Optimal Signature Searching

A Dissertation Presented

By

Abdulaziz Alqarni

To

The Graduate School

In Partial Fulfillment of The

Requirements

for

The Degree of

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

May 2020

Stony Brook University

The Graduate School

Abdulaziz Alqarni

We, the dissertation committee for the above candidate for the

Doctor of philosophy degree, hereby recommend

Acceptance of this dissertation

Thomas Robertazzi – Dissertation Advisor

Professor, Department of Electrical and Computer Engineering

Wendy Tang – Chairperson of Defense

Associate Professor, Department of Electrical and Computer Engineering

Sangjin Hong, Member

Professor, Department of Electrical and Computer Engineering

Steven Skiena, Member

Distinguished Teaching Professor, Department of Computer Science

This dissertation is accepted by the Graduate School

Dean of the Graduate School

Abstract of the dissertation

Optimal Signature Searching

By

Abdulaziz Alqarni

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

2020

Signature searching is the process of finding a signature "pattern" of interest in a data file. Signature searching can be found in many applications such as radar, sensor and data processing. The objective of this research is to provide approaches for optimal signature searching. The plan is to study the signature searching optimization problem from multiple aspects such as mathematical programming, divisible load scheduling and topology. The focus of this research is sensor applications. An active sensor emits signals for detection and information acquisition purposes. Optimal sensing means the best sensing quality with the least time and energy cost, which allow processing more data. Intuitively, optimal sensing leads to optimal signature searching. This dissertation presents novel work by using an integer linear programming "algorithm" to achieve the optimal sensing by selecting the best possible number of signals of a

type or a combination of multiple types to ensure the best sensing quality possible considering all given constraints. Then, a solution based on a heuristic algorithm is implemented to improve the performance. Finally, an optimal processing method is presented. The considered processing method is local computing, cloud computing or a combination of both methods.

Key words

Signature searching, Integer linear programming, Sensors, Divisible load scheduling, Local computing, Cloud computing.

To my Parents and my loved ones

Contents

List of Tables

List of Figures

List of Abbreviation

i	Type of signals.
n_i	Number of signals per type.
q_i	Quality per signal of type i.
e_i	Energy per signal of type i.
t_i	Time per signal of type i.
Q	Total quality.
E	Total energy.
T	Total computation time.
E_c	The energy constraint.
T_c	The time constraint.
S_c	The number of signals per type constraint.
n:	The number of processors.
ω_0	The inverse of the computing speed of the source node.
ω_i	The inverse of the computing speed of the ith processor.
z_i	The inverse of the link speed of ith link.
T_{cm}	Communication intensity constant: the entire load is transmitted in $z_i T_{cm}$ seconds over the ith link.
T_{cp}	Computing intensity constant: the entire load is processed in $\omega_i T_{cp}$ seconds by the ith processor.
$S_{DLT}(n)$	The speedup with n processors in the systems using a DLT model.

$S_{DLT_{homo}}(n)$	The speedup with n homogeneous processors in the systems using a DLT model.
T	The clock period.
f	The fraction of load that is parallelizable.
$(1-f)$	The fraction of load that is serial.
S_p	The parallel system speedup.
S_{Amdahl}	The overall system speedup.

Acknowledgements

I would like to express my sincere gratitude to the following people, without whom I would not have been able to complete this research. First of all, I would like to thank Prof. Thomas Robertazzi for providing insight and expertise that greatly assisted this research. I really appreciate your support and patience. This research couldn't have been done without his help and guidance. I would like also to thank the defense committee Prof. Wendy Tang, Prof. Sangjin Hong and Prof. Steven Skiena. Thank you for being a part of this journey. I would like to give my sincere gratitude to my family. My parents and my sisters always support me and encourage me to do my best. Thank you so much my family, I love you and I appreciate everything you do for me. Last but not least, I thank all my friends and colleagues. Thank you very much for inspiring me to always do my best.

Chapter 1

Introduction

1.1 Backgrounds and Related Work

A signature is a relatively small data pattern in a very large data file. Signature searching is the process of finding a signature of interest in a large data file. Signature searching can be found in many applications such as radar, sensor data processing, signal processing, image processing, network security, DNA sequence analysis, large scientific experiments and speech recognition. Related to this is string matching and template matching. Ko and Robertazzi used divisible load scheduling theory to solve for the expected time for searching for both single and multiple signatures in certain multiple processor database architectures [1]. Ying and Robertazzi evaluated the performance of signatures searching in the nodes of parallel processors. The Authors studied networks configured as trees, two dimensional meshes and hypercubes [2]. Kyong and Robertazzi studied the optimal division of a linear file among the nodes in a network. The objective was minimizing the time of signatures searching in the file [3].

Wireless sensor networks are useful in diverse application areas. Monitoring applications, military applications, mobile commerce, smart offices and environmental science are examples of these application areas. Monitoring applications include medical health monitoring and structural health monitoring. Surveillance, target tracking, counter sniper, and battlefield monitoring are examples of military applications using wireless sensor networks. This developing technology reduces time and effort used in collecting data and monitoring events. It has advantages as well as

challenges and shortcomings. For example, it is convenient, flexible, and accurate. On the other hand, robustness, scalability, and security are examples of the challenges for this technology [4,5]. Sensing means the act of collecting information about an object. It may be split into passive sensors that gather radiation that is emitted or reflected by the object. Passive sensors mostly use reflected sunlight as the source of measured radiation. Examples of passive sensors include film photography, infrared and radiometers. Active sensing is when the sensor emits a signal and detects its reflection by the object. RADAR and LiDAR are examples of active sensing [6,7,8].

A sensor's sensitivity indicates how much the input quantity affects the sensor's output. For instance, the temperature changes by 1 °C if the mercury in a thermometer moves 1 cm. The sensitivity in that example is 1 cm/°C. Optimal sensing means the best sensing quality with the least time and energy cost, which allow processing more data [7]. Intuitively, optimal sensing leads to optimal signature searching. Researchers have optimized signal waveforms using various criteria [9,10,11]. In this thesis, an integer linear programming "algorithm" is used to reach the optimal sensing by selecting the best possible number of signals of a type or a combination of multiple types to ensure the best sensing quality possible considering all given constraints. A mathematical optimization problem in which the variables are restricted to be integers is called integer programming. Integer linear programming (ILP) means the objective function and the constraint decision variables are linear. The main reason for using integer variables when modeling problems as a linear program: The integer variables represent quantities that can only be integer. For example, it is not possible to build 4.7 cars or in this proposal send 2.5 signals. Integer linear programming can be used in many applications areas such as production planning where a possible objective is to maximize the total production, without exceeding the available resources [12,13,14]. Another example is scheduling such as vehicle scheduling in transportation networks.

Also, an example is telecommunications networks where the goal of these problems is to design a network of lines to install so that a predefined set of communication requirements are met, and the total cost of the network is minimal. Finally, cellular networks is another application area such as The task of frequency planning in 5G mobile networks which involves distributing available frequencies across the antennas so that users can be served and interference is minimized between the antennas [15,16].

In the second chapter, the solution is found using problem-based linear programming. In the third chapter, the solution is done using solver-based linear programming and a heuristic algorithm. Problem-based and solver-based are two approaches to solving optimization problems. The appropriate approach must be selected before solving a problem. First of all, Problem-Based Optimization Setup is easier to create and debug. The objective and constraints are represented symbolically. It requires translation from problem form to matrix form, resulting in a longer solution time. The second approach is solver-based optimization. The problem setup is harder to create and debug. The objective and constraints are represented as functions or matrices. It does not require translation from problem form to matrix form, resulting in a shorter solution time. It allows direct inclusion of a gradient or a Hessian. Also, it allows use of a Hessian multiply function or Jacobian multiply function to save memory in large problems [17]. The two approaches produce solution of the same quality. Theoretically, the solver-based solution can improve the performance. This is because the objective and constraints are represented as functions or matrices in solver-based solution. That representation eliminates the translation from problem form to matrix form which allow a shorter solution time. However, a heuristic algorithm can be a faster and more efficient method to solve a problem. Heuristic algorithms are useful to find approximate solutions when it is sufficient and exact solutions are computationally expensive. Heuristic

algorithms are commonly employed to solve the Knapsack Problem. In the knapsack problem, heuristics are used to find the maximum value by grouping a given set of items while being under a certain limit and its known as the Greedy Approximation Algorithm. It starts by sorting the items based on their value per unit. Then, it adds the items with the highest value per unit as long as there is still space remaining [18]. In the third chapter, a heuristic algorithm will be used to find the maximum quality under a specific set of constraints.

Data processing is an important part of the fast-growing computer and communication technologies. The expanding multiple processor technologies requires effective and efficient data processing scheduling. There are two types of processing. The first one is serial processing, which cannot be divided and processed simultaneously. The second type and the focus of this thesis is parallel processing. Parallel processing means loads of data are divisible among processors, which allow efficient and effective parallel processing [19]. Divisible load scheduling also known as divisible load theory utilizes linear mathematical models. DLT has many advantages such as easy computation, a schematic language, equivalent network element modeling. DLT can be implemented in many applications such as intelligent sensors, image signal processing and large data bases. Given expanding sensor information collection abilities, there is a requirement for execution time prediction tools. DLT provides scalable and tractable models to be used as an accurate prediction tool [20].

Divisible load scheduling consists of two steps: load distribution and load processing. The data is usually distributed from one or more processors to multiple processors and processed in parallel. An optimal scheduling provides the minimum finish time. In DLT, there are no precedence relations between the data, which allow data to be divided among a number of processors and links. Also, network architectural issues related to parallel and distributed

computing can be solved implementing DLT. Dividing the load equally among the processors does not take different computer and communication link speeds, the scheduling policy and the interconnection network into account and that leads to suboptimal solutions. However, an optimal solution can be found using divisible load scheduling theory, which provides the required mathematical tools. Furthermore, the solution can be improved by integrating Amdahl's and other speedup laws.

Amdahl's law is an accurate formula to calculate the speedup of the execution of a task with fixed data size [21,22]. For multiple processors networks, Amdahl's law can be used as a prediction tool for the theoretical speedup in parallel computing. For example, suppose a program finish time using a single processor is 10 hours. If the part of the program that cannot be parallelized takes one hour to execute and the part that can be parallelized takes the remaining 9 hours (p = 0.9) of execution, then the minimum execution time cannot be less than that critical one hour. Hence, the theoretical speedup is limited to at most 10 times $\frac{1}{(1-p)}$ =10. Gustafson's law is similar to Amdahl's law and used to calculate the speedup of a task but with a fixed execution time [23]. In other words, Amdahl's law can be used to improve the execution time of a specific workload. However, Gustafson's law can be used to improve the executed workload during a specific period of time.

It is important to have a complete understanding to avoid economic waste, inaccurate future system designs and a lack of technological improvements. The motivation for this research is "intelligent" sensor networks doing measurements, communications, and computation to reach optimal signature searching. The fourth chapter compares local and cloud computing as well as combining both methods. Local computing simply means all the computation occurs on a local

network. However, cloud computing means all the data computation and storage happens in the cloud via the internet [24].

1.2 Motivations and Contribution

The objective of this research is to perform optimal signature searching. The plan is to study the signature searching optimization problem from multiple aspects such as topology, divisible load scheduling and mathematical programming.

A sensor's sensitivity indicates how much the input quantity affects the sensor's output. Optimal sensing means the best sensing quality with the least time and energy cost, which allow processing more data [25,26]. Intuitively, optimal sensing leads to optimal signature searching. So, an integer linear programming "algorithm" is used to perform the optimal sensing by selecting the best possible number of signals of a type or a combination of multiple types to ensure the best sensing quality possible considering all given constraints.

The objective of the third chapter is to optimize the proposed solution in the second chapter using linear programming and heuristic algorithm. Instead of using problem-based linear programming such as in the second chapter, a solver-based solution is used to optimize the solution time. Solver-based linear programming solution will be presented and tested in the third chapter. Then, a solution-based on a heuristic algorithm will be implemented to improve the performance.

Divisible load modeling and speedup expressions have been developed for a variety of multi-processor interconnection topologies such as buses, stars, multi-level tree networks, meshes, hypercubes and other networks. Also, they have been developed for different load distribution policies such as sequential load distribution and concurrent load distribution with simultaneous or staggered start. Amdahl's Law can be modified and used to calculate the entire network speedup including serial and parallel data. The speedup of a divisible load model is implemented as the

parallel part of the system replacing the speedup in Amdahl's law [20]. Other factors can now be included in Amdahl-like laws through the divisible load component of the modified laws such as interconnection topology, load distribution policy and the relative difference in computation and communication intensity and speeds. The divisible load speedup expressions will be included for three fundamental load distribution protocols in the single level tree network. Heterogeneous and homogenous networks will be analyzed for each protocol. Moreover, the integrated speedup formulas will be used for the cases of local and cloud computing. The analysis will lead to the optimal processing method between local, cloud or a combination of both. Finally, the finish time and the optimal load distribution will be calculated for each case.

1.3 Organization

The rest of this thesis is organized as follows. An introduction to optimal signature searching using integer linear programming is given in the first section of chapter II. Then, the research problem and the proposed solution are described in the second section. The performance evaluation and experimental results are presented in the third section. A comparison between problem-based and solver-based linear programming is given in chapter III. Also, a heuristic algorithm solution is introduced. Then, the research problem and the proposed solution are described. The performance evaluation and experimental results are presented in the third section of the third chapter. In chapter IV, the divisible load speedup expressions are included for three fundamental load distribution protocols in the single level tree network [19]. Heterogeneous and homogenous networks are included for each protocol. Moreover, the integrated speedup formulas will be used for the cases of local and cloud computing. A combination of local and cloud computing will be tested and compared to the results of using only one computing method. Then, the finish time and the optimal load distribution will be calculated for each case. Finally, conclusions and future work will be presented in chapter V.

Chapter 2

Optimal Signature Searching Using Integer Linear Programming

2.1 Background

Wireless sensor networks are useful in diverse application areas. Monitoring applications, military applications, mobile commerce, smart offices and environmental science are examples of these application areas. Monitoring applications include medical health monitoring and structural health monitoring. Surveillance, target tracking, counter sniper, and battlefield monitoring are examples of military applications using wireless sensor networks. This developing technology reduces time and effort used in collecting data and monitoring events. Also, it is useful for long-term research databases. It has advantages as well as challenges and shortcomings. For example, it is convenient, flexible, and accurate. On the other hand, robustness, scalability, and security are examples of the challenges for this technology. Sensing means the act of collecting information about an object. It may be split into passive sensors that gather radiation that is emitted or reflected by the object. Passive sensors mostly use reflected sunlight as the source of measured radiation. Examples of passive sensors include film photography, infrared and radiometers. Active sensing is when the sensor emits a signal and detects its reflection by the object. RADAR and LiDAR are examples of active sensing.

A sensor's sensitivity indicates how much the input quantity affects the sensor's output. For instance, the temperature changes by 1 °C if the mercury in a thermometer moves 1 cm. The sensitivity in that example is 1 cm/°C. Optimal sensing means the best sensing quality with the least time and energy cost, which allow processing more data. Intuitively, optimal sensing leads to optimal signature searching. So, integer linear programming "algorithm" is used to reach the optimal sensing by selecting the best possible number of signals of a type or a combination of multiple types to ensure the best sensing quality possible considering all given constraints. A mathematical optimization problem in which the variables are restricted to be integers is called integer programming. Integer linear programming (ILP) means the objective function and the constraints are linear. The main reason for using integer variables when modeling problems as a linear program: The integer variables represent quantities that can only be integer. For example, it is not possible to build 4.7 cars or in this proposal send 2.5 signals. Integer linear programming can be used in many applications areas such as Production planning where a possible objective is to maximize the total production, without exceeding the available resources. Another example is scheduling such as vehicle scheduling in transportation networks. Also, another example is telecommunications networks where the goal of these problems is to design a network of links to install so that a predefined set of communication requirements are met, and the total cost of the network is minimal. Finally, cellular networks is another application area such as the task of frequency planning in 5G mobile networks which involves distributing available frequencies across the antennas so that users can be served and interference is minimized between the antennas.

2.2 Problem Formulation and Solution

The goal is to select the signals that provide the maximum quality "Q" in order to have optimal signature searching as shown in equation (1) [27]. We assumed that there are i types of signals. The question is if a sensor is sending out N signals altogether, how many n signals of each type i summing to N give the best solution? Our problem formulation process starts by creating three types of signals "i". Then, the quality "q_i", computation time "t_i" and energy "e_i" specifications per signal were provided for each type. Then, the overall computation time and energy constraints were set as shown in equations (2) and (3) below. The constraints were set so the total energy does not exceed the energy constraint "E_c" and the total computation time does not exceed the time constraint "T_c". The sum of the computation time per signals for each type multiplied by the number of signals from that type "n_i" for all three types gives the total computation time "T". Similarly, the sum of the energy per signals for each type multiplied by the number of signals from that type for all three types gives the total energy "E". Four different cases were created to study basic sets of constraints. Two cases will be discussed in this section and the other two will be used in the following section. Finally, the objective function of the algorithm is created to find the optimal number of signals of each type that would result in the maximum total quality. The maximum total quality can be calculated be multiplying the number of selected signals of each type by the quality per signal for that type then summing the results of all types as shown in equation (1).

$i = 1,2,3$

n_i , t_i , e_i , Q , n_i , q_i , E_c , E, T , T_c are positive integers 0,1,2,3….

$$\text{Max } Q = \sum_i n_i q_i \qquad (1)$$

$$\sum_i n_i t_i \leq T_c \qquad (2)$$

$$\sum_i n_i e_i \leq E_c \qquad (3)$$

For the first case, the quality per signal for the first, second and third type are two, five and ten and the computation time per signal are three, two and a half and two microseconds as shown in Figure 2.1 below. Figure 2.2 shows the relationship between the energy and quality per signal. The energy per signal values for the three types are one hundred, two hundred and three hundreds. Then, two constraints were set for the maximum energy and time cost. The constraints were set so the total energy does not exceed one thousand and the total computation time does not exceed twenty five microseconds. If only one type was chosen to for the solution the maximum quality would be thirty and the selected signals are three signals of the third type. The result shown in Figure 2.3 is one signal of the first type and three signals of the third type. The selected signals energy cost is one thousand and the total computation time equals nine microseconds. Finally, the maximum total quality for this case equals thirty two.

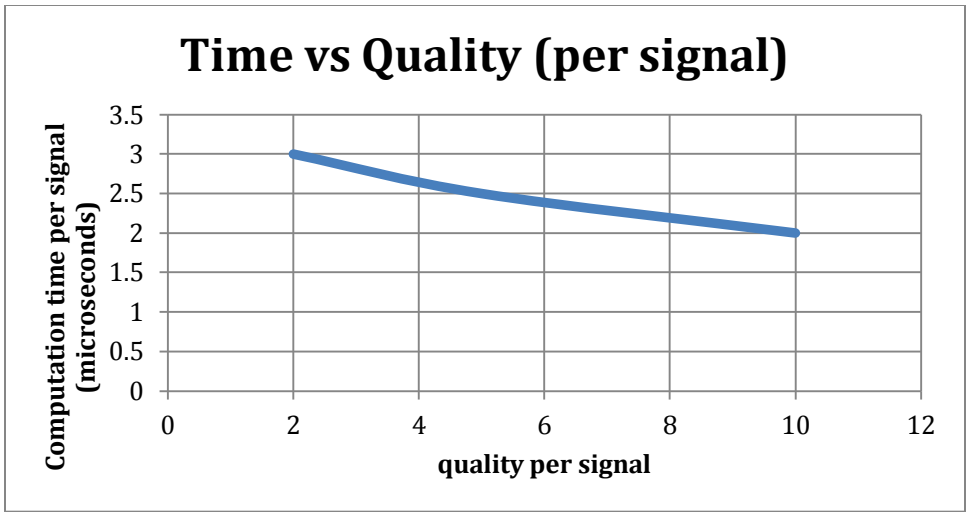

Figure 2.1: Computational time vs. quality per signal

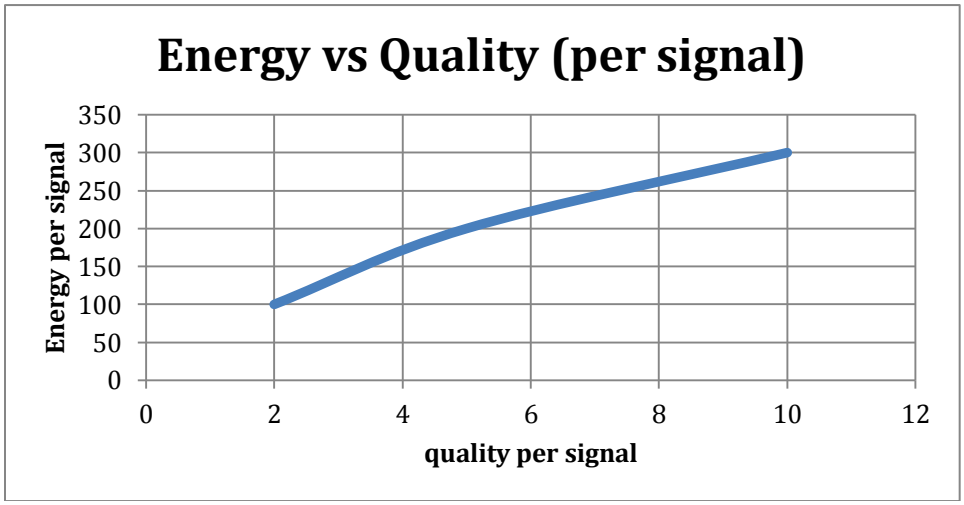

Figure 2.2: Energy vs. quality per signal

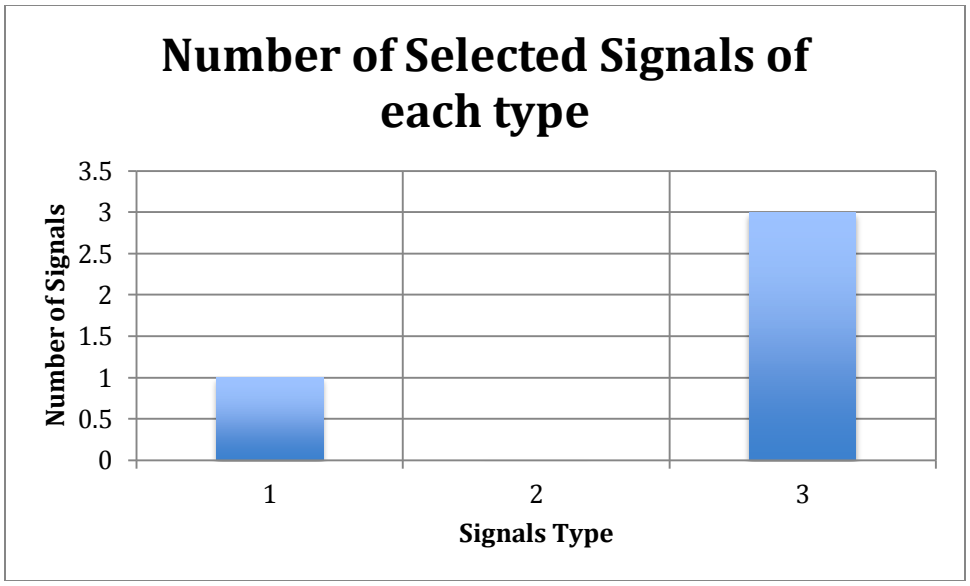

Figure 2.3: The number of selected signals per type

For the second case, the quality per signal for the first, second and third type are two, five and ten and the computation time per signal are three, two and a half and two microseconds as shown in Figure 2.4 below. Figure 2.5 shows the relationship between the energy and quality per signal, which is the difference between this case and the previous one. The energy per signal values for the three types are one hundred, one hundred and fifty and four hundred. Then, the two constraints are the same as the previous case for the maximum energy and time cost. The constraints were set so the total energy does not exceed one thousand and the total computation time does not exceed twenty five microseconds. If only one type was chosen to for the solution the maximum quality would be thirty and the selected signals are six signals of the second type. The result shown in Figure 2.6 is one signal of the first type and six signals of the second type. The selected signals energy cost is one thousand and the total computation time equals eighteen microseconds. Finally, the maximum total quality for this case equals thirty two. Clearly, modifications in the specifications and/or constraints result in different optimal number of signals

of each type. Also, the organization of the code controls the prioritization of the constraints. The performance will be evaluated in detail in the following section.

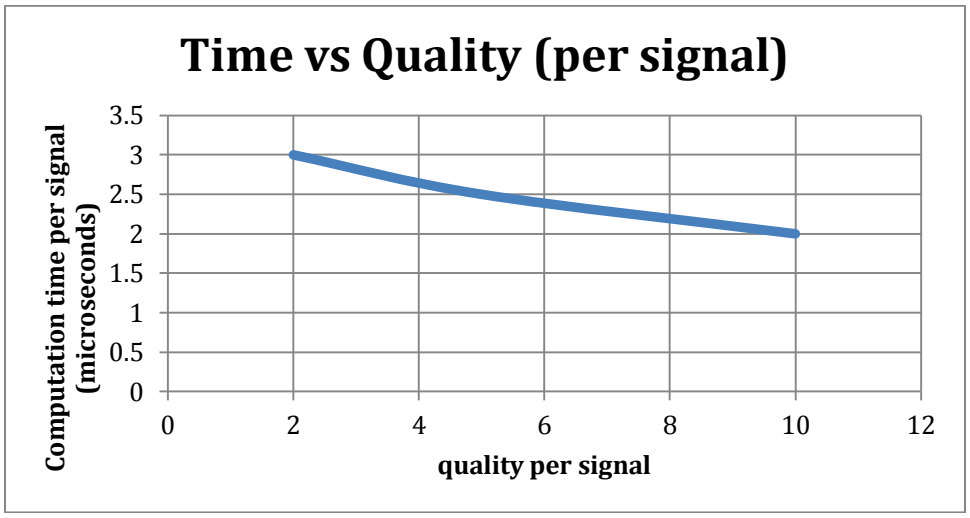

Figure 2.4: Computational time vs. quality per signal

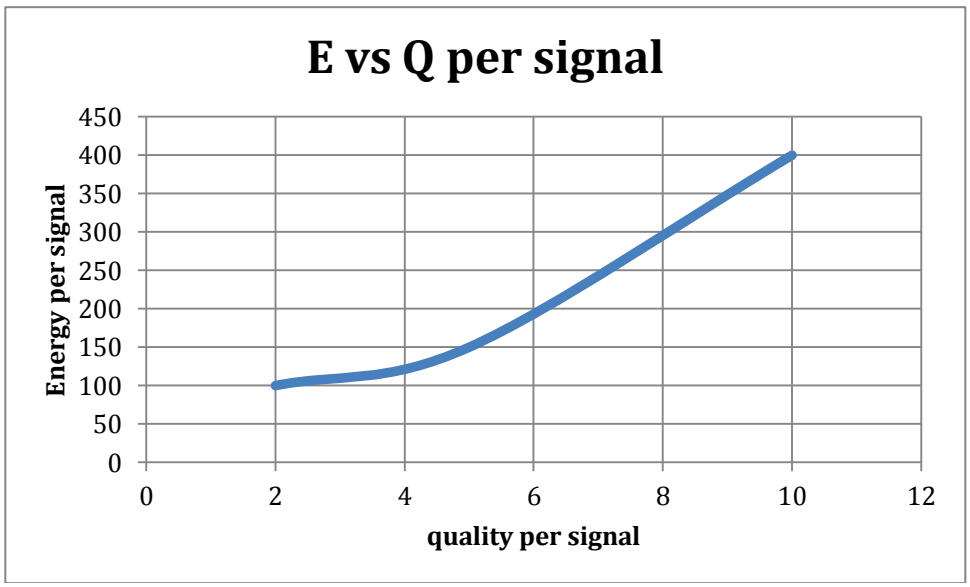

Figure 2.5: Energy vs. quality per signal

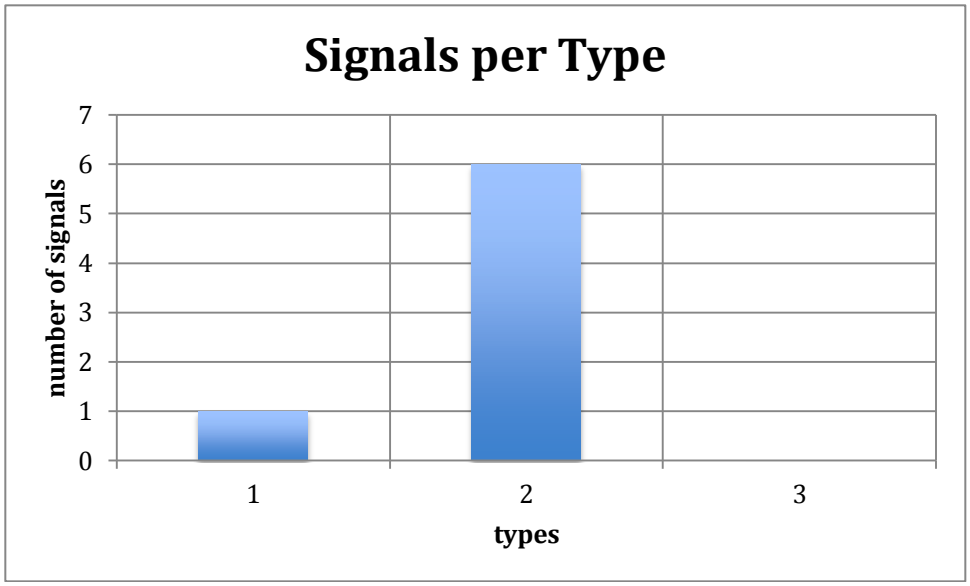

Figure 2.6: The number of selected signals per type

2.3 Performance Evaluation

The performance evaluation portion of this solution was done in three stages. The first stage is studying basic cases of constraints. Four different cases were created for testing. Two cases were discussed in the previous section and the other two will be used in this section. In every case the quality time and energy specifications per signal were provided for each type. Three types of signals were created for this problem. Then, the overall time and energy constraints were set. Finally, the objective of the algorithm is to find the optimal number of signals of each type that would result in the maximum total quality.

For the first case, the quality per signal for the first, second and third type are two, five and ten and the computation time per signal are three, two and a half and two microseconds as shown in Figure 2.7 below. Figure 2.8 shows the relationship between the energy and quality per signal. The energy per signal values for the three types are respectively one hundred, two hundred and fifty, and one thousand. Then, two constraints were set for the maximum energy and time cost. The constraints were set so the total energy does not exceed one thousand and the total computation time does not exceed twenty five microseconds. If only one type was chosen to for the solution the maximum quality would be twenty and the selected signals are four signals of the second type. The result shown in Figure 2.9 is five signals of the first type and two signals of the second type, which allows emitting more signals. The selected signals energy cost is one thousand and the total computation time equals twenty microseconds. Finally, the maximum total quality for this case equals twenty.

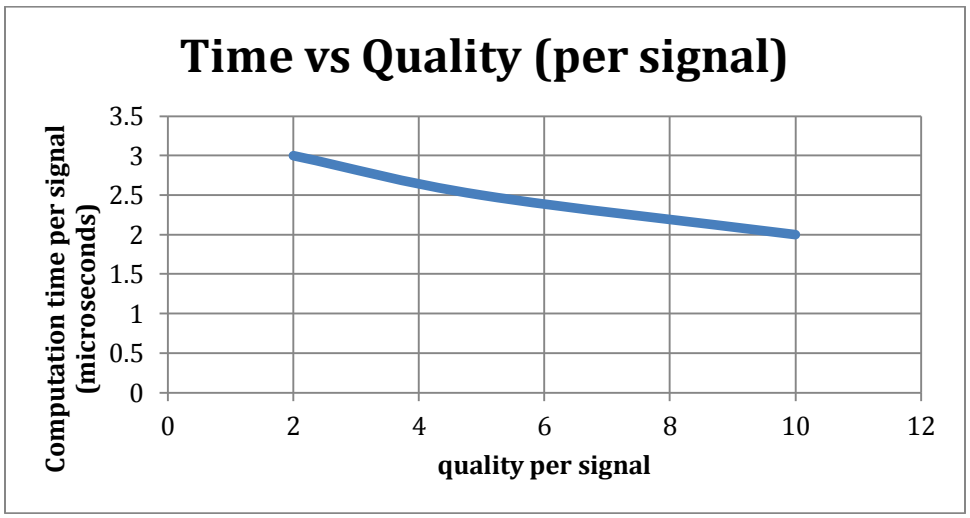

Figure 2.7: Computational time vs. quality per signal

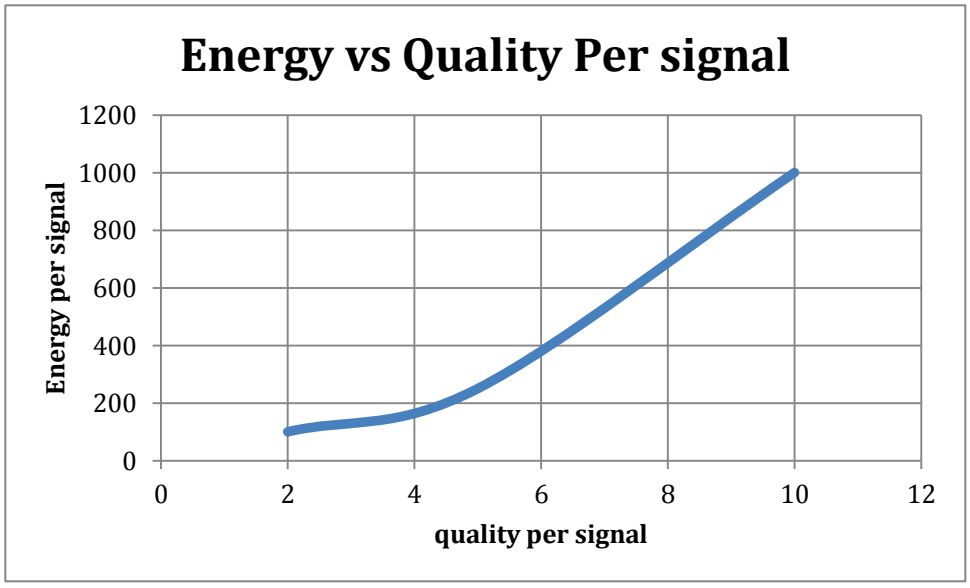

Figure 2.8: Energy vs. quality per signal

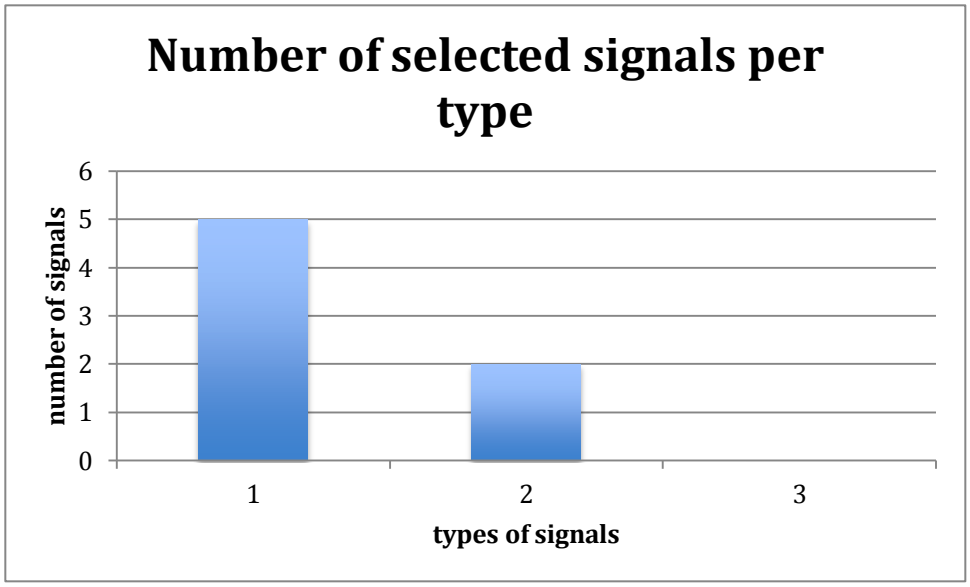

Figure 2.9: The number of selected signals per type

For the second case, the quality per signal for the first, second and third type are two, five and ten and the computation time per signal are three, two and a half and two microseconds as shown in Figure 2.10 below. Figure 2.11 shows the relationship between the energy and quality per signal. The energy for the three types are one hundred, two hundreds and five hundreds. Then, the two constraints are the same as the previous case for the maximum energy and time cost. The constraints were set so the total energy does not exceed one thousand and the total computation time does not exceed twenty five microseconds. If only one type was chosen to for the solution the maximum quality would be twenty five and the selected signals are five signals of the second type. The result shown in Figure 2.12 is five signals of the second type, which used only one type. The selected signals energy cost is one thousand and the total computation time equals twelve and a half microseconds. Finally, the maximum total quality for this case equals twenty five. It was concluded that the maximum total quality and the optimal number of signals of each type change whenever the specifications and/or constraints change. Also, the algorithm prioritizes the constraints following the order of the code.

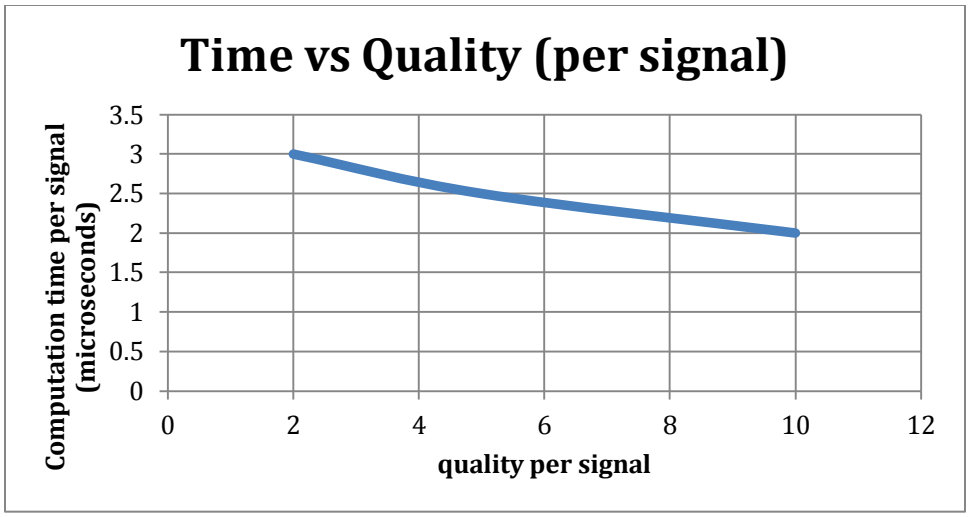

Figure 2.10: Computational time vs. quality per signal

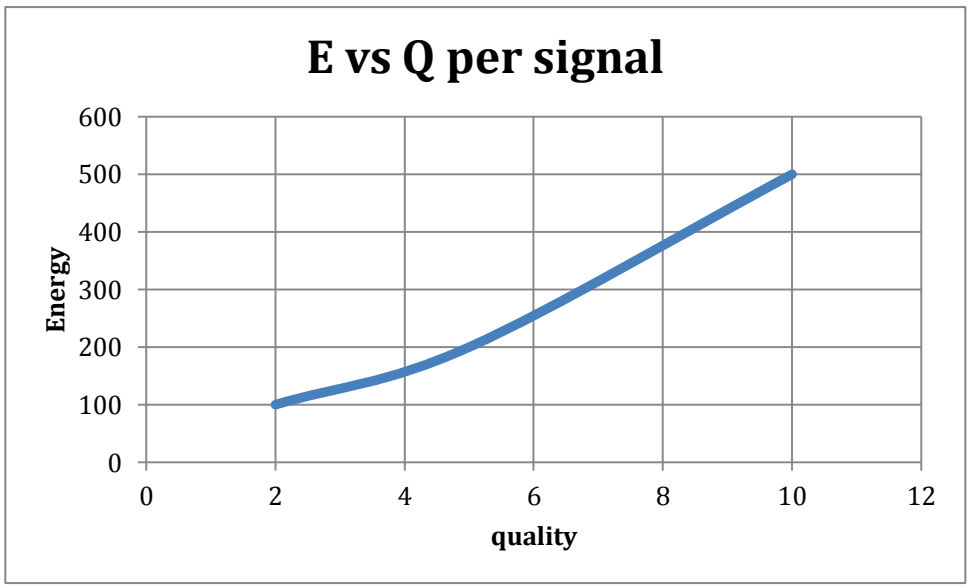

Figure 2.11: Energy vs. quality per signal

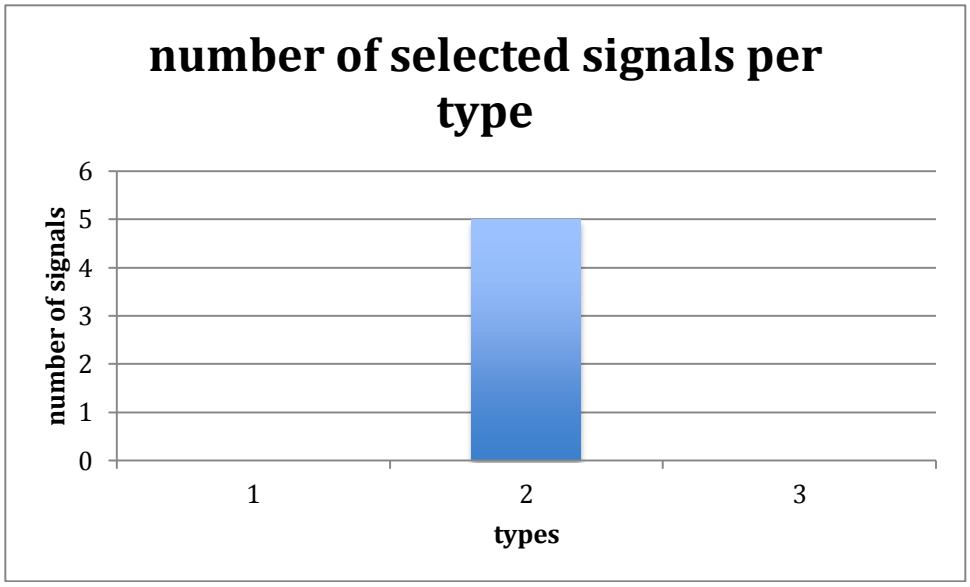

Figure 2.12: The number of selected signals per type

The second stage is studying complex cases of linear and quadratic relationship between energy/computation time and quality per signal. This stage consists of two parts. The first part is combinations of five cases to study the relationship between the energy and quality per signal with constant computation time specifications. Secondly, studying combinations of five cases to find the relationship between the computation time and quality per signal with constant energy specifications.

The first part is a combination of five cases with linear or quadratic relationship between energy and quality per signal when the computation time is constant. The first case is the linear relationship between the energy and quality when the computation time is constant. The quality per signal for the first, second and third type are two, five and ten and the computation time per signal are three, two and a half and two microseconds as shown in Table 2.1 below. Figure 2.13 shows the linear relationship between the energy and quality per signal. The energy is increased in each case. Then, two constraints were set for the maximum energy and time cost. The constraints were set so the total energy does not exceed one thousand and the total computation time does not

exceed twenty five microseconds. The selected signals energy cost and the total computation time vary for each case. For the first case, the energy cost is six hundred and the computation time equals twenty four microseconds. The energy cost for the second case is one thousand and the computation time equals twenty microseconds. The third case energy cost equals one thousand and the computation time is twenty four microseconds. In the fourth case, nine hundred and ninety is the energy cost and the computation time is eighteen and a half microseconds. Finally, the energy cost for the fifth case equals a thousand and the computation time is six and a half microseconds. The results in Table 2.2 and Figure 2.14 show that increasing the energy cost leads to a drop in the maximum total quality.

Signal type	Computation time	Quality/Energy	Case1 E=5Q	Case 2 E=10Q	Case 3 E=20Q	Case 4 E=30Q	Case 5 E=40Q
1	3	2	10	20	40	60	80
2	2.5	5	25	50	100	150	200
3	2	10	50	100	200	300	400

Table 2.1: combination of five cases with linear relationship between energy and quality

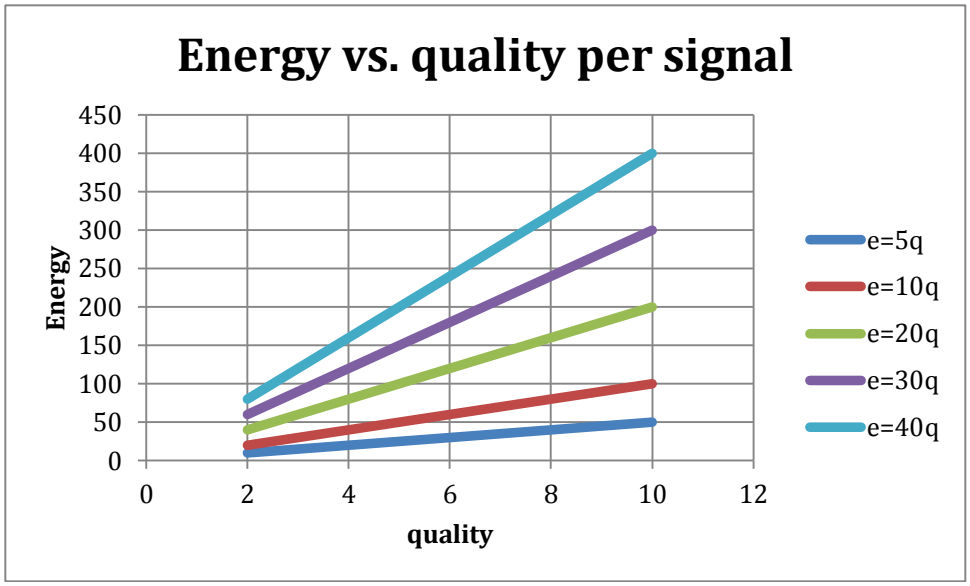

Figure 2.13: Combination of 5 cases with linear relationship between energy and quality

Type/case	1	2	3	4	5
1	0	0	5	4	0
2	0	0	0	1	1
3	12	10	4	2	2
Quality	120	100	50	33	25

Table 2.2: number of selected signals per type and the total quality for each case

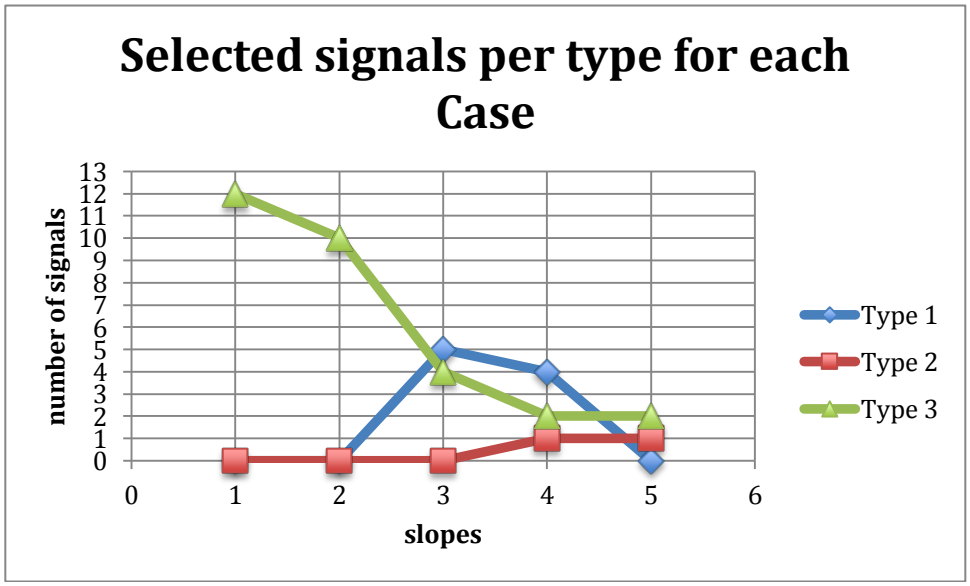

Figure 2.14: The number of selected signals per type for each case

The second case is the quadratic relationship between the energy and quality when the computation time is constant. The quality per signal for the first, second and third type are two, five and ten and the computation time per signal are three, two and a half and two microseconds as shown in Table 2.3 below. Figure 2.15 shows the quadratic relationship between the energy and quality per signal. The energy is increased in each case. Then, two constraints were set for the maximum energy and time cost. The constraints were set so the total energy does not exceed one thousand and the total computation time does not exceed twenty five microseconds. The selected signals energy cost and the total computation time vary for each case. For the first case, the energy cost is one thousand and the computation time equals twenty microseconds. The energy cost for the second case is nine hundred and fifty and the computation time equals twenty two and a half microseconds. The third case energy cost equals nine hundred and eighty and the computation time is twenty and a half microseconds. In the fourth case, nine hundred and sixty is the energy cost and the computation time is twenty four microseconds. Finally, the energy cost for the fifth case equals nine hundred and sixty and the computation time is sixteen microseconds. The results in Table 2.4 and Figure 2.16 show that increasing the energy cost leads to a drop in the maximum total quality.

Signal type	Computation time	Quality/Energy	Case1 E=5Q^2	Case 2 E=10Q^2	Case 3 E=20Q^2	Case 4 E=30Q^2	Case 5 E=40Q^2
1	3	2	20	40	80	120	160
2	2.5	5	125	250	500	750	2000
3	2	10	500	1000	2000	3000	4000

Table 2.3: combination of five cases with quadratic relationship between energy and quality

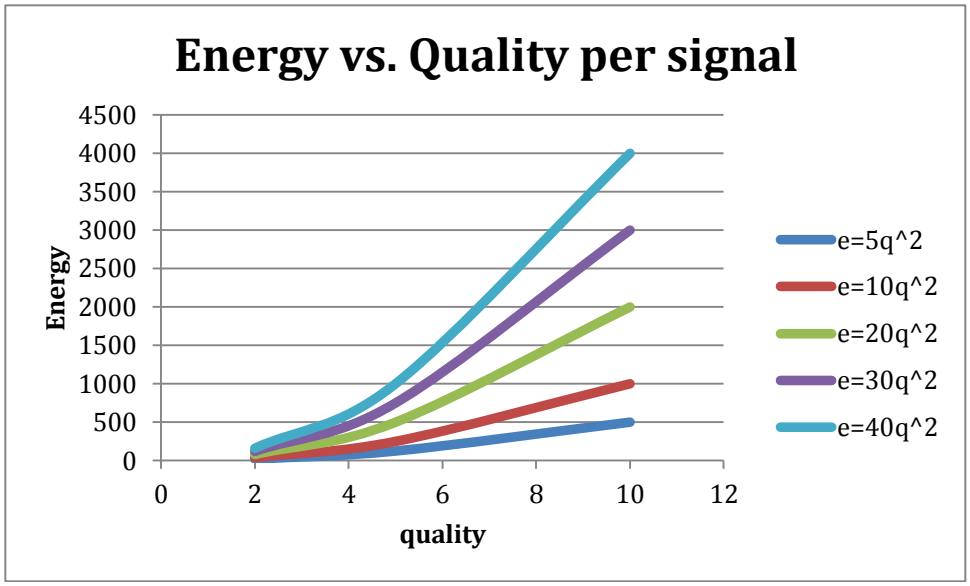

Figure 2.15: Combination of 5 cases with quadratic relationship between energy and quality per signal

Type/case	e=5q^2	e=10q^2	e=20q^2	e=30q^2	e=40q^2
1	0	5	6	8	6
2	8	3	1	0	0
3	0	0	0	0	0
Q	40	25	17	16	12

Table 2.4: number of selected signals per type and the total quality for each case

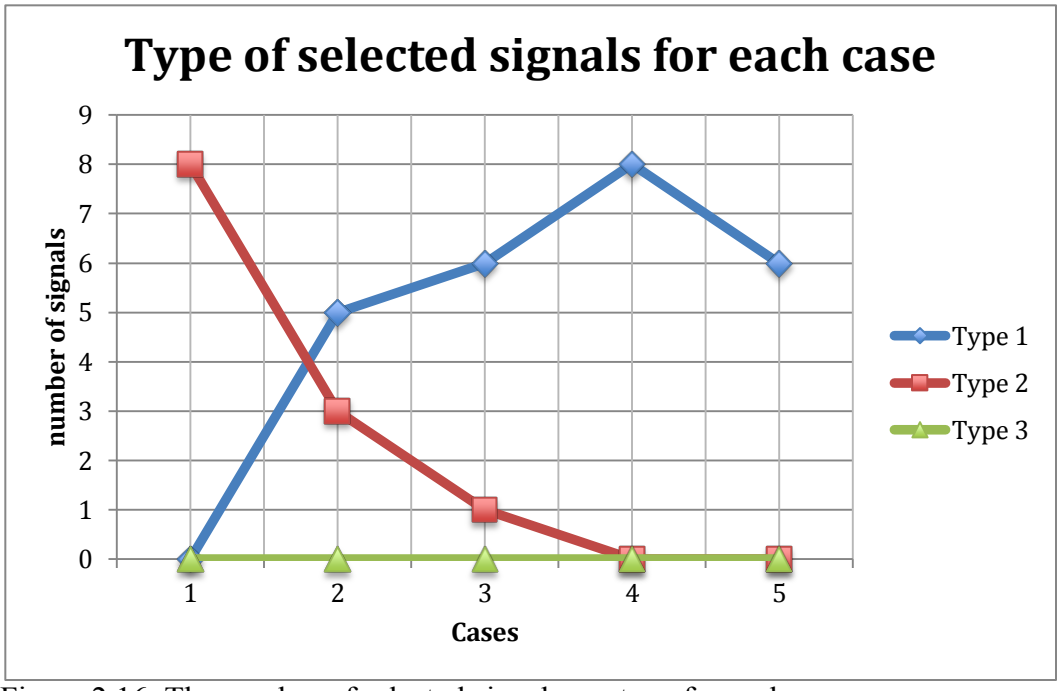

Figure 2.16: The number of selected signals per type for each case

The second part is a combination of five cases with linear or quadratic relationship between computation time and quality per signal when the energy is constant. The first case is the linear relationship between computation time and quality when the energy is constant. The quality per signal for the first, second and third type are two, five and ten and the energy per signal are one hundred, two hundred and three hundred as shown in Table 2.5 below. Figure 2.17 shows the linear relationship between computation time and quality per signal. The computation time is increased in each case. Then, two constraints were set for the maximum energy and time cost. The constraints were set so the total energy does not exceed one thousand and the total computation time does not exceed twenty five microseconds. The selected signals energy cost and the total computation time vary for each case. For the first case, the energy cost is one thousand and the computation time equals twenty microseconds. The energy cost for the second case is nine hundred and the computation time equals fifteen microseconds. The third case energy cost equals six hundred and

the computation time is twenty microseconds. In the fourth case, three hundred is the energy cost and the computation time is fifteen microseconds. Finally, the energy cost for the fifth case equals one hundred and the computation time is twenty microseconds. The results in Table 2.6 and Figure 2.16 show that increasing the computation time results in a drop in the maximum total quality.

Signal type	Energy	Quality/ time	Case1 t=25/Q	Case 2 t=50/Q	Case 3 t=100/Q	Case 4 t=150/Q	Case 5 t=200/Q
1	100	2	12.5	25	50	75	100
2	200	5	5	10	20	30	40
3	300	10	2.5	5	10	15	20

Table 2.5: Combination of five cases with linear relationship between Time and Quality

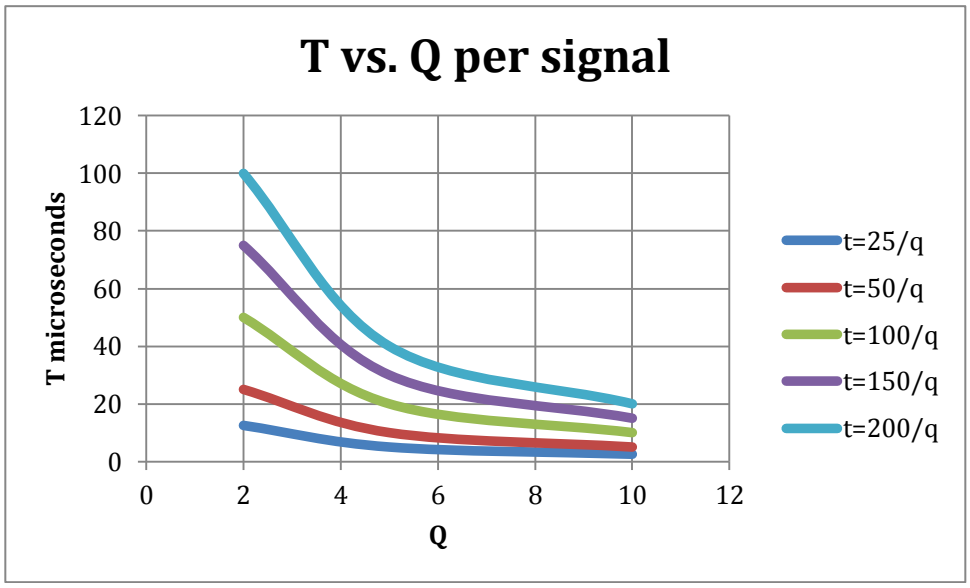

Figure 2.17: Combination of 5 cases with linear relationship between time and quality per signal

Type/case	1	2	3	4	5
1	1	0	0	0	0
2	0	0	0	0	0
3	3	3	2	1	1
Q	32	30	20	10	10

Table 2.6: number of selected signals per type and the total quality for each case

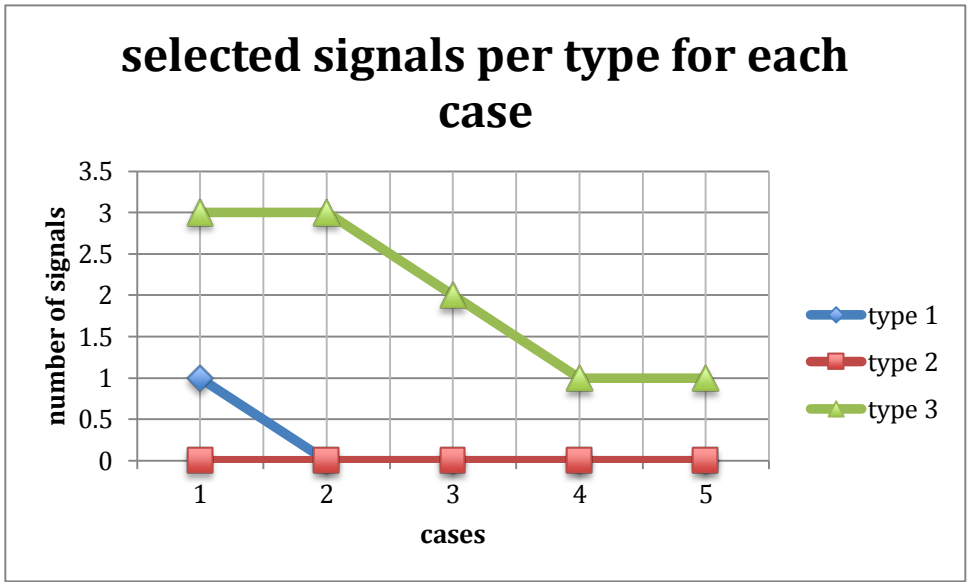

Figure 2.18: The number of selected signals per type for each case

The second case is the quadratic relationship between computation time and quality when the energy is constant. The quality per signal for the first, second and third type are two, five and ten and the energy per signal are one hundred, two hundred and three hundred as shown in Table 2.7 below. Figure 2.19 shows the quadratic relationship between computation time and quality per signal. The computation time is increased in each case. Then, two constraints were set for the maximum energy and time cost. The constraints were set so the total energy does not exceed one thousand and the total computation time does not exceed twenty five microseconds. The selected signals energy cost and the total computation time vary for each case. For the first case, the energy cost is one thousand and the computation time equals seven microseconds. The energy cost for the

second case is one thousand and the computation time equals fourteen microseconds. The third case energy cost equals nine hundred and the computation time is three microseconds. In the fourth case, nine hundred is the energy cost and the computation time is four and a half microseconds. Finally, the energy cost for the fifth case equals nine hundred and the computation time is six microseconds. The results in Table 2.8 and Figure 2.20 show that increasing the computation time results in a drop in the maximum total quality.

Signal type	Energy	Quality/ time	Case1 t=25/Q^2	Case 2 t=50/Q^2	Case 3 t=100/Q^2	Case 4 t=150/Q^2	Case 5 t=200/Q^2
1	100	2	6.25	12.5	25	37.5	50
2	200	5	1	2	4	6	8
3	300	10	0.25	0.5	1	1.5	2

Table 2.7: Combination of five cases with quadratic relationship between Time and Quality

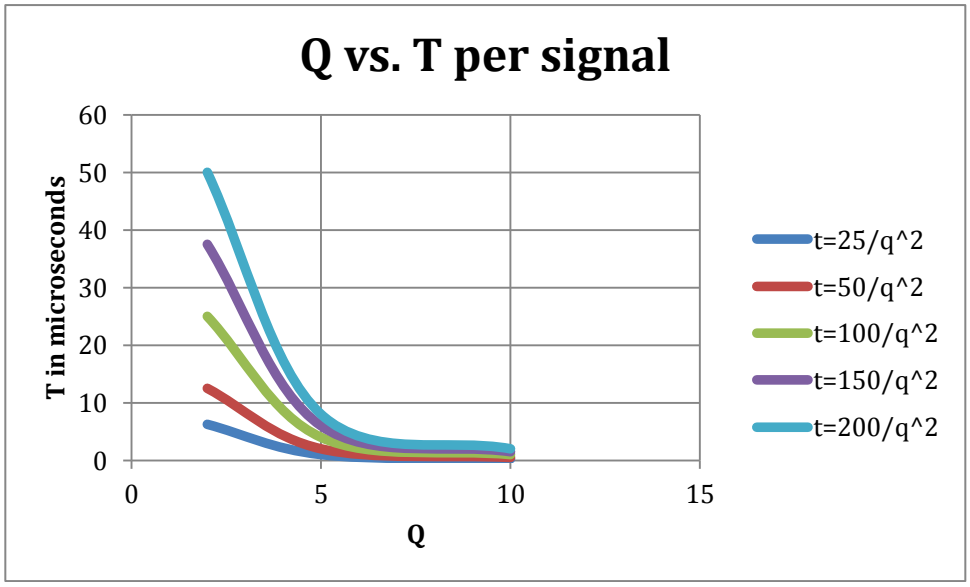

Figure 2.19: Combination of 5 cases with quadratic relationship between time and quality per signal

Type/case	1	2	3	4	5
1	1	1	0	0	0
2	0	0	0	0	0
3	3	3	3	3	3
Q	32	32	30	30	30

Table 2.8: number of selected signals per type and the total quality for each case

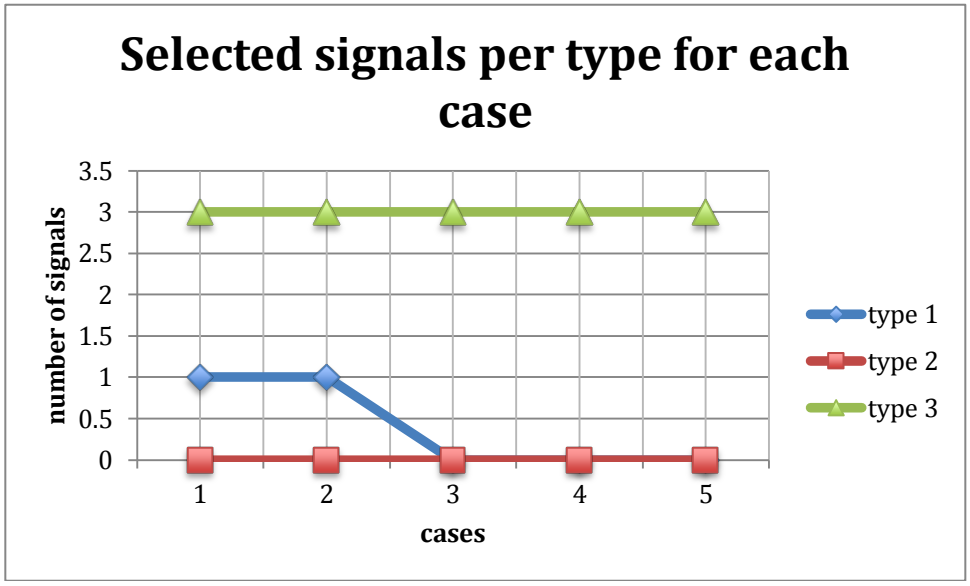

Figure 2.20: The number of selected signals per type for each case

The third stage is studying the effect of restricting the number of signals per type. In this part the energy and the computation time per signal are constant for each type. The quality per signal for the first, second and third type are two, five and ten and the energy per signal are one hundred, two hundred and three hundred as shown in Figure 2.21. Figure 2.22 shows that the computation time per signal are five, two and one microseconds. Then, two constraints were set for the maximum energy and time cost. The constraints were set so the total energy does not exceed ten thousand and the total computation time does not exceed one thousand microseconds. Figure 2.23 shows the number of selected signals per type before restricting the number of signals per

type. The total Quality equals three hundred and thirty two and the selected signals are one signal of the first type and thirty three signals of the third type.

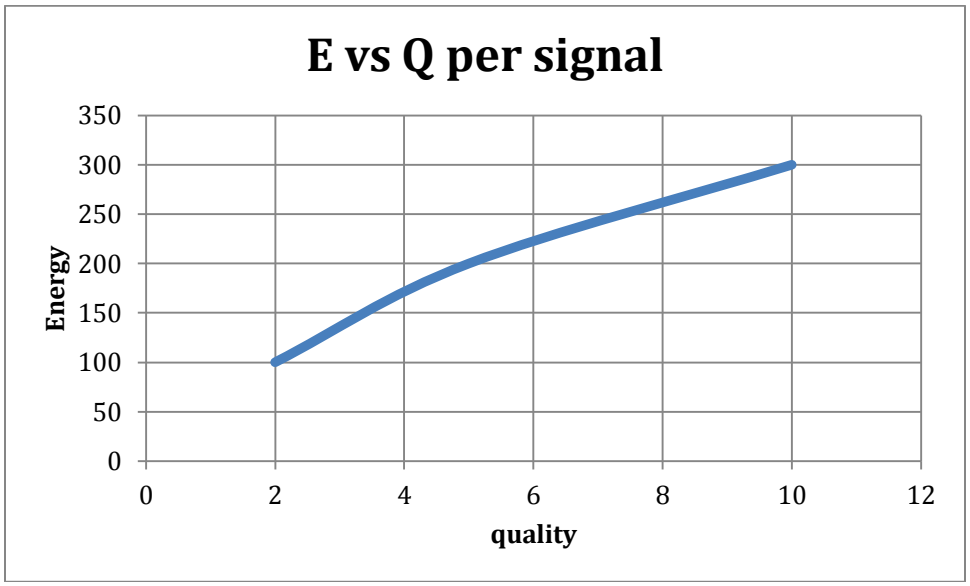

Figure 2.21: Energy vs. quality per signal

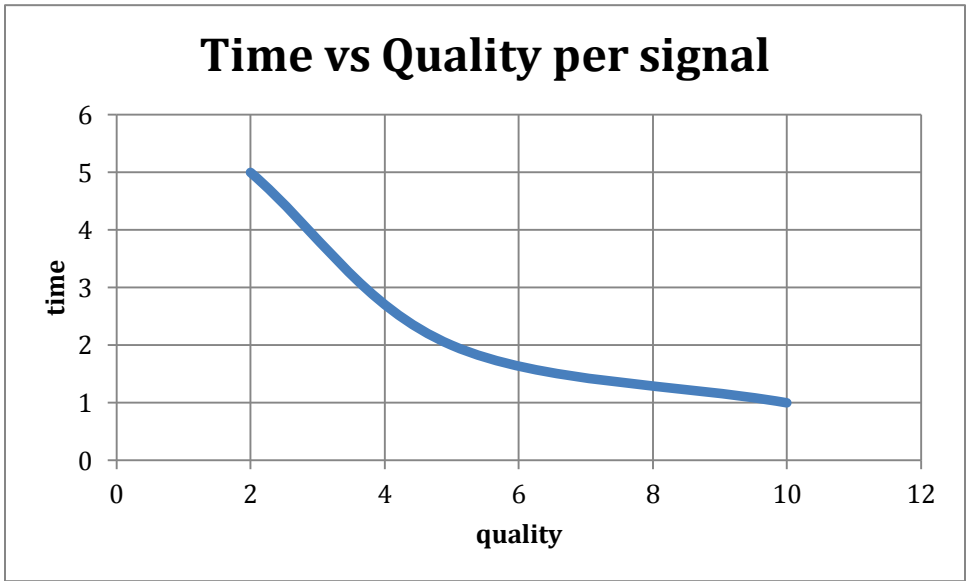

Figure 2.22: Computation time vs. quality per signal

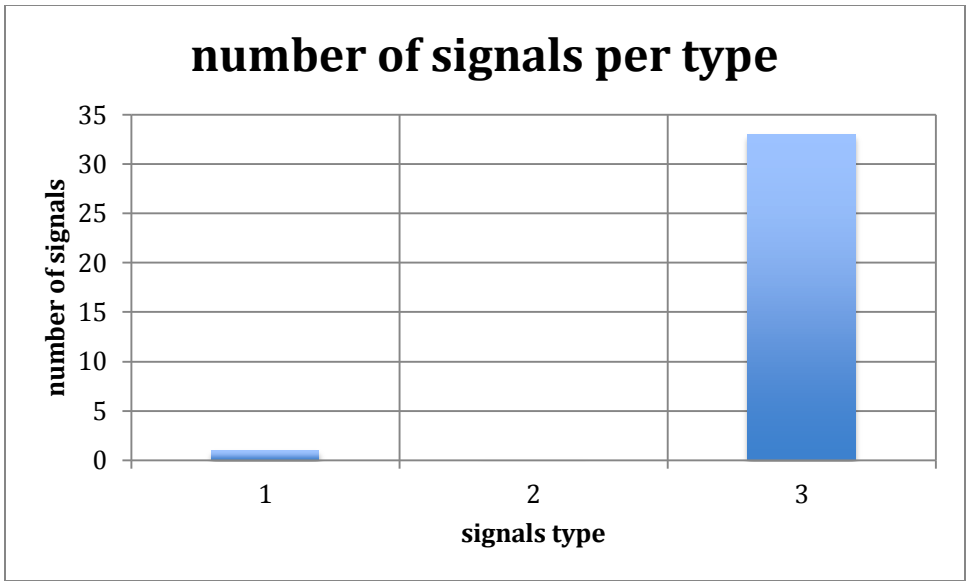

Figure 2.23: The number of selected signals per type

The next step is to restrict the number of signals per type as shown in equation (4). Figure 2.24 shows the number of selected signals per type after adding the constraint "S_c", thirty signals per type. The total quality equals three hundred and twenty five and the selected signals are five signals of the second type and thirty signals of the third type as shown in Figure 2.24. Then, the number of signals per type constraint is modified to twenty signals per type. For this case, the total quality equals three hundred and the selected signals are twenty signals of the second type and twenty signals of the third type as shown Figure 2.25. The selected signals per type vary for each case. To verify the changes in results, a third case is done where the number of signals per type constraint is set to ten signals per type. The total quality equals one hundred and seventy and the selected signals are ten signals of each type as shown in Figure 2.26. In conclusion, restricting the number of signals per type affect the maximum total quality.

$$N_i \leq S_c \qquad (4)$$

S_c is a positive integer 0,1,2,3….

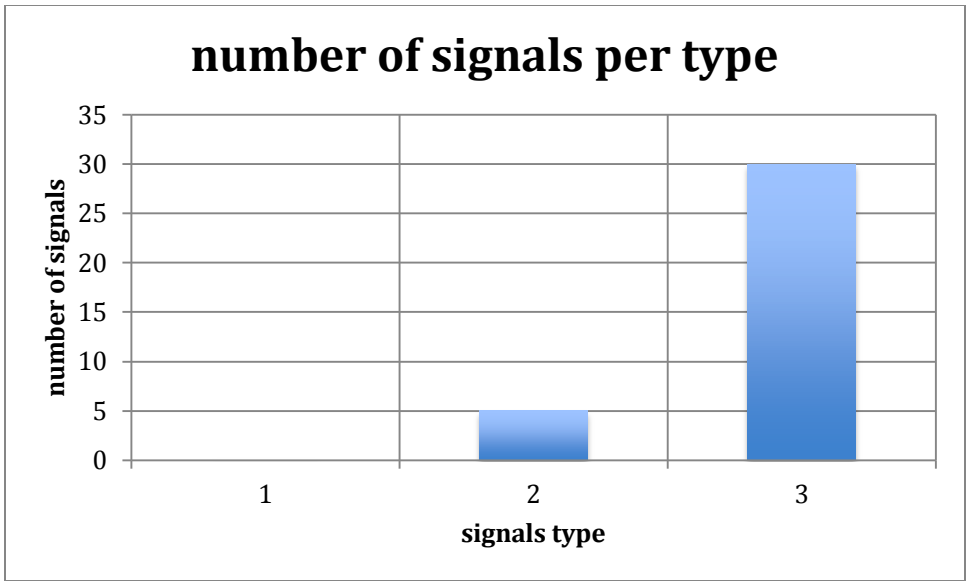

Figure 2.24: The number of selected signals per type

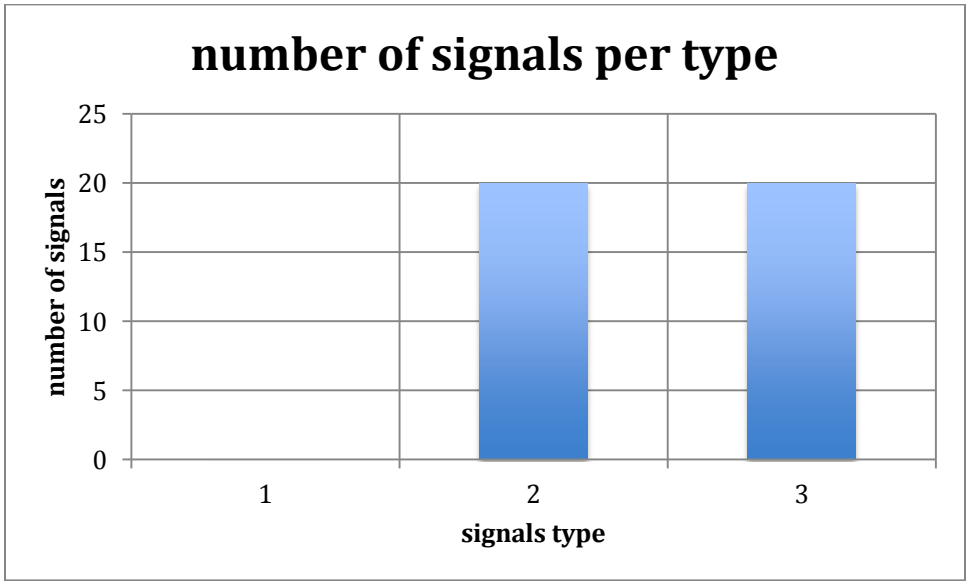

Figure 2.25: The number of selected signals per type

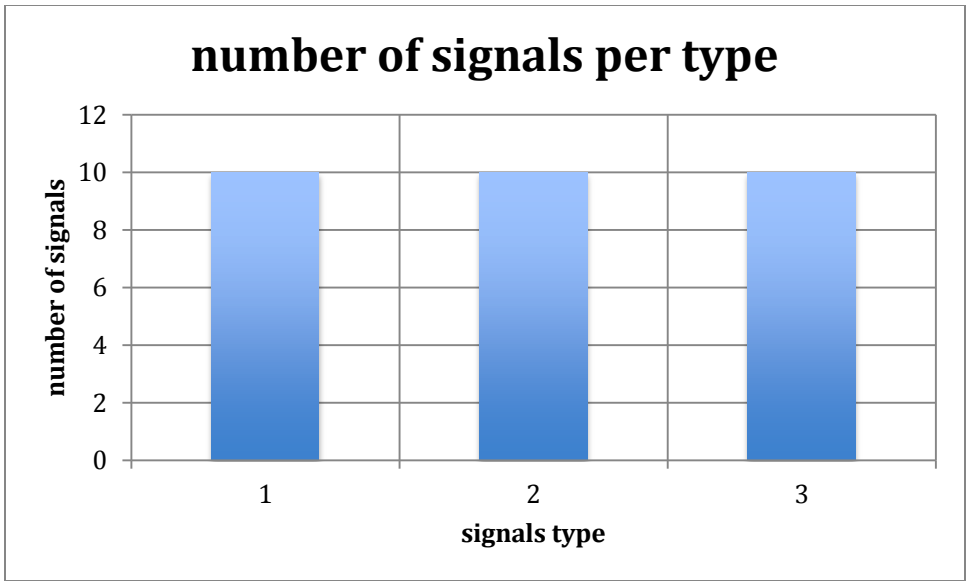

Figure 2.26: The number of selected signals per type

Chapter 3

Optimization of Integer Linear Programming Solution

3.1 Background

Wireless sensor networks are useful in diverse application areas. This developing technology provides efficient data collecting and monitoring. It is used in applications such as monitoring applications, military applications, mobile commerce, smart offices, long-term research databases and environmental science [4,5]. The focus of this research is signature searching using intelligent sensor applications. An active sensor emits signals for detection and information acquisition purposes. An intelligent sensor means the sensor can collect data and process it [6,7,8]. Optimal sensing means the best sensing quality with the least time and energy cost, which allow processing more data. Intuitively, optimal sensing leads to optimal signature searching.

Integer programming program is a mathematical optimization solution with integer variables. Integer linear programming (ILP) means the objective function and the constraints are linear [6,7,8]. The main reason for using integer variables is to represent decision quantities that can only be integer. So, an integer linear programming "algorithm" is used to perform the optimal sensing by selecting the best possible number of signals of a type or a combination of multiple

types to ensure the best sensing quality possible considering all given constraints. In the previous chapter, the solution was done using problem based linear programming. In this chapter, the performance evaluation is done in three stages and the results show improvements using solver-based linear programming and a heuristic algorithm.

The objective of this chapter is to optimize the proposed solution in the previous chapter using linear programming and heuristic algorithm. Instead of using problem based linear programming such as in the previous chapter, a solver-based solution is used to optimize the solution. Solver based linear programming will be introduced in the following section and all the results will be represented. Then, a solution based on a heuristic algorithm is implemented which shows improvements in the performance.

Heuristic algorithms can provide faster and more efficient method to solve a problem. Heuristic algorithms are useful to find approximate solutions when it is sufficient and exact solutions are computationally expensive. Heuristic algorithms are commonly employed to solve the Knapsack Problem. In the knapsack problem, heuristics are used to find the maximum value by grouping a given set of items while being under a certain limit and its known as the Greedy Approximation Algorithm. It starts by sorting the items based on their value per unit. Then, it adds the items with the highest value per unit as long as there is still space remaining [18]. In this chapter, a heuristic algorithm will be used to find the maximum quality under a specific set of constraints.

The rest of this chapter is organized as follows. A comparison between problem based and solver based linear programming is given in the following section. After that, a heuristic algorithm solution is introduced. Then, the research problem and the proposed solution are described. The

performance evaluation and experimental results are presented in the third section. Finally, conclusions and future work are presented at the end of this chapter.

3.2 Solution

3.2.1 Comparison between problem based and solver based linear programming

Problem-based and solver-based are two approaches to solving optimization problems. The appropriate approach must be selected before solving a problem. The differences between the two approaches are covered in the following part.

First of all, Problem-Based Optimization Setup is easier to create and debug. The objective and constraints are represented symbolically. It requires translation from problem form to matrix form, resulting in a longer solution time. It does not allow direct inclusion of the gradient or Hessian.

The second approach is solver-based optimization. The problem setup is harder to create and debug. The objective and constraints are represented as functions or matrices. It does not require translation from problem form to matrix form, resulting in a shorter solution time. It allows direct inclusion of gradient or Hessian. Also, it allows use of a Hessian multiply function or Jacobian multiply function to save memory in large problems [17].

The two approaches produce solution of the same quality. Theoretically, the solver-based solution can improve the performance. This is because the objective and constraints are represented as functions or matrices in solver-based solution. That representation eliminates the translation from problem form to matrix form which allow a shorter solution time.

3.2.2 Heuristic algorithms solution

Solving a given problem using heuristic could have a trade-off such as optimality, completeness and accuracy. That leads to question if the heuristic solution is good enough. When multiple solutions exist for a given problem, the following questions can be used to evaluate the solution found by heuristic algorithm; does the heuristic give the best solution? Can the heuristic find all possible solutions? Is this the fastest method for solving this type of problem? A heuristic algorithm was implemented and compared to the previous approaches to answer these questions. The first step is to sorts the signal types. To determine the order of signal types the following expression was used to calculate the quality of a certain signal type

$$\text{Type quality} = qi/(ei*ti) \qquad (1)$$

After the signal types are sorted, the algorithm will calculate the best number of signals of each type to find the maximum quality achieved under a specific set of constraints.

3.2.3 Problem formulation

The goal is to select the signals that provide the maximum quality "Q" in order to have optimal signature searching as shown in equation (2) [27]. We assumed that there are i types of signals. The question is if a sensor is sending out N signals altogether, how many n_i signals of each type i summing to N give the best solution? Our problem formulation process starts by creating four types of signals "i". Then, the quality "q_i", computation time "t_i" and energy "e_i" specifications per signal were provided for each type. Then, the overall computation time and energy constraints were set as shown in equations (3) and (4) below. The constraints were set so the total energy does not exceed the energy constraint "E_c" and the total computation time does not exceed the time constraint "T_c". The sum of the computation time per signals for each type multiplied by the number of signals from that type "n_i" for all three types gives the total computation time "T". Similarly, the sum of the energy per signals for each type multiplied by the number of signals from that type for all three types gives the total energy "E". The objective function of the algorithm is created to find the optimal number of signals of each type that would result in the maximum total quality. The maximum total quality can be calculated be multiplying the number of selected signals of each type by the quality per signal for that type then summing the results of all types as shown in equation (2). Finally, the program run time will be used to compare problem-based and solver-based solutions.

$i = 1,2,3$

$n_i, t_i, e_i, Q, n_i, q_i, E_c, E, T, T_c$ are positive integers 0,1,2,3….

$$\text{Max } Q = \sum_i n_i q_i \qquad (2)$$

$$\sum_i n_i t_i \leq T_c \quad (3)$$

$$\sum_i n_i e_i \leq E_c \quad (4)$$

Two different cases were created to study the run time for each program. The first case will be done utilizing four signal types and the other case will be done utilizing seven signal types. Problem-based, solver-based and heuristic algorithm solutions will be used to find the maximum quality utilizing four different signal types. The quality per signal, the energy per signal and the computation time per signal for each signal type are shown in Table 3.1 below.

For the first case, the quality per signal for each type are five, two, ten and seven. The energy per signal for the utilized four types are two hundred, one hundred, three hundred and two hundred and fifty. Figure 3.1 shows the relationship between the energy and quality per signal. The computation time per signal are two and five tenths, three, two, and two and three tenths microseconds. The relationship between the computation time and the quality per signal is shown in Figure 3.2 below.

For the second case, each solution will be done using seven signal types. Three more types of signals were created for this problem. All the parameters for signals types 5, 6 and 7 are provided in Table 3.1. The quality per signal for each type are twelve, six and one. ten and seven. The energy per signal for the new three types are three hundred and fifty, two hundred and twenty five and fifty. Figure 3.1 shows the relationship between the energy and quality per signal. The computation time per signal are one and a half, two and four tenths and three and a half microseconds. The relationship between the computation time and the quality per signal is shown in Figure 2 below.

Signal Type	Quality per signal	Energy per signal	Computation time per signal
1	5	200	2.5
2	2	100	3
3	10	300	2
4	7	250	2.3
5	12	350	1.5
6	6	225	2.4
7	1	50	3.5

Table 3.1. Values of parameters used for testing

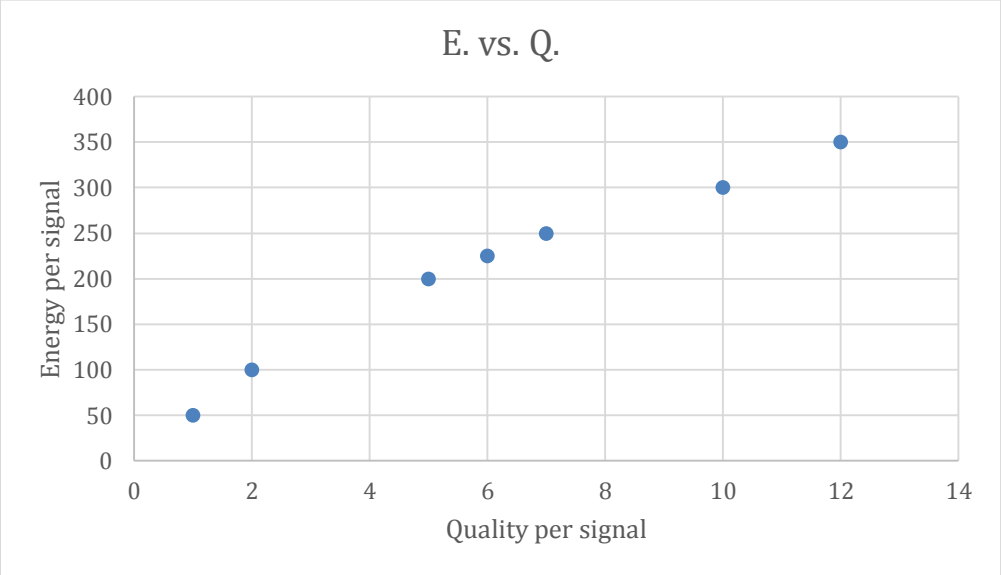

Figure 3.1. The relationship between the energy and quality per signal

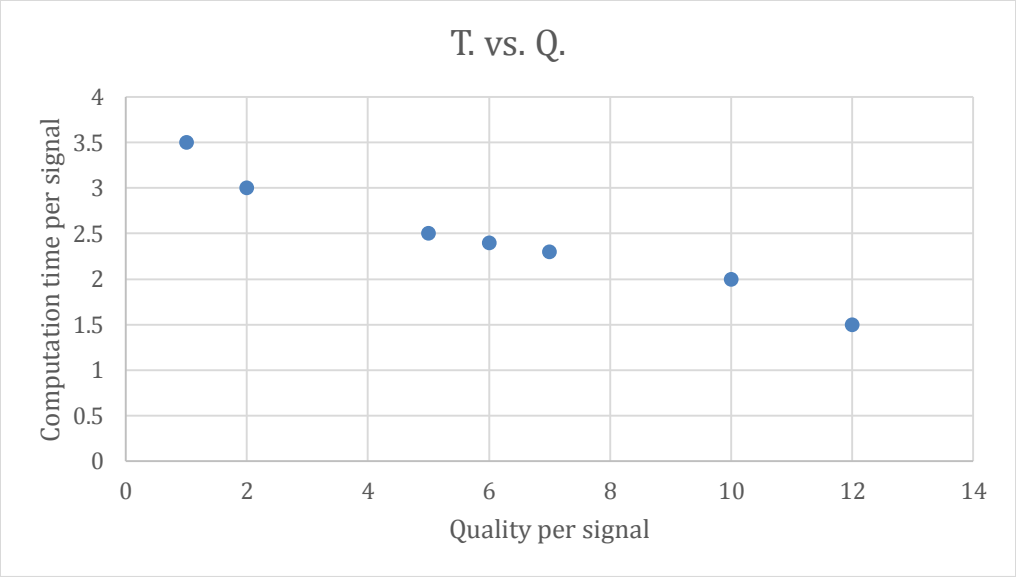

Figure 3.2. The relationship between the computation time and quality per signal

Then, three constraints were set for the maximum energy, time cost and the number of signals per type. The constraints were set so the total energy does not exceed ten thousand and the total computation time does not exceed two hundred and fifty microseconds. The programs were set so the number of signals per type doesn't exceed twenty. In other words, the program cannot select more than twenty signals from a certain type to find the maximum quality.

3.2.4 Testing methodology

The performance evaluation portion of this solution was done in three stages. The first stage is studying basic cases of parameters and constraints to find the maximum quality using the problem based, solver based and heuristic algorithm. In the first stage, only four different types of signals were created for testing. In every case the quality, time and energy specifications per signal were provided for each type. Then, the constraints were set. Finally, the objective of the algorithm is to find the optimal number of signals of each type that would result in the maximum total quality.

Once all three solutions are running and providing the correct solution which is the same for all of them. The second stage is to test and compare the running time for all three solutions. The programs were tested twenty times and the solution time for all the trials can be found in Table 3.2 below. All the results will be analyzed in the following section.

The third and final stage is to test the running time for all three proposed solution and compare them using more signal types. Three more types of signals were created for this problem. All the parameters for signals types 5, 6 and 7 are provided in Table 3.1. The programs were tested to find the maximum quality utilizing seven types of signals. Once they were running correctly, they were tested for the running time. The programs were again tested twenty times and the solution time for all the trials can be found in Table 3.2 below. All the results will be analyzed in the following section.

3.3 Results and Analysis

Linear programming was used in two approaches problem based and solver based. The problem-based solution was introduced in the previous chapter. Solver-based solution was implemented and tested for the case with four signal types. The test was done twenty times and the results show improvement in the performance. As shown in Table 3.2, the performance was enhanced from 0.64 to 0.238 seconds. Then, the heuristic algorithm was implemented and tested. The solution time was improved to 0.178 seconds as shown in Table 3.2.

To test the solutions on a larger problem, they were tested again using seven signal types. The running time for the problem-based, solver-based and the heuristic algorithm were respectively 0.667, 0.271 and 0.183 seconds. The running time for each solution was more than the result for the first case. This is because it takes more time to solve a larger computational problem. A comparison the results of twenty different trials for each solution utilizing four or seven signals types are represented in Table 3.2 and Figure 3.3. Finally, the average of all the solution time results is shown in Table 3.2 and Figure 3.4.

Trial	L.P. 4 types Problem based	L.P. 4 types Solver based	H.A. 4 types	L.P. 7 types Problem based	L.P. 7 types Solver based	H.A. 7 types
1	0.548	0.111	0.100	0.668	0.197	0.193
2	0.642	0.165	0.204	0.649	0.279	0.189
3	0.601	0.130	0.156	0.564	0.438	0.178
4	0.685	0.265	0.184	0.824	0.289	0.171
5	0.617	0.232	0.189	0.624	0.236	0.182
6	0.600	0.251	0.192	0.676	0.267	0.178
7	0.675	0.243	0.170	0.713	0.296	0.172
8	0.632	0.249	0.168	0.635	0.276	0.186
9	0.649	0.229	0.189	0.648	0.265	0.178
10	0.651	0.243	0.169	0.619	0.293	0.224
11	0.659	0.254	0.189	0.667	0.283	0.186
12	0.621	0.265	0.182	0.667	0.339	0.175
13	0.661	0.270	0.188	0.658	0.259	0.192
14	0.646	0.256	0.179	0.647	0.228	0.179
15	0.629	0.264	0.183	0.661	0.258	0.182
16	0.629	0.300	0.173	0.678	0.235	0.184
17	0.658	0.261	0.192	0.762	0.253	0.186
18	0.695	0.261	0.204	0.690	0.238	0.177
19	0.661	0.257	0.175	0.644	0.242	0.165
20	0.645	0.252	0.177	0.649	0.245	0.175
Avg.	0.640	0.238	0.178	0.667	0.271	0.183

Table 3.2. Comparison between I.L.P. solutions and heuristic algorithm solution time results

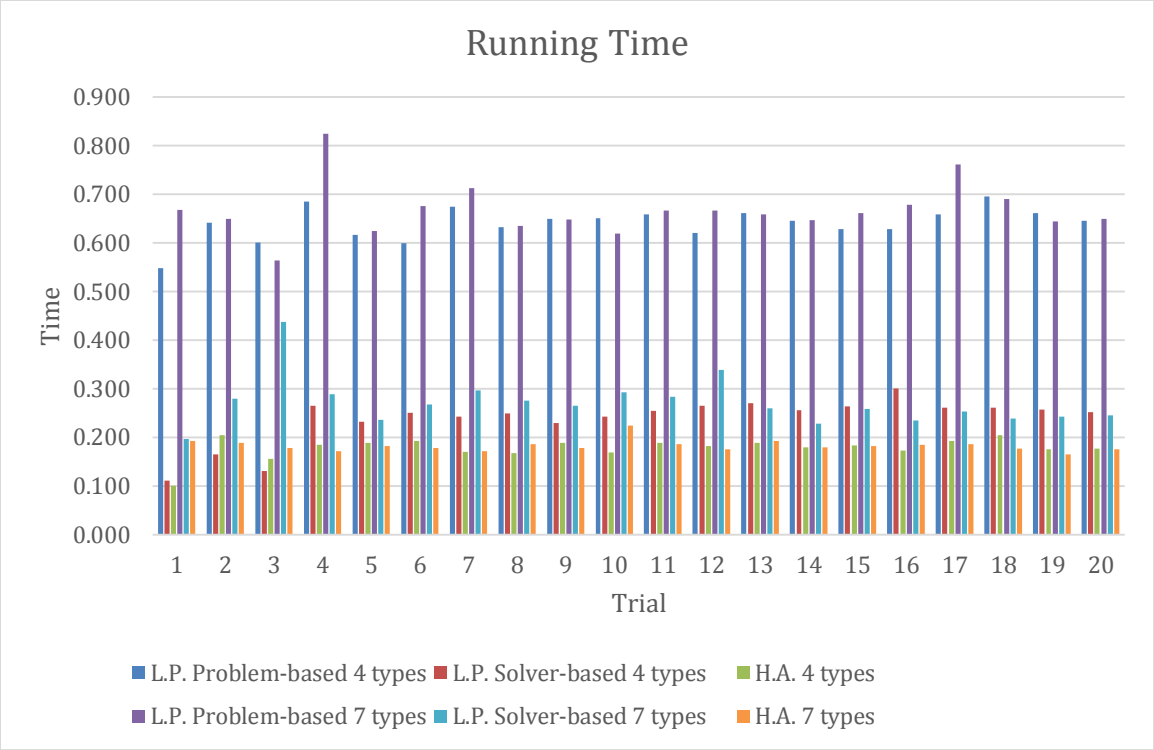

Figure 3.3. The solution time results

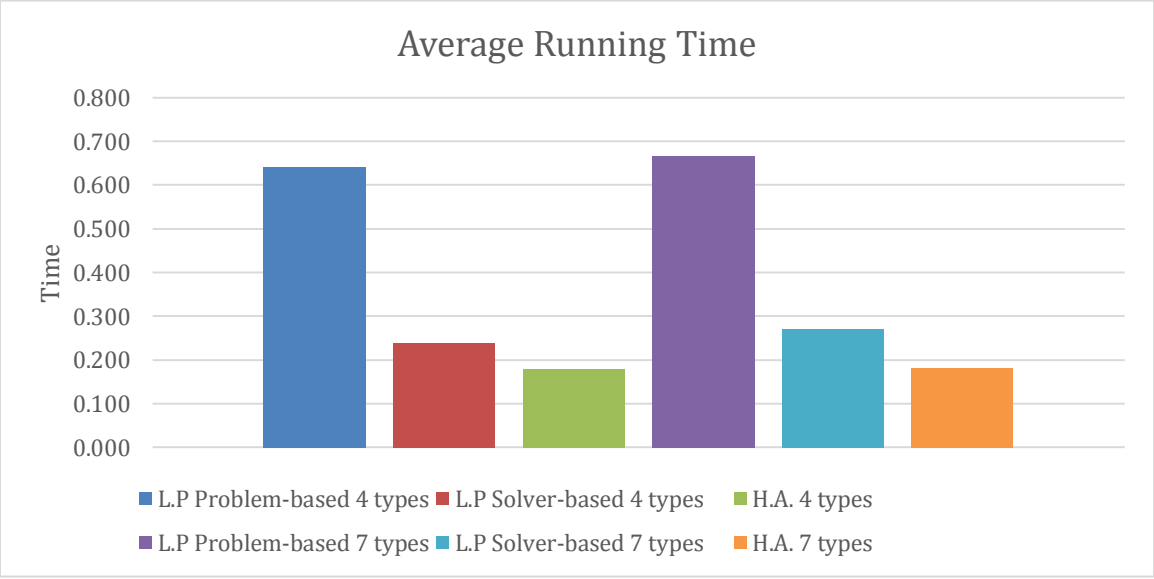

Figure 3.4. The average solution time results

Chapter 4

Cloud versus local processing in distributed networks

4.1 Background

Data processing is an important part of the fast-growing computer and communication technologies. The expanding multiple processor technologies requires effective and efficient data processing scheduling. There are two types of processing. The first one is serial processing, which cannot be divided and processed simultaneously. The second type and the focus of this thesis is parallel processing. Parallel processing means loads of data are divisible among processors, which allow efficient and effective parallel processing [19]. Divisible load scheduling also known as divisible load theory utilizes linear mathematical models. DLT has many advantages such as easy computation, a schematic language, equivalent network element modeling. DLT can be implemented in many applications such as intelligent sensors, image signal processing and large data bases. Given expanding sensor information collection abilities, there is a requirement for execution time prediction tools. DLT provides scalable and tractable models to be used as an accurate prediction tool [20].

Divisible load scheduling consists of two steps: load distribution and load processing. The data is usually distributed from one or more processors to multiple processors and processed in parallel. An optimal scheduling provides the minimum finish time. In DLT, there are no

precedence relations between the data, which allow data to be divided among a number of processors and links. Also, network architectural issues related to parallel and distributed computing can be solved implementing DLT. Dividing the load equally among the processors does not take different computer and communication link speeds, the scheduling policy and the interconnection network into account and that leads to suboptimal solutions. However, an optimal solution can be found using divisible load scheduling theory, which provides the required mathematical tools. Furthermore, the solution can be improved by integrating Amdahl's and other speedup laws.

Amdahl's law is an accurate formula to calculate the speedup of the execution of a task with fixed data size [21,22]. For multiple processors networks, Amdahl's law can be used as a prediction tool for the theoretical speedup in parallel computing. For example, suppose a program finish time using a single processor is 10 hours. If the part of the program that cannot be parallelized takes one hour to execute and the part that can be parallelized takes the remaining 9 hours (p = 0.9) of execution, then the minimum execution time cannot be less than that critical one hour. Hence, the theoretical speedup is limited to at most 10 times $\frac{1}{(1-p)}$ =10.

A task can be split up into two parts: a part that cannot be parallelized *(1-p)* and a parallelized part p. If the execution time of the whole task including both parts of the system is denoted as T, then T can be calculated using the following formula

$$T=(1-p)\ T + pT \qquad (1)$$

The parallelized part execution time can be improved by the factor *s* which is the speedup of the parallelized part. Consequently, the theoretical execution time *T*(*s*) of the whole task after the improvement can be calculated using the following formula:

$$T(s)=(1-p)\ T+\frac{p}{s}T \qquad (2)$$

Amdahl's law gives the theoretical speedup of the execution of the whole task *at fixed workload W*, which yields

$$S_{Amdahl}(s)=\frac{TW}{T(s)W}=\frac{T}{T(s)}=\frac{1}{(1-p)+\frac{p}{s}} \qquad (3)$$

Gustafson's law is similar to Amdahl's law and used to calculate the speedup of a task but with a fixed execution time [23]. As shown in Equation (4), the execution workload of the whole task before the improvement of the resources of the system "*W*" includes the execution workload of the part that cannot be parallelized and the execution workload of the parallelized part. In Equation (4), *p* is the fraction of the parallelized workload and *1-p* is the fraction of the part that cannot be parallelized.

$$W=(1-p)\ W+pW \qquad (4)$$

It is the execution of the parallelized part that is improved by a factor *s* after the improvement of the resources. Consequently, the execution of the part that cannot be parallelized remains the same. After the improvement of the resources of the system, the theoretical execution workload *W(s)* of the whole task can be calculated using the following equation:

$$W(s)=(1-p)\ W+spW \qquad (5)$$

The theoretical speedup of the execution of the whole task at fixed time *T* can be calculated using Gustafson's law as shown in the following equation:

$$S_{Gustafson}(s) = \frac{TW(s)}{TW} = \frac{W(s)}{W} = (1-p) + sp \qquad (6)$$

It is important to have a complete understanding to avoid economic waste, inaccurate future system designs and a lack of technological improvements. The motivation for this research is "intelligent" sensor networks doing measurements, communications, and computation to reach optimal signature searching. The following section of this chapter compares local and cloud computing as well as combining both methods. Local computing simply means all the computation occurs on a local network. However, cloud computing means all the data computation and storage happens in the cloud via internet [24].

The problem formulation, solution and results are discussed in the following section of this chapter. The divisible load speedup expressions are included for three fundamental load distribution protocols in the single level tree network [19]. Heterogeneous and homogenous networks are included for each protocol. Moreover, the integrated speedup formulas will be used for the cases of local and cloud computing. A combination of local and cloud computing will be tested and compared to the results of using only one computing method. Finally, the finish time and the optimal load distribution will be calculated for each case.

4.2 Solution

4.2.1 Divisible load speedup for local and cloud processing of single level tree networks

Divisible load modeling and speedup expressions have been developed for a variety of multi-processor interconnection topologies such as buses, stars, multi-level tree networks, meshes, hypercubes and other networks. Also, they have been developed for different load distribution policies such as sequential load distribution and concurrent load distribution with simultaneous or staggered start. Amdahl's Law can be modified and used to calculate the entire network speedup including serial and parallel data. The speedup of a divisible load model is implemented as the parallel part of the system replacing the "s" in Amdahl's law [20]. Other factors can now be included in Amdahl-like Laws such as interconnection topology, load distribution policy and the relative difference in computation and communication intensity and speeds.

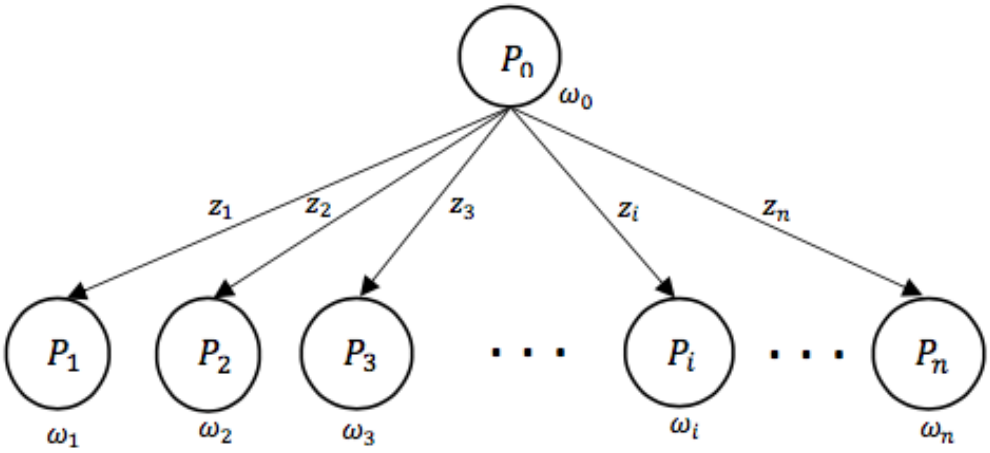

Figure 4.1. Single level tree network

Figure 4.1 shows a single level tree network [20]. The load is distributed from the root node to the children nodes. As shown in the figure above, ω_i is the ith processor's inverse computing speed and z_i is the ith link's inverse link speed. Intuitively the processors with faster link speeds will receive load prior to the ones with slower link speeds to achieve the shortest finishing time. It is assumed that computation takes more time than communication which means the inverse link speed and the communication intensity constant is smaller than the inverse computing speed and computation intensity constant. All the values used in this section are included in Table 4.1. The problem formulation, solution and results are discussed in the following section of this chapter. Also, the divisible load speedup expressions are included for three different single level tree networks. Heterogeneous and homogenous networks are included for each protocol. Moreover, the integrated speedup formulas will be used for the cases of local and cloud computing. Finally, the finish time and the optimal load distribution will be calculated for each case.

Network Type	n	ω_0	ω_c	$\omega_1, \omega_2, \omega_3, \omega_4$	z_c, z_1, z_2, z_3, z_4	T_{cp}	T_{cm}
Heterogeneous system	4	2	0.3	4,5,6,7	0.1,1.5,2,2.5,3	2	1
Homogeneous system	4	3	0.3	3,3,3,3	0.1,0.1,0.1,0.1,0.1	2	1

Table 4.1. Values of parameters used in the calculation

4.2.1.1 Sequential Load Distribution model

The timing diagram for a single level tree network with a sequential load distribution is shown in Figure 4.2 [20]. It explains the communication and computation parts of this protocol. Sequential load distribution means the source node distributes load to one child node at a time. The child node starts processing as soon as receiving the assigned load. In other words, the child node does not have to wait until the assigned load is completely received to start computing. To achieve the optimal speedup, all the nodes have to finish processing at the same time.

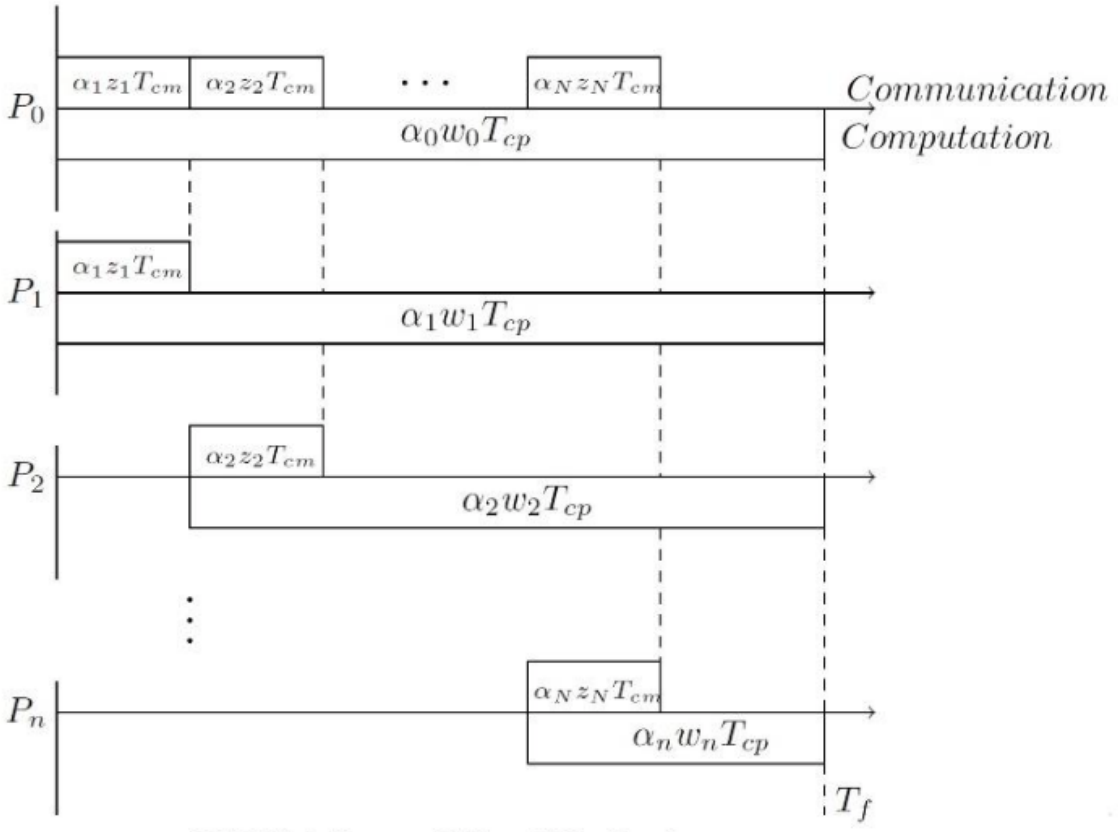

Figure 4.2. Timing Diagram for a single level tree network with a sequential load distribution

The speedup of a divisible load model of a single level tree network with a sequential load distribution parallel facility with n processors is $S_{DLT}\ (n)$. $S_{DLT}\ (n)$ can be calculated using the following equation [19]:

$$S_{DLT}\ (n)=1+ k_1\ [1+\sum_{i=2}^{n}(\prod_{l=2}^{n} q_l)] \qquad (7)$$

where $q_i = (\omega_{i-1}\ T_{cp}- z_{i-1}\ T_{cm}\ )/\omega_i\ T_{cp}$ and $k_1 = \omega_0/\omega_1$

For the system with homogeneous processors, the inverse processing speed and link speed of each processor is the same. In this case, $S_{DLT}\ (n)$ in Equation (7) can be modified and simplified to calculate the homogeneous network speedup $S_{DLT_{homo}}(n)$ using the following equation:

$$S_{DLT_{homo}}\ (n)=1+\frac{\omega_0}{\omega}\left[\frac{1-(1-\sigma)^n}{\sigma}\right] \qquad (8)$$

where: $\sigma = zT_{cm}\ /\omega T_{cp}$

Amdahl's Law can be modified to calculate the speedup of the entire network including both serial and parallel facilities using the following equation [20]:

$$S_{Amdahl} = \frac{1}{(1-f)+\frac{f}{S_{DLT}(n)}} \qquad (9)$$

To test and compare the speedup levels for different networks, Equations (7) and (8) were inserted into Equation (9). The values used are listed in Table 4.1. Heterogeneous processors are tested, and the results are shown in Table 4.2 and Figure 4.3. Also, homogeneous processors are tested, and the results are shown in Table 4.3 and Figure 4.4.

Heterogeneous Network

f	1-f	f/sp local	f/sp cloud	f/sp comb.	Ss local	Ss cloud	Ss comb.
0.000	1.000	0.000	0.000	0.000	1.000	1.000	1.000
0.100	0.900	0.046	0.013	0.012	1.057	1.095	1.097
0.200	0.800	0.092	0.026	0.023	1.121	1.211	1.215
0.300	0.700	0.138	0.039	0.035	1.193	1.353	1.361
0.400	0.600	0.184	0.052	0.046	1.275	1.533	1.547
0.500	0.500	0.230	0.065	0.058	1.369	1.769	1.793
0.600	0.400	0.276	0.078	0.069	1.479	2.091	2.130
0.700	0.300	0.322	0.091	0.081	1.607	2.556	2.625
0.800	0.200	0.368	0.104	0.093	1.760	3.286	3.418
0.900	0.100	0.414	0.117	0.104	1.944	4.600	4.899
1.000	0.000	0.460	0.130	0.116	2.172	7.667	8.643

Table 4.2. Heterogeneous processors testing results for sequential load distribution

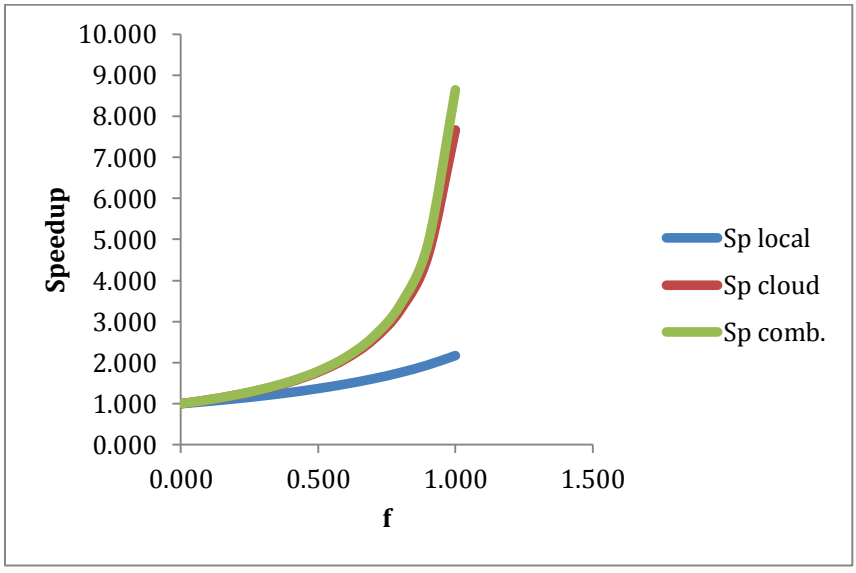

Figure 4.3. Heterogeneous processors results for sequential load distribution

Homogeneous Network

f	1-f	f/sp local	f/sp cloud	f/sp comb.	Ss local	Ss cloud	Ss comb.
0.000	1.000	0.000	0.000	0.000	1.000	1.000	1.000
0.100	0.900	0.020	0.009	0.007	1.086	1.100	1.103
0.200	0.800	0.041	0.018	0.014	1.189	1.222	1.228
0.300	0.700	0.061	0.027	0.021	1.314	1.375	1.387
0.400	0.600	0.082	0.036	0.028	1.467	1.571	1.592
0.500	0.500	0.102	0.045	0.035	1.661	1.833	1.869
0.600	0.400	0.122	0.055	0.042	1.914	2.200	2.262
0.700	0.300	0.143	0.064	0.049	2.258	2.750	2.864
0.800	0.200	0.163	0.073	0.056	2.753	3.667	3.904
0.900	0.100	0.184	0.082	0.063	3.525	5.500	6.129
1.000	0.000	0.204	0.091	0.070	4.900	11.000	14.248

Table 4.3. Homogeneous processors results for sequential load distribution

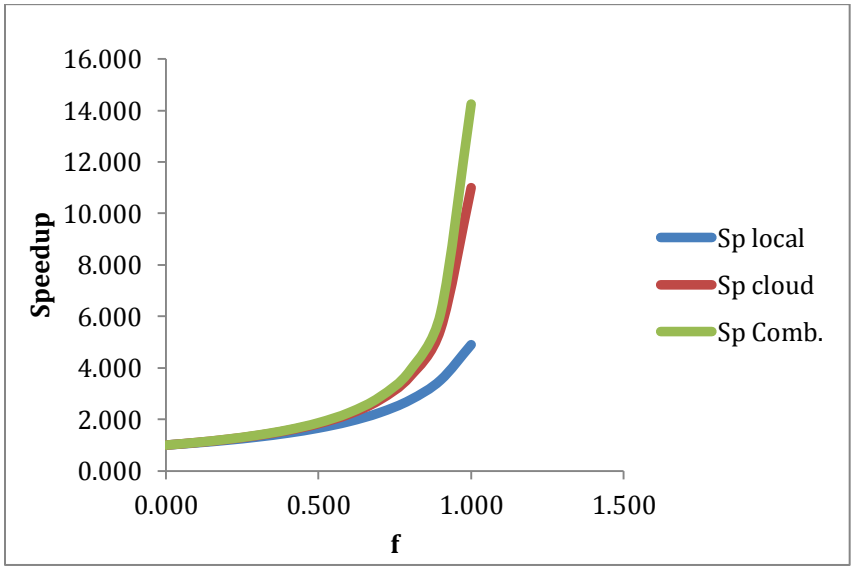

Figure 4.4. Homogeneous processors results for sequential load distribution

4.2.1.2 Simultaneous Distribution, Staggered Start model

The timing diagram for a single level tree network with a simultaneous distribution, staggered start is shown in Figure 4.5 [20]. The communication and computation parts of this protocol are included in the figure. The second model involves simultaneous distribution of load which means the source node distributes the divisible load to all children nodes simultaneously. In this model, a child node can start load processing only after receiving the entire assigned load from the source node. To achieve the optimal speedup, all the nodes have to finish processing at the same time.

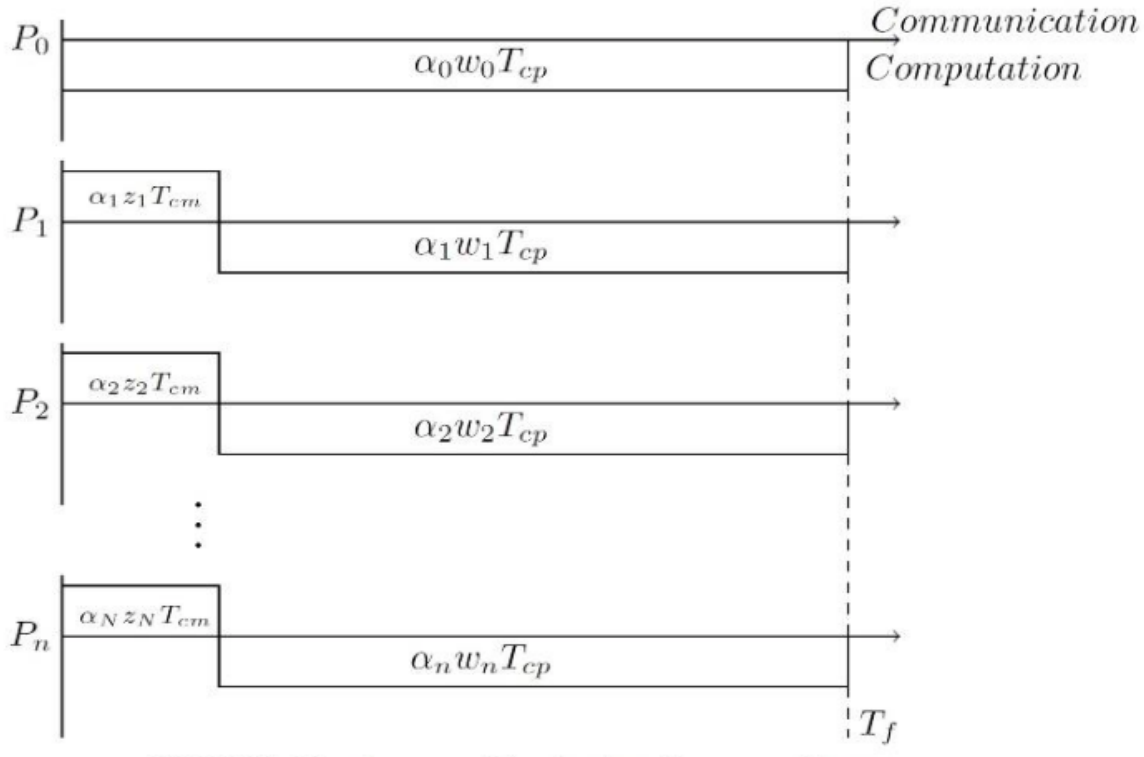

Figure 4.5. Timing Diagram for a single level tree network with a simultaneous load distribution and staggered start

The speedup of a divisible load model of a single level tree network with a simultaneous load distribution and staggered start parallel facility with n processors is $S_{DLT}\ (n)$. $S_{DLT}\ (n)$ can be calculated using the following equation [19]:

$$\boldsymbol{S}_{DLT}(n)=1+\omega_0 T_{cp} \sum_{i=1}^{n} \frac{1}{(\omega_i} \ T_{cp} + \ z_i \ T_{cm}) \qquad (13)$$

For the system with homogeneous processors, the inverse processing speed and link speed of each processor is the same. In this case, $S_{DLT}\ (n)$ in Equation (13) can be modified and simplified to calculate the homogeneous network speedup $S_{DLT_{homo}}(n)$ using the following equation:

$$S_{DLT_{homo}}(n) = 1 + k \times n \qquad (14)$$

where: $k=\omega_0 T_{cp} / (\omega T_{cp} + z T_{cm})$

Amdahl's Law can be modified to calculate the speedup of the entire network including both serial and parallel facilities using Equation (9). To test and compare the speedup levels for different networks, Equations 13 and 14 were inserted into Equation (9). The values used are listed in Table 4.1. The testing results of heterogeneous networks are shown in Table 4.4 and Figure 4.6. Also, Homogeneous processors are tested, and the results are shown in Table 4.5 and Figure 4.7.

Heterogeneous Network

f	1-f	f/sp local	f/sp cloud	f/sp comb.	Ss local	Ss cloud	Ss comb.
0.000	1.000	0.000	0.000	0.000	1.000	1.000	1.000
0.100	0.900	0.044	0.015	0.013	1.059	1.093	1.096
0.200	0.800	0.088	0.030	0.025	1.126	1.205	1.212
0.300	0.700	0.132	0.045	0.038	1.201	1.343	1.356
0.400	0.600	0.177	0.060	0.050	1.288	1.516	1.538
0.500	0.500	0.221	0.074	0.063	1.387	1.741	1.777
0.600	0.400	0.265	0.089	0.075	1.504	2.043	2.104
0.700	0.300	0.309	0.104	0.088	1.642	2.474	2.579
0.800	0.200	0.353	0.119	0.100	1.808	3.133	3.330
0.900	0.100	0.397	0.134	0.113	2.011	4.273	4.699
1.000	0.000	0.442	0.149	0.125	2.265	6.714	7.979

Table 4.4. Heterogeneous processors results for simultaneous load distribution and staggered start

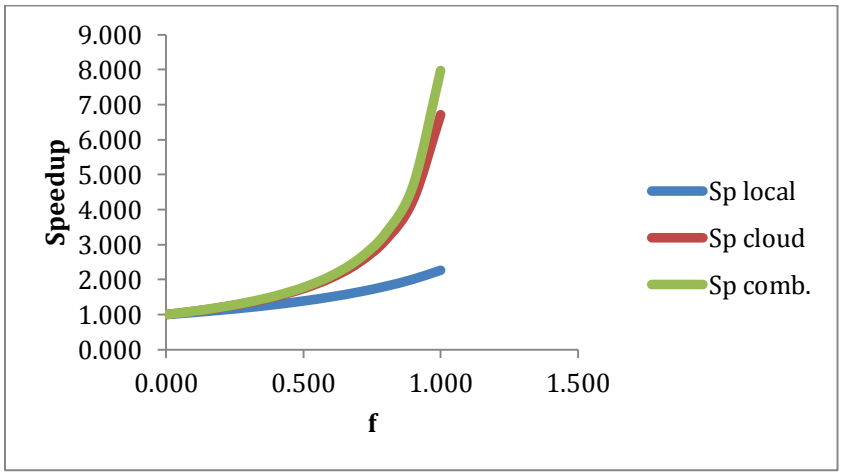

Figure 4.6. Heterogeneous processors results for simultaneous load distribution and staggered start

Homogeneous Network

f	1-f	f/sp local	f/sp cloud	f/sp comb.	Ss local	Ss cloud	Ss comb.
0.000	1.000	0.000	0.000	0.000	1.000	1.000	1.000
0.100	0.900	0.020	0.010	0.007	1.087	1.098	1.102
0.200	0.800	0.041	0.021	0.015	1.190	1.218	1.227
0.300	0.700	0.061	0.031	0.022	1.314	1.367	1.385
0.400	0.600	0.081	0.042	0.030	1.468	1.558	1.588
0.500	0.500	0.101	0.052	0.037	1.663	1.811	1.862
0.600	0.400	0.122	0.063	0.044	1.917	2.161	2.250
0.700	0.300	0.142	0.073	0.052	2.263	2.680	2.842
0.800	0.200	0.162	0.084	0.059	2.762	3.526	3.858
0.900	0.100	0.182	0.094	0.067	3.542	5.154	6.001
1.000	0.000	0.203	0.104	0.074	4.936	9.571	13.507

Table 4.5. Homogeneous processors results for simultaneous load distribution and staggered start

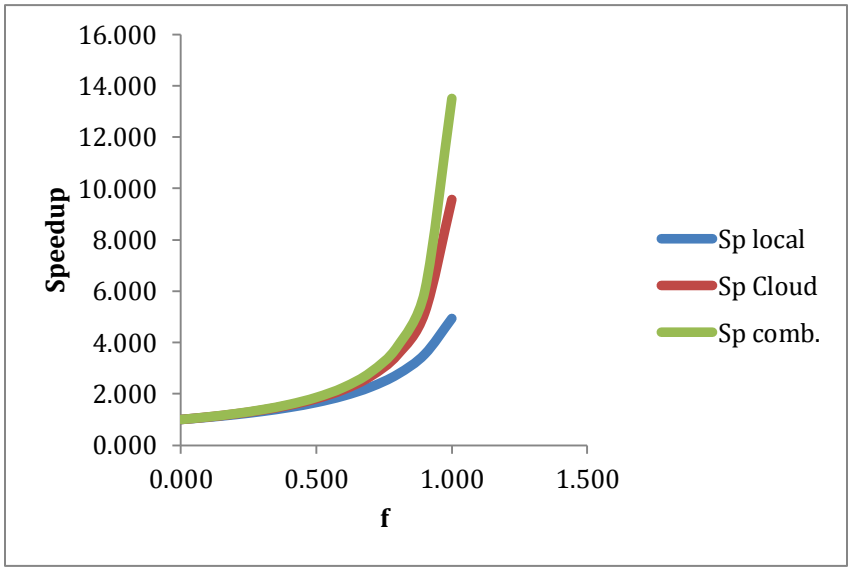

Figure 4.7. Homogeneous processors results for simultaneous load distribution and staggered start

4.2.1.3 Simultaneous Distribution, Simultaneous Start model

The timing diagram for a single level tree network with a simultaneous distribution, simultaneous start is shown in Figure 4.8 [20]. The figure includes communication and computation parts of this protocol. This model involves simultaneous distribution of load which means the source node distributes the divisible load to all children nodes simultaneously. In this model, a child node can start load processing as soon as it begins to receive its assigned load from the source node. In other words, the child node does not have to wait until the assigned load is completely received to start computing. All the nodes have to finish processing at the same time to achieve the optimal speedup.

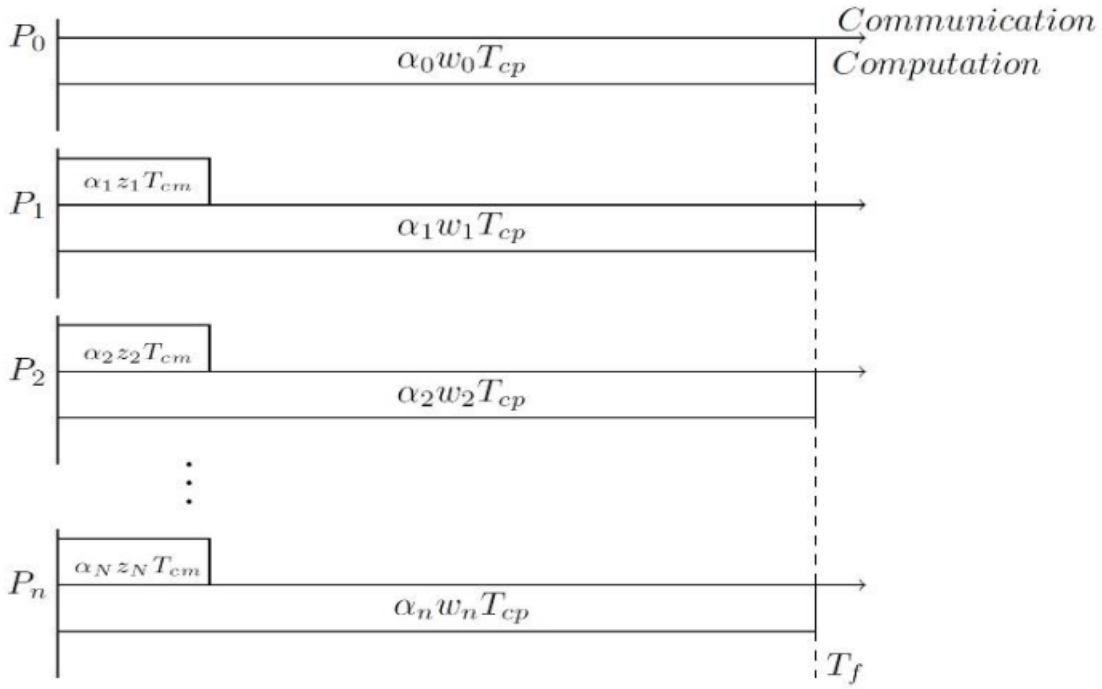

Figure 4.8. Timing diagram of a single level tree network with a simultaneous distribution, simultaneous start

The speedup of a divisible load model of a single level tree network with a simultaneous distribution, simultaneous start parallel facility with n processors is $S_{DLT}\ (n)$. $S_{DLT}\ (n)$ can be calculated using the following equation [19]:

$$S_{DLT}\ (n)=1+\omega_0 \sum_{i=1}^{n}(\frac{1}{\omega_i}) \qquad (15)$$

For the system with homogeneous processors, the inverse processing speed and link speed of each processor is the same. In this case, $S_{DLT}\ (n)$ in Equation 15 can be modified and simplified to calculate the homogeneous network speedup $S_{DLT_{homo}}(n)$ using the following equation:

$$S_{DLT_{homo}}(n) = 1 + k \times n \qquad (16)$$

where $k = \omega_0/\omega$

Amdahl's Law can be modified to calculate the speedup of the entire network including both serial and parallel facilities using Equation (9). To test and compare the speedup levels for different networks, Equations (15) and (16) were inserted into Equation (9). The values used are listed in Table 4.1. The testing results of heterogeneous networks are shown in Table 4.6 and Figure 4.9. Also, Homogeneous processors are tested, and the results are shown in Table 4.7 and Figure 4.10.

Heterogeneous Network

f	1-f	f/sp local	f/sp cloud	f/sp comb.	Ss local	Ss cloud	Ss comb.
0.000	1.000	0.000	0.000	0.000	1.000	1.000	1.000
0.100	0.900	0.040	0.013	0.011	1.064	1.095	1.097
0.200	0.800	0.079	0.026	0.023	1.137	1.211	1.216
0.300	0.700	0.119	0.039	0.034	1.221	1.353	1.362
0.400	0.600	0.159	0.052	0.045	1.318	1.533	1.550
0.500	0.500	0.198	0.065	0.057	1.432	1.769	1.796
0.600	0.400	0.238	0.078	0.068	1.567	2.091	2.137
0.700	0.300	0.278	0.091	0.079	1.730	2.556	2.636
0.800	0.200	0.318	0.104	0.091	1.932	3.286	3.441
0.900	0.100	0.357	0.117	0.102	2.187	4.600	4.951
1.000	0.000	0.397	0.130	0.113	2.519	7.667	8.826

Table 4.6. Heterogeneous processors results for simultaneous load distribution and simultaneous start

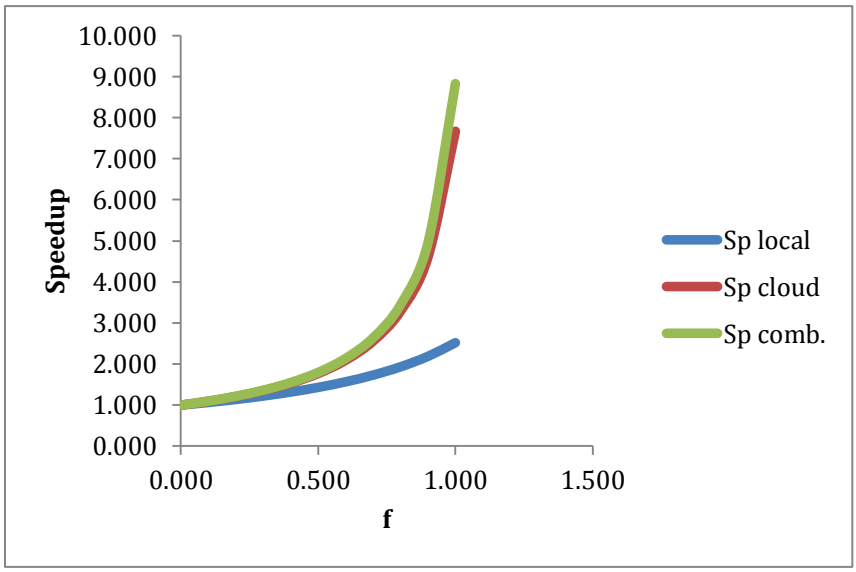

Figure 4.9. Heterogeneous processors results for simultaneous load distribution and simultaneous start

Homogeneous Network

f	1-f	f/sp local	f/sp cloud	f/sp comb.	Ss local	Ss cloud	Ss comb.
0.000	1.000	0.000	0.000	0.000	1.000	1.000	1.000
0.100	0.900	0.020	0.009	0.007	1.087	1.100	1.103
0.200	0.800	0.040	0.018	0.013	1.190	1.222	1.230
0.300	0.700	0.060	0.027	0.020	1.316	1.375	1.389
0.400	0.600	0.080	0.036	0.027	1.471	1.571	1.596
0.500	0.500	0.100	0.045	0.033	1.667	1.833	1.875
0.600	0.400	0.120	0.055	0.040	1.923	2.200	2.273
0.700	0.300	0.140	0.064	0.047	2.273	2.750	2.885
0.800	0.200	0.160	0.073	0.053	2.778	3.667	3.947
0.900	0.100	0.180	0.082	0.060	3.571	5.500	6.250
1.000	0.000	0.200	0.091	0.067	5.000	11.000	15.000

Table 4.7. Homogeneous processors results for simultaneous load distribution and simultaneous start

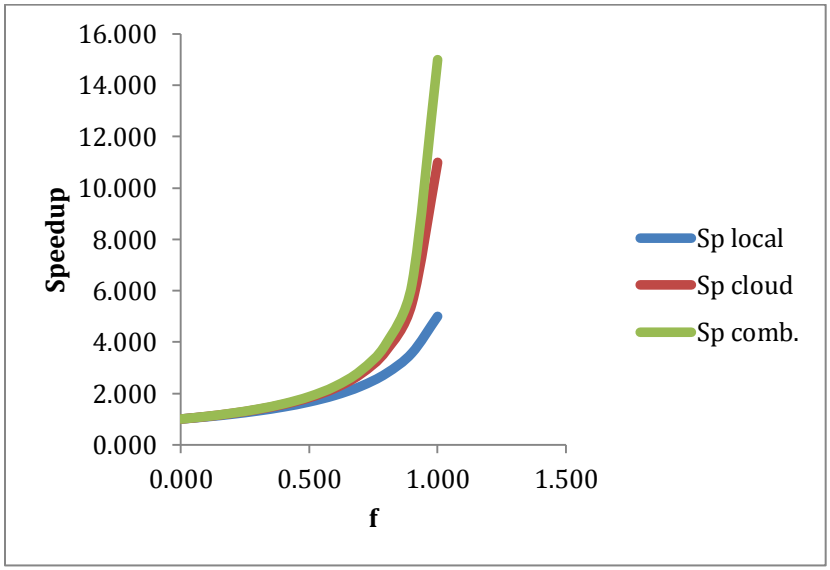

Figure 4.10. Homogeneous processors results for simultaneous load distribution and simultaneous start

Heterogeneous Network Comparison

The previous models will be used to compare local and cloud processing. Also, a combination of both processing methods will be compared to find the best processing method. Heterogeneous systems comparison will be shown in Figure 4.11, 4.12 and 4.13. Homogeneous systems comparison will be shown in Figure 4.14, 4.15 and 4.16.

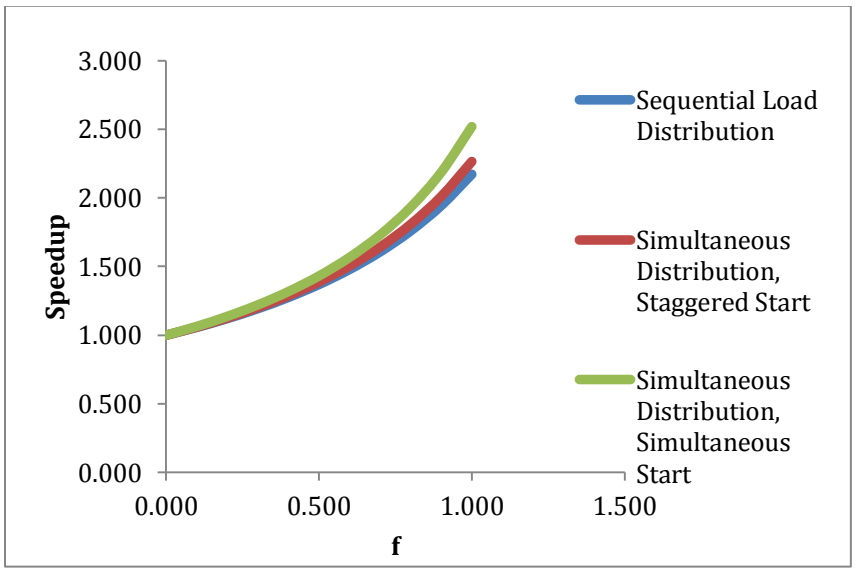

Figure 4.11. Local Processing results for all single level tree models

Figure 4.12 and 4.15. compares the results of cloud processing for all single level tree models. In Figure 4.12, 4.15 the speedup values for the sequential load distribution model and the simultaneous distribution simultaneous start model are slightly higher than the simultaneous distribution staggered start model. This is because the processor in the simultaneous distribution staggered start model cannot start processing until it receives the entire assigned load. All the

following Figures show very close results for all three single level tree models because of the same processing method being used. In other words, every figure compares all three models using one processing method.

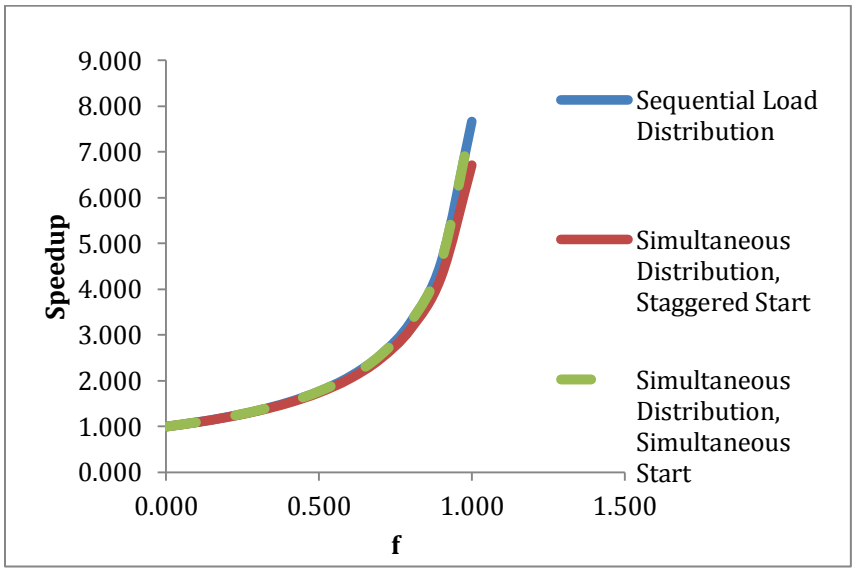

Figure 4.12. Cloud Processing results for all single level tree models

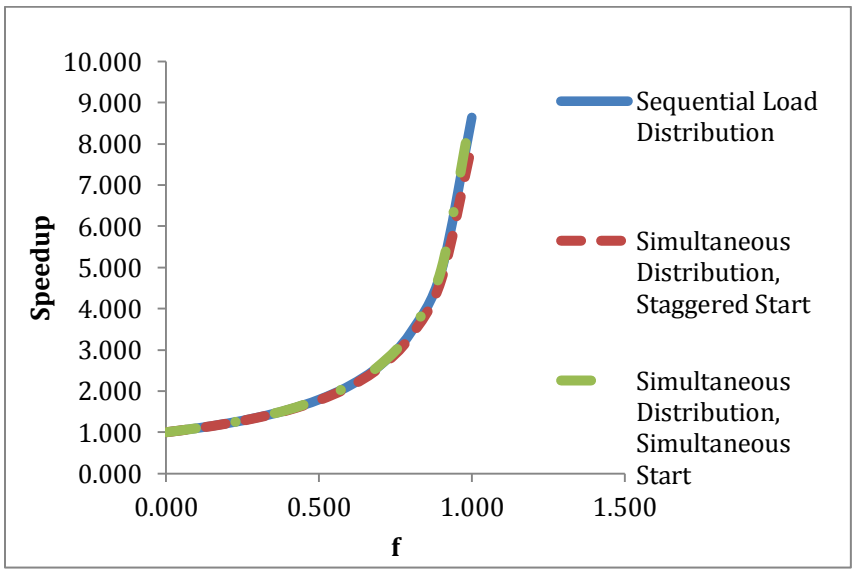

Figure 4.13. Combination of Local and Cloud Processing results for all single level tree models

Homogeneous Network Comparison

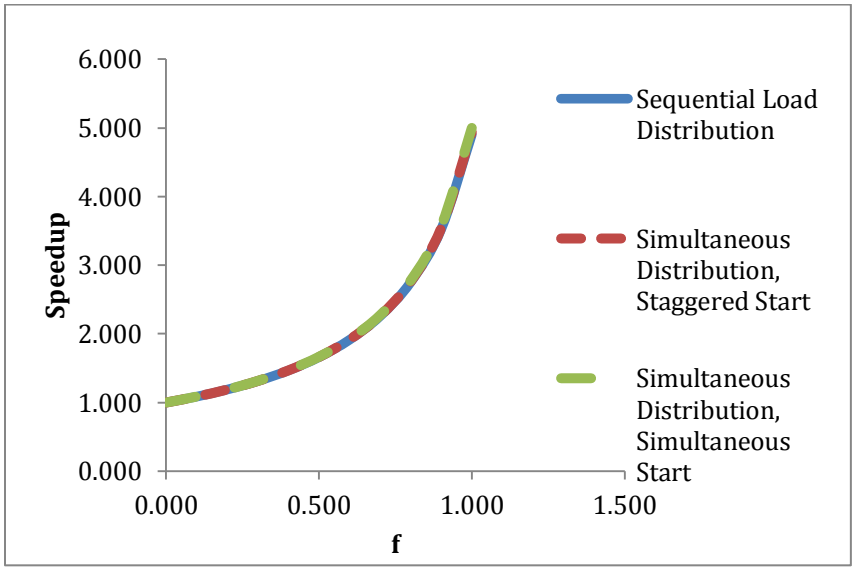

Figure 4.14. Local Processing results for all single level tree models

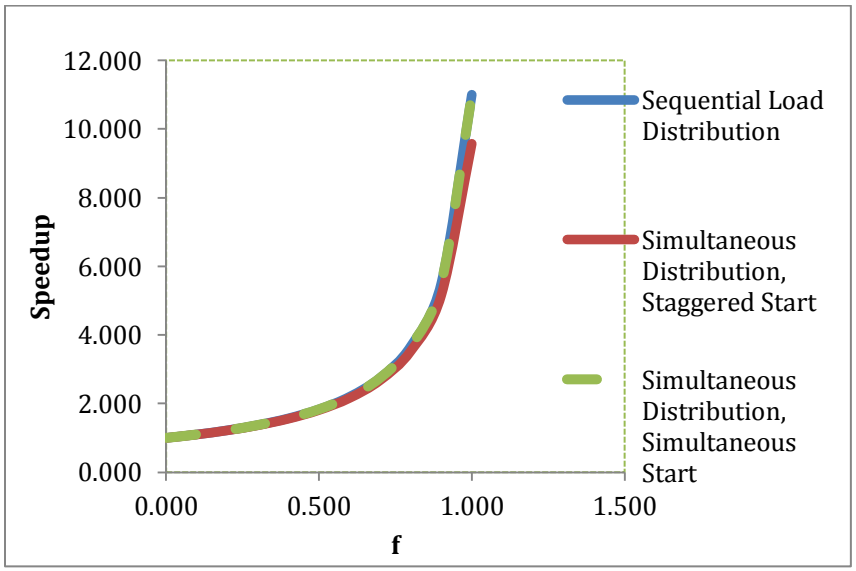

Figure 4.15. Cloud Processing results for all single level tree models

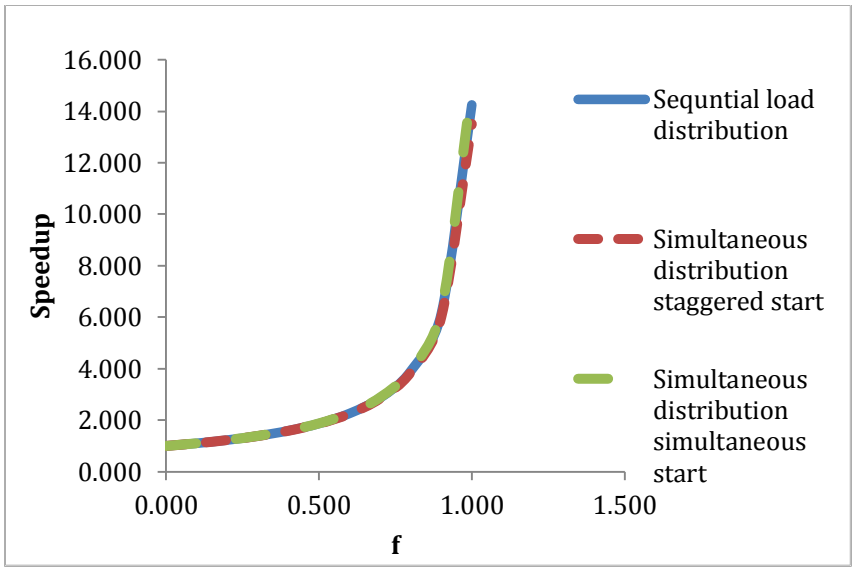

Figure 4.16. Combination of Local and Cloud Processing results for all single level tree models

4.2.2 Optimal finish time for local and cloud processing of single level tree networks

Optimal load distribution is a key to calculate an accurate finish time $T_{f,m}$. $T_{f,m}$ indicates the time it takes for a single level tree consisted of the source node and m nodes to finish processing the entire load. The finish time in this section will be used as a measurement tool to compare local and cloud processing. Also, a combination of both processing methods will be tested and compared. The calculation will be done on the single level tree models used previously. All the results are included in Table 4.8, 4.9 and 4.10.

4.2.2.1 Sequential Load Distribution

$$\alpha_0\omega_0T_{cp} = \alpha_1\omega_1T_{cp} \quad (17)$$

$$\alpha_0 = \frac{\omega_1}{\omega_0}\alpha_1 = \frac{1}{k_1}\alpha_1 \qquad \text{where, } k_1 = \frac{\omega_0}{\omega_1}$$

$$\alpha_{i-1}\omega_{i-1}T_{cp} = \alpha_{i-1}z_{i-1}T_{cm} + \alpha_i\omega_iT_{cp} \quad (18)$$

$$\alpha_i = \frac{\omega_{i-1}T_{cp} - z_{i-1}T_{cm}}{\omega_iT_{cp}}\alpha_{i-1} = q_i\alpha_{i-1} = (\prod_{l=2}^{i} q_l) \times \alpha_1 \qquad \text{where, } q_i = \frac{\omega_{i-1}T_{cp} - z_{i-1}T_{cm}}{\omega_iT_{cp}}$$

$$\alpha_0 + \alpha_1 + \alpha_2 + \cdots + \alpha_m = 1 \quad (19)$$

Using Equations (17, 18) and (19) provides the following equation:

$$[\frac{1}{k_1} + 1 + \sum_2^m(\prod_{l=2}^{i} q_l)]\alpha_1 = 1$$

$$\alpha_1 = \frac{1}{[\frac{1}{k_1}+1+\sum_2^m(\prod_{l=2}^{i} q_l)]} \quad (20)$$

Using Equation (20) and the values in table 4.1 to find the finish time using the following equation:

$$T_{f,m} = \alpha_0\omega_0T_{cp} = \frac{1}{k_1}\alpha_1\omega_0T_{cp} = \frac{1}{1+k_1[1+\sum_2^m(\prod_{l=2}^{i} q_l)]}\ \omega_0T_{cp} \quad (21)$$

All the results for the Sequential Load Distribution model are shown in table 4.8

Sequential Load Distribution	Local Processing	Cloud Processing	Combination of Local and Cloud Processing
Heterogeneous Network	1.84	0.522	0.463
Homogeneous Network	1.2	0.546	0.42

Table 4.8. The finish time results for the Sequential Load Distribution model

4.2.2.2 Simultaneous Distribution, Staggered Start

$$\alpha_0\omega_0T_{cp} = \alpha_1\omega_1T_{cp} + \alpha_1z_1T_{cm} \quad (22)$$

$$\alpha_0 = \frac{\omega_1T_{cp}+ z_1T_{cm}}{\omega_0T_{cp}}\ \alpha_1 = \frac{1}{k_1}\alpha_1 \qquad \text{where, } k_1 = \frac{\omega_0T_{cp}}{\omega_1T_{cp}+ z_1T_{cm}}$$

$$\alpha_{i-1}\omega_{i-1}T_{cp} + \alpha_{i-1}z_{i-1}T_{cm} = \alpha_i\omega_iT_{cp} + \alpha_iz_iT_{cm} \quad (23)$$

$$\alpha_i = \frac{\omega_{i-1}T_{cp}+ z_{i-1}T_{cm}}{\omega_iT_{cp} + z_iT_{cm}}\alpha_{i-1} = q_i\alpha_{i-1} = (\prod_{l=2}^{i} q_l) \times \alpha_1 \qquad \text{where, } q_i = \frac{\omega_{i-1}T_{cp}+ z_{i-1}T_{cm}}{\omega_iT_{cp}+ z_iT_{cm}}$$

$$\alpha_0 + \alpha_1 + \alpha_2 + \cdots + \alpha_m = 1 \quad (24)$$

Using Equations (22, 23) and (24) provides the following equation:

$$[\frac{1}{k_1} + 1 + \sum_2^m(\prod_{l=2}^{i} q_l)]\alpha_1 = 1$$

$$\alpha_1 = \frac{1}{[\frac{1}{k_1}+1+\sum_2^m(\prod_{l=2}^{i} q_l)]} \quad (25)$$

Using Equation (25) and the values in table 4.1 to find the finish time using the following equation:

$$T_{f,m} = \alpha_0\omega_0T_{cp} = \frac{1}{k_1}\alpha_1\omega_0T_{cp} = \frac{1}{1+k_1[1+\sum_2^m(\prod_{l=2}^{i} q_l)]}\ \omega_0T_{cp} \quad (26)$$

All the results for the Simultaneous Distribution, staggered start model are shown in table 4.9 below.

Simultaneous Distribution, Staggered Start	Local Processing	Cloud Processing	Combination of Local and Cloud Processing
Heterogeneous Network	1.767	0.596	0.501
Homogeneous Network	1.213	0.628	0.445

Table 4.9. The finish time results for the Simultaneous Distribution, Staggered Start model

4.2.2.3 Simultaneous Distribution, Simultaneous Start

$$\alpha_0 \omega_0 T_{cp} = \alpha_1 \omega_1 T_{cp} \quad (27)$$

$$\alpha_0 = \frac{\omega_1}{\omega_0}\alpha_1 = \frac{1}{k_1}\alpha_1 \qquad \text{where, } k_1 = \frac{\omega_0}{\omega_1}$$

$$\alpha_{i-1} \omega_{i-1} T_{cp} = \alpha_i \omega_i T_{cp} \quad (28)$$

$$\alpha_i = \frac{\omega_{i-1} T_{cp}}{\omega_i T_{cp}}\alpha_{i-1} = q_i \alpha_{i-1} = (\prod_{l=2}^{i} q_l) \times \alpha_1 \qquad \text{where, } q_i = \frac{\omega_{i-1}}{\omega_i}$$

$$\alpha_0 + \alpha_1 + \alpha_2 + \cdots + \alpha_m = 1 \quad (29)$$

Using Equations (27, 28) and (29) provides the following equation:

$$[\frac{1}{k_1} + 1 + \sum_2^m (\prod_{l=2}^{i} q_l)]\alpha_1 = 1$$

$$\alpha_1 = \frac{1}{[\frac{1}{k_1} + 1 + \sum_2^m (\prod_{l=2}^{i} q_l)]} \quad (30)$$

Using Equation (30) and the values in table 4.1 to find the finish time using the following equation:

$$T_{f,m} = \alpha_0 \omega_0 T_{cp} = \frac{1}{k_1}\alpha_1 \omega_0 T_{cp} = \frac{1}{1 + k_1[1 + \sum_2^m (\prod_{l=2}^{i} q_l)]}\ \omega_0 T_{cp} \quad (31)$$

All the results for the Simultaneous Distribution, simultaneous start model are shown in table 4.10.

Simultaneous Distribution, Simultaneous Start	Local Processing	Cloud Processing	Combination of Local and Cloud Processing
Heterogeneous Network	1.584	0.522	0.436
Homogeneous Network	1.2	0.546	0.4

Table 4.10. The finish time results for the Simultaneous Distribution, Simultaneous Start model

4.3 Analysis

After finishing all the calculation, all the results were analytically reviewed. Throughout working on this chapter, it became clear that the results depend on the parameters used. In other words, a certain set of parameters will provide a specific answer that a different set of parameters wouldn't provide. Based on the previous statement all the following analysis are based on the work and results mentioned in the previous section of this chapter.

The results of the first model sequential load distribution in table 4.2 and table 4.3 shows clearly that the speedup values for the system with homogeneous processors are higher than the values of the system with heterogeneous processors for the set of parameters in table 4.1. The reason is the processing speed for the homogeneous processors equals the highest processing speed among the heterogenous processors in the specified parameters. In other words, the homogenous system has higher computation power than the heterogeneous system. The results of the second model Simultaneous Distribution, Staggered Start and the third model Simultaneous Distribution, Simultaneous Start proved this statement. Moreover, the results of the second part of the solution which was about the optimal load distribution and the finish time support this statement. The finish time results for all three models are in Table 4.8, 4.9 and 4.10. For all three models, systems with homogeneous processors have a smaller finish time results than system with heterogeneous processors.

Analyzing the results of local processing speedup for different network topologies shows clearly that simultaneous distribution models have higher speedup values than the sequential distribution model. This is because the source node in the sequential distribution model send the assigned divisible loads to the children nodes one at a time. In other words, the children nodes ranked lower in the sequence must wait for a considerable length of time. However, the children nodes can all start receiving load near the starting time in the simultaneous distribution model. The local processing speedup results in Table 4.2, 4.3, 4.4, 4.5, 4.6 and 4.7 support this analysis.

After comparing sequential load distribution with the simultaneous distribution topologies, the next step is to compare the simultaneous distribution topologies. The simultaneous distribution topologies are simultaneous distribution, staggered start model and simultaneous distribution, simultaneous start model. The simultaneous start model results in Table 4.6 and 4.7 shows higher speedup values than the staggered start model. The reason is all the nodes can start processing as soon as they begin receiving their assigned load. Unlike the staggered start model, children nodes do not have to wait for the assigned load to be completely received to start processing.

All the results in the first part of the solution section shows the influence of the size of the parallelizable load (f) on the speedup values of different network topologies used in this chapter. Overall, the model has a higher speedup values when the value of (f) increases. That can be seen clearly in Figures 4.3, 4.4, 4.6, 4.7, 4.9 and 4.10.

Finally, the results were analyzed to find the best processing method. Local, cloud and a combination of both processing methods were compared in this chapter. Based on the results and implementing the parameters in Table 4.1, the best processing method is combining Local and

cloud processing. Incorporating both Local and cloud processing provides higher computation power and minimize communication delays.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Integer linear programming was used in this paper to optimize active sensing. Active sensors emit signals to detect information about an object. Integer linear programming is a mathematical method to achieve the best outcome and it was used to select the best signals to be emitted. Optimal sensing means acquiring more information with least energy and time costs which leads to processing more data. Intuitively, optimal sensing results in optimal signature searching. A signature is a pattern of interest and signature searching is a process to find that pattern in a data file.

The second chapter presented an algorithm to reach the optimal sensing by selecting the best possible number of signals of a type or a combination of multiple types to ensure the best sensing quality possible considering all given constraints. There were three types of constraints, which are energy, computation time and the number of signals per type. Also, we assumed three types of signals with different energy, quality and computation time specifications.

The performance evaluation was done in three stages. The first stage is studying basic cases of constraints. Four different cases were created for testing. The objective of the algorithm is to find the optimal number of signals of each type that would result in the maximum total quality. The second stage is studying complex cases of linear and quadratic relationship between energy/computation time and quality per signal. This stage consists of two parts. The first part is

combinations of five cases to study the relationship between the energy and quality per signal with constant computation time specifications. The results in Table 2.2 and 2.4 and Figure 2.14 and 2.16 show that increasing the energy cost leads to a drop in the maximum total quality. Secondly, studying combinations of five cases to find the relationship between the computation time and quality per signal with constant energy specifications. The results in Table 2.6 and 2.8 and Figure 2.16 and 2.20 show that increasing the computation time results in a drop in the maximum total quality. The third stage is studying the effect of restricting the number of signals per type. Restricting the number of signals per type affect the maximum total quality. The results show improvements using integer linear programming. Integer linear programming ability to optimize any set of parameters and constraints makes it a very useful tool. The technique proposed is promising in its ability to automate signal selection for remote sensing.

New approaches were presented in the third chapter to optimize the solution in the second chapter using linear programming and heuristic algorithm. Instead of using problem based linear programming such as in the second chapter, a solver-based program is used to optimize the solution. Then, a solution based on heuristic algorithm is implemented which shows improvements in the performance. The results show improvements using solver-based linear programming and a heuristic algorithm. The performance evaluation portion of this solution was done in three stages. The first stage is studying basic cases of parameters and constraints to find the maximum quality using the problem based, solver based and heuristic algorithm. The first stage purpose is to verify that all programs run properly. The second stage is to test and compare the running time for all three solutions using four signal types. the performance was enhanced from 0.64 to 0.238 seconds using solver-based linear programming. Then, the heuristic algorithm was implemented and tested. The average running time was improved to 0.178. The third and final stage is to test the running

time for all three proposed solution and compare them using seven signal types. The average running time for the problem-based, solver-based and the heuristic algorithm were respectively 0.667, 0.271 and 0.183 seconds. Linear programming and heuristic algorithm ability to optimize any set of parameters with specific constraints make them very useful tools for optimization problem.

Amdahl's law is an effective tool to evaluate the performance of parallel systems by calculating the speedup. The speedup value can be utilized to compare the performances of different system topologies. In the fourth chapter, Amdahl's law and divisible load modeling were integrated to evaluate different single level tree topologies. divisible load modeling was used because it provides tractable and realistic models. Also, it allows precise mathematical analysis.

The fourth chapter compared local and cloud computing for different single level tree topologies. Also, it evaluated the performance of combining both methods. The performance evaluation was done in two parts. The first part was using Amdahl's law and divisible load modeling to calculate the speedup for three single level tree models. The models are sequential load distribution, simultaneous distribution with staggered start and simultaneous distribution with simultaneous start. Homogeneous and heterogeneous systems were analyzed for each model. The second part was done to find the finish time “makespan”. The finish time was calculated and used to compare the performance of all the previous models.

The results of the analysis show clearly that simultaneous distribution models have higher speedup values than the sequential distribution model. This is because the children nodes can all start receiving load near the starting time in the simultaneous distribution model. However, the children nodes ranked lower in the sequence must wait for a considerable length of time in the sequential distribution model. The simultaneous start model results show higher speedup values

than the staggered start model. The reason is all the nodes can start processing as soon as they begin receiving their assigned load unlike the staggered start model. Moreover, the speedup values increase when the value of (f) increases. Finally, the best processing method is combining Local and cloud processing because it provides higher computation power and minimizes communication delays.

5.2 Future Work

For future work, the algorithm presented in the second chapter could be applied to actual applications. The collected results can be compared to previous results. A comparison will show the improvement of optimization of signature searching. Finally, modifications should be applied if results show a need for this.

The proposed algorithms in the third chapter could be applied to actual applications too. Also, new approaches could be implemented and tested. The results can be compared to presented results. A comparison may show improvement of optimization of signature searching. Finally, modifications should be applied if results show a need for this.

Efficient scheduling is highly required in sensors network applications and multiprocessors systems. This requirement drives consistent improvement and development. For future work, divisible load scheduling theory can be incorporated with Gustafson's law, a variant of Amdahl's law. Also, a different set of parameters for computing and computation speed can be utilized. moreover, the results can be tested and compared with a larger number of processors. Modifications should be applied if results show a need for that. Finally, other types of network topologies can be studied.

Other aspects of signature searching optimization problem can be studied such as different network topology, memory inclusion, multi installments scheduling, and queueing. The optimal solution could be using a certain method or a combination of multiple methods.

Bibliography

[1] K. Ko and T. G. Robertazzi, "*Signature Search Time Evaluation in Flat File Databases"*. IEEE Transactions on Aerospace and Electronic Systems, VOL. 44, NO. 2, APRIL 2008.

[2] Z. Ying and T. G. Robertazzi, *"Signature Searching in a Networked Collection of Files"*. IEEE Transactions on Parallel and Distributed Systems, VOL. 25, NO. 5, MAY 2014.

[3] Y. Kyong and T. G. Robertazzi. *"Greedy Signature Processing with Arbitrary Location Distributions: A Divisible Load Framework"*. IEEE Transactions on Aerospace and Electronic Systems VOL. 48, NO. 4, OCTOBER 2012.

[4] F. Sivrikaya and B. Yener, "*Time synchronization in sensor networks: a survey*", Network, IEEE In Network, IEEE, Vol. 18, No.4. (2004), pp. 45-50.

[5] H. U. Yildiz, B. Tavli and H. Yanikomeroglu, "Transmission Power Control for Link-Level Handshaking in Wireless Sensor Networks," in *IEEE Sensors Journal*, vol. 16, no. 2, pp. 561-576, Jan.15, 2016.

[6] O. Avecilla. "*Sensors: Different Types of Sensors*". Engineering Garage. March 2011.

[7] J. Moon and T. Basar. *"Static Optimal Sensor Selection via Linear Integer Programming: The Orthogonal Case"*. IEEE Signal Processing Letters, VOL. 24, NO. 7. JULY 2017.

[8] U. Rashid, H. D. Tuan, H. H. Kha and H. H. Nguyen, "Semi-definite programming for distributed tracking of dynamic objects by nonlinear sensor network," *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Prague, 2011, pp. 3532-3535.

[9] S. M. Sowelam and A. H. Tewfik, "Optimal waveform selection for radar target classification," *Proceedings of International Conference on Image Processing*, Santa Barbara, CA, 1997, pp. 476-479 vol.3.

[10] D. J. Kershaw and R. J. Evans, "Optimal waveform selection for tracking systems," in *IEEE Transactions on Information Theory*, vol. 40, no. 5, pp. 1536-1550, Sept. 1994.

[11] B. F. La Scala and B. Moran, "Measures of effectiveness for waveform selection," *2004 International Waveform Diversity & Design Conference*, Edinburgh, 2004, pp. 1-5.

[12] K. Guo, X. Chen, Y. Zuo, A. Xue, Y. Hu and Yajun Zheng. A New Programming Algorithm for Multi-sensor Task Assignment under Energy Constraints. International Conference on Mechatronics and Control (ICMC). July 2014.

[13] A. Richards, T. Schouwenaars, J. How, and E. Feron. *"Spacecraft Trajectory Planning with Avoidance Constraints Using Mixed-Integer Linear Programming". Journal of Guidance, Control, and Dynamics*, 25(4), pp.755-764. 2002.

[14] X. Liu, W. Hu and H. Zheng, "Fuzzy linear programming based radar subset selection for target localization in UAV system," *2016 CIE International Conference on Radar (RADAR)*, Guangzhou, 2016, pp. 1-5.

[15] M. Holender, Rakesh Nagi, M. Sudit and J. Terry Rickard, "Information fusion using conceptual spaces: Mathematical programming models and methods," *2007 10th International Conference on Information Fusion*, Quebec, Que., 2007, pp. 1-8.

[16] N. Atay and B. Bayazit. *"Mixed-Integer Linear Programming Solution to Multi-robot Task Allocation Problem".* Report Number: WUCSE-2006-54 (2006). *All Computer Science and Engineering Research.* https://openscholarship.wustl.edu/cse_research/205

[17] MATLAB. (2020). Retrieved from https://www.mathworks.com/products/matlab.html

[18] Kenny, V., Nathal, M., & Saldana S., "*Heuristic algorithms*", May 25, 2014. Retrieve from: https://optimization.mccormick.northwestern.edu/index.php?title=Heuristic_algorithms&oldid=981

[19] T.G. Robertazzi, "*Networks and Grids: Technology and Theory,*" Germany, Springer New York, 2007.

[20] F. Wu, T. Robertazzi and Y. Cao, "*Integrating Amdahl-like Laws and Divisible Load Theory*," arXiv 1902.01899, Feb. 5, 2019.

[21] G.M. Amdahl, "*Computer Architecture and Amdahl's Law,*" Computer, 2013, pp. 38-46.

[22] G.M. Amdahl, "*Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities,*" Proceedings of the AFIPS Conference, AFIPS Press, 1967, pp. 483-485.

[23] J.J. Gustafson, "*Reevaluating Amdahl's Law*," Communications of the ACM, vol. 31, no. 5, May 1988, pp. 532-533.

[24] F. Diaz-del-Rio, J. Salmeron-Garcia and J. Luis Sevillano, "*Extending Amdahl's Law for the Cloud Computing Era*," Computer, Feb. 2016, pp. 14-22.

[25] M. Holzrichter. "*An Application of the Constraint Programming to the Design and Operation of Synthetic Aperture Radars*". IEEE 978-1-4673-1576-0/12. 2012.

[26] E. Winter and L. Lupinski, "On Scheduling the Dwells of a Multifunction Radar," *2006 CIE International Conference on Radar*, Shanghai, 2006, pp. 1-4.

[27] T. G. Robertazzi. *"Planning Telecommunication Network"*. 1999.