:

```
10, 15, 22, 30, 50, 55, 62, 70, 90, 95, 102, 110
```

Each color here represents different group which their PRIs increase at the same rate. Approach I follow for extracting sliding and dwell & switch modulated pulses rely on extracting all possible groups first, then matching the groups that have the similar properties.

For the staggering case, another kind of groups are extracted and matched with other groups. Staggering groups are not periodic within themselves but as groups they are periodic. In other words, time between any two consecutive staggering group is constant. Here is an example of staggered modulated pulses:

```
10, 14, 16, 32, 50, 54, 56, 72, 90, 94, 96, 112
```

Each group has the same PRI values which are 4, 2, 16. As explained before, groups have a constant difference between them which is 40. Therefore if I can find the groups `[10, 50, 90]`, `[14, 54, 94]`, `[16, 56, 96]` and `[32, 72, 112]`, then I can refer that they are coming from the same source. Because period of all these groups are same and equals to 40.

3.2.3 Implementation Phase

The very draft implementation of the algorithm follows as:

```
int[] times;
bool[] visited = {false};

for (int i = 0; i < times.size(); ++i) {
    if (visited[i]) continue;

    /*
    ...
    Find a pattern starting from i'th element of the times array
    ...
    */

    visited[elements of the found pattern] = true;
}
```

Process of finding a recurring group is accomplished by looking the difference between current element of the array and the next unvisited element then searching for another elements which have the same period. Rest of the unmarked element are tried for other possible periods and the group with the most elements are marked.

```
int[] times;
bool[] visited = {false};

for (int i = 0; i < times.size(); ++i) {
    if (visited[i]) continue;

    int temp_times[] = subtract_element(times, times[i]);
    int maxlength = 0, period;
    int maxpattern[];
    //Try all possible periods for i'th item
    for (int j = i + 1; j < times.size(); ++j) {
        if (visited[j]) continue;
        int pred_period = temp_times[j] //Predicted period is temp_times[j]
        int pattern[];
        int length = 1;

        //Try to find a recurring pattern with period = temp_times[j]
        for (int k = j + 1; j < times.size(); ++k) {
            if (visited[k]) continue;
            int expected_time = (length + 1) * pred_period;

            //There is no possible value for rest of the array
            if (expected_time < temp_times[k]) break;

            else if (expected_time == temp_times[k]) {
                length++;
                pattern.push_back(k);
            }
        }

        if (length > maxlength) {
            maxlength = length;
            maxpattern = pattern;
            period = pred_period;
        }
    }

    visited[elements of the maxpattern] = true;
}
```

3.2.4 Testing Phase

Input 1: 15 20 25 45 61 75 89 100 105 133 140 155 165 182 195 220 222 223 260 285 301 350 416 480 -1
True Periods: 30 40 65

Output 1:
BUS COUNT: 3
PERIODS: 30 40 65

Input 1: 0 15 20 25 43 46 62 65 74 89 94 100 112 129 134 136 141 158 173 180 188 194 210 235 247 249 282 282 310 329 368 428 -1
True Periods: 23 37 47 59

Output 1:
BUS COUNT: 4
PERIODS: 47 59 23 37

4. Organization

4.1 Organization and Structure

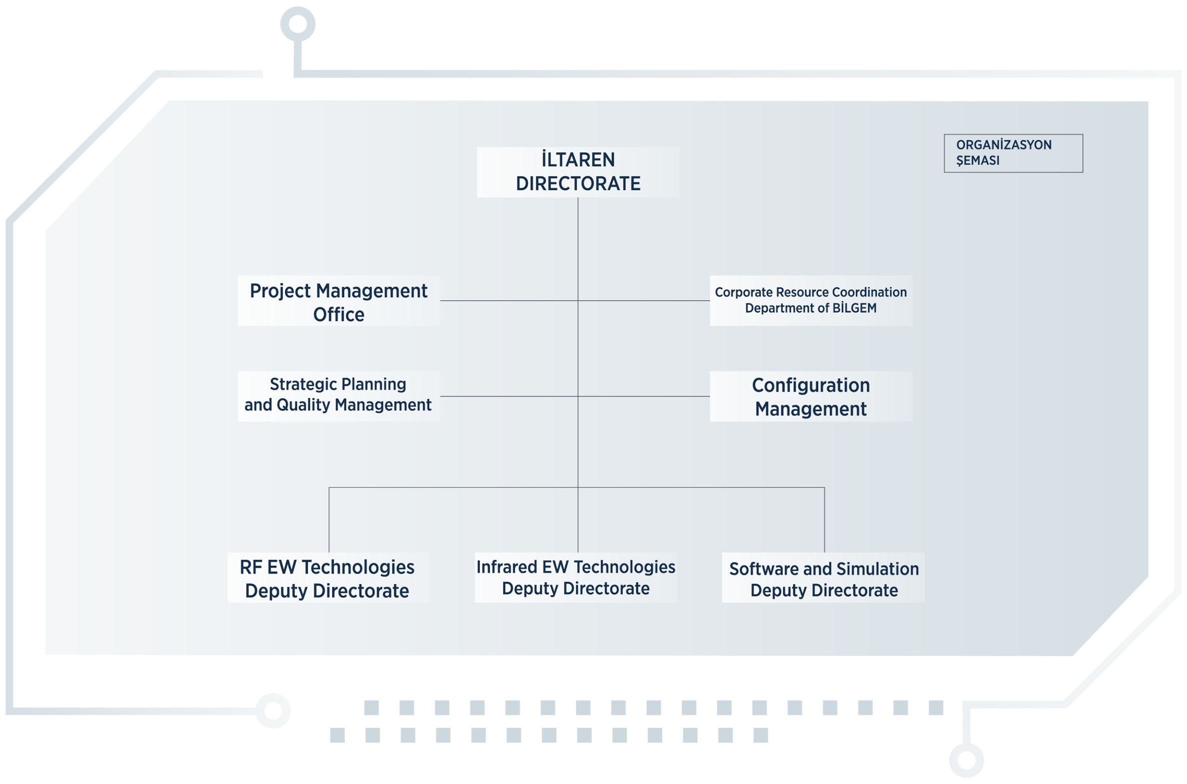

İLTAREN DIRECTORATE
ORGANİZASYON ŞEMASI
Project Management Office
Corporate Resource Coordination Department of BİLGEM
Strategic Planning and Quality Management
Configuration Management
RF EW Technologies Deputy Directorate
Infrared EW Technologies Deputy Directorate
Software and Simulation Deputy Directorate

Figure 4.1: Organizational structure

4.2 Methodologies and Strategies Used in the Company

This information can not be shared due to confidentiality policies of the company.

5. Conclusion

Throughout my internship at İLTAREN, I designed and implemented a heuristic algorithm for identifying the longest path in a graph, along with developing a radar pulse de-interleaving algorithm. These experiences expanded my practical knowledge in algorithm design and programming emphasizing the significance of theory and the real world applications.

6. Appendix