Stripmap SAR Pulse Interleaved Scheduling

A Thesis Presented
by

John P. Terragnoli

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of

Master of Science

in

Electrical and Computer Engineering

Northeastern University
Boston, Massachusetts

April 2018
This research is dedicated to God, who helps me through all things, my family who provided me with my opportunities for education, and all mentors in Boston who taught me how to think critically and independently. A special thanks to Dr. Paul Monticciolo, Prof. Miriam Leeser, Dr. Gerald Benitz, and Prof. Waleed Meleis who were all patient with me when my questions seemed infinite.
Contents

List of Figures v
List of Tables vi
List of Acronyms vii
Nomenclature ix
Acknowledgments x
Abstract of the Thesis xi

1 Introduction 1
  1.1 Organization ......................................................... 2

2 Background 3
  2.1 Stripmap Synthetic Aperture Radar (SAR) .......................... 3
    2.1.1 Parameters of a Stripmap Task ................................ 3
  2.2 Scheduling Algorithms ............................................. 7
    2.2.1 Greedy Algorithm ............................................. 9
    2.2.2 Greedy Algorithm with Local Search ......................... 9
  2.3 Related Work ...................................................... 10

3 Methodology and Design 13
  3.1 The Goal ............................................................ 13
    3.1.1 Evaluation of Scheduler Performance ........................ 14
  3.2 Data Generation ................................................... 16
  3.3 Preprocessing ....................................................... 19
  3.4 Scheduling Algorithms .............................................. 21
    3.4.1 Greedy Algorithm ............................................. 21
    3.4.2 Greedy Algorithm with Local Search Algorithm .......... 23
    3.4.3 Multiple Input Order - Greedy with Local Search ........ 26
    3.4.4 Interleaved Scheduler ........................................ 29
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Experiments and Results</td>
<td>45</td>
</tr>
<tr>
<td>4.0.1 Schedule Graphics</td>
<td>48</td>
</tr>
<tr>
<td>5 Conclusions and Future Work</td>
<td>54</td>
</tr>
<tr>
<td>5.1 Conclusions</td>
<td>54</td>
</tr>
<tr>
<td>5.2 Future Work</td>
<td>54</td>
</tr>
<tr>
<td>Bibliography</td>
<td>55</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Stripmap SAR Mode</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of Stripmap SAR Image [1]</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>Grazing Angle</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>Image Formation with Multiple PRFs [2]</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Task Generation</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>Coordinate System</td>
<td>18</td>
</tr>
<tr>
<td>3.3</td>
<td>Window of Opportunity of a Task</td>
<td>22</td>
</tr>
<tr>
<td>3.4</td>
<td>Greedy Algorithm Complexity Analysis</td>
<td>24</td>
</tr>
<tr>
<td>3.5</td>
<td>Local Search Algorithm</td>
<td>25</td>
</tr>
<tr>
<td>3.6</td>
<td>Greedy with Local Search Algorithm Complexity Analysis</td>
<td>27</td>
</tr>
<tr>
<td>3.7</td>
<td>Multiple Input Orders</td>
<td>28</td>
</tr>
<tr>
<td>3.8</td>
<td>Greedy with Local Search and Multiple Inputs Algorithm Complexity Analysis</td>
<td>29</td>
</tr>
<tr>
<td>3.9</td>
<td>Interleaved Schedule Creation</td>
<td>30</td>
</tr>
<tr>
<td>3.10</td>
<td>Possible Start Times</td>
<td>32</td>
</tr>
<tr>
<td>3.11</td>
<td>Break Into Simple Cases</td>
<td>33</td>
</tr>
<tr>
<td>3.12</td>
<td>Process to Make Tasks Compatible</td>
<td>34</td>
</tr>
<tr>
<td>3.13</td>
<td>Get Common PRF from Allowable PRF Ranges</td>
<td>35</td>
</tr>
<tr>
<td>3.14</td>
<td>Separating Transmit Pulses and Return Envelopes</td>
<td>37</td>
</tr>
<tr>
<td>3.15</td>
<td>Output Energy Exceeded and then Fixed</td>
<td>39</td>
</tr>
<tr>
<td>3.16</td>
<td>Interleaved Scheduling Algorithm Complexity Analysis - Part 1</td>
<td>41</td>
</tr>
<tr>
<td>3.17</td>
<td>Interleaved Scheduling Algorithm Complexity Analysis - Part 2</td>
<td>42</td>
</tr>
<tr>
<td>3.18</td>
<td>Interleaved Scheduling Algorithm Complexity Analysis - Part 3</td>
<td>42</td>
</tr>
<tr>
<td>3.19</td>
<td>Making $N$ Task Pieces Compatible in a Single Set Complexity Analysis - Part 1</td>
<td>43</td>
</tr>
<tr>
<td>3.20</td>
<td>Making $N$ Task Pieces Compatible in a Single Set Complexity Analysis - Part 2</td>
<td>44</td>
</tr>
<tr>
<td>3.21</td>
<td>Making $N$ Task Pieces Compatible in a Single Set Complexity Analysis - Part 3</td>
<td>45</td>
</tr>
<tr>
<td>4.1</td>
<td>Greedy Algorithm with Local Search and Multiple Input Order - Score vs Execution Time for varying $N_p$ values</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Greedy Algorithm Results - Base Case Benchmark - Test 1</td>
<td>50</td>
</tr>
<tr>
<td>4.3</td>
<td>Greedy Algorithm with Local Search Results - Base Case Benchmark - Test 1</td>
<td>51</td>
</tr>
<tr>
<td>4.4</td>
<td>Greedy Algorithm with Local Search and Multiple Input Orders with $N_p = 5$ Results - Base Case Benchmark - Test 1</td>
<td>52</td>
</tr>
<tr>
<td>4.5</td>
<td>Interleaved Schedule Results - Base Case Benchmark - Test 1</td>
<td>53</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Given Tasks - Sample .................................................. 15
3.2 Created Schedule - Sample .......................................... 15
3.3 Score of Schedule - Sample .......................................... 15
3.4 Determining Best Schedule - Sample .............................. 15

4.1 Mission Constants for Various Situations ........................ 46
4.2 Scores of Different Schedulers - Base Case ...................... 46
4.3 Execution Times for Different Schedulers ......................... 47
4.4 Scores of Interleaved Schedule in All Situations ............... 48
4.5 Interleaved Scheduler Execution Times and Peak Interleaved Tasks .... 48
List of Acronyms

**CPU** Central Processing Unit The main computing component in a computer. Most operations use the CPU in one way or another.

**DF** Duty Factor The percentage of time in the PRI that the radar is actively transmitting.

**GMTI** Ground Moving Target Indicator Type of radar mode which uses multiple PRFs and Doppler ambiguities to locate and track moving targets in a given area of land.

**JSTARS** Joint Surveillance Target and Attack Radar System This is a air frame in the United States Air Force inventory that contains a massive radar. It is used to collect intelligence.

**PRF** Pulse Repetition Frequency The number of pulses transmitted from the radar every second.

**PRI** Pulse Repetition Interval Inverse of the PRF. The amount of time in between each radar pulse transmitted by the radar.

**SAR** Synthetic Aperture Array A form of radar dependent on a moving radar platform. Pulses are transmitted to and received from a target from many points as the platform moves along a line adjacent to the target. The pulses are combined in such a way that the resolution of the image is greater than if the image was taken while the radar was stationary.

**SNR** Signal to Noise Ratio The power of the signal divided by the power of the noise.

**TA** Target Area Plot of land on the ground in which contains all of the stripmap SAR tasks.

**WOO** Window of Opportunity Time frame in which a task may execute.
Nomenclature

Flight Parameters

\( h \)  Aircraft Altitude (m)
\( v \)  Aircraft Velocity (m/s)

Other Parameters

\( TA_x \)  Target Area Length (m)
\( TA_y \)  Target Area Width (m)
\( TA_{x_{\text{max}}} \)  Target Area Maximum X Coordinate (m)
\( TA_{x_{\text{t_{\text{max}}}}} \)  Target Area Maximum Length (m)
\( TA_{y_{\text{max}}} \)  Target Area Maximum Y Coordinate (m)
\( TA_{y_{\text{min}}} \)  Target Area Minimum Y Coordinate (m)
\( TA_{y_{\text{t_{\text{max}}}}} \)  Target Area Maximum Width (m)

Physics Constants

\( c \)  Speed of Light \( 2.98 \times 10^8 \) (m/s)

Radar Parameters

\( \lambda \)  Carrier Wavelength (m)
\( \phi_{\text{az}} \)  Azimuth/Cross Range Beam Width (deg)
\( \phi_{\text{rg}} \)  Range Beam Width (deg)
\( \psi \)  Grazing Angle (deg)
\( \tau \)  Pulse Width (s)
\( \tau_{i_{\text{BR}}} \)  Beginning of the Return Envelope of Task \( i \) (s)
\( \tau_{i_{\text{ER}}} \)  End of the Return Envelope of Task \( i \) (s)
$\theta$  Squint Angle (deg)

$DF$  Duty Factor

$DF_{\text{max}}$  Maximum Duty Factor

$E_{\text{max}}$  Maximum Output Energy in Window (J)

$f_c$  Carrier Frequency (Hz)

$L_{\text{rad}}$  Length of Radar (m)

$N_x$  Number of Tasks to Reorder in Multiple Input Scheduler

$P_{\text{max}}$  Maximum Power of Radar (W)

$PFR$  Pulse Repetition Frequency

$PRI$  Pulse Repetition Interval

$R$  Distance from Radar to Target (m)

$R_{i,\text{max}}$  The farthest distance from Task $i$ with maximum $\theta$ (m)

$R_{i,\text{min}}$  The closest distance from Task $i$ with $\theta = 0$ (m)

$SNR$  Signal to Noise Ratio

$T_e$  Time of Energy Window (s)

$T_i$  Duration of Task $i$ (s)

$W_{\text{rad}}$  Width of the Radar (m)

$WOO_{i,\text{beg}}$  Beginning of the Window of Opportunity in Time from the Origin for Task $i$ (s)

$WOO_{i,\text{end}}$  End of the Window of Opportunity in Time from the Origin for Task $i$ (s)

$x_i$  Starting X Coordinate of Task $i$ (m)

$x_{it}$  Width (in Range) of Task $i$ (m)

$y_i$  Starting Y Coordinate of Task $i$ (m)

$y_{it}$  Length (in Cross Range) of Task $i$ (m)
Acknowledgments

First and foremost, I would like to thank God for blessing me with the education and mentors necessary to complete this research. I know that everything good comes from Him, and it has been a pleasure to be His servant in Boston these last few years.

This research would not have been possible without the expertise and patience of several advisers along with the support of my family and friends. Mr. Matthew Alexander introduced me to the concept of pulse level interleaving and sparked my interest. He provided me with the resources and direction to pursue this project and turn it into a reality. My advisers, Prof. Miriam Leeser and Dr. Paul Monticciolo kept me on track during regular meetings. Their corrections were always directive and encouraging. I would like to thank Prof. Meleis for working with me to combine the disciplines of radar operation and scheduling. It helped me learn the language and concepts necessary to fit my research into both academic communities. Last but not least, I would like to thank Dr. Gerald “Jerry” Benitz for spending countless hours in front of his white board with me while I struggled to understand and apply a plethora of radar concepts that were new to me. This research would not have been nearly as realistic without your help.

My parents never stopped encouraging me, and my roommates, Brian, Sabin, and Dan, were great at listening and providing feedback on all of my ideas. I am going to miss all of them when I move from Boston.
Abstract of the Thesis

Stripmap SAR Pulse Interleaved Scheduling

by

John P. Terragnoli

Master of Science in Electrical and Computer Engineering

Northeastern University, April 2018

Prof. Miriam Leeser, Advisor

Stripmap SAR is a radar mode used to image terrain from an airborne platform; it does so by transmitting and receiving a series of electromagnetic pulses. Pulse interleaving attempts to execute two or more stripmap tasks simultaneously by transmitting pulses for additional tasks while waiting for the pulses from other tasks to return. A task is simply an area on the ground to image. Other research has done this by dividing the front-end of the radar into separate sections and aiming the energy into different beams, each pointing in a different direction and devoted to a separate area to image. This thesis identifies a method for creating a schedule using pulse interleaving on only one radar beam when given a set of stripmap tasks. Scheduling is not done in real time, but is instead done before the schedule will be executed. Interleaving is performed on two tasks overlapping in execution time in the following way. The PRFs of the two tasks are altered within their allowable limits so that they match each other. Then, the transmitted pulses and return envelopes, the times when the pulses might return based on the aircraft’s distance to the target and the target’s dimensions, are separated temporally by adding delays to one or both of the tasks’ transmitted pulses. Doing this removes the possibilities of transmitting a pulse for one task while receiving the pulse from another task, needing to transmit for two tasks at the same time, or receiving pulses from two different tasks at the same time. This process is easily extrapolated to more than two tasks which are scheduled to execute in the same block of time. To compare the results of the interleaved scheduler, several greedy algorithms which do not utilize pulse interleaving were also created. In every case, the interleaved scheduler outperformed the greedy algorithms. Several situations were created to imitate different flight conditions as well as radars with varying duty factor limits and output power capability. The performance of the interleaved scheduler was consistent in all situations. This research demonstrates the benefits of pulse level interleaving when scheduling multiple stripmap SAR tasks in a short period of time.
Chapter 1

Introduction

Stripmap Synthetic Aperture Radar (SAR) is a method of utilizing radar pulses emitted and received from a moving platform to produce an image. Generally, it is useful for imaging large areas at relatively coarse resolution \[3\]. It is the same technique used when satellites map the surface of the Earth. Stripmap SAR is one of the most simple imaging techniques using radar, which makes it ideal for exploring pulse level interleaving.

The idea of this research is to utilize the time when the radar is waiting for pulses to return. When executing a single task, most of the radar’s time is spent waiting for these return pulses, not actually transmitting or receiving them. Transmitting pulses for additional tasks while waiting for the return pulses from other tasks utilizes more of the radar’s time for data collection. Often these radar platforms are carried by aircraft, which have a limited amount of fuel and are expensive to operate. The more information that can be collected in a single flight, the better.

First, this research imitates the current scheduling process on a well known airborne radar platform, the Joint Surveillance Target Attack Radar System (JSTARS). The JSTARS is an aircraft made for the collection of military intelligence in the United States Air Force (USAF). Any aircraft could be used, but the JSTARS was chosen as a basis and to ensure that the parameters used were realistic. The current scheduling process is not exactly known. However, it does not utilize pulse level interleaving and is imitated with a simple greedy algorithm in which there are priorities, durations, and windows of opportunity for each image to be taken and only one image can be taken simultaneously \[4\]. Various techniques are performed to improve upon the greedy algorithm, still without pulse interleaving. The purpose is to attain a very good schedule without interleaving for comparison so that the benefits of pulse interleaving can be seen plainly. Pulse level interleaving is the final technique applied, and the results from using this technique are compared with all the
CHAPTER 1. INTRODUCTION

greedy algorithms used.

1.1 Organization

This thesis is organized in the following way. Chapter 2 gives the background necessary to understand this research. Chapter 3 describes in detail how the four schedulers operate. Chapter 4 gives a summary of all tests performed and an analysis of the results. The radar algorithms were all tested in a single flight situation as a base case and the interleaved scheduler was tested on seven different flight situations, including the base case. Final conclusions are presented in Chapter 5.
Chapter 2

Background

2.1 Stripmap Synthetic Aperture Radar (SAR)

Stripmap SAR works in the following way. The beam of the radar points at the ground in a fixed direction with respect to the aircraft. As the aircraft moves, the beam “drags” across the ground and creates an image of the area that it drags over. This process is shown in Figure 2.1a. In reality, the radar is emitting many separate pulses every second, but the movement can be thought of as dragging the beam across the ground. An overhead view of stripmap SAR can be seen in Figure 2.1b. Note that in each of the stripmap cases in Figure 2.1b, the squint angle remains constant while imaging the entire blue area. The squint angle is defined as \( \theta \), which is the angle between the antenna boresight and the line normal to the flight path. The boresight is the direction that the radar energy is being emitted in; it is the center line of the yellow, balloon like structures in Figure 2.1b. All of the returning pulses are combined using digital signal processing techniques to create an image. These images are typically monochrome; an example can be seen in Figure 2.2.

2.1.1 Parameters of a Stripmap Task

In order to completely characterize a stripmap SAR task several constraints and parameters need to be set. For each image request, there will be a starting point, length/width of the area to be imaged, pulse repetition frequency (PRF), squint angle, grazing angle, and a priority. The following sections explain radar terminology having to do with these parameters and provides the background for how a single stripmap SAR task can change its PRF mid-execution.
CHAPTER 2. BACKGROUND

Figure 2.1: Stripmap SAR Mode

(a) Sideview

(b) Overhead View

Figure 2.2: Example of Stripmap SAR Image
CHAPTER 2. BACKGROUND

2.1.1.1 Dimensions of Area to Image

The dimensions of an area to image is defined by a starting point, a task width, and a task length. The starting point on the ground is where the aircraft will begin taking the image. The length of the task can be as long as desired, as the aircraft moves parallel to the area to be imaged. As long as the aircraft keeps flying, it can keep imaging. The width of the task is limited by the range beamwidth of the radar \( \Delta \phi_{rg} \), the formula for which is shown in Equation 2.1. This is the equation for a uniformly illuminated rectangular aperture or linear array, which is what is assumed in this research.

\[
\Delta \phi_{rg} = 50.8 \frac{\lambda}{W_{rad}} \tag{2.1}
\]

The variable \( \lambda \) is the carrier wavelength of the radar. \( W_{rad} \) is the width, or the height of the radar. Based on the JSTARS platform, this was assumed to be 1 m. \( \Delta \phi_{rg} \) is calculated in degrees. Depending on the altitude and grazing angle, the width of the areas to image on the ground is limited by Equation 2.1. Notice how in Figure 2.1a the ground area is stretched based on the angle that the radar energy has with the ground. Since the \( \Delta \phi_{rg} \) is constant, the higher the aircraft is and the smaller the grazing angle, the larger area it can cover on the ground. This relationship is discussed in greater detail later on and is characterized by Equation 3.4, when the widths of tasks are generated to produce testing data.

2.1.1.2 Pulse Repetition Frequency (PRF)

The PRF is a flexible parameter of a stripmap SAR task. There are two basic limitations, Doppler ambiguity and range ambiguity. Doppler ambiguity is responsible for the lower PRF limit and range ambiguity is responsible for the upper PRF limit. The Doppler limit is defined by the Nyquist rate and can be seen in Equation 2.2, where \( v \) is the aircraft velocity and \( L_{rad} \) is the length of the radar.

\[
PRF \geq \frac{2v}{L_{rad}} \tag{2.2}
\]

Most radar developers do not design SAR close to this limit. To accommodate for this, the coefficient in Equation 2.2 was changed from 2 to 3 in this research for a margin of safety. The range ambiguity limit is defined so that the last transmitted pulse has a chance to return before the next pulse is sent out. Therefore, this limit is largely defined by how far away the target is, as well as the
duty factor ($DF$) of the radar, as shown in Equation 2.3. \(c\) is the speed of light, \(R_{i_{max}}\) is the distance between the aircraft and the farthest point on the land to be imaged.

\[
PRF < (1 - DF) \frac{c}{2R_{i_{max}}}
\]

Combining Equations 2.2 and 2.3, the range of PRFs for a given stripmap SAR task is given in Equation 2.4. With stripmap SAR, there is significant liberty with the PRF choice, although, the PRF that is used does affect a number of other parameters, such as the pulse width. This will be discussed later.

\[
\frac{3v}{L_{rad}} \leq PRF < (1 - DF) \frac{c}{2R_{i_{max}}}
\]

### 2.1.1.3 Squint Angle

The squint angle, written as \(\theta\), is a parameter that allows the operator to initiate a stripmap SAR image slightly before or after the aircraft is adjacent to the starting point of the imaging area. Without using a squint angle, the radar would only be able to take an image when it is exactly perpendicular to the target. A graphical example of the squint angle can be seen in Figure 2.1b. There are practical limits on the squint angle and for this experiment squint angle limits are set to be ±15°, although it can be set to the parameters of whatever radar is used on the aircraft. This allows flexibility on when the start time for the task can be set by the scheduler [3].

### 2.1.1.4 Grazing Angle

The grazing angle is denoted as \(\psi\). It is the angle between the horizon and where the radar beam is pointing. There are typically limits to how small or large the grazing angle can be based on the specifications of the radar and the flight conditions. In this research, the limits are set to be 15° and 35°. Figure 2.3 shows exactly what the grazing angle is.

### 2.1.1.5 Multiple PRFs in a Single SAR Task

Traditionally, SAR tasks use a single PRF throughout the collection since there is no need for it to change [11]. However, this is not necessary. Research has shown that any distortions caused by changing the PRF mid-execution can be removed with digital signal processing [2]. This is done by weighting each returning pulse by the surrounding time interval and performing a convolution of the returning pulses with an identical image response function. Image formation is then done using
polar re-sampling. This process is shown in Figure 2.4. While this specific research only tested SAR tasks with two PRFs, the conclusions of Dr. Gerald Benitz, who conducted the research, indicate that this same process is likely to work if many different PRFs are used in the execution of a single SAR task \([2]\). This thesis will assume that the radar used is capable of implementing this technique so that the PRF can be changed throughout any SAR task.

### 2.2 Scheduling Algorithms

To approach the problem of scheduling a given set of tasks for a flight, multiple different scheduling approaches are considered. The four algorithms used are a greedy algorithm, a greedy algorithm with local search improvement, a greedy algorithm with local search using multiple task input orders, and an interleaved scheduler. In all cases, the scheduling is done offline, meaning that results are not needed immediately but should be found in a reasonable amount of time. The succeeding sections provide background on greedy scheduling, local search techniques, and the essentials for pulse level interleaving.
CHAPTER 2. BACKGROUND

(a) SAR Pulse Interpolation Method

Figure 2.4: Image Formation with Multiple PRFs [2]
2.2.1 Greedy Algorithm

This section gives a generic overview of greedy algorithms. Being the most rudimentary scheduling approach, it is easy to implement and yields a starting point for further improvement. Any greedy algorithm seeks to maximize the short-term gain without considering the ramifications in the long-term [12]. In all greedy algorithms, there must be a metric of success, known as an objective function, so that the performance of the greedy algorithm can be measured [12]. A simple example is something known as the knapsack problem.

In the knapsack problem a user is given a task of packing a knapsack so that it is as heavy as possible [12]. Here, the final weight of the knapsack would be the objective function, which the user is trying to maximize. There is a limited amount of space in the backpack. A greedy approach, which might not yield the best results, would be to start with the heaviest objects and then work to the lightest objects. If there is no room for an object, then the user will no longer consider that object, and move on to the next object to try and place it in the knapsack. The process will continue until each object has been considered once. Greedy algorithms are known for being fast but giving sub-optimal results.

2.2.2 Greedy Algorithm with Local Search

A local search algorithm “works in an iterative fashion by successively replacing the current solution by a better solution in the neighborhood of the current solution” [13]. After applying the greedy algorithm to come up with a simple solution, the user can then look at possible avenues for improvement. This might be taking out and replacing objects currently in the knapsack for objects that will fit better and provide more weight to the backpack. This local search algorithm could be applied to every object in the backpack, with the hope of making the knapsack just a little heavier than in the previous solution.

2.2.3 Pulse Level Interleaving

This section contains a brief description of how pulse level interleaving works in the simplest of scenarios. Imagine that there are two tasks that are perfectly compatible to interleave. The compatibility requirements will be described in detail later on. Figure 2.5a shows the transmitted and received pulses for Tasks 1 and 2. The pulses for the first task are orange, and the pulses of the second task are blue. The tall, darker colored bars represent the transmitted pulses, and the lighter shaded bars represent the return pulses, which are simply the transmitted pulses after they have gone
CHAPTER 2. BACKGROUND

(a) Pulses of Two Tasks - No Interleaving

(b) Pulses of Two Tasks - Interleaving

out to the target and bounced back to the radar. Each of these tasks only sends out two pulses. In Figure 2.5a, Task 2 is scheduled to execute after all the pulses in Task 1 have been transmitted and received. Assuming that none of either the transmitted nor received pulses can overlap, for reasons explained later, notice that while the radar is waiting for the return of a pulse from Task 1, it can transmit and wait for the return of a pulse of Task 2. This is interleaving, utilizing the waiting time for the return pulses to run other tasks. A graphical view of interleaving can be seen in Figure 2.5b. Obviously, it takes much less time to execute both tasks when interleaving is used. Theoretically, this can be done with more than two tasks.

2.3 Related Work

The struggle for maximizing the effectiveness of radar without additional hardware is continuous. Although this is a radar problem, it can also be considered as a scheduling problem. A greedy algorithm and local search improvement algorithm are used in this research as a basis, and then interleaving is done to improve the produced schedules. Other contemporary scheduling methods include earliest deadline first and shortest job first [14], [15], [16]. The list of scheduling techniques is seemingly endless, but many of these techniques are directed towards scheduling jobs in computer CPUs. These ideas have their place, but they do not take advantage of key principles which are distinct to radars and not CPUs. They could be used in this research, but after initial examination, they were not thought to be the most valuable use of time. In the end, it is more promising to execute
two tasks at once than to cram more tasks into the extra space of a schedule where only one task may execute at a time. Thus, that is why pulse level interleaving was explored.

This research is not the first to explore pulse level interleaving. Others have worked with this technique, although much of that has been directed towards the improvement of GMTI radar modes [17], [15]. GMTI stands for ground moving target indicator and this type of radar mode operates much differently than SAR modes. GMTI uses the Doppler shift of the returning pulses to determine the location and relative motion of objects, while SAR imitates the aperture of an impossibly large radar in order to take an image of the terrain with fine resolution [18]. However, both GMTI and SAR can be performed by the same type of radar, no distinct hardware is needed. The hope for this research is that if SAR could also be scheduled with pulse level interleaving, then it could be possible to create a program which could schedule GMTI and SAR tasks in an interleaved fashion without compromising the quality of either using the same radar beam. There has been some research to use pulse level interleaving with SAR and GMTI modes. However, these techniques have utilized multiple beams being emitted from the radar [2]. This involves splitting the radar into distinct sections and directing the energy from the separate sections in different directions. It is a useful technique, but this thesis focuses on interleaving pulses emitted from the same beam. If this is possible, then there is no reason the technique could not be used on all of the beams of the radar, multiplying its effectiveness.

Currently, many active radar platforms do not utilize pulse level interleaving for SAR tasks. Many of these platforms just use first come, first serve algorithms for scheduling radar tasks [4]. This means an arriving task will be fit into the schedule if the requested time slot is available or if it is a higher priority task than whatever is in its way. For this research, this scheduling technique is implemented as a greedy algorithm. To perform the greedy algorithm, the duration, priority, and window of opportunity (WOO), the time frame in which the task can be scheduled, are the only parameters given. The program does not know the details of how a radar task will operate. Therefore, it cannot do much to alleviate overlaps except to just remove one of the overlapping tasks. The interleaved scheduler, as promoted by Lee and Shih, requires a much greater knowledge of how the modes are actually operating and how they can be changed while still satisfying the requirements set by the user [17]. This knowledge may include the energy constrains of the system, knowing the PRF of the task at hand, and being allowed to alter the delay and duration of the pulses being sent.

This thesis builds upon Lee and Shih’s approach and even allows for the PRF to be changed using the same formulas used by those who create SAR tasks. Instead of all of the parameters being set, the original parameters are given, as well as the requirements specified by the user. If the program
CHAPTER 2. BACKGROUND

needs to alter some of a task’s parameters to execute it at a certain time, it is allowed to do so as long as the requirements of that task are still met.
Chapter 3

Methodology and Design

This chapter describes the objective function used in this research, explains how the results of different algorithms are compared, shows how test data is generated and preprocessed, specifies how each of the four algorithms work, and performs a complexity analysis of each algorithm.

3.1 The Goal

In order to collect information on a target or area, a radar system must successfully transmit and receive a specified number of pulses and then process the pulses when they return. Taking an image of a specified area of land is called a task. To execute a single task, the directions on how to perform the task are sent to the actual radar hardware where the pulses are transmitted and received. If the human operators would like to execute two tasks, they must schedule Task 1 to go first and then have Task 2 execute after Task 1 is finished. Obviously, human operators would like to execute many more than just two tasks. A problem arises with how to schedule all of these tasks so that they do not interfere with each other, especially because each task has a time frame in which it must execute and the time frames of different tasks could overlap. The purpose of this research is to develop a good autonomous scheduler using pulse level interleaving. This process is subject to a set of system-level constraints to create a schedule of execution for the given tasks. The schedule is created before run-time, so execution time does not have to be very small, it only needs to be reasonable and practical for planning surveillance missions.

The purpose of the scheduler is to take a list of all the tasks human operators would like to execute and convert it into a schedule designating which task will be executing at what time. For each task the inputs are: its duration, priority, and the time frame in which it is allowed to be scheduled,
CHAPTER 3. METHODOLOGY AND DESIGN

called the window of opportunity (WOO). This is all the information needed for the three greedy algorithms used. More task parameters are needed to design a schedule with pulse interleaving techniques, but this will be discussed much later in the paper. A scheduler must take a set of tasks and their associated parameters and create a good schedule. The method for determining the quality of a schedule is described next.

3.1.1 Evaluation of Scheduler Performance

To find a good scheduler for a specific set of tasks, there must be a method of “scoring” the schedules created. As an input, a scheduler will receive a set of tasks. Each task in the set will have certain parameters, including how long it will take to execute the task, the WOO, and a priority. As an output, the scheduler will return the set of tasks with assigned start and stop times for each task, indicating the exact times when each task is planned to execute. Ideally, a scheduler will maximize the amount of information collected in a given amount of time. The simplest way to do this is to sum the duration each task takes to execute. This is derived from the assumption that the longer the radar is operating on a target, the more information it is collecting. However, due to the priorities, not all tasks can be treated the same. The score cannot be just the sum of all task durations if time spent on some tasks is more valuable than time spent on other tasks. A schedule with 100 seconds of priority 10 tasks would be a better result than a schedule with 120 seconds of priority 9 tasks. Thus, the final score of a schedule is the sum of the durations of each task at each priority level. Consider the following example. Table 3.1 shows a list of tasks that the user would like to schedule. Table 3.2 shows the schedule that is produced by the program given the tasks in Table 3.1. Only two priority 10 tasks, one priority 8 task, and two priority 4 tasks made it into the schedule. The resulting score is shown in Table 3.3. The number 80 is the sum of the durations of Tasks S7 and S9, which are 20 and 60 seconds, respectively. This same process is applied to the priority 4 tasks S3 and S5, where $25 + 100 = 125$ seconds.

In order to decide which schedule is the best produced, the scores are compared in the following way. The schedule with the highest duration of priority 10 tasks scheduled is the best. If there is a tie between two schedules, the duration of priority 9 tasks is the tie breaker. If there is another tie, then this process continues with the priority 8 task duration and so forth. In Table 3.4, the scores of four different schedules are given, in descending order from the “best” to the “worst” schedules. Notice that Schedule 2 has a much greater overall duration than Schedule 1, but Schedule 1 is scored higher because of the durations at higher priorities.
CHAPTER 3. METHODOLOGY AND DESIGN

<table>
<thead>
<tr>
<th>Task ID</th>
<th>WOO Beginning</th>
<th>WOO End</th>
<th>Duration</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>36</td>
<td>468</td>
<td>356</td>
<td>8</td>
</tr>
<tr>
<td>s2</td>
<td>120</td>
<td>198</td>
<td>45</td>
<td>7</td>
</tr>
<tr>
<td>s3</td>
<td>400</td>
<td>510</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>s4</td>
<td>230</td>
<td>369</td>
<td>78</td>
<td>3</td>
</tr>
<tr>
<td>s5</td>
<td>440</td>
<td>648</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>s6</td>
<td>300</td>
<td>417</td>
<td>49</td>
<td>8</td>
</tr>
<tr>
<td>s7</td>
<td>560</td>
<td>696</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>s8</td>
<td>490</td>
<td>720</td>
<td>110</td>
<td>2</td>
</tr>
<tr>
<td>s9</td>
<td>590</td>
<td>780</td>
<td>60</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3.1: Given Tasks - Sample

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Start Time</th>
<th>Duration</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>36</td>
<td>356</td>
<td>8</td>
</tr>
<tr>
<td>s3</td>
<td>400</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>s5</td>
<td>440</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>s7</td>
<td>560</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>s9</td>
<td>590</td>
<td>60</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3.2: Created Schedule - Sample

<table>
<thead>
<tr>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 9 8 7 6 5 4 3 2 1</td>
</tr>
<tr>
<td>Score</td>
</tr>
<tr>
<td>80 0 356 0 0 125 0 0 0</td>
</tr>
</tbody>
</table>

Table 3.3: Score of Schedule - Sample

<table>
<thead>
<tr>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 9 8 7 6 5 4 3 2 1</td>
</tr>
<tr>
<td>Schedule 1 Score</td>
</tr>
<tr>
<td>70 180 0 0 0 24 0 0 2</td>
</tr>
<tr>
<td>Schedule 2 Score</td>
</tr>
<tr>
<td>70 175 80 44 86 0 0 0 20</td>
</tr>
<tr>
<td>Schedule 3 Score</td>
</tr>
<tr>
<td>40 53 1 57 0 0 0 18 2</td>
</tr>
<tr>
<td>Schedule 4 Score</td>
</tr>
<tr>
<td>40 53 0 60 0 21 0 0 2</td>
</tr>
</tbody>
</table>

Table 3.4: Determining Best Schedule - Sample

The list is in descending order, from the best schedule in the top row to the worst schedule on the bottom row.
CHAPTER 3. METHODOLOGY AND DESIGN

3.2 Data Generation

One of the first tasks in evaluating various scheduling algorithms is to create the test data to operate on. That is, to generate a set of feasible, realistic, stripmap SAR tasks that could be processed into a schedule. A single task is a request to image a particular area using stripmap SAR. A single set of 100 tasks is called a test, and a collection of 1,000 independent tests is called a benchmark. The performance of a scheduler is determined by the average score of the 1,000 schedules produced from a benchmark. In many airborne surveillance systems, the aircraft will fly parallel to the target area, back and forth, for the duration of its mission. This is called a race-track, as shown in Figure 3.2. This research does not consider data collection when the aircraft is turning, because that would require additional calculations to ensure that the SAR images are not distorted. Therefore, the mission can be divided up into discrete, straight flight paths equal to the length of the target area. For this experiment, the length of the target area was set to be 160 km, which is approximately 12 minutes of flight time when traveling at 230 m/s, an average velocity for surveillance aircraft. This length could easily be changed to whatever the user desires, and then the test data would be randomly generated within that boundary.

These tasks could not be generated in a completely random manner, since each of the parameters of a stripmap SAR task are interdependent. Randomly generating all of the parameters of a radar task would be like randomly generating all the pieces of a car; it is likely that they will not fit together and function! Rather, some mission constants are set and then a select few task parameters are chosen to be randomly generated. These realized values then set constraints on how the remaining parameters could be generated, or in some cases the remaining parameters are calculated exactly using various radar formulae. This process is shown in Figure 3.1. In all flowcharts, circles represent data and square objects represent processes.

Now the flowchart in Figure 3.1 will be examined more thoroughly. For a test, or a set of 100 tasks to be scheduled, the mission constants are the maximum duty factor ($DF_{\text{max}}$), maximum squint angle ($\theta_{\text{max}}$), carrier frequency ($f_c$) minimum and maximum grazing angle ($\psi_{\text{min}}, \psi_{\text{max}}$), aircraft altitude and velocity ($h, v$), width and length of the radar ($W_{\text{rad}}, L_{\text{rad}}$), minimum and maximum stripmap task length and width (distance on the ground) ($x_{\text{tmin}}, x_{\text{tmax}}, y_{\text{tmin}}, y_{\text{tmax}}$), maximum target area length and width ($T A_{x_{\text{tmax}}}, T A_{y_{\text{tmax}}}$), and the peak power ($P_{\text{max}}$) of the radar. All of these values are set by the radar operator for a specific flight. Then the width of the target area is defined using the coordinate system in Figure 3.2. The target area is the section of land where all the tasks are located. The maximum width of the target area is based off of the minimum and
maximum grazing angles, and is shown in Equations 3.1, 3.2, and 3.3. The boundaries on the X axis are based off of the origin, in the bottom left of Figure 3.2, and how far the aircraft is going to fly. The aircraft flies along the X axis starting from the origin and ending once it has flown the length of the target area. In this research $TA_{x_{max}}$ is set to be 160 km, although it can be set to any desired length.

$$TA_{y_{min}} = h \cdot \tan(90^\circ - \psi_{max})$$ \hspace{1cm} (3.1)

$$TA_{y_{max}} = h \cdot \tan(90^\circ - \psi_{min})$$ \hspace{1cm} (3.2)

$$TA_{y_{t_{max}}} = TA_{y_{max}} - TA_{y_{min}}$$ \hspace{1cm} (3.3)

The starting coordinates are then randomly generated for a task; this point must be within the target area. Based on this starting point, the grazing angle ($\psi$) is calculated assuming a squint angle ($\theta$) of zero. The grazing angle, range beam width, and aircraft altitude are used to calculate the maximum task width (in the Y direction), as seen in Equation 3.4. This is the upper limit for the task’s width along the ground. The minimum task width is given as a mission constant, 0.5 km. A task width is randomly selected between these two limits. The length of the task is randomly generated between 8 and 96 km, but these limits can be set arbitrarily.

$$x_{i_{t_{max}}} = h(\tan(90 - (\psi - (\phi_{rg} / 2))) - \tan(90 - (\psi + (\phi_{rg} / 2))))$$ \hspace{1cm} (3.4)
CHAPTER 3. METHODOLOGY AND DESIGN

Figure 3.2: Coordinate System

The PRF is then chosen as the average between the upper and lower PRF limits described in Equation 2.4. The pulse width is calculated by generating a random duty factor between 2.5% and 3.5% for each task. The duty factor is the percentage of time in the PRI that the radar is actively transmitting pulses. Once this duty factor is set, the pulse width can be calculated from the PRI, as in Equation 3.5. The PRI is simply the inverse of the PRF, as shown in Equation 3.6.

\[ \text{DF} = \frac{\tau}{\text{PRI}} \quad (3.5) \]

\[ \text{PRI} = \frac{1}{\text{PRF}} \quad (3.6) \]

When calculating the pulse width, there are two additional limits which must be checked before finalizing the task. These limits are the maximum duty factor of the radar and the maximum energy that the radar can output in a given time window. Because the duty factors of specific tasks are randomly generated between 2.5% and 3.5% and the maximum duty factor the radar can operate at is at least 10% in this thesis, it is impossible for a single task to violate the maximum duty factor of the radar. A duty factor violation could happen once interleaving occurs, but not when verifying individual tasks during data generation. This simple check is included as a reminder in case test data is generated using a different method later on. However, it could be possible to violate the maximum energy output, depending on the PRF chosen. The energy constraint is similar to the duty
factor limitation, but concerns the amount of energy that is being output without allowing the radar a break in transmission. It concerns the amount of time transmission is occurring, not necessarily the percentage of PRI. Simply, the amount of energy output over any given period of time must be below a given energy threshold [15]. For this experiment, the energy constraint used was $33.75\text{mJ}$ over $50\mu\text{s}$. This assumes that all tasks will be operating at the maximum power when outputting pulses. These energy constraints can be easily changed and applied to the experiment. These values are chosen based on the following case. If the PRF is 2000, the time to transmit a pulse takes up 7.5% of the PRI, and the power output is 900W, then the energy output for this pulse would be $33.75\text{mJ}$ [19]. This would occur over a time of $\frac{1}{2000\text{Hz}} \times 0.075 = 37.5\mu\text{s}$. The time window to check for the energy constraint was set to be slightly above this, at $50\mu\text{s}$, so that if more than $33.75\text{mJ}$ over $50\mu\text{s}$ is output, the energy constraint is violated. In practice, these values will be limited by hardware design, but the exact values for some of these constraints were not given, so a best estimate was made. Again, it would be simple to change either or both the energy constraint and the length of the window.

On occasion, the pulse width generated violates the maximum output energy constraint. If this is the case, the task in violation is regenerated. To do this, the program begins again at the “Randomly Generate Starting Coordinates” box in Figure 3.1. These failures rarely occur.

Lastly, a priority between 1 and 10 is assigned to the task, with priority 10 being the highest priority. Tasks with a higher priority will be considered first to be placed in the schedule. Note that many more tasks are generated for a single test than could be reasonably fit into a single schedule. The reason for this is to test and compare the performance of several schedulers by making it unlikely that any scheduler would perfectly schedule all given tasks into a timeline.

### 3.3 Preprocessing

More parameters of the tasks, aside from what was generated in Section 3.2, need to be calculated in order to apply scheduling algorithms. The values ideal for scheduling in this research are often not specified by radar operators. Typically, only the mission constants and the task parameters generated (see Section 3.2) are given by the radar operator; any other values needed for scheduling must be calculated from what is given. This is called preprocessing.

In this research, the schedules are created by assigning the exact times when each task will execute. In order to do this, some of the task parameters needed to be considered in time rather than in distance and physical location. The beginning of time is when the aircraft departs the origin and
heads down the X axis in Figure 3.2. The aircraft starts at this point and flies in a perfectly straight line until it has finished flying by the target area, in which case the test is complete. Each task will have a duration, or the amount of time needed to complete the task regardless of when it is scheduled. This is shown in Equation 3.7:

\[
duration_i = \frac{T_{ti}}{v} \tag{3.7}
\]

Additionally, there is a specific time frame in which the task may be scheduled. The aircraft must be in the vicinity of the task in order to capture an image. The aircraft must be nearly adjacent to the task so that the maximum squint angle would not be violated when transmitting pulses from the aircraft’s position. Thus, because the aircraft is flying in a perfectly straight line from the origin, where time starts at zero, there is a time window in which the task may be scheduled. The times at which the aircraft is in an acceptable position is called the window of opportunity (WOO). Figure 3.3 portrays the WOO. Because the aircraft velocity is known, the WOO is defined by the times when the aircraft will fly over the beginning and end of the vertical dashed lines in Figure 3.3a. Adjusting the start and end times, assuming duration to execute the task remains the same, within the WOO is the same as adjusting the squint angle within acceptable limits and starting the collection at a certain point in the flight path. If the task is too far to the left or right on the X axis so that the WOO would be less than zero or exceed the test time, then the limits of the WOO are set to be the times at which the aircraft will pass the boundaries of the target area. The time when the yellow task is scheduled can be shifted left and right on the timeline in Figure 3.3b as long as the entirety of the yellow task is within the dashed lines. If it is moved outside the dashed lines, then it would require the radar to transmit pulses when the aircraft is outside of the WOO on the flight path, as seen in Figure 3.3a, meaning that it must operate with a larger squint angle than is allowed. If the task is scheduled as early as possible in Figure 3.3b that means that the radar will begin imaging with a maximum squint angle as soon as the aircraft passes the left, vertical dashed line. If the task is scheduled as late as possible, that means the aircraft will complete the imaging with a maximum squint angle (in the opposite direction) just as it crosses the right, vertical dashed line. If the task is scheduled exactly in the middle of its WOO, then the radar will use a squint angle of zero and will begin transmission just as the aircraft is adjacent to the physical starting point of the task. The calculations for the beginning and end of the WOO are shown in Equations 3.8 and 3.9:

\[
WOO_{beg} = \max((y_i - x_i \tan(\theta_{max}))/v, 0) \tag{3.8}
\]
CHAPTER 3. METHODOLOGY AND DESIGN

\[ WOO_{\text{end}} = \min\left( (y_i + x_i \tan(\theta_{\text{max}}))/v, TA_x \right) \]  \hspace{1cm} (3.9)

After the duration and WOO are calculated, the tasks are ready for scheduling.

3.4 Scheduling Algorithms

As stated in Section 2.2, the four main scheduling algorithms used in this research are a basic greedy algorithm, a greedy algorithm with local search improvement, greedy with local search using multiple task input orders, and a pulse interleaved scheduler. This section will describe these methods in detail, and will provide a complexity analysis to understand how these methods will operate when the workloads are scaled.

3.4.1 Greedy Algorithm

This is the simplest approach and provides a starting point for developing other schedulers. The process is described here along with the complexity of the greedy algorithm used.

First, all of the tasks are sorted according to their priority, and within priority the tasks are sorted by duration. This means the longest, most important tasks are placed at the top of the list and are considered first. The tasks considered first are the most likely to end up in the schedule. Once a task is in the schedule, it cannot be removed.

Starting from the top of the sorted list, a single task is taken and placed into the schedule at the beginning of its WOO. The next task is taken off the top of the list and it is placed into the schedule at the beginning of its own WOO. If there are no overlaps between the incoming task and the task currently in the schedule, then the incoming task is permitted to remain in the schedule. An overlap can mean a number of things. In the greedy algorithm, there is no pulse level interleaving, so there can be no two tasks set to execute at the exact same time. This means that the greedy algorithm cannot permit overlaps. Later, when pulse interleaving is examined, overlaps mean that extra requirements need to be met to ensure these tasks can execute simultaneously.

Considering the greedy algorithm, if there is an overlap, it might be possible to place the incoming task at a later point in its WOO, which might alleviate the conflict with the task already in the schedule. The algorithm will try to set the incoming task to other possible starting positions within the WOO. If any one of the positions does not cause overlap, the incoming task is placed in the earliest possible position. Then, the start and end times of the incoming task are fixed. If
CHAPTER 3. METHODOLOGY AND DESIGN

(a) Window of Opportunity - Shown in Space

(b) Window of Opportunity - Shown in Time

Figure 3.3: Window of Opportunity of a Task
overlap occurs no matter where the incoming task is scheduled within its WOO, then it is rejected and removed. The incoming task is removed, rather than a task already added to the schedule earlier, due to the sorting method done at the beginning, which clarifies that the tasks lower on the list are less valuable and contribute less to the score. The process continues until all of the tasks have been considered once. Whatever tasks fit into the schedule after stepping through the list once is the final schedule. Once a task is fit into the schedule, it cannot be shifted or removed! This is a defining characteristic of a greedy algorithm. The greedy algorithm just described is depicted in Figure 3.4a.

3.4.1.1 Greedy Algorithm - Complexity

The greedy algorithm has a complexity of $O(N^3)$, where $N$ is the number of tasks in a test. This was found by calculating the complexity of each of the pieces of the flowchart in Figure 3.4a. The complexities of each of the sub-processes were combined, as seen in Figures 3.4b and 3.4c, until the complexity simplified to $O(N^3)$.

The complexities are assigned with the following rationale. “Set of Tasks” is given $O(N)$ because the program must load each task. “Preprocessing” is linear because a constant number of operations must be performed on each task, such as calculating the duration and WOO. Sorting is done by Matlab and is usually optimized for $O(N\log(N))$. Any box assigned a $O(1)$ is a quick decision or data transfer that only involves a small, finite number of operations independent of $N$. Checking for overlaps is $O(N)$ because in the worst case, the incoming task must be compared to all $N - 1$ tasks currently in the schedule to see if it overlaps with any of them. “Finished Schedule” is assigned $O(N)$ since all $N$ tasks will have to be copied to be output from the program.

In Figure 3.4b the black, dashed box on the right side could potentially iterate $N$ times. This is because whenever the incoming task is in conflict with a task existing in the schedule, the incoming task is re-positioned after the task it overlaps with and the program tests for overlap again. In the worst case, it is possible that the incoming task will be have to be re-positioned $N - 1$ times. The black, dashed box then simplifies to $O(N^2)$, which makes the inside of the blue, dashed box simplify to $O(N^2)$ as well. Since each task must be considered for the schedule, the dashed, blue box and overall problem simplifies to $O(N^3)$.

3.4.2 Greedy Algorithm with Local Search Algorithm

Adding local search to the greedy algorithm is not a difficult process. Instead of giving up when the incoming task cannot fit into a location, the program tries to move the existing tasks around

23
CHAPTER 3. METHODOLOGY AND DESIGN

(a) Greedy Scheduling Algorithm with Complexity of Each Process

(b) Greedy Scheduling Algorithm, Simplified Complexity

(c) Greedy Scheduling Algorithm, Final Simplified Complexity

Figure 3.4: Greedy Algorithm Complexity Analysis
to make room for the incoming task. Thus, the tasks that have been placed into the schedule earlier cannot be removed, but can be shifted around within their WOO to make space for the incoming task. The local search algorithm can be seen in Figure 3.5. In this situation, the green rectangle is the incoming task. The blue rectangles represent the existing tasks already in the schedule; their positions may be moved within their WOOs to accommodate the green task. The red lines represent the WOO for each task, the range of time in which the tasks may be set to execute. Suppose that the green task is being considered for the schedule. The scheduler first attempts to place the green task immediately after the first blue task. However, there is not enough room there. Thus, it tries to “widen” the space. The three, succeeding tasks are shifted as far right as possible within their WOOs. Then there is space for the green task to be fit at that time. The green task is then placed into the schedule and the three, succeeding blue tasks are shifted to the left as far as possible. If the green task does not fit in this space, the algorithm then applies this local search algorithm to the next empty space in the schedule, and so on, until it reached the end of the WOO of the green task. If, upon reaching the end of the green task’s WOO and it still cannot be placed without overlap, it is permanently discarded. This process is repeated for every task in the test. A flowchart of how this local search algorithm fits into the greedy algorithm is shown in Figure 3.6a.
CHAPTER 3. METHODOLOGY AND DESIGN

3.4.2.1 Greedy Algorithm with Local Search Algorithm - Complexity

This section calculates the complexity of the greedy algorithm with local search improvement. Calculating the complexity begins in Figure 3.6a and ends in Figure 3.6c. The final complexity is $O(N^3)$.

Many of the complexities assigned in Figure 3.6a are taken directly from Figure 3.4a. The only difference between the two flowcharts is the green box labeled "Local Search Algorithm." This is the process outlined in Figure 3.6a. In the worse case, each incoming task will run into a conflict initially and the local search algorithm must be used. When using this local search algorithm, the program needs to shift all succeeding tasks to the end of their WOOs and back. Technically, the number of succeeding tasks which are shifted will decrease when spaces farther down the timeline are considered, but for simplicity is it said that $2N$ moves are done when checking each space. Thus, the local search algorithm, when applied to a single incoming task, has a complexity of $O(2N)$, which simplifies to $O(N)$.

Similar to the greedy algorithm, it is possible that the incoming task will have an overlap with each task already existing in the schedule, meaning that the program will have to check for overlaps and use the local search algorithm $N$ times. Thus, the dashed, black box in Figure 3.6b will simplify to $O(N^2)$. In the same way as the greedy algorithm, the inside of the dashed, blue box will then simplify to $O(N^2)$. Thus, because the processes in the dashed blue box must be done for every task entering the schedule, the complexity becomes $O(N^3)$ overall.

3.4.3 Multiple Input Order - Greedy with Local Search

With the first two schedulers, the tasks are sorted once according to their priority and by duration within their priority. Once this was done, the scheduling started. While it is a good assumption that providing the most valuable tasks with the first scheduling opportunities will produce the highest scoring schedule, this is not always true. It will produce a very good schedule, but not necessarily the best. To test this, the following was done. After the initial sorting is accomplished, the top $N_x$ most important tasks are taken and rearranged. This is shown in Figure 3.7. This means that $N_x$ permutation $N_x$ task orders are then created, and the greedy algorithm with local search is run on every single one of them. In this example, $N_x = 5$. When $N_x = 5$, there are 120 different orders: $120 = \frac{5!}{(5-5)!}$. Notice that only the top five tasks are moving around in order and all other tasks retain their place. Each of the new input orders are put through the greedy scheduler with local search, and the highest scoring schedule is chosen as the output. As with the previous two schedulers, the
CHAPTER 3. METHODOLOGY AND DESIGN

(a) Greedy Scheduling Algorithm with Local Search with Complexity of Each Process

(b) Greedy Scheduling Algorithm with Local Search, Simplified Complexity

(c) Greedy Scheduling Algorithm with Local Search, Final Simplified Complexity

Figure 3.6: Greedy with Local Search Algorithm Complexity Analysis
program considers the tasks at the top of the lists first and gives them precedence in the timeline over other tasks. Tasks farther down the list have a much higher chance of being rejected. The process of how all task input orders are processed is shown in Figure 3.8a.

### 3.4.3.1 Multiple Input Order - Greedy with Local Search - Complexity

The overall complexity for this algorithm is $O(N^3)$. The breakdown of the complexity is in Figures 3.8a and 3.8b. Notice that the complexity could be higher, but it is reduced by only taking the permutation of the top five most important tasks, rather than every possible input order by rearranging all of the tasks. A more detailed explanation of why the top five tasks were chosen is given in Section 4.

Many of the processes used in Figure 3.8a are the same as those used in Figures 3.6a and 3.4a. Those processes will have the same complexity as in other algorithms. This section focuses only on the processes unique to the multiple input order technique. Creating every new input order is $O(P(5, 5))$, where $P(5, 5)$ means 5 permutation 5. This is equivalent to $O(1)$ since it does not change with $N$. This is because the tasks do not have to be rearranged or copied, but a series of indices can be created indicating the order that the tasks need to be referenced when running the algorithm. The greedy with local search algorithm is still $O(N^3)$, but it is run $P(5, 5)$ times. This does not increase the complexity. Getting the schedules from each input order involves copying $P(5, 5)$ schedules with $N$ tasks each. Scoring a single schedule requires summing the duration of each task at each priority, which involves about $10N$ operations. This must be done $P(5, 5)$ times.
CHAPTER 3. METHODOLOGY AND DESIGN

(a) Greedy Scheduling Algorithm with Local Search and Multiple Input Orders with Complexity of Each Process

\[ O(N) \rightarrow O(N) \rightarrow O(N \log(N)) \rightarrow O(P(5,5)N) \rightarrow O(P(5,5)N^3) \rightarrow O(P(5,5)N) \rightarrow O(P(5,5)(10)N) \rightarrow O(N) \]

\[ O(N^3) \]

(b) Greedy Scheduling Algorithm with Local Search and Multiple Input Orders, Final Simplified Complexity

Figure 3.8: Greedy with Local Search and Multiple Inputs Algorithm Complexity Analysis

which yields the complexity \( O(P(5,5)(10)N) \), which is still linear. Returning the highest scoring schedule involves finding the maximum scoring schedule and copying its results. That is a linear operation so it is assigned \( O(N) \). Simplifying this chain gives a final complexity of \( O(N^3) \).

3.4.4 Interleaved Scheduler

The interleaved scheduler is the final algorithm designed by this research. Its implementation is fairly complex, so first a general overview of the code is given. In proceeding sections, the processes in the block diagram in Figure 3.16a are expanded. For now, assume that the interleaved scheduler works in the following way. A set of tasks is given and a simple schedule is created, a schedule in which there is no pulse level interleaving, using the greedy algorithm with local search and multiple input orders. Then, additional tasks are added to the schedule using pulse level interleaving.
3.4.4.1 Greedy Algorithm - A Starting Point

As shown in Figure 3.16a, a list of tasks is given to the program and immediately the greedy algorithm with local search and multiple input orders is applied to it to create a simple schedule. At this point, none of the tasks in the schedule overlap in execution time. Any of the tasks which did not make it into the schedule are listed as “failed” tasks. These failed tasks will be interleaved into the simple schedule if possible.

3.4.4.2 Reconsidering Failed Tasks - Identifying Possible Start Times

The failed tasks are sorted by order of importance. That is, they are sorted by priority and then duration within each priority. In this manner, the tasks which are considered first will contribute the most to the score of the schedule. Starting from the top of the list, a task is placed at the beginning of its WOO. This task is called the incoming task. Next, the possible start times are marked. The possible start times are the only points the program will consider placing the incoming task. To get these points, first mark all points on the timeline that are either the beginning or ending of tasks already in the schedule or the beginning of the WOO of the current task. As explained later, these are the points when the PRF will change. These points are considered possible start times if and only if
they are within the WOO of the incoming task, and they are early enough in the WOO so that the incoming task will complete before the end of its WOO. This process of determining the possible start times is best explained through the following example.

In Figure 3.10, the incoming task is the red task and the existing tasks are the blue, yellow, green, gray, and purple tasks, which already have fixed execution times. The red task is shown in four different possible starting positions, the last of which, denoted by an “X”, is not possible because it resides outside of the WOO of the red task, which is shown in red under the X axis. Thus, the four possible starting positions are whittled down to three. Only the first three start times are possible for the red task. This is how the possible start times mentioned in Figure 3.9 are found.

Only the possible start times outlined in Figure 3.10 are considered for this reason. The fourth start time is not a possibility because if the red task is placed at the fourth start time, it would complete after its WOO. Simply sliding the task slightly to the left of the fourth start time so that it is within the WOO would be the same as just checking if the red task is compatible at the third start time. Both at the third start time and in between the third and fourth start time, the incoming task will still overlap with the yellow, green, and gray tasks. As described later, the difficulty of interleaving the incoming task into the existing tasks is determined by how many different tasks overlap the current task, not how long they overlap. Checking a starting position after the third start time and before the fourth start time could only introduce the additional possibility of the red task overlapping with the purple task, which would just be additional task with which to check compatibility. Thus, only a finite number of possible start times are considered as viable starting points for the incoming task.

The program attempts to place the incoming task at each of these three locations, but not in order from earliest to latest. The more existing tasks that the incoming task overlaps with, the more difficult it will be to make all overlapping tasks compatible. Because of this, it makes the most sense to count the number of overlaps that would occur if the incoming task was placed at a possible start time. Then, the program attempts to put the incoming task at the different start times by order of the least number of overlaps. In Figure 3.10, the number of overlaps when the red task is at positions 1, 2, and 3 are 4, 3, and 3 overlaps, respectively. At the first position, the red task will overlap with the blue, yellow, green, and gray tasks. At positions two and three, the red task will overlap with the yellow, green, and gray tasks. Thus, the program will attempt to place the red task at the second possible start time, then the third possible start time, and then the first possible start time. If the incoming task is found to be compatible with all overlapping tasks at one of these times, then it is placed there and none of the remaining possible start times are checked. If the incoming task is
CHAPTER 3. METHODOLOGY AND DESIGN

Figure 3.10: Possible Start Times

The red boundaries designate the WOO of the incoming, red, task. The red task is shown in three possible positions in the schedule and one position which is not viable. The impossible case is marked with an “X” because it resides partially outside the WOO.

incompatible at any of these locations, it is discarded.

3.4.4.3 Checking Incoming Task at a Specific Location

The process of interleaving an incoming task into the existing tasks in the schedule consists of breaking up the timeline into simple cases, making all task pieces in the simple cases compatible, and then stitching the task pieces back together. This is described in the following sections.

First, the incoming task is temporarily placed at a starting point in the timeline and the problem is broken up into many simple cases. To do this, all of the relevant tasks, or tasks from the existing schedule which overlap in execution time with the incoming task, are marked. These are the tasks that the incoming task must be made compatible with in order for it to be placed at that point in the schedule. This is shown in Figure 3.11. The incoming task is the red task; the blue, yellow, green, gray, and purple tasks are the existing tasks. However, the blue and purple tasks are not relevant, because they do not overlap with the red task when it is at position two. Once all of the relevant tasks are collected, they are broken up into sets, which are just simple cases where all tasks overlap exactly in execution time. This is also shown in Figure 3.11.

Once the relevant tasks are broken down into simple cases, the task pieces in each set can be interleaved and then stitched back together. The set boundaries are determined by break times.
CHAPTER 3. METHODOLOGY AND DESIGN

Figure 3.11: Break Into Simple Cases

Similar to the possible start times, the break times are either the start or end times of the existing or incoming tasks which overlap the planned execution of the incoming task. In Figure 3.11, all of the break times are shown as vertical dashed lines. The gray task’s end time is not included because it does not occur during the red task’s execution. In between break times, if the PRF and pulse delays are set so that the task pieces are compatible for a single PRI, then they are compatible for the entire duration of the set. As noted earlier, it is possible for the PRF and pulse delays to change throughout a stripmap SAR task’s duration without degrading image quality. Thus, introducing the red task in Figure 3.11 has become five separate problems. Two of the sets, A and D, only consist of one task piece each, so they do not need to be changed at all. The other three sets, B, D, and E, need to be changed so that all of the task pieces of a set are compatible with each other.

3.4.4.4 Making Task Pieces Compatible - Simple Case

Now that the problem has been broken down into many simple cases, if compatibility can be found for all of the sets, then the red task would be compatible in that location of the schedule. The simple case is defined as when two or more task overlap exactly in execution time, that is, they have the exact same start and end times. All of the sets produced in Figure 3.11 are simple cases. For tasks in the simple case to be compatible, the following four requirements must be met. (1) The PRFs must be equal; (2) the maximum duty factor of the radar cannot be exceeded; (3) none of the pulses, either transmitted or returning, can overlap temporally; (4) the maximum output energy for any given time window in the PRI cannot be exceeded. This process is shown in Figure 3.12.
CHAPTER 3. METHODOLOGY AND DESIGN

Figure 3.12: Process to Make Tasks Compatible

these four requirements are met for each of the sets in Figure 3.11, then the red task is compatible at location two. The four requirements for the simple case are described next.

This section will describe how a common PRF is found for all tasks in the simple case to satisfy requirement number one. First, to come up with a common PRF for each task piece in a set, the PRF ranges are calculated for each piece. As mentioned earlier, the PRF for stripmap SAR can be varied within the limits denoted by Equation 2.4. Notice that the lower limit of Equation 2.4 is determined by aircraft constants and is not unique to a task. Therefore, all tasks in a test have the same lower PRF limit. The formula for choosing a PRF that is guaranteed to overlap all of the PRF ranges for each task is as follows. Obviously, the lower limit could just be chosen for each task. However, an aim of the scheduler is to change the tasks as little as possible when interleaving them. With this in mind, a PRF is chosen that is relatively near the original, designed PRFs. The new PRF is equal to the average between the lowest, maximum PRF range of any task and the minimum, existing PRF of any task. This is displayed in Figure 3.13. It is a drawing of the allowable PRF ranges for four different tasks currently being considered for interleaving. The small, black, vertical lines denote the current PRF of each task. The two red, dashed lines mark the minimum current PRF and the smallest, PRF upper limit. The minimum current PRF belongs to Task 4 and the smallest PRF upper limit is defined by Task 1. The solid, red line denotes the average of the two red, dashed lines, and it is the common PRF for all four tasks. If this result is within the PRF limits of all tasks, then it is the new PRF for all tasks in that set.
If the PRF can be changed, the pulse width is altered so that the duty factor of each task remains what it was before the PRF shift. This adjustment is defined by Equation (3.10). Typically, radar tasks are designed to output a certain amount of energy, or operate at a certain duty cycle. If the PRF changes but the pulse width remains the same, then the amount of energy being emitted from the radar over the length of the task is changing. This could lead to excessive or insufficient energy being emitted, which could lead to unexpected, poor performance. This situation is avoided by simply keeping the duty factor of each task constant.

\[ \tau_{\text{old}}(PRF_{\text{old}}) = \tau_{\text{new}}(PRF_{\text{new}}) \]  

(3.10)

After the pulse widths have been set to their final length, the program checks if the sum of the pulse widths of all tasks will exceed the maximum duty factor of the radar. If this is the case, the second requirement is not met and the task pieces cannot be interleaved. If that is not the case, the program moves to the next step in Figure 3.12 which is to verify the third and fourth requirements of the simple case.

This section will discuss how the third requirement of the simple case is met. The transmitted and received pulses must be separated because the radar cannot transmit pulses in two directions simultaneously or receive returning pulses from two different directions at the same time with a single radar beam [20]. To ensure that all of these pulses are separated temporally, first, the
CHAPTER 3. METHODOLOGY AND DESIGN

return envelope, or the time period that a pulse might return, needs to be calculated. The beginning of the return envelope is the time it will take for the beginning of the transmitted pulse to hit the closest point on the target and return to the radar. The end of the return envelope is the time it will take for the end of the transmitted pulse to hit the farthest point on the target and return to the radar. The equation for the maximum distance any part of the task is from the radar at a maximum squint angle is shown in Equation 3.12. The minimum distance is shown in Equation 3.11. Notice that the squint angle is set to zero in this case, because that is when the aircraft will be perfectly adjacent and closest to the task. Calculating the beginning of the return envelope based off a squint angle of zero ensures that the soonest possible time has been found for the beginning of the return envelope in case the task is set in the middle of its WOO. Using a squint angle of zero and a maximum squint angle for the beginning and end of the return envelope ensure that the soonest and latest possible time that tasks could be returning are considered, which gives the program liberty to move the task around its WOO without changing the return envelope. The return envelope calculations are shown in Equations 3.13 and 3.14.

\[
R_{i\text{min}} = \sqrt{\left(x_i - \frac{x_{iw}}{2}\right)^2 + h^2} 
\]  \hspace{1cm} (3.11)

\[
R_{i\text{max}} = \sqrt{\left(x_i + \frac{x_{iw}}{2}\right)^2 + x_i^2 \tan^2 \phi_{\text{max}} + h^2} 
\]  \hspace{1cm} (3.12)

\[
\tau_{\text{begReturn}} = \frac{2}{c} R_{i\text{min}} 
\]  \hspace{1cm} (3.13)

\[
\tau_{\text{endReturn}} = \frac{2}{c} R_{i\text{max}} 
\]  \hspace{1cm} (3.14)

Figure 3.14 shows this process of separating transmitted pulses and return envelopes more clearly. Let the blue pulses belong to Task 1 and the orange pulses belong to Task 2. The darker, high power pulses are the transmitted pulses and the lighter, lower power pulses are the return envelopes. First, it is assumed that the PRFs of the two tasks have been set to the same value. Second, in Figure 3.14a there is an initial delay of \(\tau_0\) applied to Task 2 so that the pulses transmitted for Tasks 1 and 2 do not overlap; the radar cannot transmit maximum power in two different directions in the same instant. Notice that \(\tau_0\) is simply the pulse width of Task 1, so that the pulse of Task 2 is transmitted immediately after Task 1’s transmitted pulse. Once this delay is applied, the return envelopes for each task are calculated.
as calculated in Equations 3.13 and 3.14. These return envelopes, the possible times the pulses may return based on the target distance for each task, overlap by $\tau_a$. The radar cannot operate in this state, since it would have to direct its beam in two different directions at the same time in order to receive the return pulses of each task simultaneously. Therefore, an additional delay is applied to the transmit pulse of Task 2 so that both the transmitted pulses and the return envelopes are separated temporally. The total delay added to Task 2 is $\tau_0 + \tau_a$. The result is shown in Figure 3.14c. Notice that there is a limit to how large of a delay may be added to Task 2. If too large of a delay is added, then the return envelope of Task 2 will overlap with the next sent pulse of Task 1. This requires that the pulse delay of any task must be small enough so that the end of the return envelope does not exceed the PRI of the tasks.

Figure 3.14 describes the process of separating the pulses of two tasks, but this needs to be extrapolated for separating the pulses when there are more than two tasks in the simple case. It is almost the exact same process. First, the return envelopes of all tasks in the set are calculated as if the pulses were transmitted without delay at the beginning of the PRI. These tasks are then sorted so that the tasks with the earliest return envelopes are considered first. Starting from the top of the list, the program adds the transmitted pulse and return envelope the PRI schedule one by one, just as in Figure 3.14. The PRI schedule is the organization of transmitted pulses and return envelopes for all
the tasks in the simple case for a single PRI. Once the transmitted pulses and return envelopes of a task are set into the PRI schedule, they are not moved or removed. If the delay needed to separate the return envelopes is so large that the return envelope occurs past the end of the PRI, the tasks in that set are not compatible with each other. If it is possible to place each task into the PRI schedule so that all transmitted pulses and return envelopes are separated temporally without exceeding the PRI, the delays applied are recorded and the third requirement for the simple case is satisfied.

Finally, the fourth requirement for the simple case is to ensure that the energy output at any given $50 \mu s$ window in the PRI schedule does not exceed the maximum allowable energy, $E_{\text{max}}$. This requirement is checked in the following way. An energy timeline of the PRI is created for the tasks in the PRI schedule thus far. This timeline is an array the length of the PRI in very small discrete time intervals, where all values are zero except for the instances where a pulse is being transmitted. These elements receive a value equal to the amount of energy transmitted in the small discrete time interval. A window of duration $50 \mu s$ is created in intervals of the same discrete time interval. The window is an array of all ones. Convolution is performed between the energy timeline and the window. If any value in the resulting array is above the maximum energy output, $33.75 mJ$, then the energy constraint has been exceeded and the current pulse organization in the PRI will not work. This indicates that the period of continuous transmission is too long and the radar hardware could overheat. To alleviate this conflict, the transmit pulse of the task being considered for the PRI schedule is delayed by half of the window length. The energy requirement is then checked again, and more delays are added if necessary until the output energy is not exceed or the end of the PRI is reached. An example of this can be seen in Figure 3.15. In the top case, the output energy is $45 mJ$ for the 0 to $50 \mu s$ window, which exceeds $E_{\text{max}}$. In the bottom case, the orange transmitted pulse is delayed by half the window length, $25 \mu s$. The most energy output in any given window in the bottom case is $31.5 mJ$, so $E_{\text{max}}$ is not exceeded and the pulse delay alleviated the conflict. It is possible that by adding this delay, the return envelopes might overlap now. Thus, whenever a delay is added, both the third and fourth requirements of the simple case must be rechecked, just as in the lower portion of the flowchart in Figure 3.12.

If all four of the requirements for the simple case are met, then the task pieces in that simple case can be interleaved and executed at the same time. Again, these requirements are that they all have the same PRF, the duty factor is not exceeded, the transmitted pulses and return envelopes are separated temporally, and the maximum output energy is not exceeded for any time window in the PRI schedule. If these four requirements are met for each set in Figure 3.11 then the pieces of the tasks can be stitched back together and the red task is compatible at position 2. It can be
3.4.4.5 Interleaved Scheduler - Complexity

This section will examine the complexity of the interleaved scheduler. For convenience, Figures 3.9 and 3.12 are reproduced in this section. Next to each of the steps in the flowchart are the complexities of the respective process. The overall complexity of the interleaved scheduler is $O(N^5)$. The process of finding the complexity is shown in Figures 3.16, 3.17, 3.18, 3.19, and 3.20. The process for arriving at this final complexity is described below.

Only the processes, the blue boxes, which are unique to the interleaved scheduler will be analyzed in this section. The task starting with “Place Task at Beginning of WOO” is assigned $O(N)$ because to get the possible start times, the start and end time of each task must be considered as a possible start time. That will use at least $2N$ operations. For “Find Number of Overlaps,” each possible start time must be compared to every single task in the worst case. This is $2N$ possible start times each being compared to $N$ tasks. Thus, it is of complexity $O(N^2)$. “Get Relevant Tasks” means that the incoming task at a particular starting position must be compared to each task in the schedule, so it is of complexity $O(N)$. “Break Into Sets” is of complexity $O(N^2)$ because each break time must be compared to each task in the schedule. Making the pieces in each set compatible is of complexity $O(N^4)$, because there could be $N$ sets and the complexity for making one set...
compatible is $O(N^3)$. This complexity for making a single set compatible is derived in the next paragraph. By simplifying the dashed, blue box in Figure 3.17b yields a complexity of $O(N^5)$, since the compatibility process might occur $2N$ times in the worst case. Because the black, dashed box in Figure 3.18a is only repeated 20 times, rather than $N$ times, it does not add to the complexity. Originally, all of the failed tasks were included, but the processing time was too large to be justify the diminishing returns of re-considering all failed tasks. This rationale will be discussed more in the results section. If all failed tasks were reconsidered, then the complexity for the interleaved scheduler would become $O(N^6)$.

Next, the complexity of the simple case will be discussed. This process is shown graphically in Figures 3.19 and 3.20. Finding a common PRF is linear because it is just looking up two values of each task piece, performing minimum and maximum operations on those values, and then finding an average to get a common PRF. Adjusting the pulse widths is also linear as it is just a simple calculation which must be done to each task piece independently. Finding the duty factor is of order $O(N)$ because the pulse widths of all tasks must be summed. Checking the return envelopes is of order $O(N)$ because the return envelopes for each task must be found and the return envelope of the incoming task piece must be compared to the return envelopes of all task pieces existing in the PRI schedule. Checking the energy constraint involves a convolution operation, but it is not dependent on $N$. The number of operations is uncertain, but the amount of computation time seems to be negligible and it is constant regardless of what $N$ is. Looking at Figure 3.19b, the blue, dashed box could potentially iterate $N$ times. This is if the incoming task piece conflicts with each task piece currently in the PRI schedule, one at a time. A new pulse delay would be issued and again the return envelope and energy constraint would be checked with each iteration. Thus, the blue box reduces to $O(N^2)$. In the worst case, it is possible to have $N$ task pieces in a set that need to be interleaved. This causes the black, box to reduce to $O(N^3)$. $O(N^3)$ is the highest complexity in Figure 3.20b, so it is the final complexity of the process to make $N$ task pieces in a set compatible.
CHAPTER 3. METHODOLOGY AND DESIGN

(a) Interleaved Scheduling Algorithm with Complexity of Each Process

(b) Simplified Complexity - 1

Figure 3.16: Interleaved Scheduling Algorithm Complexity Analysis - Part 1
CHAPTER 3. METHODOLOGY AND DESIGN

(a) Simplified Complexity - 2

Figure 3.17: Interleaved Scheduling Algorithm Complexity Analysis - Part 2

(b) Simplified Complexity - 3

Figure 3.18: Interleaved Scheduling Algorithm Complexity Analysis - Part 3
CHAPTER 3. METHODOLOGY AND DESIGN

(a) Making $N$ Task Pieces Compatible in a Single Set - Complexity of Each Process is Labeled

Figure 3.19: Making $N$ Task Pieces Compatible in a Single Set Complexity Analysis - Part 1
Figure 3.20: Making $N$ Task Pieces Compatible in a Single Set Complexity Analysis - Part 2
Chapter 4

Experiments and Results

This chapter will discuss the types of benchmarks created to test the four scheduling algorithms, the results of each of these algorithms, and conclusions based on these results.

Seven benchmarks were created for testing the four algorithms used in this research. These benchmarks were created with variations in the mission constants to simulate various flight conditions and hardware capabilities. These values were chosen based off of the normal altitude limits of the JSTARS and normal maximum duty factors and maximum output power for various surveillance radar. The quantities measured were the score as described in Section 3.1, the peak number of simultaneous tasks scheduled in a test, and the time needed to create a schedule. The original situation is referred to as the “base case” and is described by the mission constants shown in the second column of Table 4.1. Columns 3 through 8 describe the different mission constants used to create the different benchmarks. There are seven benchmarks in total, the base case, low maximum duty factor, high maximum duty factor, low maximum power, high maximum power, low altitude, and high altitude. There are 1,000 tests for each situation’s benchmark, and each test composed of 100 tasks. First, the performance of the different schedulers will be compared in the base case. Once it is obvious that the interleaved scheduler is the optimum choice for maximizing the score, the performance of the interleaved scheduler will be evaluated in all seven of the situations.

The scores, peak number of scheduled tasks, and the execution time for the base case can be seen in Tables 4.2 and 4.3. All tests were performed on the Discovery Cluster provided by Northeastern University. For the multiple input order scheduler, the number of tasks which were reordered, denoted as \( N_x \), was varied between 1 and 7. The scores of the schedules tended to improve as the number of reordered tasks increased, as this increased the number of permutations for different schedules to be created. However, the execution time increased significantly as \( N_x \) increased. When
Table 4.1: Mission Constants for Various Situations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Base Case</th>
<th>Low DF</th>
<th>High DF</th>
<th>Low Power</th>
<th>High Power</th>
<th>Low Altitude</th>
<th>High Altitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DF_{\text{max}}$</td>
<td>0.15</td>
<td>0.10</td>
<td>0.20</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>$\theta_{\text{max}}$</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
</tr>
<tr>
<td>$\psi_{\text{min}}$</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
</tr>
<tr>
<td>$\psi_{\text{max}}$</td>
<td>35°</td>
<td>35°</td>
<td>35°</td>
<td>35°</td>
<td>35°</td>
<td>35°</td>
<td>35°</td>
</tr>
<tr>
<td>$P_{\text{max}}$</td>
<td>900 W</td>
<td>900 W</td>
<td>900 W</td>
<td>700 W</td>
<td>1100 W</td>
<td>900 W</td>
<td>900 W</td>
</tr>
<tr>
<td>$W_{\text{rad}}$</td>
<td>6 m</td>
<td>6 m</td>
<td>6 m</td>
<td>6 m</td>
<td>6 m</td>
<td>6 m</td>
<td>6 m</td>
</tr>
<tr>
<td>$L_{\text{rad}}$</td>
<td>6 m</td>
<td>6 m</td>
<td>6 m</td>
<td>6 m</td>
<td>6 m</td>
<td>6 m</td>
<td>6 m</td>
</tr>
<tr>
<td>$f_{c}$</td>
<td>1 GHz</td>
<td>1 GHz</td>
<td>1 GHz</td>
<td>1 GHz</td>
<td>1 GHz</td>
<td>1 GHz</td>
<td>1 GHz</td>
</tr>
<tr>
<td>$\alpha_{\text{min}}$</td>
<td>10,000 m</td>
<td>10,000 m</td>
<td>10,000 m</td>
<td>10,000 m</td>
<td>10,000 m</td>
<td>8,000 m</td>
<td>13,000 m</td>
</tr>
<tr>
<td>$v$</td>
<td>230 m/s</td>
<td>230 m/s</td>
<td>230 m/s</td>
<td>230 m/s</td>
<td>230 m/s</td>
<td>230 m/s</td>
<td>230 m/s</td>
</tr>
<tr>
<td>$T_{A_{x}}$</td>
<td>40 km</td>
<td>40 km</td>
<td>40 km</td>
<td>40 km</td>
<td>40 km</td>
<td>40 km</td>
<td>40 km</td>
</tr>
<tr>
<td>$T_{A_{y}}$</td>
<td>160 km</td>
<td>160 km</td>
<td>160 km</td>
<td>160 km</td>
<td>160 km</td>
<td>160 km</td>
<td>160 km</td>
</tr>
<tr>
<td>$x_{\text{it}_{\text{min}}}$</td>
<td>5 km</td>
<td>5 km</td>
<td>5 km</td>
<td>5 km</td>
<td>5 km</td>
<td>5 km</td>
<td>5 km</td>
</tr>
<tr>
<td>$x_{\text{it}_{\text{max}}}$</td>
<td>8 km</td>
<td>8 km</td>
<td>8 km</td>
<td>8 km</td>
<td>8 km</td>
<td>8 km</td>
<td>8 km</td>
</tr>
<tr>
<td>$y_{\text{it}_{\text{min}}}$</td>
<td>96 km</td>
<td>96 km</td>
<td>96 km</td>
<td>96 km</td>
<td>96 km</td>
<td>96 km</td>
<td>96 km</td>
</tr>
<tr>
<td>$y_{\text{it}_{\text{max}}}$</td>
<td>0.00375 J</td>
<td>0.00375 J</td>
<td>0.00375 J</td>
<td>0.00375 J</td>
<td>0.00375 J</td>
<td>0.00375 J</td>
<td>0.00375 J</td>
</tr>
<tr>
<td>$E_{\text{max}}$</td>
<td>50 µs</td>
<td>50 µs</td>
<td>50 µs</td>
<td>50 µs</td>
<td>50 µs</td>
<td>50 µs</td>
<td>50 µs</td>
</tr>
</tbody>
</table>

Table 4.2: Scores of Different Schedulers - Base Case

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>487 42 21 13 9 7 5 6 6 3</td>
</tr>
<tr>
<td>Greedy with Local Search</td>
<td>516 47 22 13 9 7 5 5 5 3</td>
</tr>
<tr>
<td>Greedy Multiple Input Order ($N_{x} = 5$)</td>
<td>583 17 9 7 6 5 4 4 4 2</td>
</tr>
<tr>
<td>Interleaved</td>
<td>1420 500 111 10 6 5 4 3 3 2</td>
</tr>
</tbody>
</table>

The scores listed are the average scores over the 1,000 tests in the benchmark.
CHAPTER 4. EXPERIMENTS AND RESULTS

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Execution Time (s)</th>
<th>Execution Time Std. Dev. (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>0.29</td>
<td>0.033</td>
</tr>
<tr>
<td>Greedy with Local Search</td>
<td>0.45</td>
<td>0.068</td>
</tr>
<tr>
<td>Greedy Multiple Input Order ((N_x = 5))</td>
<td>24</td>
<td>5.8</td>
</tr>
<tr>
<td>Interleaved</td>
<td>37</td>
<td>8.1</td>
</tr>
</tbody>
</table>

Table 4.3: Execution Times for Different Schedulers
The times listed are the average time over the 1,000 tests in the benchmark.

Figure 4.1: Greedy Algorithm with Local Search and Multiple Input Order - Score vs Execution Time for varying \(N_x\) values
CHAPTER 4. EXPERIMENTS AND RESULTS

<table>
<thead>
<tr>
<th>Situation</th>
<th>Score</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Base Case</td>
<td>1420</td>
<td>500</td>
<td>111</td>
<td>10</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Low DF</td>
<td>1402</td>
<td>519</td>
<td>113</td>
<td>12</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>High DF</td>
<td>1420</td>
<td>500</td>
<td>111</td>
<td>11</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Low Power</td>
<td>1411</td>
<td>503</td>
<td>123</td>
<td>11</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>High Power</td>
<td>1402</td>
<td>529</td>
<td>106</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Low Altitude</td>
<td>1416</td>
<td>504</td>
<td>119</td>
<td>12</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>High Altitude</td>
<td>1414</td>
<td>496</td>
<td>119</td>
<td>10</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.4: Scores of Interleaved Schedule in All Situations
The scores listed are the average scores over the 1,000 tests in the respective benchmark.

<table>
<thead>
<tr>
<th>Situations</th>
<th>Execution Time (s)</th>
<th>Execution Time Std. Dev. (s)</th>
<th>Peak Number of Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Case</td>
<td>37</td>
<td>8.1</td>
<td>5.1</td>
</tr>
<tr>
<td>Low DF</td>
<td>39</td>
<td>9.4</td>
<td>5.0</td>
</tr>
<tr>
<td>High DF</td>
<td>38</td>
<td>8.0</td>
<td>5.0</td>
</tr>
<tr>
<td>Low Power</td>
<td>37</td>
<td>8.6</td>
<td>5.0</td>
</tr>
<tr>
<td>High Power</td>
<td>38</td>
<td>8.4</td>
<td>5.0</td>
</tr>
<tr>
<td>Low Altitude</td>
<td>37</td>
<td>8.9</td>
<td>5.0</td>
</tr>
<tr>
<td>High Altitude</td>
<td>38</td>
<td>8.4</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Table 4.5: Interleaved Scheduler Execution Times and Peak Interleaved Tasks

amount. Thirty-seven seconds seems reasonable, considering the large increase in score over the other schedulers. By these standards, the interleaved scheduler is clearly the best of the four algorithms. To better understand the interleaved scheduler, it was tested in seven situations. Doing so provided insight on its reliability and robustness in typical flight scenarios and within normal hardware limitations. The results from these benchmarks are shown in Tables 4.4 and 4.5. The interleaved scheduler appears to be fairly consistent across all situations, both in terms of the score, execution time, and peak number of tasks scheduled at once. The differences in score are so small that they cannot be considered significant and must be attributed to the randomness in data generation. Overall, this should give future operators confidence in the interleaved scheduler’s ability to consistently organize tasks to a set standard within regular operating limits.

4.0.1 Schedule Graphics

This section shows the schedules produced by the four different algorithms. Both the timeline of the execution of the tasks and the physical layout of the tasks in the target area are
CHAPTER 4. EXPERIMENTS AND RESULTS

displayed. These results are shown in Figures 4.2, 4.3, 4.4, and 4.5 and were created from Test 1 of the base case benchmark. In the top left corner of the timeline is the ten element score vector of that schedule produced, as described by Section 3.1. The vertical lines in the timelines are the break times where the PRF, pulse widths, and pulse delays of tasks are changing. Notice that just because two tasks have the same X coordinate in the target area they are not necessarily operating at the same time, since the squint angle of the radar can change between tasks. From these figures, it is evident that the interleaved scheduler is capable of gathering much more information in a single flight past the target area than the other three algorithms.
CHAPTER 4. EXPERIMENTS AND RESULTS

(a) Schedule - Timeline

(b) Tasks in Target Area

Figure 4.2: Greedy Algorithm Results - Base Case Benchmark - Test 1
CHAPTER 4. EXPERIMENTS AND RESULTS

Figure 4.3: Greedy Algorithm with Local Search Results - Base Case Benchmark - Test 1
Figure 4.4: Greedy Algorithm with Local Search and Multiple Input Orders with $N_x = 5$ Results - Base Case Benchmark - Test 1
CHAPTER 4. EXPERIMENTS AND RESULTS

Figure 4.5: Interleaved Schedule Results - Base Case Benchmark - Test 1
Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis four different scheduling algorithms were applied to planning the execution of several stripmap SAR tasks during the flight of a surveillance aircraft. By order of worst to best, these algorithms are greedy, greedy with local search, greedy with local search and multiple input orders, and pulse level interleaving. Pulse level interleaving produces the highest scores by far in a reasonable amount of time. Seven different flight and radar scenarios were used to generate test data. For each of them, the score, execution time, and peak number of tasks of the interleaved scheduler are fairly consistent, proving that its performance is robust in normal operating circumstances. Pulse level interleaving is recommended for any airborne radar system executing stripmap SAR tasks.

5.2 Future Work

As suggested earlier, it would be useful to apply this pulse level interleaving technique to multiple beams from the same radar. Also, multiple types of SAR modes could be used, as well as GMTI and other modes. If possible, testing should be done on actual hardware systems to verify that the results are satisfactory when interleaving so many different tasks at the same time.

Additionally, when generating the test data, the duty factors of each task were randomly generated between 2.5% and 3.5%. In reality, the minimum pulse width is typically calculated exactly, based on the normalized radar cross section of the target and the amount of energy the user would like to receive. In future experiments, it would be useful to calculate the exact, minimum pulse width needed for the tasks and the algorithms should be re-run with this data.
Bibliography


BIBLIOGRAPHY


