Application of Control Allocation Methods to Linear Systems with Four or More Objectives

by
Roger E. Beck

Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Aerospace Engineering

Dr. Wayne Durham, Chair
Dr. Frederick Lutze
Dr. Craig Woolsey
Dr. Christopher Hall
Dr. Eugene Cliff

June 11, 2002
Blacksburg, Virginia

Keywords: Control Allocation, Linear Programming, BESA
Copyright 2002, Roger E. Beck
Methods for allocating redundant controls for systems with four or more objectives are studied. Previous research into aircraft control allocation has focused on allocating control effectors to provide commands for three rotational degrees of freedom. Redundant control systems have the capability to allocate commands for a larger number of objectives. For aircraft, direct force commands can be applied in addition to moment commands.

When controls are limited, constraints must be placed on the objectives which can be achieved. Methods for meeting commands in the entire set of achievable objectives have been developed. The Bisecting Edge Search Algorithm has been presented as a computationally efficient method for allocating controls in the three objective problem. Linear programming techniques are also frequently presented.

This research focuses on an effort to extend the Bisecting Edge Search Algorithm to handle higher numbers of objectives. A recursive algorithm for allocating controls for four or more objectives is proposed. The recursive algorithm is designed to be similar to the three objective allocator and to require computational effort which scales linearly with the controls.

The control allocation problem can be formulated as a linear program. Some background on linear programming is presented. Methods based on five formulations are presented.

The recursive allocator and linear programming solutions are implemented. Numerical results illustrate how the average and worst case performance scales with the problem size. The recursive allocator is found to scale linearly with the number of controls. As the number of objectives increases, the computational time grows much faster. The linear programming solutions are also seen to scale linearly in the controls for problems with many more controls than objectives.

In online applications, computational resources are limited. Even if an allocator performs well in the average case, there still may not be sufficient time to find the worst case solution. If the optimal solution cannot be guaranteed within the available time, some method for early termination should be provided. Estimation of solutions from current information in the allocators is discussed. For the recursive implementation, this estimation is seen to provide nearly optimal performance.
Acknowledgments

I would like to thank all of those who have helped me get to the point where I am today. I am not sure that anyone can fully comprehend or appreciate all of those who have helped shape them into who they are. I particularly would like to individually recognize some of those who have been important in this stage of my academic journey.

My wife Catherine has been a constant source of support and love. I am grateful for the amount of stuff she has put up with from me while I have been a graduate student. My parents and brother are always there when I need them, and their willingness to help is greatly appreciated.

Dr. Wayne Durham has done a wonderful job as my advisor. I have benefitted from his enthusiasm for research and learning and his concern and involvement with his students outside of school. His candor in dealing with students is always appreciated. I would like to thank Dr. Lutze, Dr. Hall, Dr. Cliff, Dr. Woolsey, and Dr. Anderson for serving on my committee at one point or another and for their accessibility and willingness to discuss interesting ideas, good or bad, which have not always found their way into my work.

The students in the flight simulation laboratory have made it a wonderful environment in which to work. In particular, I would like to thank Henrik Pettersson for his help in preparing this document even as he is finishing his own work; Kevin Scalera and Drew Robbins for involving me in their work and for convincing me that everything is indeed “plug-and-play”; and Mike Phillips and Ralph Sandfry for talking me into taking the preliminary exams and offering untold advice and support about school and life. Bill Oetjens has always been willing to help out; I am grateful for his support.

No list would be complete without including Wanda Foushee and Betty Williams. They have made it simple to deal with the myriad details which arise in research and graduate education.
## Contents

1 Introduction ................................................................. 1

2 Problem Statement ......................................................... 4
   2.1 Control Law Structure .............................................. 4
   2.2 Linear Control Allocation .......................................... 6
      2.2.1 Geometry of attainable objectives ............................ 7
      2.2.2 Terminology .................................................... 11

3 Background and Literature Review ..................................... 13
   3.1 History ................................................................. 13
   3.2 Non-Optimal Methods ............................................... 14
      3.2.1 Daisy Chain Solutions ......................................... 15
      3.2.2 Multiple Pass Inverses ....................................... 16
      3.2.3 Cascaded Generalized Inverse Solutions ....................... 16
   3.3 Optimal Methods ..................................................... 17
      3.3.1 Direct Allocation Solutions .................................... 18
      3.3.2 Optimization Based Solutions ................................ 19
   3.4 Rate Limiting and Discrete Allocation ............................ 23
   3.5 Realtime Control Allocation ....................................... 23
   3.6 Applications for Higher Dimensional Control Allocation ....... 24
4 Control Allocation For Four Objectives

4.1 Introduction to the Bisecting Edge Search  

4.1.1 Aligning the Desired Objective Direction  
4.1.2 Finding the intersection of $x_1$ with $\partial(\Phi)$  

4.2 Four Objective Bisecting Edge Search  

4.2.1 Aligning the desired objective  
4.2.2 Edge Searching in four Dimensions  
4.2.3 Implementation of Four Objective Edge Searching  
4.2.4 Results  

4.3 Searching For Three-Dimensional Objects  

4.3.1 Bisection Search  
4.3.2 Facet Search Subproblem  
4.3.3 BESA Subproblem  
4.3.4 Three-Dimensional BESA Revisited  
4.3.5 Results For an Improved Four Objective Allocator  

5 Methods for Higher Numbers of Objectives

5.1 Recursive Method  

5.1.1 Aligning the Objective with $x_1$  
5.1.2 Search  
5.1.3 Checking the solution  
5.1.4 Estimation  
5.1.5 Special Cases  

5.2 Linear Programming  

5.2.1 Solvers  
5.2.2 Control Allocation Formulations  
5.2.3 Early Termination
6 Results

6.1 Recursive Method .............................................. 54
  6.1.1 Tuning ......................................................... 54
  6.1.2 Timing Results ............................................... 55
  6.1.3 Estimation ...................................................... 60
  6.1.4 Discussion ..................................................... 67

6.2 Linear Programming ............................................ 67
  6.2.1 Solvers ......................................................... 67
  6.2.2 Control Allocation .......................................... 69
  6.2.3 Discussion ..................................................... 76

7 Conclusions ......................................................... 77
# List of Figures

2.1 Block diagram showing control law architecture ............................................. 5  
2.2 $\Omega$ for three controls. .............................................................................. 8  
2.3 $\Phi$ for three controls and two objectives. ......................................................... 8  
2.4 $\Phi$ for four controls and two objectives. ......................................................... 9  
2.5 $\Phi$ for four controls and three objectives. ......................................................... 9  
2.6 $\partial(\Phi)$ for a four-objective problem with six controls. ............................... 10  
3.1 Block diagram showing daisy chaining allocator ............................................. 15  
4.1 Edges defining a three-dimensional facet. ......................................................... 29  
4.2 Tests of BESA extensions. .............................................................................. 32  
4.3 Number of bisections required to solve the three-dimensional problem. ....... 35  
4.4 Execution time of the four-dimensional recursive allocator. ............................ 36  
4.5 Number of bisections required in the outer loop of the four-objective case. .. 37  
4.6 Total number of three-objective bisections required to solve a four-objective problem. ......................................................... 37  
5.1 Vectors defining a three-dimensional facet are used to find the desired inter-  

section.................................................................................................................... 42  
6.1 CPU Time for the recursive allocator with three objectives. .......................... 56  
6.2 CPU Time for the recursive allocator with four objectives. ........................... 57  
6.3 CPU Time for the recursive allocator with five objectives. ............................ 58  
6.4 Scaling of computational time with objectives and controls ........................... 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5</td>
<td>CPU Time for the recursive allocator w/ estimator for four objectives.</td>
<td>61</td>
</tr>
<tr>
<td>6.6</td>
<td>CPU Time for the recursive allocator w/ estimator for five objectives.</td>
<td>62</td>
</tr>
<tr>
<td>6.7</td>
<td>Percentage of cases which estimate a solution in $\mathbb{R}^3$.</td>
<td>64</td>
</tr>
<tr>
<td>6.8</td>
<td>Percentage of cases which estimate a solution in $\mathbb{R}^4$.</td>
<td>64</td>
</tr>
<tr>
<td>6.9</td>
<td>Percentage of cases which estimate a solution in $\mathbb{R}^5$.</td>
<td>65</td>
</tr>
<tr>
<td>6.10</td>
<td>Normalized mean error as a function of controls for three objectives.</td>
<td>65</td>
</tr>
<tr>
<td>6.11</td>
<td>Normalized mean error as a function of controls for four objectives.</td>
<td>66</td>
</tr>
<tr>
<td>6.12</td>
<td>Normalized mean error as a function of controls for five objectives.</td>
<td>66</td>
</tr>
<tr>
<td>6.13</td>
<td>Comparison of different linear programming solution techniques.</td>
<td>68</td>
</tr>
<tr>
<td>6.14</td>
<td>Performance of direction preserving linear program for attainable and unattainable commands.</td>
<td>70</td>
</tr>
<tr>
<td>6.15</td>
<td>Performance of reduced direction preserving linear program for attainable and unattainable commands.</td>
<td>71</td>
</tr>
<tr>
<td>6.16</td>
<td>Performance of dual branch linear program for attainable and unattainable commands.</td>
<td>72</td>
</tr>
<tr>
<td>6.17</td>
<td>Performance of single branch linear program for attainable and unattainable commands.</td>
<td>74</td>
</tr>
<tr>
<td>6.18</td>
<td>Performance of mixed optimization linear program for attainable and unattainable commands. In each case the mean performance is indicated by the solid line and the maximum by the dotted line.</td>
<td>75</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Dimensions of Control Allocation Formulations</td>
<td>48</td>
</tr>
<tr>
<td>6.1</td>
<td>Least squares fit of cputime(msec) = a m + b to mean attainable Direction Preserving data.</td>
<td>70</td>
</tr>
<tr>
<td>6.2</td>
<td>Least squares fit of cputime(msec) = a m + b to mean attainable Reduced Direction Preserving data.</td>
<td>71</td>
</tr>
<tr>
<td>6.3</td>
<td>Least squares fit of cputime(msec) = a m + b to mean attainable Dual Branch data.</td>
<td>73</td>
</tr>
<tr>
<td>6.4</td>
<td>Least squares fit of cputime(msec) = a m + b to mean attainable Single Branch data.</td>
<td>74</td>
</tr>
<tr>
<td>6.5</td>
<td>Least squares fit of cputime (msec) = a m + b to mean attainable Mixed Optimization data.</td>
<td>75</td>
</tr>
</tbody>
</table>
Nomenclature

\(0_{n \times m}\) \(\quad\) \(n \times m\) zero matrix

\(1_{n \times m}\) \(\quad\) \(n \times m\) matrix of ones

\(A\) \(\quad\) \(l \times k\) equality constraint matrix for linear programs

\(B\) \(\quad\) \(n \times m\) control effectiveness matrix

\(b\) \(\quad\) \(l\) vector right-hand side of equality constraints for linear programs

\(c\) \(\quad\) Cost coefficient \(k\) vector for linear programs

\(h\) \(\quad\) Variable upper bound \(k\) vector for linear programs

\(I_n\) \(\quad\) \(n \times n\) matrix identity

\(k\) \(\quad\) Number of linear programming unknowns

\(l\) \(\quad\) Number of linear programming equality constraints

\(m\) \(\quad\) Number of control effectors

\(n\) \(\quad\) Number of objectives

\(T_1\) \(\quad\) Transformation to align problem with \(x_1\)

\(T_{2-n}\) \(\quad\) Transformation to search for edge crossings

\(u\) \(\quad\) \(m\) vector of control effectors

\(u_p\) \(\quad\) Preferred control vector

\(u_l\) \(\quad\) \(m\) vector of control effector lower bounds

\(u_u\) \(\quad\) \(m\) vector of control effector upper bounds

\(W_d\) \(\quad\) \(n\) vector of objective weights

\(W_u\) \(\quad\) \(m\) vector of control weights

\(x\) \(\quad\) Unknown \(k\) vector for linear programs

\(x_i\) \(\quad\) Coordinate axis in objective space

\(y\) \(\quad\) vector of objectives

\(y_d\) \(\quad\) \(n\) vector of desired objectives
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{y}_d$</td>
<td>Unit vector in desired objective direction</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Objective scaling parameter (used in direction-preserving linear programs)</td>
</tr>
<tr>
<td>$\Pi$</td>
<td>Obtained objective set</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>Attainable objective set</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Angle used to parameterize transformations in BESA</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>Attainable control set</td>
</tr>
<tr>
<td>$\partial(X)$</td>
<td>Boundary of the set $X$</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Historically aircraft control has been accomplished by providing the pilot with individual cockpit controls (e.g. stick and pedals) intended to provide control about the three rotational degrees of freedom. Traditionally, each of these cockpit control inceptors is tied to an individual surface or group of surfaces (elevator, ailerons, or rudder). Each of these surfaces is intended to generate moments primarily about a single axis.

The coupling of roll and yaw effects due to rudder and aileron deflections led to the common usage of an aileron-rudder-interconnect (ARI) to remove the coupling from the pilot’s commands [21]. Longitudinal control was usually considered decoupled from the other axes. When present, additional surfaces such as flaps or speed brakes were usually treated as configuration changes to be triggered by the pilot or scheduled.

Redundant suites of control surfaces have become more common. Traditional surfaces like ailerons and rudders have been split to reduce individual actuator demands and increase redundancy[52]. New surfaces, novel aerodynamic controls[62] and new non-aerodynamic controls such as thrust vectoring have been added[15]. To refer to this growing set of surfaces and other moment generators, the term control effectors is used. Future control concepts include using large arrays of small surfaces that could drive the number of control effectors even higher still[51].

As the number of control effectors grows, the relationship between the moment generated about a given axis and an individual effector is no longer necessarily clear. In some cases, groups of effectors can be thought of as related to specific moment commands, and they can be ganged together. In other cases, the same effectors can be used to command multiple objectives (i.e. a pair of elevators could be used symmetrically for pitch or asymmetrically for roll control). In this case, some sort of control allocation scheme needs to be developed to decide how to utilize the effectors.

When there are more effectors present than necessary to generate independent pitch, roll, and yaw moments, multiple combinations of effectors can be found which provide the same
moments\cite{9}. The extra freedom offered by having multiple solutions can be used to provide redundancy in case of damage or to meet secondary objectives having to do with surface positions. By working on minimizing trim drag or the use of specific effectors, the control designer can potentially affect quantities such as life-cycle cost through the choice of control usage.

Previous research into aircraft control allocation has focused on the problem of choosing control positions to satisfy demands for specific angular accelerations generated by the control law\cite{9}. Methods for satisfying the largest possible range of these demands have been developed and are termed optimal. One of the problems encountered with such optimal methods is the number and complexity of the computations required. Faster techniques for optimally allocating controls for three moments have been developed that could be used in real time operations\cite{27, 26, 50}.

While traditional aircraft control design often disregards the force-generating effects of the control effectors, redundant effectors could be used to offer control over extra degrees of freedom for the aircraft motion. Some aircraft have attempted to use their redundant controls to control one or more of the translational degrees of freedom. Researchers using the X-22 and VAAC Harrier have investigated combinations of cockpit controls to allow pilot commands for this translational motion\cite{39, 41}. Direct force control in response to traditional pilot commands has been suggested for improving the instantaneous response to load factor commands\cite{60}, as well as for in-flight simulation \cite{53, 17}.

This research focuses on algorithms for solving redundant control allocation problems to meet four or more objectives with control effectors constrained by upper and lower limits. The initial focus is on extending an existing fast three-objective algorithm to handle a fourth objective. Improvements are made to allow allocation for an arbitrary number of objectives. Because the intention is to eventually perform these computations online as part of a control system, the focus is on the computational performance of these methods. Some of the issues related to the application of these methods are also addressed.

This dissertation is divided into four main sections. Chapters 2 and 3 provide introductory material useful in the discussion of control allocation. Chapter 4 stands alone as a description of the extension of a three-dimensional control allocator to four dimensions. Chapters 5 and 6 treat implementations designed to handle an arbitrary number of dimensions. Conclusions and some direction for future research is found in Chapter 7.

Chapter 2 of this dissertation provides the reader with an overview of the control allocation problem studied in this research. A concept for how such a control allocator fits into the overall control law, as well as a more specific discussion of the problem to be solved, is presented. Some terminology adopted from previous research is also presented. A partial history of prior research into control allocation is provided in Chapter 3. The focus is on flight control applications, though some discussion of other applications is included.

The description of the work in this dissertation begins in Chapter 4. The Bisecting Edge
Search Algorithm (BESA) is a fast method of allocating controls for three objectives[25]. Chapter 4 begins with two attempts to extend this algorithm into four dimensions. The second approach motivates a closer look at the original BESA implementation. Results from a successful method for allocating four controls finishes the section.

Chapters 5 and 6 work in parallel to described allocation for an arbitrary number of dimensions. Chapter 5 is broken into two parts. The first describes the development of a recursive bisecting search for allocating controls. The second focuses on linear programming approaches. The major sections in Chapter 6 parallel those in Chapter 5 and present numerical results from the implementation of these methods. The reader has the option of reading these chapters in order or following a specific topic.

Chapter 7 provides conclusions to the work presented and offers suggestions for the direction of future work.
Chapter 2

Problem Statement

This research focuses on algorithms for solving the linear control allocation problem described by Durham [21]. The original definition conveys the specific task of finding combinations of control effectors to generate a specific moment command. To reflect the control effectors’ abilities to generate effects in addition to moments, this research will refer to control outputs as “objectives”.

The task of finding control combinations to achieve a specific objective is only a portion of the larger control problem. Section 2.1 describes the role of the control allocator as part of the larger system. Treating control allocation independently of the other parts of the larger problem is convenient for this research, but the interaction between the various parts can be important to the performance of the larger system[48].

The linear control allocation problem is a special case of the more general problems described above. Several useful observations about the geometry of the three-objective problem have been made by Durham[23]. Many of these same observations apply to cases with more objectives. The linear problem is stated in Section 2.2 and terminology for discussing the problem is introduced.

2.1 Control Law Structure

Many authors have suggested a modular flight control structure similar to that presented in Figure 2.1[15, 56, 47]. The portion of the system responsible for following the desired response is conceptually separated from that which is responsible for handling control redundancy. The control law which relates the desired response to a set of commands is not dependent on the design of the control allocation system relating these commands to actual effector positions. This structure may make it easier to deal with online identification of control effectiveness and in-flight reconfiguration to handle failed or damaged effectors. Such a
Aircraft flight control systems are often thought of as meeting their objectives by commanding some desired moment to be generated by the control surfaces. In these cases, the separation between the command generating and control allocation portions of the system is readily apparent. Many control techniques, including model following\cite{29}, dynamic inversion\cite{40}, backstepping\cite{57}, and sliding mode control\cite{55}, can be posed in a way which requires finding controls for some objectives through the inverse of the control effectiveness map. If this inversion can be considered separately as a control allocation block, then such systems can be put in the form of Figure 2.1.

In many classical aircraft control problems, the desired response is achieved by feeding back output quantities directly to compute control positions\cite{58}. In these systems, it is often difficult to separate and label sections of the system as “command generating” and “control allocating.” When modifying previously designed control systems, obscure coordinate systems or non-obvious assumptions can make it difficult to adopt the form in Figure 2.1. In such difficult cases, an approach where the effector commands calculated by the original system are “deallocated” to form the command inputs for the control allocation block has been successfully used\cite{18, 56}.

Often the structure of the system being controlled will lead to a natural choice of variables for the desired objectives. Viewing aircraft control effectors as moment generators leads to the choice of control-generated moments as the desired outputs. An alternative way of viewing control effectors is to transform their effects into modal coordinates and view their effectiveness as commands for individual mode commands.

As effectors are added to the system, the control allocation block is responsible for handling the resulting redundancy. Criteria for handling redundancy can either be viewed as secondary objectives provided by the control law, or as mission objectives that can be scheduled. When these additional requirements are specified as functions of the effector positions, it is important that the allocation scheme be invariant to changes in the units used to measure effector values\cite{20}. Changes in units can arise when considering mixed types of effectors which may have very different physical forms. Mixing aerodynamic surfaces with values specified in radians of deflection with thrusters whose values are specified in pounds of thrust is one example of mixed units.
The addition of effectors may offer the ability to allocate a larger number of primary objectives. For aircraft systems, the most obvious extension is to view the effectors as force generators and consider the control-generated lift, drag, and side force in addition to moments. Commands for which the control effectors are allocated are not limited to the six rigid body degrees of freedom. Other quantities which are functions of effector values and for which specific desired values are found can also be used as desired commands. When the desired combination of objectives is beyond the capabilities of the effectors, some compromise between commands must be achieved. Objectives measured in differing units can create the same problems with unit changes described above.

Because real control effectors often have practical limitations on their capabilities, one important task of the control allocation block becomes not only resolving redundancy when multiple solutions can be found, but also determining the control solution to be returned in the case where no solution can be found. It has been suggested[24] that the responsibility for resolving unattainable commands more properly rests with the control law which should not request unattainable objectives. In order to limit the commands, command-limiting control laws require similar information about the capabilities of the controls[7] to that required by control allocators. Even if a control system does not limit the commanded objectives, information on effector saturation can be useful for preventing control law integrators from becoming saturated[14, 61].

2.2 Linear Control Allocation

This research focuses on the control allocation block for a specific subset of control problems. The problems of interest are those where the control effectors map to the outputs commanded through a linear relationship,

\[ y = Bu \]  

(2.1)

where \( y \in \mathbb{R}^n \) is the vector of outputs, \( u \in \mathbb{R}^m \) is the control vector, and \( B \in \mathbb{R}^{n \times m} \) is the control effectiveness matrix relating the \( y \) and \( u \). Systems where the control effectiveness is linear are frequently encountered. Leedy[42] and Bolling[8] described ways to treat cases where the control effectiveness is a function of the states or effector positions through the use of local linear approximations.

This research considers systems where the control effectors have upper and lower bounds, \( u_u \) and \( u_l \):

\[ u_l \leq u_i \leq u_u, \quad i = \{1...m\} \]  

(2.2)

Physical controls present in real systems typically have such constraints, so this restriction is not too strict. Methods similar to those presented here have also been extended to deal
with controls with bounded rate capabilities[22].

2.2.1 Geometry of attainable objectives

The bounds on the individual effectors restrict the control vector to an $m$-dimensional “box”, denoted $\Omega$. The boundary of this region is the set of control vectors with at least one of the elements equal to one of the limits. Typically, this boundary is denoted $\partial(\Omega)$. Because $\Omega$ is closed, bounded, and convex, its image under the linear map, $B$, into objectives is closed, bounded, and convex as well. The image of $\Omega$ under the control effectiveness map is an $n$-dimensional polytope denoted $\Phi$. The symbol $\partial(\Phi)$ will be used for the boundary of $\Phi$. The geometry of $\Phi$ is discussed in detail by Durham and Bordignon[21, 22, 9].

This research deals primarily with systems where the dimension of the control vector is greater than the dimension of the objectives, $m > n$. For this case, the map from $\Omega$ to $\Phi$ is “onto”, but not “one-to-one”. This is true because the matrix $B$ has a null-space of at least dimension $m - n$. Any two control vectors whose difference lies in the null-space of $B$, $N(B)$, will map to the same vector in the objective space.

$$\begin{align*}
    u^\perp &= u_2 - u_1 \in N(B) \Rightarrow Bu^\perp = 0 \\
    Bu_1 &= Bu_1 + 0 = Bu_1 + B(u_2 - u_1) = Bu_2
\end{align*}$$

Consider the case where $n = 2$, $m = 3$, and $B \in \mathbb{R}^{2 \times 3}$. The relationship of $\Omega$ and $\Phi$ in this case is easy to visualize. The controls are restricted to lie in a rectangular prism(Figure 2.2). The boundary of this box, $\partial(\Omega)$ is made up of vertices where all three controls are at a limit, edges with two limited controls, and faces where only one control is limited.

In the case above, where there are only two objectives, $B$ projects this box into a plane. One such projection is shown in Figure 2.3. This two-dimensional figure is $\Phi$ and its boundary is $\partial(\Phi)$. Not all points on $\partial(\Omega)$ map to $\partial(\Phi)$; instead some of them map to the interior of $\Phi$. Each of the six edges which make up $\partial(\Phi)$ map from an edge in control space. To be on the boundary of $\Phi$, no more than one control is free to vary.

When there are more controls, $\partial(\Omega)$ becomes a $m$-dimensional hyper-box. This becomes difficult to draw; however the mapping into two dimensions remains the same. The boundary, $\partial(\Phi)$, is still composed of edges where no more than one control is free to vary, only now there are $2m$ such edges. Figure 2.4 shows the projection of a four-dimensional hyper-box into the plane.

Figure 2.5 shows an example of $\Phi$ for a case where $m = 6$ and $n = 3$. The boundary, $\partial(\Phi)$, is made up of two-dimensional facets. Each of these facets maps from a two-dimensional face on $\partial(\Omega)$. All but two controls are at their limits on these facets. In three dimensions, there are $2m(m - 1)$ two-dimensional facets on $\partial(\Phi)$.

It is difficult to visualize $\Phi$ for more than three objectives. Figure 2.6 presents an attempt
Upper and lower limits on $u_1$, $u_2$, and $u_3$ limit the controls to a rectangular box.

\[\Omega\] for three controls.

$\Phi$ is the image of $\Omega$ in two dimensions. The edges which make up $\partial(\Phi)$ (solid lines) are a subset of the images of the edges on $\partial(\Omega)$ (dotted and solid lines).

\[\Phi\] for three controls and two objectives.
Figure 2.4: $\Phi$ for four controls and two objectives. Note that the boundary, $\partial(\Phi)$ is still made up of one-dimensional line segments. The dotted lines represent edges on $\partial(\Omega)$ whose image is on the interior of $\Phi$.

Figure 2.5: $\Phi$ for four controls and three objectives. The wire frame shows that a two-dimensional facet of $\partial(\Phi)$ is a parallelogram.
to display $\Phi$ for a four-objective, six-control problem. One of the three-dimensional “facets” of $\partial(\Phi)$ is highlighted. This entire parallelepiped is considered to be part of the boundary of the four-dimensional figure. In four dimensions, there are $\frac{m}{3}(m - 1)(m - 2)$ such facets.

In general $\partial(\Phi)$ is made up of $2 \frac{m!}{(n-1)!(m-n+1)!} (n - 1)$-dimensional facets. On each facet, all but $n - 1$ of the controls are fixed at their limits. It can be seen that the number of facets on the boundary is of the order $m^{n-1}$.

### 2.2.1.1 Objects in $\Phi$ and $\Omega$

In working with linear control allocation problems and algorithms for their solutions, the discussion of edges, facets, and vertices of various dimension quickly becomes cumbersome. A system for describing the component objects which make up $\Phi$ and $\Omega$ is described by Durham[25] and will be adopted for this dissertation. An overview of this system is presented below.

$\Omega$ and $\Phi$ are referred to as made up of objects. Objects are considered to be closed subsets of certain linear varieties in control space and their images in objective space[9]. Objects can be described by a set number of controls at fixed positions with the other controls free to vary within their limits.
The dimension of an object will be used to refer to the number of controls which are free to vary. Thus \( \Phi \) or \( \Omega \) can be thought of as objects of dimension \( m \), while vertices of \( \partial(\Omega) \) have dimension 0. Using this definition, the facets of \( \partial(\Phi) \) are of dimension \( n - 1 \) even though there are \( n \) objectives. It is common to see references to the controls which “define” some object. These controls are considered to be those which are not fixed for the given object.

Two operations are defined that operate on these objects. The union of two objects is found by comparing the positions of the fixed controls. Any of the fixed controls which differ in position are considered to be free in the union. Any controls which are free in either of the initial objects are also free in the union. The dimension of the object formed by taking the union of two objects is not necessarily equal to the sum of the dimensions of the original object.

The intersection between two objects is also carried out through an element by element comparison of the controls. If any control is fixed at one limit in the first object and the opposite limit in the other object, then the two have no intersection. If the intersection exists, it has free controls for any controls that are free to vary in both original objects and fixed controls for those that are fixed in at least one of the objects. In general, the intersection of two objects is an object of lower dimension.

A numerical scheme was developed to deal with these object definitions. The initial scheme presented by Durham [21] was later modified and a new version was described in Reference [25]. This newer scheme involves using a vector of integers to represent an object in \( \Phi \) and \( \Omega \). The elements of this vector are allowed to take one of three values based on the corresponding control: 1 if the control is at its upper bound, -1 if it is at its lower bound, or 0 if the control is free to vary on that object. Thus \((1, 0, -1, 1, 1)^T\) represents a one-dimensional portion of a five-control problem, where the first, fourth, and fifth controls are at the upper bound, the third control is at its lower bound, and the second control is free to vary. Similarly \((0, 0, 0, 0, 0)^T\) represents all of \( \Omega \) or \( \Phi \) for a five-control problem.

This numerical scheme is useful for discussing the details of problem solutions. It is also useful in the development and comparison of numerical algorithms to solve these problems. Keeping track of variables in an integer format which is standardized across implementations avoids difficulties with floating point equality comparisons and also can make it easier to interpret the action of a piece of code.

### 2.2.2 Terminology

In addition to the terminology presented above, this dissertation will also make use of the more informal “edge”, “facet”, and “vertex” to describe elements in \( \partial(\Omega) \) and \( \partial(\Phi) \). For problems with three objective dimensions, these elements have dimensions one, two, and zero, respectively. For problems with more than three objectives, a slight abuse of this terminology will allow the discussion of \( \partial(\Phi) \) as made up of \((n - 1)\)-dimensional “facets”. 
These facets are themselves thought of as containing lower dimensional “facets”.

Another slightly informal use of terminology arises in the discussion of the size of control allocation problems. The dimension of a problem will be considered to be the number of objectives. Thus, a 14-control, six-objective problem will be said to be a “six-dimensional problem”. Since the focus is on control allocation problems, the danger of confusing cockpit controls, “ceptors”, with airframe controls, “effectors”, is minimal. The term “controls” will be used to describe control effectors; any chance for confusion in the discussion will made clear by context.
Chapter 3

Background and Literature Review

3.1 History

Much of the history and early work on the control allocation problem was well summarized by Bordignon[9]. In his dissertation, Bordignon described several methods proposed early-on which allocate controls but which fail to allow allocation of commands throughout \( \Phi[9] \). These methods include various ad hoc schemes, methods that fall under the general category of right-generalized inverse methods, and methods which daisy chain groups of controls.

The ad hoc methods cited by Bordignon generally depend on the control designer to use engineering judgment to assign individual control effectors to specific moment commands. By their design, such systems are unable to make use of the individual effector’s capabilities to generate moments in axes other than that chosen by the designer [9]. These methods must either explicitly use combinations of effectors chosen to minimize off-axis effects or they must ignore these effects and treat them as disturbances to be handled by the control law. Symmetric deflections of horizontal tail or coordinated aileron and rudder movements are common attempts to minimize these effects.

For systems with an equal number of effectors and objectives, the obvious method for utilizing all of the effects of controls is to invert the control effectiveness matrix. The extension of this method for under-determined systems is to use a generalized inverse (GI) matrix. A right generalized inverse of a matrix \( B \) is a matrix \( B^\dagger \) where \( BB^\dagger = I_n[9] \). A solution to \( y_d = Bu \) can be found using \( B^\dagger \):

\[
y_d = Bu \Rightarrow BB^\dagger y_d = Bu \Rightarrow u = B^\dagger y_d
\]  \hspace{1cm} (3.1)

Because this approach is the one suggested by many textbooks on control systems[2], the history of its use is difficult to divine. Generalized inverses commonly show up in the literature as part of control systems designed by researchers not focused just on the control
allocation portion of the problem. One of the most common choices for a generalized inverse is the Moore-Penrose pseudo-inverse, $B^T (BB^T)^{-1}$. In the absence of constraints on the control vector, this inverse minimizes the two-norm of the control deflection. A similar inverse can be chosen to minimize a weighted norm, $(u^T W u)^{\frac{1}{2}}$, where $W$ is a weighting matrix which is usually chosen to be diagonal[10]. Durham and Bordignon presented several interesting ways in which generalized inverses can be chosen [10]. Generalized inverse solutions have the advantages of being relatively simple to compute and allowing some specification on the use of controls. Because of these advantages, GI solutions are frequently encountered[9].

The biggest drawback to generalized inverse solutions lies in determining how to handle constrained controls. Durham demonstrated that, except in certain degenerate cases, a generalized inverse cannot allocate controls within $\Omega$ which will map to all of $\Phi$[21]. Two methods are suggested to handle objectives which cannot be allocated. One of these is to calculate the GI solution and truncate any controls which exceed their limits. The other maintains the direction of the objective command by finding the largest scaling factor $a$ with $0 \leq a \leq 1$ which satisfies $u = a B^T y_d$ without violating the control constraints[9].

Even when the controls do not saturate, care must be taken in choosing the generalized inverse to use. Doty, et al, cautioned that, when using weighted pseudo-inverse solutions for problems where the control effectors are measured in different physical dimensions, the elements of the weighting matrix must be chosen carefully if the resulting solution is to be invariant to changes in units and coordinate systems[20].

### 3.2 Non-Optimal Methods

The notion that some control allocation algorithms are unable to allocate controls for all of the objective vectors in $\Phi$ leads to a metric for comparing control allocation algorithms. By computing the size of the set, $\Pi \subseteq \Phi$, for which a given allocator can allocate admissible controls ($u \in \Omega$), allocation schemes can be compared. For the three-dimensional case often considered in aircraft flight control, comparing the volume of $\Pi$ and $\Phi$ is such a metric.

The use of the volume of the attained objective set as a metric for comparing control allocators was suggested by Durham[21]. Results comparing several typical methods were presented by Durham and Bordignon[10]. Many common methods have attainable sets that are convex for which the volume is relatively simple to compute. For those methods where the determination of $\Pi$ is difficult, or where $\Pi$ is not convex, Bordignon and Durham suggested a numerical approach for determining points in $\Pi$ and then approximating the volume of $\Pi$ as the volume of the convex hull of these points.

It is possible to create allocators that only attain objectives in a set which is non-convex, or even in a set which is not well connected. For such allocators, the volume may be a poor indicator of performance; however, for most practical allocators, the volume has been a good method for comparing allocators[9]. Previous works show that the volume of generalized
inverse solutions is often much less than half of that of $\Phi^{[10]}$ (for the popular pseudo-inverse solutions, the volume is generally even smaller). Methods for allocating controls exist which will allocate controls for the entirety of the attainable set. Such methods are termed “optimal”. Methods which allocate over a smaller set are “non-optimal”.

Other metrics have been proposed for comparing the “open-loop” performance of control allocators. One such method is to compare the ability of allocators to attain objectives along some specific time history. Such a comparison may yield different results than just comparing “volumes” of $\Pi$, as one allocator may do better in a specific objective direction, but have less volume. For optimal allocators, those for which $\Pi$ equals $\Phi$, the performance on such a metric will be identical$^{[48]}$.

3.2.1 Daisy Chain Solutions

Because a single generalized inverse is unable to yield admissible controls for all of the region of attainable objectives, there may still be control power available to improve even when some of the effectors are saturated. A second allocation step may be able to make use of that additional control power$^{[9, 61]}$. One scheme making use of multiple steps is Daisy Chaining. A daisy chain allocator partitions the control effectors into two sets, so that objectives unattainable by the first are allocated with the second.

Daisy chaining control allocators originated with a desire to prioritize groups of control effectors, using a second set of effectors only when the first set is unable to met the demands$^{[3]}$. While generally proposed using generalized inverse methods, any sort of control allocation scheme could be daisy chained—first allocating only the “primary” controls, and then later allocating the “secondary” set using the residual objective $y_{d2} = y_d - Bu_1$. Figure 3.1 illustrates the daisy chain approach.

![Block diagram showing daisy chaining allocator](image)

Figure 3.1: Block diagram showing daisy chaining allocator

Conceptually, daisy chaining control allocation is appealing for several reasons: chiefly the relative simplicity and the ability to limit the use of certain control effectors$^{[9]}$. Bordignon and Durham pointed out several shortcomings of these methods. The most important drawback is the inability to allocate controls that generate the correct command in some portions of $\Phi$$^{[9, 21]}$. Additionally, they demonstrated that the output for unobtainable commands could be different from the desired objective in both magnitude and direction. Berg et al$^{[3]}$
demonstrated a further drawback, a phase delay in the output of a daisy chain in response to inputs that rate saturate the individual sets of controls in rate limited systems.

### 3.2.2 Multiple Pass Inverses

Other methods of using multiple steps to handle unobtainable commands have been suggested[9]. Rather than limiting the use of certain controls, these methods are motivated by the desire to make use of control authority remaining even when some controls are saturated[61].

One such scheme was suggested by Virnig and Bodden[61] as a method of “redistributing” effectors to deal with unattained commands in a V/STOL aircraft control law. Their baseline control allocator was an inverse-based method which allocates effectors to match the commands which would be generated by “generalized” controls intended to represent the controlled degrees of freedom. When controls saturated, the difference between the commanded and actual actuator positions was mapped back into a set of objectives to be allocated as increments to the unsaturated controls. Virnig and Bodden handled the still unobtained commands by weighting the control mapping to prioritize the axes. One interesting suggestion made was to use measured control effector positions to sense the discrepancy rather than internal models. This approach was also used by Eberhardt and Ward in the design of the control system for a tailless aircraft[30].

### 3.2.3 Cascaded Generalized Inverse Solutions

Commands still unobtainable after the first redistribution step can be addressed through additional redistribution steps. This is the basic idea of cascaded-generalized-inverse (CGI) methods. As originally presented by Bordignon[9], CGI allocation occurred as a series of generalized inverse steps. At each step, a generalized inverse $B^{-1}$ was used to allocate controls, $u_k = B_k^{-1}y_k$, and the controls in $u_k$ exceeding their limits were set at the corresponding limit and removed from $B_{k+1}$ and $u_{k+1}$. The unobtained moment for the step was computed, $y_{k+1} = y_k - B_ku_k$, and another generalized inverse step was taken with the reduced control list, $u_{k+1} = B_{k+1}^{-1}y_{k+1}$. This process repeated until the resulting problem obtained the command, $y_{k+1} = 0$, or there were $n$ or fewer remaining controls. For these cases, a least squares approach was adopted—once again removing saturated controls at each step.

Such a method, while considerably slower than the generalized inverse method, will take, at most, $m$ iterations. If each individual GI iteration takes on the order of $m$ operations, then the worst case computational requirements for CGI allocation will scale with $m^2$. Computational results presented by Durham [25] show that the average case tends to scale more as a linear function of $m$. This method of CGI has been observed to allocate controls for almost all obtainable commands [9]. However, unobtainable commands may yield answers
with much different magnitude and direction than desired, possibly even on the interior of the $\Phi$\cite{9}.

To improve the performance when allocating for unobtainable commands, the CGI method can treat each GI step as a “Scaled GI” rather than a “Truncated GI”. By removing only the single control most saturated at each step, the solution can be forced to preserve direction at the expense of extra computational steps. This method would be forced to take the full $m$ iterations similar to the worst case described above.

### 3.3 Optimal Methods

None of the methods presented above are optimal because they do not produce admissible control deflections for all attainable objectives. Bordignon noted that this behavior “translates directly into a loss in potential maneuverability or to the extra weight of controls which must be borne to achieve some desired level of maneuverability”\cite{9}. The desire to avoid these problems leads to an interest in control allocators that are optimal in the volume sense.

Some control allocation schemes are seen to yield admissible controls for nearly all of the attainable set. These nearly optimal methods may be close enough that, in a practical sense, their performance is not significantly different than an optimal allocator. The previous CGI methods have been shown to be nearly optimal\cite{9}, though to handle unattainable commands, the computational cost is high.

Durham presented a compromise between the speed and simplicity of a generalized inverse allocator and the objective capability of optimal allocators which approximates $\Phi$ for a three-dimensional problem with a smaller polyhedron made up of 48 triangular pyramids whose apexes are at the origin. Six pyramids were chosen in each octant so that determining which pyramid contains the desired objective direction was reduced to an exercise of comparing the signs and magnitudes of the objective components. Once the correct pyramid was found, a generalized inverse solution was used to solve for the control solution. The volume of $\Pi$ for this method was found to be close to that of $\Phi$, and the online computations were relatively simple; however, the \emph{a priori} computation and storage of 48 generalized inverses was required for each control effectiveness matrix considered. In order to come the closest to an optimal solution, a numerical optimization step was also required to choose the elements of a transformation applied to $B$.

Bodson and Petersen proposed another method based on choosing from a set of pre-computed solutions based on the objective direction\cite{4}. Their method for three-dimensional problems used spherical coordinates to map the boundaries of all of the facets of $\Phi$ to two dimensions. This two-dimensional map can be rapidly searched to find the desired facet\cite{4}. Given the facet, constructing the solution is simple\cite{22}. Because all of the facets of $\Phi$ can be located, this method is optimal. It has also been extended to deal with cases where some of the facets of $\Phi$ are coplanar\cite{50, 5}. Like the method proposed by Durham, this method requires the
pre-computation of $\Phi$. Bodson and Petersen suggested that this computation could be done online if undertaken at a slower rate and in parallel with the allocation loop.

Other methods which attempt to search pre-computed solutions for optimal or sub-optimal results include diverse approaches such as a neural network where the training data is precomputed[34] and a method that represents generalized inverses as wavelets[36]. Requiring a priori knowledge of the control effectiveness matrix limits one of the potential advantages to a redundant suite of effectors, the ability to reconfigure controls to deal with failures or other changing requirements. Two groupings for the optimal allocation schemes considered in this work are direct allocation and optimization based methods.

### 3.3.1 Direct Allocation Solutions

Much of the original research at Virginia Tech has been in methods that allocate controls using knowledge of the geometry of $\Phi$ and $\Omega$. Early papers by Durham[21, 22] describe this geometry. The work by Durham, Bordignon, Bolling, and Leedy involved using the method referred to as Direct Allocation to solve the control allocation problem[21, 23, 22, 8, 42]. Direct allocation attempts to find the intersection of a ray in the direction of the desired objective starting at the origin with $\partial(\Phi)$. The desired control solution is then scaled back from this intersection.

#### 3.3.1.1 Facet Searching

A method known as a “facet-search” was originally used to solve the direct allocation problem. This brute force approach generates facets of $\partial(\Phi)$ and tests to see if they intersect the desired direction. Facets are generated until the one containing the solution is found. While this approach has been streamlined[22] and successfully implemented in a real-time piloted simulation [42], the computational requirements of this approach are quite large[25].

As originally proposed[21], the facet search generated facets of $\partial(\Omega)$, mapped them to $\Phi$, and tested to see whether the mapped object contained the desired direction and was on $\partial(\Phi)$. For the three-dimensional case, this requires checking $2m^2m(m-1)$ objects for the intersection, each check involving the solution of a $3 \times 3$ system. Later improvements involved only checking facets on $\partial(\Phi)[22]$. For the three-dimensional case, this works out to $m(m-1)$ objects checked. When expanded to higher dimensions, the relationship is $2^{\frac{m!}{(n-1)!(m-n+1)!}}$.

#### 3.3.1.2 Bisecting Edge Search

An improved method for direct allocation has been presented by Durham[27] and implemented in piloted simulations by Scalera[56] and Nelson[47]. Called the Bisecting Edge Search Algorithm (BESA), this method constructs a series of edges on $\partial(\Phi)$ as part of a
search designed to converge on edges which define the solution facet. The BESA search is generally terminated after a fixed number of iterations. Since each iteration scales linearly with the number of controls, the entire method scales as a linear function of $m$. Durham showed this method to be significantly faster than the facet search; at 15 controls the average case saw a four-fold increase in speed[25].

As presented, the edge search terminates after a fixed number of iterations[25]. Sometimes this will occur before the desired solution is found. The number of cases where this occurs was found to be low (on the order of a few cases out of 100)[25]. When the algorithm terminates without finding the solution, some amount of information about $\partial(\Phi)$ in the vicinity of the solution facet is available. This information can be used to estimate the solution for these cases. Results from tests using the BESA with an included estimator showed the average magnitude of the errors to be small and to get even smaller as $m$ was increased. Because the direction of the desired moment is preserved in the estimate, the moment obtained is the same for the scaled solution if the desired moment is interior to the estimation of the boundary.

3.3.2 Optimization Based Solutions

If the control redundancy is to be exploited by optimizing a secondary objective that is a function of the control positions, the control allocation problem can be stated as a standard constrained optimization problem.

$$\min_u J(u)$$

$$Bu = y_d$$

$$u_l \leq u \leq u_u$$

Above, $J(u)$ represents a scalar-valued function of the control positions to be minimized and the standard control allocation problem shows up in the constraints. Once the problem is posed in this way, any of a number of standard optimization techniques could be used to solve it. If $y_d$ is attainable, the constraints ensure that the actual objectives attained equal those desired.

A potential problem with this straightforward approach arises when the desired objective is unattainable. For the case where the cost function is linear, Bordignon noted that, while traditional linear programming solvers will indicate that the constraints are infeasible, they offer no guidance as to what solution to use[9]. To resolve this problem, most authors who consider using standard optimization techniques consider alternate formulations. Two of the most popular serve to preserve direction (Figure 3.3), or to minimize objective error (Figure
3.4) for unattainable commands:

\[
\begin{align*}
\min_{u, \lambda} & -\lambda \\
Bu &= \lambda y_d \\
0 &\leq \lambda \leq 1 \\
u_l &\leq u \leq u_u
\end{align*}
\] (3.3)

\[
\begin{align*}
\min_u &\|Bu - y_d\| \\
u_l &\leq u \leq u_u
\end{align*}
\] (3.4)

\[
\begin{align*}
Bu &\leq y_d \\
u_l &\leq u \leq u_u
\end{align*}
\] (3.5)

3.3.2.1 Quadratic Programming

Enns proposed a solution to the problem by minimizing a quadratic function of the objective error[31].

\[
\min_u \frac{1}{2}(Bu - d)^TW_d(Bu - d), \ W_d > 0
\] (3.6)

Enns approximated \( \Omega \) with a circumscribed ellipse so that, with the introduction of a single Lagrange multiplier, the problem can be solved as a root-finding problem in one variable. In an attempt to reduce the errors caused by the constraint approximation, a multi-pass solution was proposed that—much like a cascaded generalized inverse—removes saturated effectors and repeats the process.

A different approach based on quadratic programming was offered by Burken et al [16]. The control law developed used quadratic programming to redistribute effector commands in presence of a failure detected in one of the surfaces. Given the control effectiveness of the un-failed controls, \( B_z \), and the disturbance caused by the failed surfaces, \( d_z \), a solution for the un-failed controls, \( u_r \), is sought which gives the same response as the original controls and effectiveness, \( u^* \) and \( B \).

\[
B_zu_r + d_z \simeq Bu^*
\] (3.7)

This is equivalent to the statement of the control allocation problem presented in Section 2.2 if \( y_d = Bu^* - d_z \) and the reduced controls and effectiveness, \( u_r \) and \( B_z \), are used. Burken et al attempt to minimize a quadratic function of the error and use a fixed-point iterative approach to solve this problem. This method will yield a solution arbitrarily close to the correct one over all of \( \Phi \), and will approximately minimize the error for solutions exterior to \( \Phi \). The approximate nature of the solution is due to a small term involving the norm of the control positions. This term is added to the cost function to ensure it is strictly convex.
3.3.2.2 Linear Programming

As stated in Section 2.2, the constraints in the control allocation problem are all linear. Several authors have introduced linear cost functions, making the problem a standard linear programming problem.

\[ \min_{\mathbf{x}} \mathbf{c}^T \mathbf{x} \mid \mathbf{A} \mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0 \]  

(3.8)

Equation 3.8 is a linear programming problem in what many texts consider “standard” form[43]. The unknown variables, \( \mathbf{x} \), are non-negative and bounded by equality constraints, \( \mathbf{A} \mathbf{x} = \mathbf{b} \). Some authors use other definitions of standard form where the equality constraints are replaced with inequality in their definitions (and call the form in equation 3.8 “canonical”)[32] or where upper bounds are introduced for the variables[43, 33].

Simplex methods for linear programming originated in the 1940s and have become the dominant approach used for computational solutions of linear programs[63, 33]. One of the advantages of simplex methods is the existence of an upper theoretical bound on the number of iterations needed to find the solution to problems given some simple assumptions. For a problem with \( l \) equality constraints and \( k \) unknowns, this theoretical bound grows with \( l^k \). As the number of unknowns increase, the number of iterations can become quite large. However, for practical problems the average number of iterations has been seen to scale linearly with the number of unknowns[33].

Most of the variation between different simplex implementations is in how they perform the numerical computations at each step[33]. Because many large problems are made up of relatively sparse matrices, many implementations take advantage of techniques for factoring and inverting sparse matrices[43]. Some improvements have also been made in ways to find starting points for the algorithm or to utilize the primal and dual problems at the same time.

By considering only basic feasible solutions, the simplex algorithm can be thought of as visiting the vertices of a convex polytope corresponding to the constraints[49]. In the last 20 years, a new technique for solving linear programming problems has been proposed in which the solution lies interior to the polytope at each iteration. These interior point methods have been shown to have better theoretical performance than the simplex algorithm, though in practice these gains may not be seen[63]. Most of the work on interior point methods has focused on large, sparse systems[63].

Most of the control allocation approaches using linear programming make use of simplex solvers. The major differences occur in how the problem is stated and how unattainable solutions are resolved. Buffington[11] proposed a system in which a nonlinear secondary objective is used to choose among attainable solutions by choosing a linear cost function at each frame which will drive the solution toward the optimum. It was suggested that preserving the direction for unattainable commands may degrade performance for certain systems. To counter this, Buffington proposed prioritizing different portions of the control law command using a sequential series of linear programs resembling the direction preserving
form of equation 3.3.

A common approach appearing in many papers[15] [18, 13] is a “dual branch” method originally suggested by Buffington[12]. In this method, the actual linear programming formulation used depends on whether or not the desired objective is obtainable or unobtainable. One branch would determine if the objective was obtainable, termed “sufficient control” by Buffington, by solving a LP to minimize the weighted 1-norm error.

$$\min_u J = \|W_d(Bu - y_d)\|_1$$

(3.9)

$$u_l \leq u \leq u_u$$

If the control is found to be sufficient, J=0, then a second branch is selected to allocate the given objective minimizing the error with some “preferred” control position. When the control is deficient, the original minimum objective error solution is returned or a linear program is solved which limits the desired objective so that the direction is preserved.

$$\min_{u, \lambda} -\lambda, Bu = \lambda y_d$$

(3.10)

$$u_l \leq u \leq u_u, 0 \leq \lambda \leq 1$$

Bodson suggested a “mixed” form where the two branches of the dual branch formulation are combined into a single cost function,

$$\|Bu - y_d\|_1 + \epsilon\|u - u_p\|_1$$

(3.11)

The parameter $\epsilon$ specifies the tradeoff between meeting the objective and minimizing error from some preferred control location[6]. This is similar to the quadratic cost function in the fixed-point iteration of Burken et al[16] though here the second term is motivated by control sufficiency and not numerical properties.

Most of the applications of linear programming presented make use of solvers based on the standard problem form. The control allocation problem can be converted to this form at the expense of adding variables and constraints to the linear program. In the case of the feasibility branch in the dual branch approach above, as many as $3n + m$ constraints and $4n + 2m$ variables are added to make this transformation[13]. Bodson[6] advocated the use of a simplex method designed to handle bound variables to reduce the number of extra variables required. He presents formulations of the direction preserving and mixed allocation problems designed to minimize the number of constraint equations.

Interior point methods for control allocation do not appear in the literature. One reason for this is that much of their success has been in handling large sparse problems[63], something that control allocation problems generally are not. It has been suggested that primal-dual interior point algorithms’ uniform convergence properties and their ability to evaluate how far a given iteration is from the solution may be valuable in situations where the algorithm can not run to completion[6], but little work has been done to verify this.
3.4 Rate Limiting and Discrete Allocation

The extension of methods which allocate effector positions to systems with rate limited effectors has been addressed in several ways. Berg et al proposed incorporating rate limits explicitly into a daisy chaining method[3]. It has been suggested that rate limits could be included in optimization-based methods through frame-wise modification of the effector bounds[6]. Durham et al[28] investigated discrete allocation where the objective allocated is the change in $y_d$ and the rate limits can be converted to discrete position limits for each frame. In this formulation, the size of the local $\Phi$ is associated with the objective rate capabilities of the effectors. Commands in the interior can be thought of as having extra rate capability and methods for using this capability to minimize secondary functions like control energy, drag, side-force, or error with some preferred location have been proposed[9, 56].

3.5 Realtime Control Allocation

The computation requirements of a particular control allocator are a concern if it is to be used for online control. Leedy[42], Scalera[56], and Nelson[47] have all demonstrated successful piloted simulations using Direct Allocation. Leedy used the facet search in a desktop simulation while Scalera and Nelson applied the BESA approach in a manned flight simulator. These tests demonstrate that the algorithms will execute promptly on the hardware used but offer little guidance for comparing different algorithms. Durham reported timing results of a BESA implementation of three objective direct allocation that shows it scales linearly with the number of controls[25]. The results showed the BESA to be slower than GI methods but faster than CGI methods applied to the same problems. A direction preserving method based on a simplex solution to a linear programming problem was compared to a redistributed pseudo-inverse (RPI) method by Bodson[6]. For a three objective, eight control problem the method was found to take approximately five times as long to execute as the RPI.

Feron and McGovern noted that the use of optimization methods in real-time control becomes tempting as computational capabilities increase[44]. One of the applications they considered was control allocation. Three criteria were suggested for the use of optimization schemes in safety critical applications: guarantee of convergence to a solution, a known upper bound for time to find a solution, and numerical properties such that the size of errors can be controlled. If online optimization algorithms meet these requirements, Feron and McGovern suggested that they considered “information storage and retrieval mechanisms”.
3.6 Applications for Higher Dimensional Control Allocation

More than three objective dimensions can be defined for aircraft control when direct force commands are considered. Direct lift has been used to provide additional control during powered approach[52]. It has also been used in combination with more traditional longitudinal control to improve the response to load factor commands[60]. Side force commands have been suggested as a way to improve dive bombing accuracy[54] and runway lineup maneuvers[35]. In general, pilot response to direct force commands has not been favorable[37]. It has been noted that some pilots prefer force commands to be presented with separate inceptors so that they can be decoupled from normal pilot technique[35].

The research into direct force commands has generally considered a separate surface that may work independently or grouped with some traditional effectors[60]. Systems which have made use of redundant effectors to provide force commands have used traditional generalized inverse methods[35]. Hyde describes a Harrier project in which the longitudinal force and moment commands were mixed by computer to remove the pilots need to manipulate stick, throttle, and nozzle angle simultaneously[39]. This project still resulted in only three objectives for the controls since the longitudinal system was considered to be decoupled from the lateral and directional controls.

One aircraft application for which the need to allocate controls for force and moment commands is readily apparent is in the design of in-flight simulators. The Total In-Flight Simulator (TIFS) is one example. Control allocation in the TIFS model-following control is accomplished through a generalized inverse method[45]. A proposed YT-2B in-flight simulator allocated controls by defining three “independent force controllers” to be used in addition to the standard aircraft controls[53]. These force controllers consisted of a blend of separate controllers scheduled to work together.

Previous work into three-dimensional allocation considered a fourth dimension in the definition of restoring commands[8]. These extra dimensions are not allocated optimally with the other objectives. Instead, some suboptimal method is used to find a control solution in the null space of $\mathbf{B}$ that drives the control solution toward a desired value over time. In the course of this work, it was demonstrated that, even without effectors designed to directly generate forces, redundant effectors have significant ability to change drag[8] and side-force[56].

From a control allocation standpoint, the common theme among these approaches to allocate controls for objectives beyond the standard three is that they are non-optimal. These approaches do not make full use of the effectors present. Improved allocation methods will offer improved performance and may even make new approaches to aircraft control possible.
Chapter 4

Control Allocation For Four Objectives

Much of the interest in control allocation stems from a desire to use reconfigurable control laws that can compensate for damage or inaccurate parameter estimates[4]. Reconfigurable control systems offer the potential to increase reliability and performance by using redundancy in the effectors to compensate for failed or damaged surfaces[16]. In order to exploit this potential, a control system must be able to respond to changes in real time.

Once failures have been detected, they are often modelled by removing control effectors from consideration, modifying the limits on effectors, or changing the elements of the control effectiveness matrix[56]. Online estimation of system parameters can also lead to modifications of $B[12]$. Because these effects change in real time, control allocation methods which require off-line computations cannot be used unless all failure modes can be anticipated and modelled ahead of time or handled with asynchronous computation.

One of the difficulties with optimal or near optimal allocation methods is the computational power required to run them in real time[4]. While more efficient methods for implementing them have been found and used in realtime simulations[42], the basic facet search is still very complex for large numbers of controls (see Section 3.3.1.1). The result of work to find a faster method for direct allocation that does not require off-line computation is the Bisecting Edge Search Algorithm (BESA)[27].

The BESA allocator calculates near-optimal solutions to the three objective control allocation problem. As originally described, the complexity of the BESA approach is seen to scale linearly with the number of controls. The number of computations required compares favorably with other allocation methods including Cascaded Generalized Inverse methods[25].

One drawback of the BESA approach is that it is limited to solving problems with three objectives. The original description of the method involves visualizing $\partial(\Phi)$ as a solid figure in $\mathbb{R}^3$ and performing specific rotations of that figure. The descriptions generated are harder
to visualize in higher dimensions. The efficiency of the method makes it attractive and this research was focused on extending BESA to higher dimensional problems.

4.1 Introduction to the Bisecting Edge Search

A description of the original three-dimensional algorithm is presented in order to better understand the attempts to extend the BESA approach to higher dimensions. For more detail, the reader is referred to the papers of Durham [27, 25].

A solution to the three-objective problem using the Bisecting Edge Search approach is carried out in two steps. First, the desired objective direction is aligned with the \( x_1 \) axis, and then a search is carried out to find the facet of \( \partial(\Phi) \) that the \( x_1 \) axis intersects. Once the solution facet is found it is simple to find the Direct Allocation solution.

4.1.1 Aligning the Desired Objective Direction

Aligning the desired moment with \( x_1 \), while not strictly necessary, simplifies the discussion and implementation of the search for the solution facet. In the implementation described by Durham [27], this alignment is carried out by left-multiplying \( B \) and \( y_d \) by a rotation matrix, \( T_1 \),

\[
y_d^{new} = T_1 y_d^{old} = \begin{pmatrix} y_{d1} \\ 0 \\ 0 \end{pmatrix}
\]

The rotation matrix used is,

\[
T_1 = \begin{bmatrix} \hat{y}_{d1} & \hat{y}_{d2} & \hat{y}_{d3} \\ 0 & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix}
\]

where

\[
t_{31} = \sqrt{1 - \hat{y}_{d1}^2}
n_{32} = -\hat{y}_{d1}\hat{y}_{d2} \frac{t_{31}}{t_{31}}
nt_{33} = -\hat{y}_{d1}\hat{y}_{d3} \frac{t_{31}}{t_{31}}
nt_{22} = -\text{sign}(\hat{y}_{d3})\sqrt{1 - \hat{y}_{d2}^2 - t_{32}^2}
nt_{23} = \text{sign}(\hat{y}_{d3})\frac{\hat{y}_{d2}\hat{y}_{d3} + t_{32}t_{33}}{t_{22}}
\]

(4.3)
In the equations above \( \hat{y}_d = (\hat{y}_{d1}, \hat{y}_{d2}, \hat{y}_{d3})^T \) is a unit vector in the desired objective direction. The same alignment operation can be thought of as an application of a standard Householder reflection [38, 59] chosen to align \( y_d \) and applied to \( y_d \) and \( B \).

4.1.2 Finding the intersection of \( x_1 \) with \( \partial(\Phi) \)

In general, the intersection of the desired objective direction with \( \partial(\Phi) \) is found on a parallelogram facet. Using the operations defined in Section 2.2.1.1, this facet can be constructed from the union of two edges. The solution facet can thus be found by finding two distinct edges whose union makes up this facet.

Durham describes a relatively simple method for finding the intersection of an edge with the \( x_1 \)-axis in the two-dimensional case [27]. When the problem is projected into the \( x_1-x_2 \) plane, this method can be used to identify edges which are also on the boundary of the three-dimensional figure. It is suggested that, by rotating \( \Phi \) about the \( x_1 \)-axis and identifying edges which solve the two-dimensional problem, eventually two edges that define the solution will be found.

There is an angle through which \( \Phi \) can be rotated about \( x_1 \) resulting in a solution facet being parallel to the \( x_3 \)-axis. In this orientation, two edges intersect the \( x_1 \)-axis when the figure is projected into the \( x_1-x_2 \) plane. In three dimensions, these two intersection points will be seen to lie on opposite sides of the \( x_1-x_2 \) plane. Rotation through a slightly smaller angle will result in one of the edges being found; a slightly larger angle will correspond to the other edge.

All rotations of an angle less than that required to align the facet will result in edges being found on the same side of \( x_1-x_2 \), while all larger rotations will find edges on the opposite side. This suggests a simple search procedure for locating the solution facet. Starting with the initial orientation, the problem could be rotated successively by some angle. If the sign of the \( x_3 \) component of the intersection changes, then it is known that the orientation which aligns the facet is in between the last two angles checked. The rotation angle can be cut in half and the direction reversed. Continually bisecting the range of angles eventually leads to the identification of the edges that make up the solution facet.

4.2 Four Objective Bisecting Edge Search

Extending the BESA approach to four dimensions involves looking for edges which define objects on the boundary of the attainable set. The same approach of aligning the desired objective with \( x_1 \) and then identifying edges using the two objective solution is used.
4.2.1 Aligning the desired objective

The first step of the allocation is to find a transformation, $T_1$, that will align the desired objective with the $x_1$-axis, $T_1 y_d = (y_{d1}, 0, 0)$. The transformation used by Durham[27] for the three dimensions can easily be found for a four-dimensional case.

$$T_1 = \begin{bmatrix} \hat{y}_{d1} & \hat{y}_{d2} & \hat{y}_{d3} & \hat{y}_{d4} \\ 0 & 0 & t_{23} & t_{24} \\ 0 & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix}$$

(4.4)

$$t_{41} = \sqrt{1 - \hat{y}_{d1}^2}$$
$$t_{42} = -\frac{\hat{y}_{d1}\hat{y}_{d2}}{t_{41}}$$
$$t_{43} = -\frac{\hat{y}_{d1}\hat{y}_{d3}}{t_{41}}$$
$$t_{44} = -\frac{\hat{y}_{d1}\hat{y}_{d4}}{t_{41}}$$

$$t_{32} = \sqrt{1 - \hat{y}_{d2}^2 - t_{32}^2}$$
$$t_{33} = -\frac{\hat{y}_{d2}\hat{y}_{d3} + t_{42}t_{43}}{t_{32}}$$
$$t_{34} = -\frac{\hat{y}_{d2}\hat{y}_{d4} + t_{42}t_{44}}{t_{32}}$$

$$t_{23} = (-1)^n \text{sign}(\hat{y}_{d3}) \sqrt{1 - \hat{y}_{d3}^2 - t_{33}^2 - t_{43}^2}$$
$$t_{24} = -(-1)^n \text{sign}(\hat{y}_{d4}) \frac{\hat{y}_{d3}\hat{y}_{d4} + t_{33}t_{34} + t_{43}t_{44}}{t_{22}}$$

(4.5)

The transformation given is selected so that $T_1$ is unitary with a determinant equal to one. For the three-dimensional case, this yields transformations which are rotation matrices. In general this is not required. Any non-singular $T_1$ which aligns the desired direction with $x_1$ will work[27], suggesting that more efficient methods of finding $T_1$ may reduce the computation required to find a solution.

For the computational results presented in this dissertation, it will be seen that this initial transformation does not account for a significant portion of the computational time. Because of this, Equation 4.4 will always be used to find a transformation aligning $y_d$ in this dissertation.
4.2.2 Edge Searching in four Dimensions

For the four-dimensional case, an individual facet on $\partial(\Phi)$ is a three-dimensional parallelepiped. The solution facet can be found as the union of three edges. In the three objective case, a two-dimensional facet can be found from the union of any two distinct edges of the facet. With four objectives, however, it is possible to have three distinct edges of the three-dimensional facet whose union is not the facet. In addition to being distinct, the three edges identified need to span the parallelepiped (Figure 4.1). Edges will be found by solving the projection of the problem into two dimensions just as done in the three-dimensional allocator.

![Figure 4.1: Edges defining a three-dimensional facet.](image)

Three unique edges on a parallelepiped do not necessarily define the parallelepiped. The union of edges a, b, and c is only two dimensional, whereas a, b, and d define the entire three-dimensional figure.

The rotations about $x_1$ in the three-dimensional case were parameterized by a single angle. A similar set of “rotations” which involve only a single parameter and leave $x_1$ unmodified can be created for the four-dimensional case.

$T_{2-3} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) & 0 \\ 0 & -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$T_{2-4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & 0 & \sin(\phi) \\ 0 & 0 & 1 & 0 \\ 0 & -\sin(\phi) & 0 & \cos(\phi) \end{pmatrix}$

$T_{3-4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\phi) & \sin(\phi) \\ 0 & 0 & -\sin(\phi) & \cos(\phi) \end{pmatrix}$

Only two of these transformations, $T_{2-3}$ and $T_{2-4}$, will have any effect on the projection into the $x_1$-$x_2$ plane. The third transformation $T_{3-4}$ has no effect on this projection and does not help in locating edges.

Recall that the three objective BESA involves searching for the angle at which the intersection with the identified edge changes sides of the $x_1$-$x_2$ plane. With four objectives, there are two components whose signs could be checked. It makes sense to combine checks of $x_4$ with $T_{2-4}$ and $x_3$ with $T_{2-3}$. These combinations result in the same search carried out in
the three objective case if the $x_3$ or $x_4$ coordinate were ignored. Either of these searches can only identify two unique edges, so some combination is required to identify the three edges required.

### 4.2.3 Implementation of Four Objective Edge Searching

The correct way to locate three distinct edges using the transformations above is not obvious and four variations are considered. Each variation combined the two searches to find four, not necessarily unique, edges. Two of the methods intermixed the two searches. The third completely separated them and the fourth was designed to combine them into a single search.

The first method performs both the $T_{2-3}$ and the $T_{2-4}$ transformations as part of the same bisection search. For a given angle, $\phi$, the first method searches for the rotation where the sign of $x_3$ changes followed by finding where the sign of $x_4$ changes. Then the angle is bisected and the rotations are carried out again. This process is terminated after a fixed number of bisections as was done in the original BESA allocator.

The second method implemented functions just like the three objective BESA in that it searches for where the $x_3$ component changes sign after rotations, $T_{2-3}$, and then bisects the angle and reverses direction. At every step of this search, a separate bisection search is carried out for $x_4$ using $T_{2-4}$. Both loops are terminated by an upper limit on the number of bisections.

The third method completely separates the two searches. First, a search using $T_{2-3}$ for a fixed number of bisections is done; then one using $T_{2-4}$. This is essentially the same as first solving the problem ignoring the fourth objective using the standard BESA, then starting with the final rotation in the three objective BESA solving another three objective problem ignoring the third objective.

The final method implemented attempts to completely combine the two searches. Transformations $T_{2-3}$ and $T_{2-4}$ are placed in the same bisection loop. The angle, $\phi$, is only bisected when both the $x_3$ and $x_4$ components simultaneously change sign. In essence, each step applied a transformation equal to the product $T_{2-3}T_{2-4}$ where the angle $\phi$ was of the same magnitude for the two matrices and only its sign changed.

All of the methods outlined returned four edges. After a fixed number of bisections the last edge found on each side of the $x_1$-$x_2$ plane for each of the two transformations was returned. These edges were not necessarily unique and their union was returned as the solution object. Given the correct object on $\partial(\Phi)$, finding the solution to the Direct Allocation problem is trivial.
4.2.4 Results

The results of some numerical tests of the algorithms are presented in Figure 4.2. Each of the four methods was implemented as a MATLAB m-file which would return the object given by the union of the four edges. The final allocation step was not carried out and the results only consider whether the allocators found the correct object on $\partial(\Phi)$. The figure shows the results of 100 four objective, six control problems. For each problem, a control effectiveness matrix was chosen with random elements bounded by +1 and −1 and symmetric control limits,

$$B_{ij} \in [-1,1], u_{ij} = -1, u_{ui} = 1 \text{ for } \{i = 1 \ldots n, j = 1 \ldots m\} \tag{4.6}$$

Each problem was solved twice with each method, once in the manner described above and once with the $T_{2-3}$ and $T_{3-4}$ searches reversed giving 200 trials. The returned objects were compared to the facets found by a facet search method.

Figure 4.2 consists of six plots. The first shows the percentage of cases where the methods fail to find any sign changes and cannot return edges on both sides. The second plot is the percentage of cases where the union of the edges found is the same as the three-dimensional facet found using the facet search. The remaining plots compare the objects found to the correct object found by the facet search. Plots c, d, and e indicate the percentage of times where the object found has at least three, two, or one defining control in common with the correct one, respectively. Plot f is the percentage where at least one of the controls defining the object returned is incorrect.

The performance of the first two test cases were similar, never failing to detect an edge crossing or to return an object defined by more than three controls. These cases find the correct three controls 28% and 28.5% of the time. The first method finds an incorrect control 13% of the time while the second finds one 9% of the time. These algorithms return a two-dimensional figure which is part of the correct solution in nearly 90% (87% and 91%) of the test cases.

The third algorithm which treats the rotations completely separately found the correct solution in 17% of the cases. Finding an incorrect solution for 61.5% of the tests was much more frequent than the first two approaches. The final algorithm, treating both rotations together in the bisection loop and only reporting an edge crossing when both signs changed for the same iteration, found the correct solution once (0.5% of the tests) and found an incorrect solution 7.5% of the time. In most cases (91.5%) this algorithm failed to find a solution as it reached the bound on the number of iterations without finding an edge change.
Figure 4.2: Tests of BESA extensions.
The allocators tested are those described in the text. The charts show the percentage of cases for which each design: a) fails to return a solution, b) returns a correct solution, c) identifies three correct controls, d) identifies two of the three correct controls, e) identifies one of the three correct controls, and f) identifies an incorrect control.
4.3 Searching For Three-Dimensional Objects

A three-dimensional facet of $\partial(\Phi)$ in the four-dimensional case can be defined by the union of two unique two-dimensional objects. One can search for two two-dimensional objects on $\partial(\Phi)$ rather than searching for edges looking for the solution facet. Two-dimensional objects can be found by solving three-dimensional projections of the problem.

The basic algorithm for this method is much the same as that outlined for the three and four-dimensional edge searching approaches. First the problem is transformed so that the desired objective is along the positive $x_1$ axis and then a search is conducted looking for two two-dimensional objects on $\partial(\Phi)$ that define the solution facet. Once the solution facet is found, it is simple to find the intersection of the desired direction and a solution.

4.3.1 Bisection Search

Once the transformation outlined in Section 4.2.1 is carried out, a bisection search is used to look for two objects that will describe the solution facet. To find these objects, a three-dimensional problem will be created by projecting $\mathbf{B}$ and $\mathbf{y}_d$ into $x_1-x_2-x_3$. When the control vector found for the intersection of $x_1$ with $\partial(\Phi)$ in three dimensions is multiplied by the full four-dimensional control effectiveness matrix, the $x_4$ component of the objective, $y_{d_4}$ is not specified. This sign will be used to define a bisection search looking for the angle, $\phi$, where $\mathbf{T}_{2-4}$ gives $y_{d_4} = 0$.

4.3.2 Facet Search Subproblem

One of the nice properties of the original three-dimensional bisecting edge search allocator is that it includes a plan for estimating a solution if the search does not return the solution facet. The errors in these estimates are typically quite small[27]. The four-dimensional allocator proposed uses both the solution facet and the solution of the three-dimensional problem.

In order to initially avoid dealing with the estimation case, initial tests were carried out using the facet search approach to solving the three-dimensional problem. A version of this allocator was created in $MATLAB^TM$ and tested for a number of four-dimensional problems. As no system was put in place to handle estimation, the allocator returned an error condition when the search did not find a solution. A system with eight bisections solved four-dimensional problems with a similar success rate to the three-dimensional allocator, much better than the edge-searching attempts.
4.3.3 BESA Subproblem

Once the idea of solving a three-dimensional problem as part of the loop was seen to work using the facet search to find two-dimensional objects, the three-dimensional BESA was substituted. The facet search itself is often considered too slow for any real online application so an allocator which calls it multiple times for one solution is also slow. Using a more efficient method like the BESA to solve the inner problem will result in a faster allocator.

As in the initial tests using the facet search, the implementation was designed to set an error flag if the solution could not be found and no estimation was attempted. Testing showed the allocator to fail in a much larger number of cases than the three-dimensional allocator. The failure cases all corresponded to either the case where the outer search failed to find the solution facet, or more often, where the inner BESA search called its estimator. Intuitively, this increased failure rate makes sense as the allocator requires calling the BESA algorithm once per rotation.

4.3.4 Three-Dimensional BESA Revisited

The success rate of the four-dimensional allocator depends on that of the three-dimensional BESA. As originally presented, the three-dimensional edge search terminates after a fixed number of bisections. Presumably the final rotation is small enough not to step over the solution facet. Previous authors have obtained satisfactory performance with the limit set around nine bisections[56, 47]. Durham reports the number of errors appearing to level out after seven bisections[27].

None of the results in literature look at numbers of bisections greater than nine. To determine whether the number of errors in the three-dimensional case could be reasonably lowered, a modified BESA allocator was tested. This allocator tested the current edges and continued until a solution was found, reporting the solution and the number of bisections required when it exited.

The modified BESA allocator was tested for a series of randomly generated problems with symmetric control limits. The elements of $y_d$ and $B$ were chosen to be uniformly distributed random numbers between zero and one. These problems were created for four objective problems with between 4 and 28 controls. The compilation of several such tests is presented in figure 4.3. For each number of controls, more than a thousand random problems were generated. The maximum, minimum, and mean number of bisections for each number of controls is shown. It can be seen that while the average case did not take many bisections, there were some cases which required as many as 22 iterations. Special cases including $B$ matrices which were not full rank or had linearly dependent columns were not tested and the desired objectives were not chosen to point along an edge or at a vertex except by chance. Of the nearly 27,500 cases tested with between 4 and 28 controls, all could be solved in a finite number of bisections.
Figure 4.3: Number of bisections required to solve the three-dimensional problem. The mean number is lower than the 8 bisections traditionally used as the maximum, but the maximum required is seen to be quite large in some cases.

4.3.5 Results For an Improved Four Objective Allocator

The four objective allocator that searched for two-dimensional facets described in Section 4.3 was modified so that it called the BESA allocator without a fixed limit on bisections. Timing results based on a $MATLAB^{TM}$ implementation of this recursive algorithm are presented in figure 4.4. These results were generated by running the algorithm on a large sample of randomly chosen four-dimensional problems. The CPU time reported by $MATLAB^{TM}$ for each case was recorded. The results were run in $MATLAB^{TM}$ 6.1 under Windows NT 5.0 on a computer with a Pentium II -450 MHz processor and 128Mbyte of RAM. The results shown are a compilation of the results for several different experiments; hence, the number of cases run at each number of controls varies.

The mean time for solving four objective problems appears to scale linearly with the number of controls, as was reported for the three objective BESA. If the trend in Figure 4.4 is truly linear, then the number of bisections required should be constant as the number of controls increases. Figures 4.5 and 4.6 show the number of bisections at the four objective level and the total number of bisections at the three objective level. The mean value increases slightly with the number of controls for both bisections. This indicates that the execution time should increase with controls at a slightly faster than linear rate.
Figure 4.4: Execution time of the four-dimensional recursive allocator. The mean execution time of the four-dimensional recursive design are presented as well as level curves describing the distribution of the recorded cases. The indicated percentage of the tested cases ran faster than the times along each curve. From these levels, it can be seen that while the maximum time is quite large compared to the mean, the distribution of results tails off rapidly.
Figure 4.5: Number of bisections required in the outer loop of the four-objective case.
The number of bisections increases slightly as the number of controls increases. This implies that the execution time should vary slightly faster than linearly with $n$.

Figure 4.6: Total number of three-objective bisections required to solve a four-objective problem.
The number varies slightly with the number of controls. Also the four-objective problem requires a large number of iterations in the four-objective subproblems.
Figure 4.4 also shows level curves for various percentiles of the timing results. The number of cases found taking longer than a specific time to run drops rapidly as that time increases from the mean. While the maximum time is quite a bit larger than the mean time, 99% of the runs took less than two and a half times the average execution time.

Comparing the results presented in Figure 4.4 with those previously observed for the three objective case, we see that the recursive solver took more than 7 times as long to solve the four objective case as it did the three objective case. The four objective case requires a large total number of bisections in three dimensions (Figure 4.6). Compared to the number of bisections required in a single three objective problem, Figure 4.3, the four objective case is much more complex.
Chapter 5

Methods for Higher Numbers of Objectives

Two methods for allocating controls to satisfy more than four objectives are considered. The first of these is a recursive extension of the bisecting object search presented in Section 4.3. The development of this method is detailed in this chapter, followed by results of tests of this method at the beginning of Chapter 6.

Linear programming as a solution to the control allocation problem is discussed in the second half of this chapter. The numerical routines to solve linear programs can come in many varieties and a brief overview is presented separately for specific formulations of the control allocation problem as linear programs. Tests of these formulations are described in Chapter 6.

5.1 Recursive Method

The approach used in Section 4.3 can be extended to create an allocator designed to work in an arbitrary number of objective dimensions. This new allocator is recursive in nature; searches for two \((n-2)\)-dimensional objects defining the \((n-1)\)-dimensional solution facet are carried out by invoking the same allocator on \((n-1)\)-dimensional projections of the problem. Problems in two dimensions are still solved using the technique described by Durham[25].

5.1.1 Aligning the Objective with \(x_1\)

Once again, the first step in the allocation will be to align the desired objective direction with the \(x_1\) axis. The method to align the objective is described in Section 4.2.1 for the four-dimensional case. This same approach easily extends to higher dimensions.
5.1.2 Search

After the initial transformation to align the desired objective with the $x_1$ axis, the algorithm proceeds in a similar manner to the four-dimensional case described above. A search is carried out to identify two $(n - 2)$-dimensional objects which define the $(n - 1)$-dimensional solution facet. The transformations are constructed in a similar fashion to the $T_{2-4}$ transformation described above only with the non-identity elements in the second and $n^{th}$ rows.

$$T_{2-n} = \begin{pmatrix}
1 & 0 & \cdots & 0 \\
0 & \cos(\phi) & \cdots & \sin(\phi) \\
\vdots & \ddots & \ddots & \vdots \\
0 & -\sin(\phi) & \cdots & \cos(\phi)
\end{pmatrix}, \quad T_{2-n} \in \mathbb{R}^{n \times n}$$

Taking the first $n - 1$ rows of $B$ and the desired objective gives a lower order problem which can be solved to find an $(n - 2)$-dimensional object which intersects $x_1$. When the controls at this solution are multiplied by the $n^{th}$ row of the control effectiveness matrix, the sign of the $x_n$ component can be found. The bisection search seeks $\phi$ where this sign changes. The algorithm continues bisecting a range of angles until the solution is found.

When $\phi = \pi$ Equation 5.1 gives,

$$T_{2-n} = \begin{pmatrix}
1 & 0 & \cdots & 0 \\
0 & -1 & \cdots & 0 \\
\vdots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & -1
\end{pmatrix}$$

After applying this transformation, the only change in $B$ is to change the sign on the second and last rows.

$$B = \begin{pmatrix}
B_{r1} \\
B_{r2} \\
B_{r3} \\
\vdots \\
B_{rn}
\end{pmatrix}, \quad T_{2-n}B = \begin{pmatrix}
B_{r1} \\
-B_{r2} \\
B_{r3} \\
\vdots \\
-B_{rn}
\end{pmatrix} \quad (5.1)$$

The problem made up of the first $n - 1$ rows of $B$ can be solved to find $u_i$ such that:

$$\begin{pmatrix}
B_{r1} \\
B_{r2} \\
\vdots \\
B_{rn-1}
\end{pmatrix} u_i = \begin{pmatrix}
y_1 \\
0 \\
\vdots \\
0
\end{pmatrix} \quad (5.2)$$
Multiplying by the full matrix gives the intersection,

\[
\begin{pmatrix}
B_{r1} \\
B_{r2} \\
\vdots \\
B_{rn-1} \\
B_{rn}
\end{pmatrix}
\begin{pmatrix}
u_i
\end{pmatrix} =
\begin{pmatrix}
y_1 \\
0 \\
\vdots \\
0 \\
y_n
\end{pmatrix} \quad (5.3)
\]

When applied to the matrix transformed by \( T_{2-n}(\pi) \) the objective coordinates of the intersection are

\[
\begin{pmatrix}
B_{r1} \\
-B_{r2} \\
\vdots \\
B_{rn-1} \\
-B_{rn}
\end{pmatrix}
\begin{pmatrix}
u_j
\end{pmatrix} =
\begin{pmatrix}
y_1 \\
0 \\
\vdots \\
0 \\
-y_n
\end{pmatrix} \quad (5.4)
\]

The objects for these two cases, \( u_i \) and \( u_j \), are the same. So, after solving the initial projected problem, \( \phi = 0 \) and \( \phi = \pi \) can be used to define the initial interval for a bisection search. The transformation at the midpoint, \( T_{2-n}(\pi/2) \), is used to find the next n-1 dimension problem to solve.

This method is recursive because each time a solution is needed for a problem of lower dimension, the same algorithm can be invoked with the smaller problem. The simple non-recursive method described in the original BESA development is used for the solution of two-dimensional problems. This non-recursive approach serves as the terminating condition for the recursion. Eventually this two-dimensional case will be reached because the method only invokes itself with problems of one fewer objectives than the input.

### 5.1.3 Checking the solution

At every step of the bisection, two objects are available to form the potential solution facet. One of these corresponds to the last intersection found with \( y_n \) negative; the other, the last with \( y_n \) positive. At the first bisection step where \( \phi_+ \) and \( \phi_- \) are 0 and \( \pi \), these two objects are the same. Later iterations, however, have two distinct objects. As a condition for terminating the search, the union of these two objects is tested to see if it is on the boundary and if it contains the desired intersection.

Testing to see whether an object is on the boundary can be done by finding a series of transformations which align controls defining the facet normal to \( x_1 \). If the signs of all of the other elements in the first row of the transformed \( B \) are the same as those of the facet, it is on the boundary.

Given the potential solution facet, a set of vectors which span the solutions on that facet can be defined (Figure 5.1). One vector, \( y_0 \), extends from the origin to one vertex of the
facet. A set of vectors, $y_i, i = \{1 \ldots n - 1\}$, which extend along the edges of the facet can be defined. A simple equation can then be solved to find the desired intersection.

\[
v_1 y_d = y_0 + \sum_{i=1}^{n-1} v_{i+1} y_i
\]

\[
M = \begin{bmatrix}
-y_d & y_1 & \cdots & y_{n-1}
\end{bmatrix}
\]

\[
\Rightarrow \quad Mv = -y_0
\]

\[
\Rightarrow \quad v = -M^{-1}y_0
\] (5.5)

Once the vector $v$ has been found, its coefficients can be checked to see if the solution is contained in the current facet, $0 \leq v_1, 0 \leq v_i \leq 1, i = 1 \ldots n - 1$.

### 5.1.4 Estimation

In the four-dimensional, test cases the maximum execution time was seen to be much larger than the time required by 95% of the cases (Figure 4.4). No upper bound on this time was explicitly shown to exist. In online applications, the time available to compute a solution is generally limited.

A scheme was designed to estimate a solution when the solution facet failed to be found in a set number of bisections. The recursive nature of the algorithm requires that the estimated solution be used by higher level searches. The estimated solution should preserve direction and return an object to be used by other loops. The estimation routine should return no error when called with the correct solution facet.

One approach to the estimation routine is to take the union of the two objects which were last found. In general, this will yield an object which is of a larger dimension than $n - 1$. Once
this object is found, equation 5.5 can be extended by adding terms for the extra controls. The resulting under-determined system is solved using a cascaded pseudo-inverse approach to limit the elements of $\mathbf{v}$ between 0 and 1.

In the main part of the search, the object used as input to the estimator is used. If one of the smaller subproblems estimates a solution, all of the problems which had called it will estimate a solution as the object will be too big. It is possible for the algorithm to recover the true intersection if subsequent steps replace the estimated solution.

Two methods for including this estimator are considered. In one the estimator is called if the solution is not found in a set number of bisections. The other is to always carry out the same number of bisections and then estimate at the end. If the frame rate for the allocator is fixed, stopping early does not offer much advantage. Also, the second approach eliminates the need to check solutions at every step.

5.1.5 Special Cases

When considering the control allocation method, it is important to look at the special cases which may violate assumptions implicit in the method.

5.1.5.1 Objects of Lower Dimension

The geometric arguments used to motivate the three-dimensional allocator were based on the idea that the desired objective intersected $\partial(\Phi)$ somewhere on a two-dimensional facet. It is possible to find a specific objective direction for which the desired intersection occurs at a boundary between such facets. In higher dimensional cases, the boundary between facets may have as many as $n-2$ dimensions; the smallest object containing the intersection has fewer dimensions than the solution sought and the search may not terminate correctly.

Computationally, it is easy to handle such points if small errors in the vicinity of these points can be tolerated. These degenerate points cause the algorithm to break down when the sign of the $x_n$ component is checked to determine where to make the next bisection. If the $y_n$ is found to be close enough to zero, the current search can be stopped as the current solution to the $(n-1)$-dimensional problem is “close-enough”.

5.1.5.2 Coplanar Controls

The original facet searching algorithm in three dimensions determines which facets are on the boundary by finding a vector normal to the two columns of $\mathbf{B}$ being considered. In higher dimensions, finding the facet on the boundary involves finding a basis for a one-dimensional null space of $n-1$ columns of $\mathbf{B}$. This requires that every set of $n$ columns of $\mathbf{B}$ be linearly
independent.

In the three-dimensional case, this requirement is the same as saying that none of the facets of \( \partial(\Phi) \) can be coplanar. The geometry of coplanar 3-d facets is described by Bodson and Petersen[50]. The adjective “coplanar” will be used to describe the similar case with higher numbers of objectives where multiple facets overlap.

Direct allocation requires finding a solution on \( \partial(\Phi) \) which is scaled to find the final solution. If sets of \( n \) controls are not linearly independent, then the solution on \( \partial(\Phi) \) is not unique. This can be seen by considering equation 5.5. If an additional control can be written as a linear combination of those in the solution,

\[
y_{\text{new}} = \sum_{i=1}^{n-1} \alpha_i y_i \Rightarrow y_j = \frac{1}{\alpha_j} \left( \sum_{i=1, i \neq j}^{n-1} \alpha_i y_i - y_{\text{new}} \right), \tag{5.6}
\]

one of the controls defining the solution can be replaced by the new control and the system solved as before:

\[
v_1 y_d = y_0 + \sum_{i=1}^{n-1} v_{i+1} y_i
\]

\[
v_1 y_d = y_0 + \sum_{i=1, i \neq j}^{n-1} v_{i+1} y_i + \frac{v_{j+1}}{\alpha_j} \left( \sum_{i=1, i \neq j}^{n-1} \alpha_i y_i - y_{\text{new}} \right)
\]

\[
v_1 y_d = y_0 + \sum_{i=1, i \neq j}^{n-1} \tilde{v}_{i+1} y_i + \tilde{v}_{j+1} y_{\text{new}}
\]

\[
M = \begin{bmatrix} -y_d & y_1 & \cdots & y_{j-1} & y_{\text{new}} & y_{j+1} & \cdots & y_{n-1} \end{bmatrix}, \quad \tilde{v} = (v_1 \ldots \tilde{v}_n)^T
\]

\[
\Rightarrow M\tilde{v} = -y_0
\]

\[
\Rightarrow \tilde{v} = -M^{-1}y_0 \tag{5.7}
\]

In aircraft applications, coplanar (or even collinear) controls may be introduced by assumptions made when modelling individual sections of a larger split surface. Non-aerodynamic controls are also a particularly easy way to introduce such issues as they often act in single axes. For example, consider the case of two engines capable of yaw thrust vectoring. If both engines are located at the same vertical distance from the reference point, their effectiveness will be the same.

The original facet search routine and early versions of the BESA allocator solved this problem by perturbing the elements of the control effectiveness matrix by small random values. These values were chosen small enough to not significantly impact the solution, but did serve to ensure that the columns were independent. Nelson[47] used the BESA approach on a system with large numbers of coplanar controls without perturbing \( \mathbf{B} \). The system was observed to call the estimator for a large number of cases, but overall performance was not seen to be effected.
When given edges which are coplanar, the original BESA estimator will return an answer which lies in that plane. Similarly, the estimation scheme described above will also return an answer in the same plane. The problem of coplanar controls is handled by including the ability to estimate solutions into the algorithm.

5.2 Linear Programming

Linear programming is a popular approach for solving control allocation problems. Several different ways of formulating linear control allocation problems as linear programming problems have been suggested.

5.2.1 Solvers

5.2.1.1 Simplex Method

Repeating Equation 3.8 the standard form for a linear program is,

$$\min_x c^T x \mid Ax = b, x \geq 0.$$  \hfill (5.8)

In order to avoid confusion with the dimensions of the control allocation problem whose solution is sought, $k$ and $l$ will be used to represent the number unknowns and the number of equality constraints.

$$x, c \in \mathbb{R}^k, b \in \mathbb{R}^l, A \in \mathbb{R}^{l \times k}, l \leq k$$

Variations on the simplex method are popular ways to solve linear programs and descriptions of them can be found in many textbooks [43, 32, 49, 46]. The description which follows is based on those found in Gale [32] and Luenberger[43].

If the matrix $A$ is full rank, then a set of $l$ linearly independent column vectors can be found. $l$ linearly independent vectors in $\mathbb{R}^l$ form a basis for $\mathbb{R}^l[46]$. Any other vector in $\mathbb{R}^l$, in particular $b$, can be written as a linear combination of the vectors in this set. Defining $B$ as the square matrix whose columns are these $l$ vectors, $b$ can be written,

$$Bx_B = b.$$ \hfill (5.9)

Solving this system for $x_B$ yields a unique solution for $l$ variables. These basic variables correspond to the columns of $A$ which were originally selected. Forming $x$ by setting all elements corresponding to the basic variables to their values in $x_B$ and making the rest of the elements of $x$ zero gives a solution to the original equality constraints. This can be seen by defining $D$ to be the columns of $A$ corresponding to non-basic variables and $x_D$ to be the vector of non-basic variables.

$$Ax = Bx_B + Dx_D = Bx_B + D(0, \ldots, 0)^T = Bx_B = b$$ \hfill (5.10)
This solution is called a basic solution to equation 5.8. If \( x \geq 0 \), it is a basic feasible solution. Since each combination of \( l \) independent column vectors corresponds to a particular basic feasible solution, there is a finite number of such solutions.

The importance of basic solutions is made clear by the Fundamental Theorem of Linear Programming\[43\].

Given a linear program in the form 5.8 where \( A \in \mathbb{R}^{l \times k} \) and \( \text{rank}(A) = l \).

i) if there is a feasible solution, there is a basic feasible solution;
ii) if there is an optimal feasible solution, there is an optimal basic feasible solution.

The proof of this theorem can be found in Luenberger [43]. The theorem implies that a search through the basic feasible solutions of the system will locate the optimal solution. The simplex method provides an efficient procedure for carrying out such a search.

The simplex procedure starts with an initial basic feasible solution. The non-basic vectors are then examined to find one which will, if added to the basic set, result in a feasible basis and a lower value for the cost function. This vector is then swapped for one which results in the greatest decrease. This procedure repeats until a basic feasible solution is found which cannot be improved.

In the worst case, the simplex procedure could take a large number of iterations to run. Wright [63] mentions that example problems requiring as many as \( 2^k \) iterations have been found. Generally, however, simplex algorithms tend to converge in a number of iterations which scales linearly with \( k \).

Traditionally simplex procedures are executed as a series of pivot operations acting on the entire coefficient matrix\[32\], much the same way as numerical techniques such as Gaussian elimination are defined. This means that every iteration requires a number of computations which will scale as \( k \). The total computation time could be expected to be a function of \( k^2 \).

A revised simplex algorithm has the potential to reduce those computations\[43\] for systems where \( k \gg l \). The revised procedure separates the coefficient matrix into two parts, the \( B \) and \( D \) defined above. Each step still involves operations on every element to find the vector to enter and leave the basis, but the expensive pivot operations are only carried out on \( B \) which is of size \( l \times l \).

Most formulations of control allocation problems as linear programming problems involve variables with upper bounds.

\[
\min_{x} c^T x \mid Ax = b, h \geq x \geq 0
\]  

(5.11)

To convert from this form into the standard form (5.8) expected by solvers, extra variables
and equations must be added to the system[19],

$$\min_{\bar{x}, \tilde{x}} c^T x \quad \left[ \begin{array}{cc} A & 0 \\ I_k & I_{1k} \end{array} \right] \begin{pmatrix} x \\ \bar{x} \end{pmatrix} = \begin{pmatrix} b \\ \tilde{h} \end{pmatrix}, \begin{pmatrix} x \\ \bar{x} \end{pmatrix} \geq 0. \quad (5.12)$$

Bodson [7] suggests using solvers which will handle upper bounds explicitly instead of adding slack variables. Luenberger presents a modification to the simplex method to handle upper bounds[43]. Each non-basic variable is allowed to be at either its upper or lower bound. Basic variables are considered to be differences from whichever bound they were at before they entered the basic set. Each variable is referenced to either its lower bound, $x_i^+ = x_i$, or to its upper bound $x_i^- = h_i - x_i$. At each iteration, either a new variable is moved into the basis or one of the non-basic variables switches to its opposite bound. If a new variable enters the basis, the exiting variable is moved to the bound which results in the largest reduction in cost. This method proceeds much like the regular simplex approach, though additional computations are required to update right and left hand side of the problem at each iteration requiring a variable to switch bounds.

Simplex methods require an initial basic feasible point to start their iterations. If such a point is unknown, a second linear program can be defined to find a point to initialize the first point. One such “crash procedure” is to define the new program,

$$\min_{\bar{x}} \bar{c}^T \bar{x} \quad \bar{A} \bar{x} = \bar{b}, \bar{h} \geq \bar{x} \geq 0 \quad (5.13)$$

$$\bar{A} = (A \ diag(b))$$
$$\bar{b} = b$$
$$\bar{c}^T = (0_{1 \times k}, 1_{1 \times l})$$
$$\bar{h} = (u_u - u_l)$$
$$\bar{x} = (x^T \ x_s T)^T$$

The initial basic feasible point for this new program is $\bar{x}_0 = (0_{1 \times k}, 1_{1 \times l})^T$. The minimum cost is zero and occurs when all of the slack variables $x_s$ have been driven out of the basis. The resulting solution is a basic feasible solution for the original problem.

This starting procedure requires the solution of an additional linear program, larger than the original one. In general, however, the simple cost function enables a starting point to be found in a small number of iterations.

### 5.2.2 Control Allocation Formulations

Several ways to formulate the control allocation problem as a linear program have been suggested. Several of the methods below were originally stated in terms of the standard
form. They have all been restated as problems in the standard form with upper bounds. Different approaches to the problem can result in linear programs of different sizes. Table 5.1 lists the number of equality constraints and unknowns for each of the formulations to be presented below. Whether there is an obvious choice of initial basis for simplex solvers or whether the solver must search for one are also listed.

<table>
<thead>
<tr>
<th>Method</th>
<th>Constraints(l)</th>
<th>Variables (k)</th>
<th>Easily Found</th>
<th>Initial Point?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction Preserving</td>
<td>$n$</td>
<td>$m+1$</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Direction Preserving Reduced Size</td>
<td>$n-1$</td>
<td>$m$</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Dual Branch: Feasibility</td>
<td>$n$</td>
<td>$m+2n$</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Dual Branch: Sufficiency</td>
<td>$n$</td>
<td>$2m$</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Single Branch</td>
<td>$n$</td>
<td>$2m+1$</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Mixed Optimization</td>
<td>$n$</td>
<td>$2m+2n$</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### 5.2.2.1 Direction Preserving

The original direct allocation approaches to finding the optimal control solution preserve the direction of the desired objective when the effectors are saturated. Unobtainable commands can be handled in much the same way using linear programming. A straightforward approach to preserving direction is to add a scaling on the desired command to the variables.

$$\min_{u, \lambda} -\lambda \mid Bu = \lambda y_d, u_l \leq u \leq u_u, 0 \leq \lambda \leq 1$$

(5.14)

To present this as a standard linear programming problem with upper bounds, both the equality and inequality constraints need to be modified. To set the lower bound on the variables at zero, $\tilde{x} + u_l$ is substituted for $u$.

$$Bu = \lambda y_d \Rightarrow B(\tilde{x} + u_l) = \lambda y_d \Rightarrow B\tilde{x} + Bu_l = \lambda y_d$$

(5.15)

Writing in terms of the new vector of unknowns, $(\tilde{x}, \lambda)^T$, the terms for the standard form with upper bounds, $\min_x c^T x \mid Ax = b, 0 \leq x \leq h$, are,

$$A = [B - y_d]$$

$$b = -Bu_l$$

$$c^T = (0_{1 \times m}, -1)$$

$$h = \left(\begin{array}{c} u_u - u_l \\ 1 \end{array}\right)$$

(5.16)
The controls are found from the optimal variables by taking the first $m$ terms of the solution and adding the minimum limit.

5.2.2.2 Reduced Size Direction Preserving

Because traditional linear programming solvers require many more computations as the dimension of the problem increases, Bodson suggests searching for smaller representations of the control allocation problem. He suggests a form for direction-preserving which requires only $n - 1$ constraint equations and $m$ variables.

The rows of $B$ and $y_d$ are reordered so that $y_{d1}$ is nonzero. If this is not possible, the control allocation problem is trivial as $y_d = 0$. The direction preserving form can then be solved to eliminate the scaling parameter.

$$MBu = 0$$ (5.17)

where,

$$M = \begin{pmatrix} y_{d2} & -y_{d1} & \cdots & 0 \\ \vdots & 0 & \ddots & 0 \\ y_{dn} & 0 & \cdots & -y_{d1} \end{pmatrix}$$ (5.18)

This formulation sets up the problem in the form of Equation 5.2.1.1 with

$$A = MB$$ (5.19)
$$b = -Au_l$$ (5.20)
$$c^T = -B^T y_d$$
$$h = u_u - u_l$$ (5.21)

Finding the control corresponding to the solution of this problem is a little more difficult than in the other direction-preserving method. First, the lower limit is added as before, $u = x + u_l$. The control solution returned is the one on the boundary. The correct solution is found by calculating the scaling factor $\rho = \frac{y_d^T Bu}{y_d^T y_d}$. If $\rho > 1$, then the desired objective is on the interior of $\Phi$ and the correct solution is $u = u / \rho$.

This solution is the same as the direct allocation solution because it is scaled from the boundary. Bodson refers to this method as direct allocation; but, for this dissertation, the term direction-preserving will be used to distinguish it from methods which explicitly calculate portions of the geometry of $\partial(\Phi)$. For unattainable moments, the two direction-preserving linear programs will return the same answer; however, for attainable moments, the two schemes will differ. This method returns the solution scaled from the maximum, while that presented in the previous section is not specified.
5.2.2.3 Dual Branch

Providing different criteria for dealing with unattainable objectives and attainable objectives motivates an approach that uses two different linear programs to solve the problem based on the desired objective. The dual branch method described in the review of literature (Section 3.3.2) preserves direction for solutions that are unattainable. Buffington also describes a similar approach in which the error between the desired and attained objectives is minimized.

\[
\min_u \|w_d^T (Bu - y_d)\|_1 \mid u_l \leq u \leq u_u
\]  

(5.22)

Once the “feasibility branch” has found the minimum error solution, the error can be checked. If the solution is attainable, the error will be zero and a second branch is called. The second optimization is designed to minimize the error between the control vector and some desired control configuration, \(u_p\) while meeting the objective.

\[
\min_u \|w_u^T (u - u_p)\|_1 \mid Bu = y_d, u_l \leq u \leq u_u
\]  

(5.23)

The feasibility branch can be converted into linear programming form by adding variables corresponding to the positive and negative objective error \(e = e^+ - e^- = Bu - y_d\). Then by modifying the limits on the controls, as in the previous formulation, an equality constraint can be written,

\[
e^+ - e^- = Bu - y_d = B(\tilde{x} - u_l) - y_d \Rightarrow e^+ - e^- - B\tilde{x} = Bu_l - y_d
\]  

(5.24)

Taking the unknown vector to be \(x^T = (e^{+T}, e^{-T}, \tilde{x}^T)\) and setting upper bounds for the allowable objective error, \(e_{\max}\), the parameters for the linear program in standard form (Equation 5.2.1.1) for this branch is found,

\[
A = \begin{bmatrix} I_n - I_n - B \end{bmatrix} \\
b = -Bu_l - y_d \\
c^T = \begin{bmatrix} w_d^T, w_d^T, 0_{1 \times m} \end{bmatrix} \\
h = \begin{bmatrix} e_{\max} \\
e_{\max} \\
\end{bmatrix} \\
\end{bmatrix} \\
\begin{bmatrix} u_u - u_l \end{bmatrix}
\]  

(5.25)

The control sufficiency branch can be found by rewriting the control as two variables that define the error from the desired control, \(u - u_p = u^+ - u^-\). Substituting into the constraint equation:

\[
Bu = B(u^+ - u^- + u_p) = y_d \Rightarrow Bu^+ - Bu^- = y_d - Bu_p
\]  

(5.26)
From here the parameters of the linear program can be written with the vector of unknowns \( x = (u^T u^-)^T \),

\[
\begin{align*}
A &= [B - B] \\
b &= y_d - Bu_p \\
c^T &= (w_u^T, w_u^T) \\
h &= (u_u - u_p) (u_p - u_l)
\end{align*}
\]

The variables representing the error in both of the linear programs act as slack variables and make it easy to select an initial basic feasible point. The maximum allowable objective error should be chosen as a large number so that the first program is feasible for all reasonable commands. If the allowable error is chosen sufficiently large, an initial point for the feasibility branch can be found by setting all of the controls to their lower limits. The error, \( Bu_l - y_d \), can then be broken into the components \( e^+ \) and \( e^- \). In a similar manner, a basic feasible point for the sufficiency branch can be constructed using the feasible control solution output from the first program and finding the appropriate error components.

### 5.2.2.4 Single Branch

The desire to specify separate cost functions for attainable and unattainable controls without the computational requirement to solve two problems has led to approaches which combine them into a single cost function. Buffington suggests a single branch system which would seek to minimize some function of the control vector while preserving direction if possible.

\[
\min_{u, \lambda} ||w_u^T (u - u_p)||_1 - \lambda \quad | Bu = \lambda y_d, u_l \leq u \leq u_u, 0 \leq \lambda \leq 1
\]

The constraint equation can be found in a similar fashion to the sufficiency branch above by defining, \( u - u_p = u^+ - u^- \).

\[
Bu = B(u^+ - u^- + u_p) = \lambda y_d \Rightarrow Bu^+ - Bu^- - \lambda y_d = -Bu_p
\]

The vector of unknowns in this case contains the control errors and the objective scaling parameter, \( x^T = (u^T, u^-^T, \lambda) \). To put the linear program in the proper form choose,

\[
\begin{align*}
A &= [B - B - y_d] \\
b &= -Bu_p \\
c^T &= (w_u^T, w_u^T, -1) \\
h &= (u_u - u_p) (u_p - u_l) 1
\end{align*}
\]
5.2.2.5 Mixed Optimization

The formulation Bodson calls mixed optimization\[6\] is similar to the single branch form. The difference is that mixed optimization attempts to combine the minimization of control error with objective error rather than preserving direction. Bodson considers the case where the individual elements of the control and objective error cannot be weighted—only the combination of the two. A slight variation on this approach is to allow the elements to be individually weighted,\[72x376\]

$$\begin{align*}
\min \| w_u^T (u - u_p) \|_1 + \| w_d^T (Bu - y_d) \|_1 & \mid u_i \leq u \leq u_u, 0 \leq \lambda \leq 1 \\
\end{align*}$$  \hspace{1cm} (5.31)$$

The terms in the control weighting vector, \(w_u\) should be chosen to be much smaller than those in \(w_d\) so that the error minimization dominates when the solution is unattainable.

To convert this problem into the standard form, variables for the constraint error are introduced, \(e = e^+ - e^- = Bu - y_d\). The same objective error variables used above can be substituted into this equation,

$$Bu - y_d = B(u^+ - u^- + u_p) - y_d = e^+ - e^- \Rightarrow -Bu^+ + Bu^- + e^+ - e^- = Bu_p - y_d \hspace{1cm} (5.32)$$

Expressing this equation and the cost function in terms of the unknowns, \(x^T = (e^+T, e^-T, u^+T, u^-T)\) and setting the maximum allowable error, \(e_{\text{max}}\), the linear program corresponding to Equation 5.2.1.1 is found with

$$\begin{align*}
A &= [I_n - I_n - BB] \\
b &= Bu_p - y_d \\
c^T &= (w_d^T, w_d^T, w_u^T, w_u^T) \\
h &= \begin{pmatrix} e_{\text{max}} \\ e_{\text{max}} \\ u_u - u_p \\ u_p - u_i \end{pmatrix}
\end{align*}$$ \hspace{1cm} (5.33)$$

Once again the addition of the error variables allow the initial basic feasible solution to be found by setting all of the controls to their preferred position and then calculating the corresponding error terms.

5.2.3 Early Termination

One of the nice properties about the simplex method is that there is a theoretical upper bound on the number of iterations required to find the optimal solution. This can be combined with estimates of the time required to complete each iteration to provide an estimate of the maximum computational effort required to solve the control allocation problem. If the
amount of time available at each frame for control allocation is known, this information could be used to size the required computational resources.

This approach, however, may lead to a conservative design. In general, the number of iterations taken is much smaller than the upper limit. A less conservative design might be to find a number of iterations in which a high percentage of the cases can be solved and define some method for handling the other cases. One approach is to use a relatively cheap allocation technique such as a GI solution if the system failed. Another approach is to run the LP up until the very end of the available time and return the controls corresponding to the current basis.

If the simplex procedure is ended early, all that is known is that the current solution is basic and feasible. The feasibility of the current solution ensures that the constraint equation is satisfied and the variables are within their bounds. The fact that the solution is basic means that at least \( k - l \) of the variables will be at one of their bounds. A feasible solution to the direction-preserving approach of Section 5.2.2.1 will be in the correct objective direction. The smaller direction-preserving formulation (Section 5.2.2.2) will also always have a solution which preserves direction. Because the scaling factor is not explicitly included as an unknown variable, an additional control will potentially be not at a limit.

The first branch of the dual branch problem is feasible for any control vector within the limits. Because the cost function should improve at every step, it is hopeful that the solution is close to optimal, though this cannot be assured. For a solution to be a feasible solution of the second branch, it must satisfy the desired objective, so any interruption of this branch will only affect the proximity to the preferred control position.

The constraints on the single branch formulation require that the direction of the desired objective be preserved. How rapidly the magnitude of the obtained objective vector grows relative to the shrinking of the control error will be a function of the weights. The mixed optimization method, like the first branch of the dual branch, places no restriction on the obtained objective or the controls other than their bounds.
Chapter 6

Results

The control allocation approaches for higher dimensional numbers of controls described in Chapter 5 were implemented as a series of MATLAB\textsuperscript{TM} m-files. The tests were run in MATLAB\textsuperscript{TM} v6.5 R13 running on a Pentium II - 450MHz with 128Mbytes of RAM. To compare the computational complexity of different cases, the built-in MATLAB\textsuperscript{TM} \textit{cputime} function was used. Two separate sets of results are presented. Section 6.1 outlines the results of a recursive approach to the control allocation problem. Section 6.2 compares the results of different implementations of linear programming to solve control allocation problems.

6.1 Recursive Method

The recursive allocator described in Section 5.1 was implemented. The implementation made extensive use of the object notation presented in Section 2.2.1.1. This notation improves the ease of maintaining and understanding the code. The implementation makes use of recursive calls to evaluate the lower dimensional problems. The recursion reduces the amount of code and makes it follow the description more closely.

Recursion can impose some additional overhead due to the need to keep copies of the function and local variables on the heap. In general, iterative approaches to implementing an algorithm tend to be more efficient than their recursive counterparts[1]. The online help files for MATLAB\textsuperscript{TM} note that function calls are not as optimized as other operations. As such, recursion may carry even more penalty.

6.1.1 Tuning

The MATLAB\textsuperscript{TM} \textit{profiler} was used to compare what portions of the allocator required the most computational time. A four-dimensional problem with 25 controls which took nine
outer loop bisections to solve was used. The initial transformation to align the desired objective only accounted for 4.4% of the total execution time for this problem. The most significant individual functions were the check to see if the current object was a solution and the two-dimensional sub-problem. Together, these accounted for nearly 75% of the total execution time. Most of the work in the two objective problem was in a sort carried out to order the controls. This sort itself is one of the standard functions in MATLAB and uses the “quick-sort” algorithm. By itself, this sort did not take much time, but it was called 83 times in the course of the solution. Similarly most of the executable time in the check of the current object was spent in an internal MATLAB function used to determine if a given scalar was an element of a particular vector.

Modifying the routine to avoid these functions in some places involves producing user code which duplicates the built-in function. In other places, the work-around is less optimized for the interpreter than the original function. To see what effect the overhead of calling user functions had on the execution time, a version of the allocator which only worked in four dimensions was created by inserting the subfunctions inline into the main allocator. This new allocator solved problems approximately twice as fast. Inserting the subfunction code inline resulted in some known inefficiencies due to the reuse of variables and operations which became redundant with ones in the parent function.

If the tuning process were carried out in more detail, it is likely that the performance of the allocator would be improved somewhat. This could be useful if MATLAB were the final platform for the algorithm, but in general such improvements may not help in other environments. The results of direct comparisons between this algorithm and others invoking a different set of operations may be greatly influenced by the optimization carried out by the interpreter.

### 6.1.2 Timing Results

The recursive allocator was run for a series of problems with between 3 and 100 controls. All of the controls numbers from three to fifteen were tested, then problems with 20, 40, 60, 80, and 100 controls were tested. The number of objectives was varied from three to five. For each allowable combination of controls and objectives, 640 randomly created problems were solved. Each problem consisted of a control effectiveness matrix with terms randomly chosen to be uniformly distributed between -1 and 1. The control limits were chosen so that \( u = 1 \) and \( u = -1 \). The desired objective was also chosen to consist of uniformly distributed random components and then scaled so that the objective was unattainable.

Figure 6.1 shows the results of the three objective problems. The solid line shows the mean time for each number of controls. A curve representing the time in which 95% of the cases finish as well as the maximum time are plotted. The mean times increase nearly linearly with the number of controls. The 95 percentile level curve is also linear. The maximum time taken is seen to vary more. The highest times and most variability occur for smaller problems,
however there is an overall trend which increases as the number of controls increases.

A least squares approach was used to fit a line to the mean data. This trend line gives the mean CPU time in milliseconds as $1.39m + 63.218$. The correlation R value for this fit was 0.999014 indicating that the calculated means fall almost exactly on this trend line. The 95 percentile case could be closely fit with a line, $2.707m + 102.686$, with a correlation of 0.995. The slope of the 95 percentile is 1.942 times that of the mean case.

The three-dimensional case is included for comparison to previously reported results. While Durham[25] reported performance in terms of floating point operations rather than time some important observations can be made. In those results, the computational requirements seem to scale much more rapidly with increased controls. The intercept of the linear equations is much larger relative to the slope in the results above. Some portion of this offset is representative of the overhead required by the algorithm and is not a function of the controls.

The four-objective cases are seen in Figure 6.2. As in the three-dimensional case, the mean
Figure 6.2: CPU Time for the recursive allocator with four objectives. The average, maximum, and 95 percentile times are plotted. The average time in milliseconds varies linearly with the number of controls as $11.5m + 97.1$ and 95 percentile increases nearly linearly with the number of controls. A peak is seen in the max time at nine controls. This peak corresponds to a single case which took significantly longer than the maximum found at eight and ten controls. For large numbers of controls, there is a more obvious increase in the maximum time than there was for the three objective case.

The trend line for the four objective mean data is $time(msec) = 11.479m + 97.123$. The 95 percentile is fit as line $22.101m + 248.237$. These fits are nearly as good as those in three dimension, the correlation is 0.999 for both trends. The ratio between the slopes of the two trend lines is 1.925.

The five objective recursive approach is shown in Figure 6.3. Qualitatively this graph is similar to the previous two. The mean and 95 percentile case are nearly linear; however, the times and the slopes relative to the controls are much larger. One 100 control problem took nearly 37 seconds. The average case took nearly five seconds to handle a 50 control problem.
The average, maximum, and 95 percentile times are plotted. The average time in milliseconds linearly varies with the number of controls as $91.3m + 109.9$.

Least squares lines are $91.302m + 109.963$ and $199.032m + 824.105$. Both lines have a correlation of 0.999. Once again the ratio of the slopes of the two trend lines is nearly two—though this time it is slightly higher, 2.18.

The results for the recursive allocator show it to become very slow as the number of objectives is increased (Figure 6.4). Problems with six or more objectives often took several minutes to solve. These runs were terminated before getting sufficient data to plot.

All of the 32640 cases presented Figures 6.1-6.3 were able to successfully find a solution. The maximum times across all of the objectives could take more than twice the average time. The average time grows rapidly with the number of objectives. It is expected that the execution time scales with the power of the number of bisections required for the average or 95 percentile case.
Figure 6.4: Scaling of computational time with objectives and controls
6.1.3 Estimation

The results from two different plans for terminating the search early and estimating the answer are presented. One plan is to conduct the search as normal, but to estimate the solution if the search does not terminate after a set number of bisections. The second plan is to always run the search for a certain number of bisections without checking the solution and then to estimate the solution. This second approach may be more appropriate in realtime implementations where the desired frame rate is known.

The process of estimating the solution was carried out using the pseudo-inverse to solve for the intersection of the objective direction with the current object. If the pseudo-inverse saturated one of the controls on the current object without finding the solution, that control’s contribution was removed and a CGI approach was used. A handful of the cases tested failed to find a control in the estimator. Most of these cases corresponded to problems where the number of bisections was not great enough to distinguish two distinct objects. For practical searches, this is not a problem as a larger can be chosen to be larger. A small group of cases (much less than 1%) corresponded to legitimate failures of the estimator. It is hypothesized that these cases are caused by small difference between $\Pi$ for the CGI and $\Phi$.

The results of timing the estimated algorithm in four and five dimensions are presented in Figures 6.5 and 6.6. Cases where there are four, six, and nine bisections are highlighted. For each case, the average CPU time of both estimation approaches is presented. Comparing Figures 6.2 and 6.5, the speed for smaller numbers of controls is similar for the allocator without the estimator and for the one which will exit early if a solution is found.

The curves corresponding to the method that always runs for a fixed number of bisections are linear. This is expected; other than the bisection loop the only other places where the number of computations is variable is in the solution of the two-dimensional problem. The variable portion of this problem is dwarfed by the time taken by the sorting routine. Additional bisections also take more time as expected.

For small numbers of controls where the average number of bisections required to find the solution facet is small, the allocator which exits early takes similar time regardless of the number of bisections. This allocator initially performs better than the fixed one because it can exit early. Eventually, however, the situation changes. For four objective systems with more than 40 controls, four bisections is quicker with the fixed algorithm. For six bisections, the two plots cross at around 85 controls. The number of controls was not made large enough for a fixed nine-bisection algorithm to be more efficient.

The five objective case (Figure 6.6) is similar to that in four objectives. The fixed iteration approach now starts with so much overhead and grows so fast that the other algorithm can not catch it in the range plotted. The amount of time required for nine bisections is dramatically different than that for fewer correct.

Figure 6.4 summarizes the results of testing the recursive allocator. The time is seen to
Figure 6.5: CPU Time for the recursive allocator w/ estimator for four objectives.
The average time is shown for allocators that exit early if the solution is found or allocators that always perform a fixed number of bisections. Results for 4, 6, and 9 bisections are plotted. The tradeoff between the expense of checking a solution every iteration and being able to exit early is evident.
Figure 6.6: CPU Time for the recursive allocator w/ estimator for five objectives.
The average time is shown for allocators that exit early if the solution is found or allocators that always perform a fixed number of bisections. Results for 4, 6, and 9 bisections are plotted. For the 6 and 9 bisection case, the time required is much higher and the fixed iteration allocators are never more efficient for the controls plotted. With large numbers of controls, it is more efficient to perform a fixed number of bisections in the four bisection case.
vary with both the number of controls and with the number of objectives. As dimension is increased, the time it takes to solve problems of a given size also increases in a non-linear fashion. If the average number of bisections is relatively constant, the computation time would be expected to scale as \( n^{n_{\text{bisections}}} \). The increase in the number of dimensions also affects the ratio of time to number of controls.

### 6.1.3.1 Error Frequency

One of the desirable properties of the BESA estimator was that the number of cases invoking the estimator was small and decreased as the number of controls increased. Figure 6.7 shows the number of estimates required by the recursive allocators for a three objective case. This plot counts any calls to the estimator which return the correct solution as having not been estimated. The other major difference between these allocators and the original BESA is that the original BESA begins its search with \( \phi = \pi / 4 \) and these start with \( \phi = \pi \).

Figures 6.7, 6.8, and 6.9 show the percentage of cases involving estimates for three, four, and five objective problems. Three different numbers of controls are plotted. In general, the larger number of controls require more bisections to reduce the percentage of estimates below a certain level. The higher dimensional problems also require more bisections. All of the cases seemed to get the correct solution greater than 95% of the time by the time nine bisections are reached. In the lower dimensional cases, the error rate is seen to level off after five to seven bisections. The error never completely disappears as there is a small number of problems requiring a large number of bisections to find the correct solution.

The five objective, forty-control problem still had a large number of error cases, greater than one fourth, even after six bisections. When the estimator is going to be called a large number of times, the error being returned is important. Figures 6.10- 6.12 show the average amount of error due to the estimator.

The problems with forty controls have the least error for small numbers of bisections. As the number of bisections is increased, the error also drops fast. Going from a three-objective to a five-objective problem increases the mean error almost by a factor of 10.

The magnitude of the error can be decreased by increasing the number of bisections. The desirable error properties of the BESA are that error frequency and magnitude decrease as the number of controls increases. These properties appear to hold for the recursive allocator as well. The number of errors is seen to increase as the number of objectives is increased.
Figure 6.7: Percentage of cases which estimate a solution in $\mathbb{R}^3$
Larger numbers of controls require estimating more solutions. After 7 iterations, the error percentage is nearly constant.

Figure 6.8: Percentage of cases which estimate a solution in $\mathbb{R}^4$
Larger numbers of controls require estimating more solutions. After 7 iterations, the error percentage is nearly constant. In general, these percentages are higher than those for the three-dimensional case.
Figure 6.9: Percentage of cases which estimate a solution in $\mathbb{R}^5$
Larger numbers of controls require estimating more solutions. In general, these percentages are higher than those for lower dimensional cases.

Figure 6.10: Normalized mean error as a function of controls for three objectives
The mean error is smaller with higher bisections and tends to decrease as the number of controls is increased.
Figure 6.11: Normalized mean error as a function of controls for four objectives
The mean error is smaller with higher bisections and tends to decrease as the number of controls is increased.

Figure 6.12: Normalized mean error as a function of controls for five objectives
The mean error is smaller with higher bisections and tends to decrease as the number of controls is increased.
6.1.4 Discussion

The recursive algorithm defined scaled linearly with the number of controls as expected. The actual execution time involved, however, was quite large. For the three-dimensional case with a fixed number of bisections, the recursive allocator should follow nearly the same approach to finding a solution as the BESA allocator. The differences in the starting points of the bisection and estimator are not thought to be large enough to account for the difference in performance. Some of the difference in performance may be due to increased overhead from the large numbers of recursive and non-recursive function calls. Difference in the ordering and syntax used for operations may also influence how the $MATLAB^{TM}$ interpreter is able to optimize and evaluate expressions.

If the performance of the recursive allocator for three dimensions can be made to equal that of the BESA approach and a proportional improvement is seen in higher dimensions, then this algorithm may be useful for systems with four or five objectives. The linear scaling with controls and the error properties of the estimator could be attractive for the same reasons the BESA is of interest in three dimensions. Increasing the system much beyond five dimensions quickly becomes inefficient. Even if the performance is improved in the three objective case, unless the rate at which it scales with increased objectives is changed, computational time required for higher numbers of objectives is still large.

6.2 Linear Programming

6.2.1 Solvers

Initially five different approaches were compared to choose a solver to use inside of the linear programming based allocators. A standard simplex implementation, a simplex routine with upper bounds incorporated, the revised simplex, the revised simplex with upper bounds and $MATLAB^{TM}$’s internal linear programming solver were tested. In order to find feasible programs to test the solvers, problems based on the reduced direction preserving format (section 5.2.2.2 for control allocation were created.

The results of these tests appear in Figure 6.13. The average time for 150 test cases for controls from three to fifty are shown for three and four objective problems. Each of the three different types of solver displays a different trend with increasing numbers of controls. The execution times of the two simplex-based solvers appear to grow quadratically with the number of controls. The revised simplex routines only show a linear increase.

The solvers with upper bounds have some additional computation required at each iteration; however, the alternative of adding slack variables to the problem doubles the number of unknowns. For both the revised and regular simplex solvers, the additional computations appear to be made up for by the reduction in variables. The cases where upper bounds were
Three Objective

Four Objective

Figure 6.13: Comparison of different linear programming solution techniques
The same trends are shown for three and four objective cases. Note that the revised simplex scales linearly with $m$ for these problems.

included in the solver start at approximately the same point as the methods they were based on, but as the number of controls increase, they grow with a slower rate.

MATLAB’s internal solver, function `linprog`, was set to use a primal-dual interior point method for solution. This method is optimized for large scale problems but was included here as an alternative to the simplex methods. The relative tolerance on the cost function for convergence was set to 0.01. The interior point solution only shows a slight increase in execution time as the number of controls is increased from three to fifty and no significant change from three to four objectives. Some amount of preprocessing of the problem is done internally before attempting to solve it. Because the control allocation problem is so small, it is expected that this preprocessing is providing the bulk of the overhead for the solution. For large numbers of controls, this solver begins to be competitive with the revised simplex formulations.

The revised simplex method with upper bounds is the fastest solver of those tested. Additionally, the change in time required with increasing controls is slow compared to the the
6.2.2 Control Allocation

The five formulations of the control allocation problem presented in 5.2.2 were implemented as Matlab m-files which called the revised simplex with upper bounds tested above. The allocators were tested for a series of randomly created problems. The allocators were tested for cases with three, four, and five objective. For each number of objectives, cases were defined with controls in the range of three to fifteen and then from twenty to 100 in steps of twenty. For each case, 200 problems were generated with the elements of $B$ and $y_d$ chosen as uniformly distributed random numbers between -1 and 1. The maximum attainable objective in the direction $y_d$ was found, and two new objectives were defined—one which desired 150% of the attainable objective and one which asked for 75%. Each allocator was tested with the attainable and unattainable objectives, and the results tabulated separately.

6.2.2.1 Direction Preserving

The direction preserving allocator defined in Equation 5.2.2.1 was implemented. The crash procedure described in Equation 5.13 was used to initialize the solver. The results from the tests of this formulation are presented in Figure 6.14. The three plots show the execution time for cases with three, four, and five objective dimensions. For each plot the results from the unattainable and attainable desired objective are presented separately. In each case the solid line represents the mean CPU time and the dotted line is the maximum.

The average performance in each case scales linearly with the number of controls. A least squares approach was used to put a line through the average data for each of the feasible cases. The results of this fit is presented in Table 6.1. The trend with the number of objective dimensions appears to be a linear change in both the slope and intercept of the fit.

The attainable cases are found about twice as fast as the unattainable cases. Because the only constraint is that the attained objective is in the same direction as the desired, there are an infinite number of solutions which will meet this constraint. The simplex solver limits this to a smaller set of basic solutions; however, the allocator will still return whichever is found first. The maximum time required for the attainable case is nearly the same as the mean for the unattainable case except for a couple of data points in the four objective case.

6.2.2.2 Reduced Direction Preserving

Figure 6.15 shows the results of tests done for the reduced formulation for direction preserving control allocation. Once again results for both attainable and unattainable commands are plotted. The results are nearly the same for two sets of commands. By eliminating the
Figure 6.14: Performance of direction preserving linear program for attainable and unattainable commands. In each case the mean performance is indicated by the solid line and the maximum by the dotted line.

Table 6.1: Least squares fit of cputime(msec) = a m + b to mean attainable Direction Preserving data.

<table>
<thead>
<tr>
<th>Objectives</th>
<th>a</th>
<th>b</th>
<th>$R^2$</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.348</td>
<td>5.247</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.394</td>
<td>6.266</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.457</td>
<td>7.230</td>
<td>0.998</td>
<td></td>
</tr>
</tbody>
</table>

scaling parameter from the equations, this program finds the same solution on the $\hat{\Phi}$ in each case. In the attainable cases, the desired solution is found by scaling from this solution. The only difference between the computations required in the two cases is the final division required to scale the objective in the unattainable case. A linear fit to the mean attainable data is found in Table 6.2.
For the three objective problem, this method proves to be slower at solving the attainable objectives than the larger direction preserving formulation in the previous section. The elimination of one constraint equation requires additional matrix multiplications to set up the problem. For higher numbers of objectives, the performance is nearly the same as that for the feasible case above.

![Graphs showing CPU time for 3, 4, and 5 objectives for attainable and unattainable commands.](image)

Figure 6.15: Performance of reduced direction preserving linear program for attainable and unattainable commands. In each case the mean performance is indicated by the solid line and the maximum by the dotted line.

Table 6.2: Least squares fit of \( cputime(\text{msec}) = a m + b \) to mean attainable Reduced Direction Preserving data.

<table>
<thead>
<tr>
<th>Objectives</th>
<th>a</th>
<th>b</th>
<th>( R^2 ) Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.665</td>
<td>3.067</td>
<td>0.999</td>
</tr>
<tr>
<td>4</td>
<td>0.763</td>
<td>3.497</td>
<td>0.999</td>
</tr>
<tr>
<td>5</td>
<td>0.852</td>
<td>4.052</td>
<td>0.999</td>
</tr>
</tbody>
</table>
6.2.2.3 Dual Branch

The results of tests with the dual branch program are shown in Figure 6.16. The weighting vectors were chosen to weight all elements of the control and objective evenly, and a desired control deflection of 0.1 was chosen for all effectors. The initial conditions for both the feasibility and sufficiency programs are available and eliminate the need for the crash procedure which may offset some of the additional costs of performing two separate optimization steps. The results indicate that it is only slightly slower than solving the attainable problem with the direction preserving scheme. Though the program has more unknowns than the direction preserving schemes, the crash procedure was not required offering computational savings. The attainable and unattainable commands result in similar results. The added cost of the sufficiency branch is not high and is mostly offset by quicker solutions of the feasible branch for attainable commands.

Figure 6.16: Performance of dual branch linear program for attainable and unattainable commands.
In each case the mean performance is indicated by the solid line and the maximum by the dotted line.

Linear equations fit to the attainable results (Table 6.3) indicate that time required scales
with controls at nearly the same rate as the Direction Preserving example. The main difference is that the offsets are approximately 3 milliseconds higher. The worst case times for the dual branch approach occurred for problems with few controls and were larger than most of the maximums encountered with the other allocators.

Table 6.3: Least squares fit of cputime(msec) = a m + b to mean attainable Dual Branch data.

<table>
<thead>
<tr>
<th>Objectives</th>
<th>a</th>
<th>b</th>
<th>$R^2$</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.330</td>
<td>8.230</td>
<td>0.996</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.390</td>
<td>9.220</td>
<td>0.997</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.480</td>
<td>9.870</td>
<td>0.998</td>
<td></td>
</tr>
</tbody>
</table>

6.2.2.4 Single Branch

The single branch approach offers the potential to balance a direction preserving scheme with one which minimizes control error. The results of tests of the single branch allocator are presented in Figure 6.17. The control weighting for the single branch was chosen so that all controls were weighted equally at 0.01 times the weighting of the objectives.

The results once again are nearly linear (Table 6.4). The average time for the unattainable cases is not a linear function of the controls. Initially unattainable cases take longer to solve than the attainable cases. As more controls are added to the problem, the time to solve an unattainable case eventually begins to level out and the attainable cases become more expensive. If the objective error cannot be met, the direction-preserving term dominates. As more controls are added with the relative costs kept the same, there is the possibility that these controls, even at a lower weight, can drive the cost function faster than the objective error.

On the boundary of $\Phi$ at least $m - 3$ controls must be at their limits. If the direction of the command is preserved, the cost from the objective term will be -0.75. The cost contribution from the controls will be large just by nature of sheer numbers. If there are 50 controls, then the control cost must be at least 0.47. This is a large contribution; moving the solution to the interior of $\Phi$ may result in a greater reduction of control cost than the increase in cost of the scaling parameter.

6.2.2.5 Mixed Optimization

Like the single branch method, the mixed optimization formulation offers the ability to combine desired behaviors for attainable and unattainable solutions into a single step. In order to be an alternative to the dual branch method this approach, was set up to allow for each element of the control error to be weighted individually. For these tests, these weights
Figure 6.17: Performance of single branch linear program for attainable and unattainable commands. In each case the mean performance is indicated by the solid line and the maximum by the dotted line.

Table 6.4: Least squares fit of \( \text{cputime(msec)} = a \cdot m + b \) to mean attainable Single Branch data.

<table>
<thead>
<tr>
<th>Objectives</th>
<th>a</th>
<th>b</th>
<th>( R^2 ) Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.767</td>
<td>8.677</td>
<td>0.997</td>
</tr>
<tr>
<td>4</td>
<td>0.881</td>
<td>8.753</td>
<td>0.998</td>
</tr>
<tr>
<td>5</td>
<td>1.047</td>
<td>8.287</td>
<td>0.999</td>
</tr>
</tbody>
</table>

are all chosen to be identical and 0.01 the weights on the objective error. The results are shown in Figure 6.18.

The addition of individual control weighting makes the program solved in the mixed optimization approach larger than any single program solved in the other cases. The results show the mean case to vary linearly with the number of controls; however, the slope of this
Figure 6.18: Performance of mixed optimization linear program for attainable and unattainable commands. In each case the mean performance is indicated by the solid line and the maximum by the dotted line.

The attainable case was executed a little faster than the unattainable; in general, this method was slower than the others. For large numbers of controls, mixed optimization was slower than the single branch method.

Table 6.5: Least squares fit of $\text{cputime (msec)} = a \cdot m + b$ to mean attainable Mixed Optimization data.

<table>
<thead>
<tr>
<th>Objectives</th>
<th>a</th>
<th>b</th>
<th>$R^2$ Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.802</td>
<td>1.919</td>
<td>0.999</td>
</tr>
<tr>
<td>4</td>
<td>0.929</td>
<td>1.969</td>
<td>0.999</td>
</tr>
<tr>
<td>5</td>
<td>1.082</td>
<td>1.481</td>
<td>0.999</td>
</tr>
</tbody>
</table>
6.2.3 Discussion

Because linear programs based on the control allocation problem often have many more unknowns than equations, they can be solved with routines whose run time scales linearly with controls. This is as opposed to the quadratic scaling which has been suggested as a drawback of linear programming in the past. One of the potential advantages of linear programming is the ability to define conditions for resolving redundancy for attainable cases as part of the main solution. Once the desired objective is attained, a cost function expressed in terms of control positions can be minimized. Resolving redundancy in this manner can add to the overall complexity of the program, so these conditions should be chosen carefully.

It is not entirely obvious from the size of the programs being solved what the performance will be. The mixed optimization approach would seem to be an improvement over the dual branch method since it optimizes the same functions with a single step instead of two steps. However, the increased size of this program cancelled any benefit from calling a single program. The dual branch approach also performed well compared to the direction preserving programs.
Chapter 7

Conclusions

Most of the research into control allocation strategies has focussed on the three objective problem. From an aircraft control standpoint, this reflects the classical approach of providing the pilot with three inputs to control the system. The desirability of providing a pilot with direct control over more degrees of freedom has been debated. Even if pilots are limited to three inceptors, control law design can still take advantage of the degrees of freedom. Current allocation strategies provide the ability to implicitly optimize additional degrees of freedom by using excess capabilities once the desired commands have been met. Allocating more objectives will enable control systems to make full use of available control power to explicitly track commands for these additional objectives.

Current research in control allocation has looked for methods which are efficient in their computation requirements. The desire to allocate an increasing number of effectors in real-time has led to the desire for allocation methods which scale well as controls are added. The Bisecting Edge Search Algorithm is a method for allocating control effectors which scales linearly with the number of controls.

This research was originally inspired by a requirement to extend the BESA to work with four objectives. Several approaches to extending this algorithm were undertaken. Approaches based on simply extending the edge search into four dimensions were seen to be ineffective. An extension of the basic algorithm to search for facets instead of edges was seen to offer a four-objective approach which worked. Experimental results showed this algorithm to scale linearly with the number of controls.

Along the way of extending the BESA allocator, the question of estimated solutions was considered. Conditions which resulted in the original allocator being forced to estimate a solution were explored. It was found that these cases could generally be made to return the optimal solution through further bisections or by calling the estimation routine. These special cases have been mentioned as frequently occurring for some configurations of aircraft—in particular tailless aircraft or those with many split surfaces.
The four-dimensional allocator offered an obvious path to an allocator which would work in an arbitrary number of dimensions. This recursive allocator was implemented and tested. The results show linear scaling in the number of controls; however, the scaling with the number of objectives quickly renders the method unusable. The overall computational time used by the recursive allocator was much higher than expected. Even for the three-dimensional case in which the recursive allocator should duplicate the behavior of the BESA, computational requirements were not of the same order of magnitude. It is not known whether this is an inefficiency inherent to the recursive nature of the algorithm or an artifact of this particular implementation.

A survey of proposed formulations of the control allocation problem as a Linear Program was undertaken. Originally, the goal of this survey was to compare these methods with the four-dimensional extensions to the BESA algorithm being developed. Most of the control allocation references using linear programming have been implemented using existing solvers, resulting in concerns about their efficiency. Recently, it has been suggested that there are, readily available in the linear programming literature, methods which can improve the performance for problems on the scale of most control allocation problems.

A revised simplex method which explicitly handled upper bounds was found to offer performance which was linear in the number of controls. Five formulations of the control allocation problem were compared and seen to offer reasonable performance. In general, the average performance of a linear programming solution is difficult to predict from the structure of the program. Linear programming does, however, offer the control designer potentially more flexibility for handling redundancy.

The results presented demonstrate linear programming can solve control allocation problems at least an order of magnitude faster than the recursive allocator. Because of the discrepancies in the measured performance, it is difficult to make any conclusive statements comparing the two. The observed behavior of the linear programming solutions, taken by itself, suggests it to be a viable approach for control allocation.

There are many things which could be gained by using linear programming to solve control allocation problems, including the potential for more flexibility in posing the problem. One intangible, but important, benefit is the large body of existing work and expertise in linear programming present in the math and optimization communities. The many advantages of linear programming make it attractive as an area for continued research into higher dimensional control allocation. If existing questions about fixed frame times and early termination of linear programming solvers—as well as the acceptance of online optimization into control systems—can be addressed, there is little reason for continued work on the recursive approach developed in this work.

No matter what solution method is used, the question of how to handle redundancy for attainable commands and what is the appropriate method for resolving unobtainable commands is an open one. Systems which allocate more than three objectives will likely mix objectives of different physical nature and units. The standard error minimization and di-
rection preserving approaches may make little sense in such systems. Future research into this area as well as other interactions between control systems and control allocators could be useful.
Bibliography


Vita

Roger Beck was born in Concord, Massachusetts, in early spring 1977. With an older brother born in Georgia and parents from South Carolina and Massachusetts, Roger grew up in the approximate mean family birth site, Mount Vernon, Virginia. Roger became involved in his local church and in Boy Scouts, eventually becoming an Eagle Scout.

After graduating from the Thomas Jefferson High School for Science and Technology, Roger searched for a school with a shorter name before eventually enrolling at the Virginia Polytechnic Institute and State University. During the summer of 1997, the opportunity arose to work as a “simulator-slave” in the Flight Simulation Laboratory. A summer spent fixing interlocks and building duck boards and Roger was hooked. After a senior year which included too many late night trips to the observatory and too little sleep, Roger began graduate school. Along the way he has gotten married to his wonderful wife Catherine, and they look forward to continuing life in the high desert of California.