Turbojet Range, Loiter, and Altitude Tradeoff Estimation in Efficient Modeling and Optimization Formulations

THESIS

Lauren M. Bramblett, 2nd Lt, USAF

AFIT-ENS-MS-19-M-102

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-19-M-102

TURBOJET RANGE, LOITER, AND ALTITUDE TRADEOFF ESTIMATIONS IN EFFICIENT MODELING AND OPTIMIZATION FORMULATIONS

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Lauren M. Bramblett, B.S.

2nd Lt, USAF

March 21, 2019

AFIT-ENS-MS-19-M-102

TURBOJET RANGE, LOITER, AND ALTITUDE TRADEOFF ESTIMATIONS IN EFFICIENT MODELING AND OPTIMIZATION FORMULATIONS

THESIS

Lauren M. Bramblett, B.S.
2nd Lt, USAF

Committee Membership:

Dr. L. E. Champagne
Chair

Dr. R. R. Hill
Reader

AFIT-ENS-MS-19-M-102

Abstract

Identifying accurate tradeoffs between loiter time, range, and loiter altitude for jet aircraft are critical for mission planning, but current methods are cumbersome and time and labor intensive. This research introduces a new method for determining tradeoffs between loiter time, range, and loiter altitude for jet aircraft utilizing specific aircraft performance characteristics and the Bréguet range and endurance equations. Results from this new methodology are compared to the existing model used by the National Air and Space Intelligence Center for flight planning. This method is then extended and applied to several routing assignment problems, taking advantage of the incorporation of performance characteristics in their formulation. Example problems are generated using binary integer programming models and nonlinear mixed integer programming models.

Acknowledgements

I would like to thank my husband and parents for their help and support, without them this would not have been possible. In addition, I would like to thank my faculty research advisor, Dr. Champagne, and committee member, Dr. Hill, for their support in the development of this research. Lastly, I would like to thank my classmates for always supporting my daily visits to the coffee shop and weekly visits to get fried chicken.

Lauren M. Bramblett

Table of Contents

Page

List of Figures

List of Tables

TURBOJET RANGE, LOITER, AND ALTITUDE TRADEOFF ESTIMATIONS IN EFFICIENT MODELING AND OPTIMIZATION FORMULATIONS

I. Introduction

1.1 Overview

Range and loiter time have been explored in the field of aeronautics by many individuals and companies, seeking to maximize both separately. However, since there is little utility for such use in a commercial environment, the trade-offs of range and loiter time for an aircraft have not been widely explored. The National Air and Space Intelligence Center (NASIC) seeks an optimization tool that efficiently identifies the range, loiter time, and loiter altitude tradeoffs while minimizing error. Customers of NASIC are constructing aircraft mission plans and are looking for a spectrum of flight plans that meet their mission criteria. Analysts are currently using labor-intensive methods to provide range, radius, and time estimates for specific aircraft configurations. As a result, improved methods integrated into the process could result in significant time and cost savings.

1.2 Background

The mission planning tool used by the National Air and Space Intelligence Center allows for a selection of aircraft, standardized load out, and flight profile. The outputs of the current tool are mission range or loiter time depending on initial user inputs and, depending on the feasibility of those inputs, may output an error. The program mainly operates from a fuel constrained perspective and provides the outputs

based on fuel reserve after user-defined parameters such as warm up, take-off, cruise altitude, length of combat, and fuel reserve.

The high fidelity of the mission planning tool allows finding specific points, and a trade-off of loiter time and range by interpolating between these points, which are in close proximity. This aspect of finding the frontier of loiter time and range is time consuming and must be done by hand by analysts familiar with the mission planning tool. The intent to find a trade-off is similar to Snyder [2] finding the performance tradeoffs in various rotocraft. Some have used the simplification of equations to maximize range [3, 4], while Raymer [5] found methods of approximating endurance from range at specific points. Extensions of this problem in Alighanbari et al [6] formulated an optimization problems with endurance constructed as an objective and timing as the constraints of multiple UAV control.

1.3 Motivation

The current method of deriving an optimal range and loiter time for a customer with at least one given as a parameter gives results that are accurate within five percent of actual, but are time consuming and inefficient to calculate. Additionally, the sensitivity of the current problem is unknown, given that the current method requires tabular calculations. The use of goal programming and optimization methods are used to leverage the parameters in this dynamic problem and simplify the initial model. The geometric equivalence of the current solution method is used as a backbone for finding the vertical distance of two intersecting polynomials, representing aircraft's ingress to a range and egress to a base, using the result as an objective.

The primary objective of this research is to develop an efficient tool to accurately find the frontier of optimal ranges and loiter time for customers seeking options for their aircraft. A secondary objective is to find a full solution for all possible ranges

and loiter times at multiple altitudes with as few runs as possible, illustrating the trade-offs between a longer range or a longer loiter time and applying these findings to a relevant assignment routing problem.

Chapter 2, reviews current methodologies involving the estimation of range and flight time through various equations and their applications. Chapter 3 presents a new methodology to estimate the tradeoff frontier of range and loiter time for specific aircraft flight characteristics and applied to a geographic scenario. Chapter 4 tests this method against a known high fidelity model for accuracy. Chapter 5 focuses on the application of this methodology to an assignment problem. Finally, Chapter 6 concludes with insights gained during the process and propose further research and methods to hone and apply the new methodology.

II. Literature Review

2.1 Overview

This chapter defines aircraft performance characteristics and discusses the generalized range equation for jet aircraft and the simplifications known as the Bréguet range and endurance equations used by Cavcar [3], Raymer [5], and Chuck et al. [7] for design optimization as well as the incorporation of multiobjective optimization in flight optimization. The derivation of these equations rely on several key assumptions which define the flight plan of an aircraft and ascertain a conservative assessment of maximum range and endurance. Specifically, these equations rely on a constant thrust-specific fuel consumption ($TSFC$) and a constant lift over drag, which was used by Vitte [4] for optimization of continuous target coverage and shown effective for evaluating the number of UAVs needed for a mission. Noteably, the range and endurance equations can be derived in terms of each other [5], making multiobjective optimization more useful if these two equations can be constrained by similar terms.

2.2 Aircraft Performance Characteristics

Lift of a specific aircraft is generated by flight conditions and angle of attack. The lift coefficient depends on wing angle of attack, wing span, aircraft speed and other factors of the aircraft design including the fuselage, engine nacelles, and horizontal tail. Thus, any changes to angle of attack, speed, or design can impact the lift coefficient.

The total drag on an aircraft is influenced by speed, profile drag, and induced drag and varies based on lift. These are represented by a drag polar:

$$C_D = C_{D_0} + kC_L^2$$

where C_D is total drag, C_L is the coefficient of lift, and C_{D_0} is the parasite drag coefficient and represents all drag generated by an aircraft when not experiencing lift. The constant k is an expression where $k = 1/(\pi A_R e_0)$. A_R is the aspect ratio of an aircraft and e_0 is the Oswald's efficiency factor for a three-dimensional wing's change in drag as compared to an ideal wing with the same aspect ratio. Since both parasitic drag and k remain constant during flight, a change in the coefficient of lift will impact the coefficient of drag and vice versa with the added impact of mach number shown in Figure 1. Generally, the aircraft becomes less efficient at a higher mach number

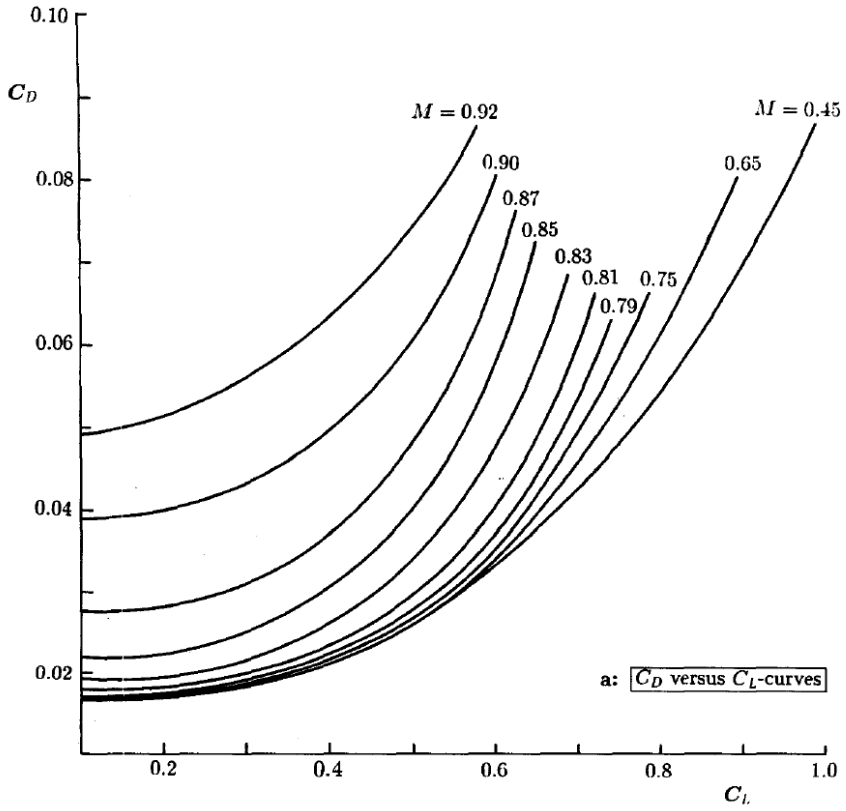


Figure 1. Coefficient of Lift versus Coefficient of Drag with Mach Number [1]

which impacts the C_L/C_D and in turn effects the maximum range and/or loiter time of the aircraft.

The production of thrust allows the aircraft system to be most suitable for specific

flight regimes. There are two main characteristics of propulsion systems. The first is the ratio of an engine's sea level output to its own weight which is set for all aircraft examined in this research. The second is thrust-specific fuel consumption and is more interesting to the given problem. Thrust-specific fuel consumption ($TSFC$) is the ratio of fuel consumption to thrust output:

$$TSFC \equiv \frac{\dot{W}}{T}$$

where $\dot{W}$ is the rate of fuel consumption and T is thrust output. This equation determines operating envelopes for mach number and an engine's maximum thrust-to-weight ratio. Additionally, thrust output is impacted by altitude and thus changes depending on the altitude of a flight profile.

2.3 Bréguet's Range Equation

Bréguet's range equation combines specific fuel consumption, lift, drag, cruising speed, and initial and final fuel weights with aircraft weight. The derivation of this equation starts with several steps that are important for the sake of assumptions used in each simplification. Eq. 1 is integration over the change in weight to solve for distance. The integration for the change in weight does not account for changes in the flight characteristics of the aircraft such as lift over drag, velocity, and thrust-specific fuel consumption which are variable depending on weight. These can be generalized by assumptions or by integrating over small weight changes with given parameters. The in-depth derivation of these equations are discussed in Section 3.3.

$$R = \int_{W_f}^{W_i} \frac{V dW}{(TSFC)D} \tag{1}$$

where V is the speed of the aircraft, D is drag, and W_i, W_f are the initial and final weight of the aircraft, respectively.

Assuming constant altitude, constant angle-of-attack, and constant thrust specific fuel consumption, the equation simplifies to Eq. 2.

$$R = \sqrt{\frac{2}{\rho_\infty S}} \left(\frac{1}{TSFC}\right) \left(\frac{C_L^{1/2}}{C_D}\right) (\sqrt{W_i} - \sqrt{W_f}) \tag{2}$$

where ρ_∞ is the atmospheric density at altitude, S is the wing area, and C_L/C_D is the coefficient of lift over coefficient of drag. The simplicity of this equation lends itself well to optimization of more complex problems.

If, however, the assumption for constant altitude is relaxed, but constant airspeed is assumed, the equation relies on fewer constants and becomes Eq. 3.

$$R = \frac{V}{TSFC} \left(\frac{C_L}{C_D}\right) \ln \frac{W_i}{W_f}. \tag{3}$$

Chuck et al. [7] use Eq. 3 to compare fuel design effects on range. Mostly due to its logarithmic design, this equation accounts for fuel efficiency in that as an aircraft burns fuel, the lighter the aircraft becomes and thus can travel farther. This makes the use of this equation more realistic for longer enduring aircraft.

The final well-established derivation of the range equation is the equation assuming a constant airspeed, constant altitude, and parabolic drag polar. A parabolic drag polar more closely mimics the drag of an aircraft during flight and fuel burn than assuming constant drag in previous equations. Eq. 4 shows the result of these assumptions on Eq. 1. The equation assumes that there is an initial coefficient (C_{L_1}) of lift and final coefficient of lift (C_{L_2}) with a coefficient of lift for maximum range ($C_{L_{MR}}$).

$$R = \frac{2V}{TSFC} \left(\frac{L}{D}\right)_{max} \left[\arctan \frac{C_{L_1}}{C_{L_{md}}} - \arctan \frac{C_{L_2}}{C_{L_{md}}}\right] \tag{4}$$

where $C_L/C_D = L/D$. The main limitation of this equation is the inability to project an aircraft into future space with limited given parameters such as a drag polar.

The assumption that lift and drag are constant can simplify the coefficients of lift and drag for maximum range. Eqs. 5 and 6 illustrate these simplifications as shown in the basic performance equation introduced by [4].

$$C_{L_{MR}} = \sqrt{\frac{C_{D_0}}{3k}} \tag{5}$$

$$C_{D_{MR}} = \frac{4}{3} C_{D_0} \tag{6}$$

where C_{D_0} is parasitic drag and $C_{D_{MR}}$ is the coefficient of drag for maximum range. Jonas [8] argues that these equations are not accurate at predicting realistic results when accounting for changes in weight over the course of an aircraft's flight due to fuel burn. The new derivation of the range equation [8] where weight is dynamic produces the following equation:

$$R = (b/a)[\arctan(W_i/a) - \arctan(W_f/a)]. \tag{7}$$

The constants a^2 and b are given by

$$\begin{aligned} a^2 &= \frac{C_0 qS + C_1 C_{D_0} (qS)^2}{kC_1} \\ b &= \frac{VqS}{kC_1} \end{aligned} \tag{8}$$

where C_0 and C_1 are constants for altitude and airspeed for a problem instance and $k = 1/(\pi A_R e)$. The results of these derivations were applied to a hypothetical jet [8]. The results showed a range only 0.77% higher than the exact value obtained from the original range equation. The new method was considered accurate as compared to

the Bréguet range equations and validated the use of aircraft and engine parameters available. This showed that regardless of the original hypothesis that the Bréguet range equation was an inaccurate depiction of true range, the Bréguet range equation performs well when compared against equations that use less assumptions.

As stated before, the weight change over time changes the fuel efficiency of an aircraft. Jonas [8] derives an equation involving the overall efficiency slope, m that describes fuel efficiency. Eq. 9 shows the results of this derivation.

$$R = \frac{aVJC_H}{10,560m} log_e \frac{W_i}{W_i - W_F} \tag{9}$$

where V is airspeed in feet per second, J is a constant representing mechanical heat at 778.26 foot pounds per BTU, C_H is the fuel heat value in BTU per pound, and a is equivalent to $2C_L/C_D$. These values are not included in the available data for this research and so the most useful equation was derived by Jonas [8] shown in Eq. 10.

$$R = \frac{W_i}{1.467} \frac{V}{Q_I} \ln \frac{W_i}{W_i - W_f} \tag{10}$$

where Q_I is θ_f/θ_i which is the ratio of altitude density over the course of the cruise. This equation is useful when reducing the assumptions in a flight plan since simplifying assumptions are made about the ratios between density, weight, and lift over drag. Eqs. 9 and 10 allow for the removal of the constant altitude assumption. These results were also compared to the original Bréguet range equation's simplifications using various assumptions for accuracy.

2.4 Endurance Equation

In addition to maximum range, loiter time is needed and relates to the endurance equation. Endurance is the calculated time that an aircraft can remain in the air.

The general endurance equation is represented as

$$\Delta t = \frac{\Delta W_f}{(TSFC)D} \tag{11}$$

where ΔW_f is the change in weight due to fuel burn. Brandt [9] introduces an average value method for this prediction as

$$E = \frac{\Delta W_f}{(TSFC)D_{avg}} \tag{12}$$

and states that the results for turbojet aircraft are often accurate, but the more accurate prediction of endurance is

$$E = -\int_{W_i}^{W_f} \frac{dW}{(TSFC)D}. \tag{13}$$

Assuming a constant angle of attack, similar to Eq. 3, hence a constant C_L and L/D and noting that lift equals weight,

$$E = \int_{W_f}^{W_i} \frac{1}{TSFC}\frac{L}{D}\frac{dW}{W} = \frac{1}{TSFC}\frac{C_L}{C_D}\int_{W_f}^{W_i}\frac{dW}{W}$$

$$E = \frac{1}{TSFC}\left(\frac{L}{D}\right)\ln\left(\frac{W_i}{W_f}\right). \tag{14}$$

Eq. 14 shows the final equation and for the remainder of the paper endurance, E, will be equivalent to loiter time.

The endurance equation's lift and drag elements can be replaced by an angle of attack for best endurance [4] shown in equation 15.

$$\frac{L}{D} = \frac{C_{L_{ME}}}{C_{D_{ME}}} = \sqrt{\frac{1}{4KC_{D_0}}} \tag{15}$$

where $C_{L_{ME}}/C_{D_{ME}}$ is the coefficient of lift over coefficient of drag for maximum endurance. This is important for an aircraft to maximize its potential when performing a maneuver. However, the assumption that this is constant requires that the aircraft fly slower as weight decreases. The combination of the average value method and the assumptions for Eq. 14 is the approximation method utilized for testing the methodology.

The simplification of these equations involves introducing an unknown amount of error into a continuous problem. There are analytical solutions to reduce this error shown by Vitte [4] and Raymer [5]. These give insight into the differences between using the endurance and range equation for a discrete set of weights versus a continuous set of weights. Raymer [5] uses the non-linear Bréguet range equation to approximate the maximum loiter time of an aircraft. Employing approximate conditions of cruise (L/D) versus loiter (L/D), Raymer [5] finds that

$$\frac{R_{cruise}}{V_{cruise}} = \frac{TSFC_{loiter}}{TSFC_{cruise}} \frac{(L/D)_{cruise}}{E_{loiter}} \tag{16}$$

where E_{loiter} is the loiter time. Finding the approximate relationship between the (L/D)s of both loiter and cruise made Raymer's method approximately 5% optimistic when compared to tested jets and shows that the results are accurate when equating loiter time to endurance.

2.5 Multiobjective Optimization

Pareto introduced the concept of dominated solutions in the field of economics in 1906 to describe a solution that best serves all parties in a multiplayer game [10]. A Pareto efficiency is defined as a state where a reallocation of products to any party where this party is better off to the detriment of at least one other party.

A Pareto frontier is the set of these pareto efficiencies. Economics and engineering have coordinated this concept into design and performance optimization [11]. A Pareto frontier in an aircraft's performance space can be readily visualized when there are few objectives. In this case, two objectives gives a reasonable expectation for a straightforward visualization. Agrawal et al. [12] use an initial mapping from a design space to a performance space in multiple dimensions and proves that the visualization using performance is a better method for an n-dimensional design space. However, the performance space of the Pareto frontier proposed in this paper is only two-dimensional and does not require the complicated mapping of a design space to a performance space from multiple dimensions.

Korhonen et al. [13] point out that there are two main methods when approaching a multiobjective optimization problem. These are the weighting method and the constraint method. In these two methods, the decision maker can control the solution process with their preferences.

Weighting Method.

The weighting method involves solving the following multiobjective optimization problem,

$$\begin{aligned} &\min \sum_{i=1}^{k} w_i f_i(\mathbf{x}) \\ &\text{subject to } \mathbf{x} \in S, \end{aligned} \tag{17}$$

where $w_i \geq 0 \ \ \forall i = 1, \ldots, k$ and each objective is normalized so that magnitudes are not a factor in the solutions. By using multi-objective optimization, the analyst can weight one function as important over another over a tradeoff region and explore the Pareto frontier of solutions. The solution to Eq. 17 is weakly Pareto optimal and not necessarily unique. The solution is Pareto optimal if for all $i = 1, \ldots, k$, $w_i > 0$ such that the solution is unique [13]. The weighting method can be used as a decision tool

to generate different Pareto optimal solutions for the decision maker to choose from. The main requirement for the use of a weighting method is a convex problem. The optimal solutions of some nonconvex problems can sometimes be found no matter the weights, but cannot be proven and does not always behave like the convex solution method [13].

ϵ-Constraint Method.

The second method is the ϵ-constraint method where one of the objective functions is optimized and the rest are used as constraints in the optimization problem. The form of this problem can be seen in Eq. 18.

$$\begin{aligned} &\text{minimize } f_i(\mathbf{x}) \\ &\text{subject to } f_j(\mathbf{x}) \leq \epsilon_j, \quad \forall j = 1, \ldots, k,\ j \neq l, \\ &\qquad \mathbf{x} \in S. \end{aligned} \tag{18}$$

The ϵ-constraint method is preferable for this research problem since only two objectives are examined. Additionally, since they share similar parameters, the constraint of one objective will subsequently optimize the second objective. The weighting method is unnecessarily complicated for the research method given its dimension, but is applicable to the optimization problem in Section 5.6 for two objectives that are not dependent on similar parameters.

2.6 Conclusion

The Bréguet equations are a reliable way to define the range and endurance of an aircraft, given specific parameters. Additionally, the similarities between the range and endurance equation are used to define a tradeoff region for multiobjective optimization of range and loiter time. The next chapter will define a methodology of how

these equations are formulated to allow for a tradeoff region and how each parameter is extracted from aircraft profiles.

III. Methodology

3.1 Introduction

This chapter outlines the use of Bréguet's range equations in two methods for finding the Pareto frontier of optimal range and loiter time tradeoffs for an aircraft, given specific performance characteristics. Additionally, a method for determining a Pareto frontier is established for the range of an aircraft's flight and its respective loiter time and is referred to as an aircraft's tradeoff frontier. Lastly, the application of this approach is explained in detail.

3.2 Current Methodology

The current methodology is derived from the service rules determined by the National Air Intelligence Agency. The defining rules that govern the flight path for combat air patrol control the inputs and outputs of the flight plan generated by NASIC. The steps include warmup (1), climb to cruise altitude (2), cruise and drop external fuel tanks (3), descend to loiter altitude and loiter at the profile for maximum endurance (4), descend to 10,000 feet (5), accomplish two minutes of combat (6), drop external armaments (7), climb to cruise altitude (8), cruise back to initial point (9), land with reserve fuel generally at 20 minutes plus 5% initial fuel load (10) and are shown in Figure 2.

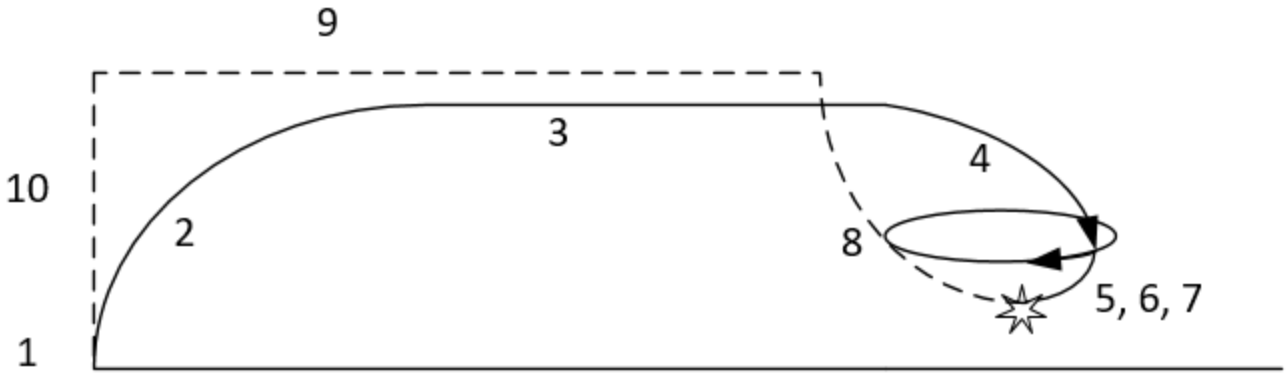


Figure 2. Visualization of Combat Air Patrol Service Rules

The options for a flight developed by NASIC and defined by combat air patrol service rules require either a distance to a interest location or a distance to the interest location and a wanted loiter time at the location. NASIC then develops the tradeoffs for loiter altitude, loiter time, or other flight options. The change in loiter altitude can affect the ability of an aircraft to loiter at a given range. NASIC calculates an approximate tradeoff profile of loiter time given range between altitudes by linearly interpolating the loiter time between altitudes of 5,000 feet. The method is tedious and labor and time intensive for different flight profiles and calculations for loiter time between altitudes.

3.3 Deriving Bréguet Equations

Two main equations are derived from the general range equation seen in Eq. 1, Eq. 21 and Eq. 23. Eq. 21 is used when assuming a linear fuel loss for an aircraft and Eq. 23 is used when assuming a logarithmic fuel loss for an aircraft. The first logically flows from the need to maximize range while assuming a constant thrust specific fuel consumption (constant altitude and throttle setting). Maximizing V/T_A will maximize range and $T \approx T_R = D$ where D is drag. Thus, to maximize range, the

aircraft must be flown at $(D/V)_{min}$. The following equivalent equations follow from the assumption that the aircraft will fly at straight, level, unaccelerated flight.

$$T_A = D = C_D q_\infty S \frac{W}{L} = \frac{C_D q_\infty S(W)}{C_L q_\infty S} = \frac{C_D}{C_L} W \tag{19}$$

$$L = W = C_L q_\infty S = C_L \frac{1}{2} \rho_\infty V_\infty^2 S \Rightarrow V_\infty = \sqrt{\frac{2W}{\rho_\infty C_L S}}. \tag{20}$$

These equations are used in the general range equation shown in Eq. 21.

$$\begin{aligned} R &= \int_{W_f}^{W_i} \sqrt{\frac{2W}{\rho_\infty C_L S}} \left(\frac{1}{TSFC}\right) \left(\frac{C_L}{C_D}\right) \left(\frac{1}{W}\right) dW \\ R &= \int_{W_f}^{W_i} \sqrt{\frac{2W}{\rho_\infty S}} \left(\frac{1}{TSFC}\right) \left(\frac{C_L^{1/2}}{C_D}\right) \left(\frac{1}{W}\right) dW \end{aligned} \tag{21}$$

Further simplifying assumptions are used in general aircraft performance in [14] to derive the Breǵuet range equation in Eq. 22. These include a constant altitude which implies a constant density, ρ_∞, a constant angle of attack which makes C_L and C_D constant, and similar to the general equation, TSFC is constant.

$$\begin{aligned} R &= \sqrt{\frac{2}{\rho_\infty S}} \left(\frac{1}{TSFC}\right) \left(\frac{C_L^{1/2}}{C_D}\right) \int_{W_f}^{W_i} \frac{dW}{\sqrt{W}} \\ R &= \sqrt{\frac{2}{\rho_\infty S}} \left(\frac{2}{TSFC}\right) \left(\frac{C_L^{1/2}}{C_D}\right) (\sqrt{W_i} - \sqrt{W_f}). \end{aligned} \tag{22}$$

In addition to the linear range equation, another simplification of the general range equation involves assuming constant cruise velocity, a constant angle of attack, and a constant thrust specific fuel consumption. This equation is referred to as the constant speed cruise range equation [14]. Using the general range equation in Eq. 21 and substituting V in for the equivalent formula presented in Eq. 20, the following

equation is derived.

$$R = \frac{V}{TSFC}\left(\frac{C_L}{C_D}\right)\ln\frac{W_i}{W_f}. \tag{23}$$

This equation is maximized when $V\dfrac{C_L}{C_D}$ is maximized or equivalently when $\dfrac{C_L^{1/2}}{C_D}$ is maximized.

3.4 Range Determination

Parameters for the flight plan can either be user-defined or be defined to maximize performance for a best case scenario. In the interest of simplicity, the T-37 is used as a general example for how parameters are extracted for maximizing range performance and the F-15C will be used to identify parameters from the existing NASIC model.

The theoretical drag polar for an aircraft is shown in Eq. 24.

$$C_D = C_{D_0} + kC_L^2 \tag{24}$$

where C_{D_0} is the parasitic drag coefficient and $k = 1/\pi A_R e$ where A_R is the aspect ratio of the aircraft and e is the Oswald's efficiency factor. Yechout [14] introduces the T-37's drag polar equation for an example that maximizes the C_L/C_D and $C_L^{1/2}/C_D$. The drag polar equation for the T-37 is

$$C_D = 0.02 + 0.057C_L^2. \tag{25}$$

To maximize C_L/C_D and $C_L^{1/2}/C_D$, Eq. 5 is used to solve for C_L where

$$C_{L_{MR}} = \sqrt{\frac{0.02}{3(0.057)}} = 0.342. \tag{26}$$

Using this value for C_L, C_D can be calculated using the T-37 drag polar.

$$C_D = 0.02 + 0.057(0.342)^2 = 0.0267. \tag{27}$$

As a result, flying at these values for maximizing C_L and C_D will maximize range and endurance [9]. These values allow for the use of a continuous range equation introduced in Eqs. 22 and 23.

Linear Range Determination.

Beginning with the formulation for the linear range equation, the intercept for maximum range and the intercept for starting fuel must be determined. Since the flight pattern includes an initial warm-up, take-off, and climb, the developed method assumes that the fuel burn and distance from initial starting point are given, although these can also be calculated using optimal climbing conditions discussed by Yechout [14]. To identify the point where the aircraft starts its ingress to a point of interest, this model requires the vertical value to be in terms of fuel reserve. Thus, to determine the slope of this line, the fuel burn must be converted to the fuel reserve of the aircraft. For example, assume the T-37 flies 1.5 miles on its climb to cruise and burns 370 lbs of fuel during take-off procedures and climb to cruise. Given that the take-off weight and dry (no fuel) weight of the aircraft is 6,598 lbs and 3,869 lbs respectively, the fuel reserve percentage is at 0.864. The calculations are shown more succinctly below where f_{TO} is take-off fuel weight and f_{TO} is fuel burned on take-off.

$$\frac{f_{TO} - f_{TO}}{f_{TO}} = \frac{2729 - 300}{2729} = 0.890 \tag{28}$$

The point (1.5,0.890)=(miles,fuel fraction) is the first point to determine the equation for the ingress line. The next point needed is determined by Eq. 22. Given that the

thrust specific fuel consumption (TSFC) for the T-37 at 30,000 ft is 0.000232 lbs/s, the ρ_∞ is 0.001267 slug/ft^3 using the US Standard Atmosphere, and the wing area of the T-37 is 184 ft^2, the solution for maximum range is found where W_D is the dry weight of the aircraft and W_C is the weight after climb to cruise.

$$\begin{aligned} R &= \sqrt{\frac{2}{\rho_\infty S}}\left(\frac{2}{TSFC}\right)\left(\frac{C_L^{1/2}}{C_D}\right)(\sqrt{W_C}-\sqrt{W_D}) \\ &= \sqrt{\frac{2}{0.001267(184)}}\left(\frac{2}{0.000232}\right)(21.9)\,(\sqrt{6598-300}-\sqrt{3869}) \\ &= 1391.2 \text{ miles.} \end{aligned} \tag{29}$$

Thus, the second point for the equation of the ingress line is (1797.0,0).

The egress equation is found in a similar fashion using the slope for the ingress line and using the user-defined fuel reserve required as the point for use in the point-slope formula. For example purposes, the fuel reserve required for the T-37 is chosen to be 20% in cohesion with the air service combat rules, making the point for the egress equation (0,0.2).

To determine the slope for the ingress equation, the point-slope formula is used, making the equations of the lines where f_e is the fuel remaining in egress, f_{in} is the fuel remaining in ingress, and x is miles traveled:

$$\begin{aligned} f_{in} &= 0.891 - 0.000728x \\ f_e &= 0.20 + 0.000728x. \end{aligned} \tag{30}$$

The intercept of these lines will determine the maximum range that an aircraft can ingress to a point of interest and return to base with the required fuel reserve. The vertical distance between these lines represents the fuel available for loiter and/or combat time. Figure 3 shows the results of the previous calculations and plots the

resulting ingress equation, egress equation, and fuel available for loiter and combat at 300 miles. The loiter and combat altitudes are variable and depend on the flight plan, but must be specified.

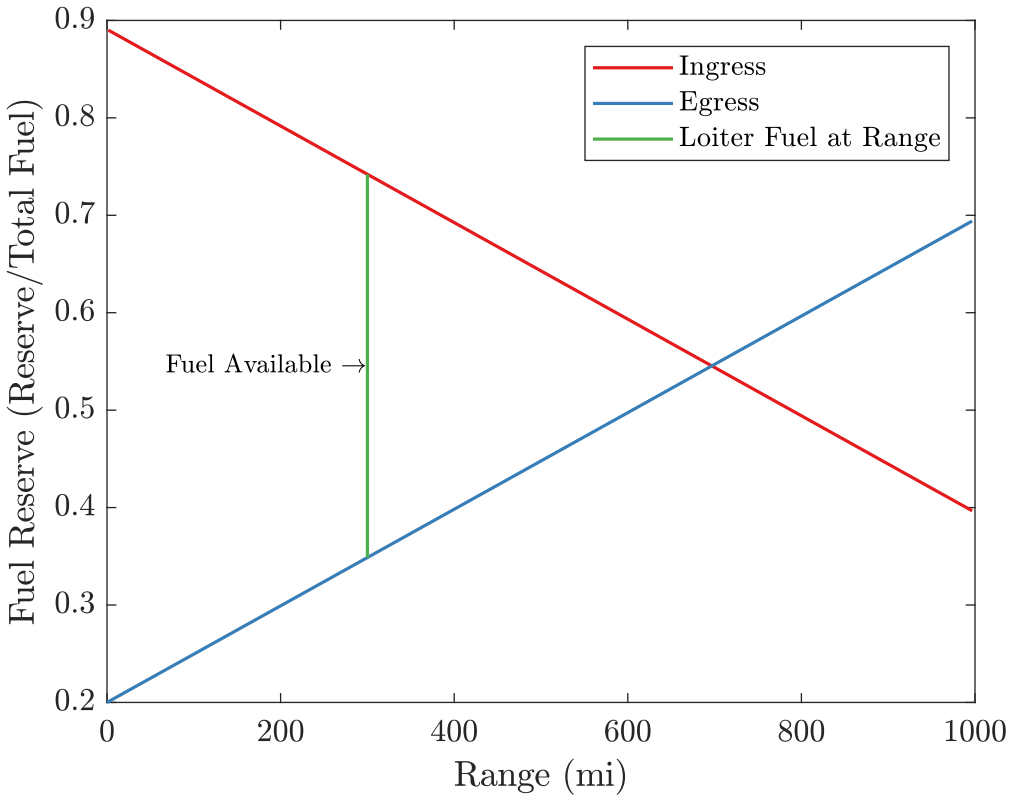


Figure 3. Example Flight Plan T-37

Non-linear Range Determination.

The non-linear equation for range is the equivalent formula presented in Eq. 23. The only additional parameter needed to solve for maximum range in this equation is the velocity which is assumed constant at the C_L/C_D. This is equivalent to finding the velocity at the maximum $\frac{C_L^{1/2}}{C_D}$. In the example case, since the drag polars at each mach number are unknown, a velocity of 0.5 mach at sea level is used. Solving Eq. 23 for fuel reserve where W_F is the full weight and W_D is the dry weight of the

aircraft, the ingress equation becomes

$$f_{in} = \frac{\exp\left[\left(\frac{-R}{V}\right)\frac{(TSFC)}{C_L/C_D}\right]W_F - W_D}{W_F - W_D}. \tag{31}$$

The equation for egress is similarly derived as

$$f_e = \frac{\exp\left[\left(\frac{R}{V}\right)\frac{(TSFC)}{C_L/C_D}\right]W_F - W_D}{W_F - W_D}. \tag{32}$$

For the equations to accurately represent fuel reserve, the given intercept must be determined for fuel burned during climb to cruise and take-off as well as the intercept for fuel reserve required.

The computations for the egress intercept requires fewer computations than the ingress intercept. To compute the egress intercept, the fuel reserve percentage must be converted to fuel using the known weights of the aircraft. The parameter p_f is the fuel reserve percentage and f_p is the resulting fuel..

$$f_p = (W_F - W_D)p_f \tag{33}$$

Then the minutes of reserve is converted to an aircraft weight by rearranging Eq. 15. This becomes

$$f_t = \exp\left[\frac{t * (TSFC)}{(L/D)_{max}}\right]W_F - W_D \tag{34}$$

where t is the endurance time in reserve for the aircraft and f_t is the total fuel needed to meet this requirement. This must in turn be converted into a ratio of fuel to be reserved, p_e.

$$p_e = \frac{f_t + f_p}{W_F - W_D}. \tag{35}$$

The translation of the egress equation is then calculated by solving for the vertical intercept in Eq. 32 where range is zero.

$$\begin{aligned} t_e &= \textit{egress translation} \\ &= -\left(\frac{C_L}{C_D}\right) V \ln\left[\frac{W_D + W_F p_e - W_D p_e}{W_F}\right] \frac{1}{(TSFC)} \end{aligned} \tag{36}$$

The new equation for egress with the fuel reserve calculation becomes

$$r_e = \textit{fuel \% remaining} = \frac{\exp\left[\left(\frac{R - t_e}{V}\right)\frac{(TSFC)}{C_L/C_D}\right] W_F - W_D}{W_F - W_D} \tag{37}$$

with R as the variable for range. After calculating the percentage fuel remaining after climb to cruise, a similar approach to solving for the egress translation is used as in the ingress intercept. The difference is solving for the translation using the known point of distance and fuel to cruise (referenced as the parameter p_{in}). Shown below, the translation for ingress is solved by rearranging Eq. 23.

$$\begin{aligned} t_{in} &= \textit{ingress translation} \\ &= -\ln\left[\frac{p_{in}(W_F - W_D) + W_D}{W_F}\right] \frac{V}{TSFC}\left(\frac{C_L}{C_D}\right) - R_{cruise} \end{aligned} \tag{38}$$

where R_{cruise} is equal to the ground distance that the aircraft travels when climbing to cruise. The resulting equation for ingress with the translation accounting for take-off and climb to cruise is

$$r_{in} = \textit{fuel \% remaining} = \frac{\exp\left[\left(\frac{-(R + t_{in})}{V}\right)\frac{(TSFC)}{C_L/C_D}\right] W_F - W_D}{W_F - W_D}. \tag{39}$$

The flight map run on the T-37 in Figure 4 shows a similar result as the linear range equation, but the flight is further defined by the use of a specific velocity, V, or

C_L/C_D. This is useful when defining flight plans in a combat area where velocity or angle of attack is not necessarily intended for maximum range.

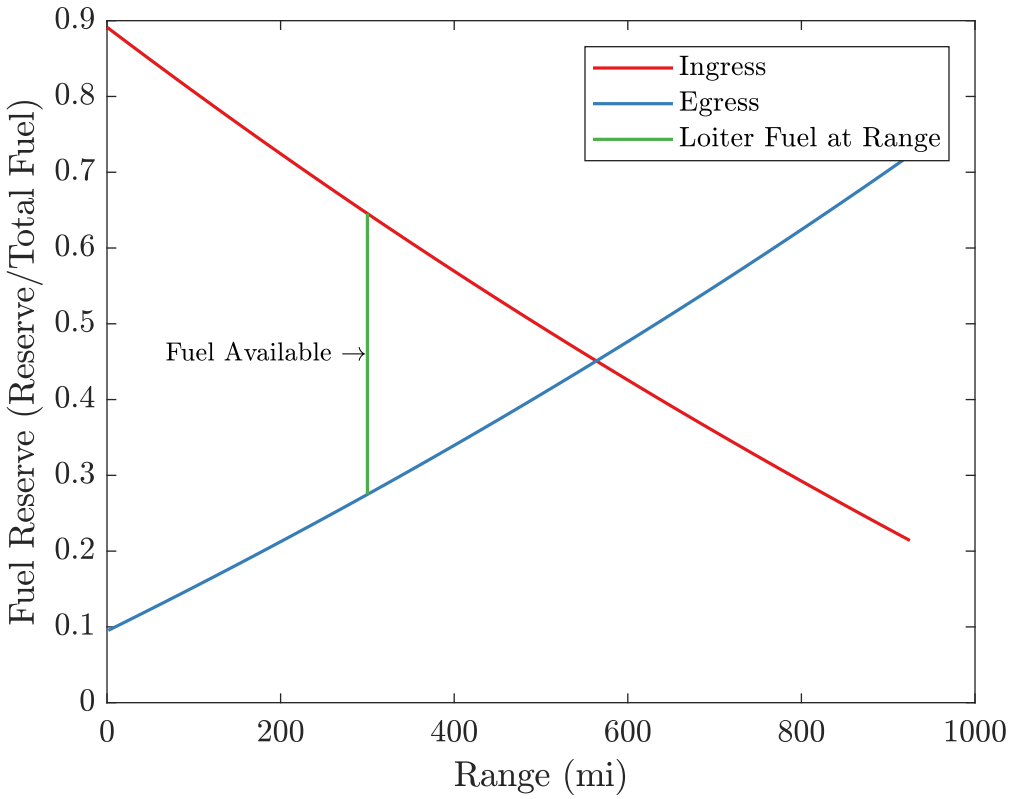


Figure 4. Example Flight Plan for Nonlinear Bréguet Range Equation

Parameters for the F-15C's C_L and C_D performance are referenced in Figure 5. Each line in the graph represents a mach number which is formed by the value for each coefficient of lift (C_L) and the associated coefficient of drag (C_D). These are found from historical flying data for each specific aircraft. From the matrix of these values, the lift over drag and $C_L^{1/2}/C_D$ maximums can be found. Where $C_L^{1/2}/C_D$ is maximized, the respective mach number and C_L/C_D can be pulled into the model as a parameter.

An example of F-15 drag polar outputs from the existing model are shown in Figure 5. Using the technique discussed above, the maximum values for C_L/C_D and

$C_L^{1/2}/C_D$ are found from the matrix. An example of the matrix is shown in Section A.

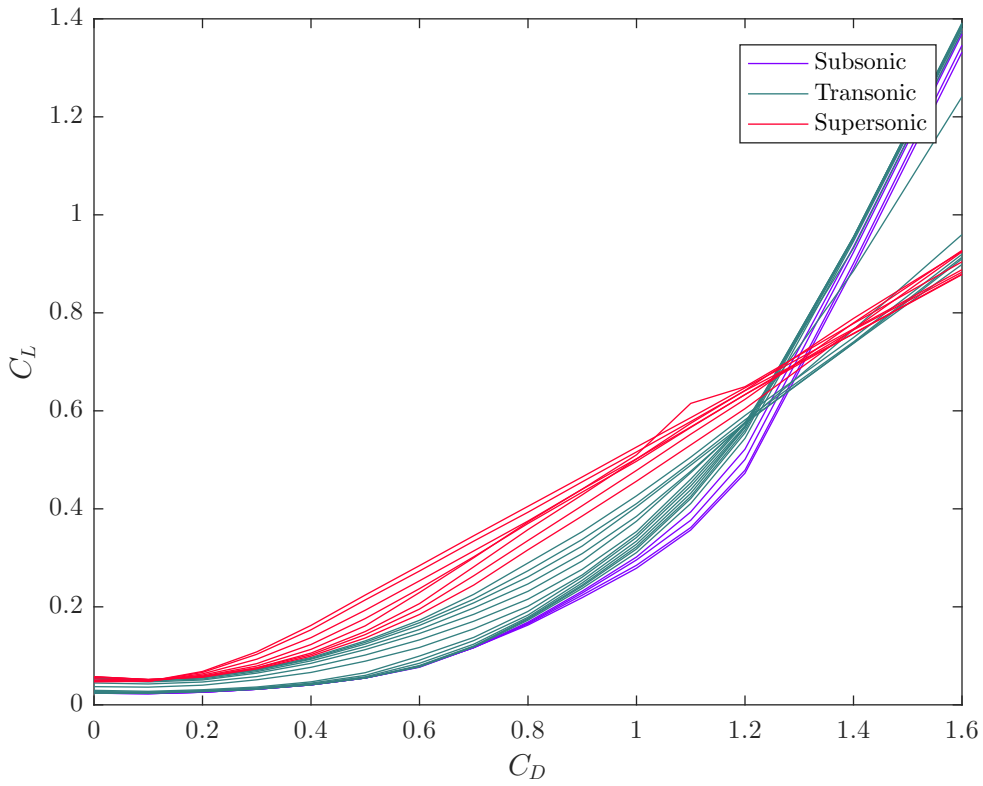


Figure 5. Coefficients of Lift versus Drag for the F-15C.

3.5 Loiter Time Determination

Loiter time is determined based on Eq. 14. Solving for the weight at the given range along the ingress and egress equations, the available loiter time can be determined over the target area. The endurance, $E_{available}$, is calculated using the following incorporation of the ingress and egress equation.

$$E_{available} = \frac{1}{TSFC}\left(\frac{C_L}{C_D}\right)\ln\left(\frac{(r_{in}+W_D)/W_F}{(r_e+W_D)/W_F}\right). \tag{40}$$

This is used as a starting point for available minutes given that the aircraft has

gone a certain range. The next step incorporates any combat time or climb from loiter or combat altitude.

3.6 Model Usage for Correction Factors

The developed model adjusts for several parameters that are provided for comparing the NASIC model to the proposed model. These include given coefficients of lift, coefficients of drag, thrust specific fuel consumption at altitude, payload weights (W_P), external fuel tank weights (W_{FT}), total fuel weight (W_{Fuel}), internal fuel weight (W_{int}) take-off weight (W_F), combat fuel burn (f_{com}), and climb back to altitude after combat and/or loiter (f_{climb}). Dry weight, a corrected egress and ingress equation, and thrust specific fuel consumption at sea level are determined from there. Additionally, an adjustment for including combat and the secondary climb to altitude must be determined for a true comparison.

Dry weight is found by using the take-off weight as a total weight and subtracting the total fuel weight such that $W_D = W_F - W_{Fuel}$. This is useful as the corrected egress equation uses this as the final weight for range. The egress equation must account for an external fuel tank and payload drop prior to combat.

The corrected egress equation uses an additional derived parameter for full weight given that there are no external tanks or additional payload. In this way, the equations are corrected so that the ingress equation uses an aircraft with the additional payload and external fuel tanks with the total fuel weight as its take-off weight and the dry weight as the final empty weight such that $W_{D_{in}} = W_{F_{in}} - W_{Fuel}$. The egress equation uses the full weight of the aircraft given that there are no external fuel tanks, additional payloads, or any external fuel such that $W_{D_e} = W_F - W_{FT} - W_P - W_{Fuel}$ and $W_{F_e} = W_{D_e} + W_{int}$.

The two equations must also correct for the given flight plan, which includes a

new L/D. L/D is given at several weight steps for each aircraft by the NASIC output and the L/D for the proposed model averages the weight steps that include any weight that the aircraft would be at during flight. For example, if the full weight of the aircraft is $40,000$ the L/Ds at each weight step of $40,000$ or below would be averaged. The same is done for mach number and $TSFC$, but $TSFC$ must be adjusted for altitude.

Since $TSFC$ is brought into the model from an output at altitude, it must be adjusted to sea level in order for the proposed model to remain flexible at various altitudes. To do this, the temperature at altitude (T_{alt}) and the temperature at sea level (T) are needed for Eq. 41.

$$TSFC = TSFC_{sl}\sqrt{T_{alt}/T} \tag{41}$$

where temperature at altitude is found using the US Standard Atmosphere. Solving for $TSFC$ at sea level allows for conversions at each point using Eq. 41 along the flight path when the altitude is changed for loiter or combat altitude. The final value for $TSFC$ is divided by 3600 to put the value in $TSFC/s$ instead of $TSFC/hr$. Similarly, mach number, M, is converted to ft/s using Eq. 42 given T_{alt} and the gas constant (R), assuming the specific heat ratio of air for standard day conditions is 1.4.

$$V = 3.2808M\sqrt{1.4T_{alt}R} \tag{42}$$

where 3.2808 is the number of feet in a meter. This also allows the model to remain agnostic to altitude given a mach number.

Converting the secondary climb to cruise altitude fuel burn and the combat fuel burn to a correction on the total available endurance is done by converting the fuel burn to minutes lost. Using the dry weight of the egress equation, minutes lost for

secondary climb to cruise and combat is found using Eq. 43 where $TSFC_{loiter}$ is the corrected $TSFC$ for loiter altitude.

$$\begin{aligned} L_{com} &= \frac{1}{TSFC_{loiter}} \left(\frac{C_L}{C_D}\right) \ln\left(\frac{W_{D_e} + f_{com}}{W_{D_e}}\right) \\ L_{climb} &= \frac{1}{TSFC_{loiter}} \left(\frac{C_L}{C_D}\right) \ln\left(\frac{W_{D_e} + f_{climb}}{W_{D_e}}\right). \end{aligned} \tag{43}$$

The final endurance value is then computed as $E = E_{avail} - L_{com} - L_{climb}$, where the full equation for determining E_{avail} is shown in Eq. 44. The algorithm has all the required parts for determining the Pareto front of endurance and range of the aircraft and a summary is shown in Algorithm 1.

$$\begin{aligned} E_{avail} \quad = \quad & \frac{(C_L/C_D)}{TSFC_{loiter}} \left(TSFC_{alt} \frac{t_e - x}{V(C_L/C_D)} - \ln\left(W_{D_{in}}\right)\right) + \\ & \frac{(C_L/C_D)}{TSFC_{loiter}} \ln\left(W_{D_e} - W_{D_{in}} + W_{F_{in}} \exp\left[-TSFC_{alt} \frac{t_{in} + x}{V(C_L/C_D)}\right]\right) \end{aligned} \tag{44}$$

Algorithm 1 Range and Endurance Algorithm

Require: ac = Structure of Aircraft Parameters (ac.Parameter)

$TSFC_{alt} = ac.TSFC\sqrt{T_{alt}/T}/3600$

$TSFC_{loiter} = ac.TSFC\sqrt{T_{loiter}/T}/3600$

$V = 3.2808 * ac.M\sqrt{1.4T_{alt}R}$

$p_f = (ac.W_{F_e} - ac.W_{D_e})ac.f_r$

$$f_t = \exp\left[\frac{ac.t * (TSFC_{alt})}{ac.(L/D)}\right] ac.W_{D_e} - ac.W_{D_e}$$

$$p_e = \frac{f_t + p_f}{ac.W_{F_e} - ac.W_{D_e}}$$

$$t_e = -ac.(L/D)V \ln\left[\frac{ac.W_{D_e} + (ac.W_{F_e} - ac.W_{D_e})p_e}{ac.W_{F_e}}\right]$$

$$t_{in} = -\ln\left[\frac{p_{in}(ac.W_{F_{in}} - ac.W_{D_{in}}) + ac.W_{D_{in}}}{ac.W_{F_{in}}}\right]\frac{V}{TSFC_{alt}}ac.(L/D) - ac.R_{cruise}$$

$$L_{com} = \frac{1}{TSFC_{loiter}}ac.(L/D)\ln\left(\frac{ac.W_{D_e} + ac.f_{com}}{ac.W_{D_e}}\right)$$

$$L_{climb} = \frac{1}{TSFC_{loiter}}ac.(L/D)\ln\left(\frac{ac.W_{D_e} + ac.f_{climb}}{ac.W_{D_e}}\right)$$

for x in k:n **do** ▷ k,n is user-defined for distance between Pareto pts

$$r_e = \frac{\exp\left[\left(\frac{x - t_e}{V}\right)\frac{TSFC_{alt}}{ac.(L/D)}\right] ac.W_{D_e} - ac.W_{D_e}}{ac.W_{F_e} - W_{D_e}}$$

$$r_{in} = \frac{\exp\left[\left(\frac{-(x + t_{in})}{V}\right)\frac{(TSFC_{alt})}{ac.(L/D)}\right] ac.W_{F_{in}} - ac.W_{D_{in}}}{ac.W_{F_{in}} - ac.W_{D_{in}}}.$$

$$E = \frac{1}{TSFC_{loiter}}ac.(L/D)\ln\left(\frac{(r_{in} + ac.W_{D_{in}})/ac.W_{F_{in}}}{(r_e + ac.W_{D_e})/ac.W_{F_e}}\right) - L_{climb} - L_{com}$$

$E_{array} = E_{array}.append(E)$

$R_{array} = R_{array}.append(x)$

end for

return $T = [R_{array}, E_{array}]$ ▷ Return matrix for $T(row, :) = (range, endurance)$

3.7 Haversine Formula for Interest Locations

The Haversine formula calculates great-circle distance between two points on a sphere by latitude and longitude coordinates. The haversine formula of an angle is given as $\text{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right)$. The haversine formula is used to find distance, d, given

the latitude and longitude of two points shown in Eq. 45.

$$
\begin{aligned}
a &= \sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right) \\
d &= r * \text{atan2}(\sqrt{1-a}, \sqrt{a})
\end{aligned}
\tag{45}
$$

where r is the radius of the sphere, ϕ_1, ϕ_2 are the latitudes of points 1 and 2, and λ_1, λ_2 are the longitudes of points 1 and 2. Given that the intersect can be calculated where endurance is equal to zero, the maximum range of the aircraft can be determined and then mapped. Figure 6 shows the maximum range of the F-15C from Salina, KS, in any direction. The perimeter of the range circle was found by solving for the longitude and latitude given the distance and the angle from the original location.

Using Eq. 45, the distance from the initial point and various interest locations are found given their latitude and longitude. The distance is then evaluated as a range and the loiter time at that location is found using the equations from the previous section. The result of finding the maximum endurance of an F-15C at the three locations of Dallas, TX, Oklahoma City, OK, and Denver, CO is shown in Figure 7. This method assumes that an aircraft will go out to the interest point and return to the initial location.

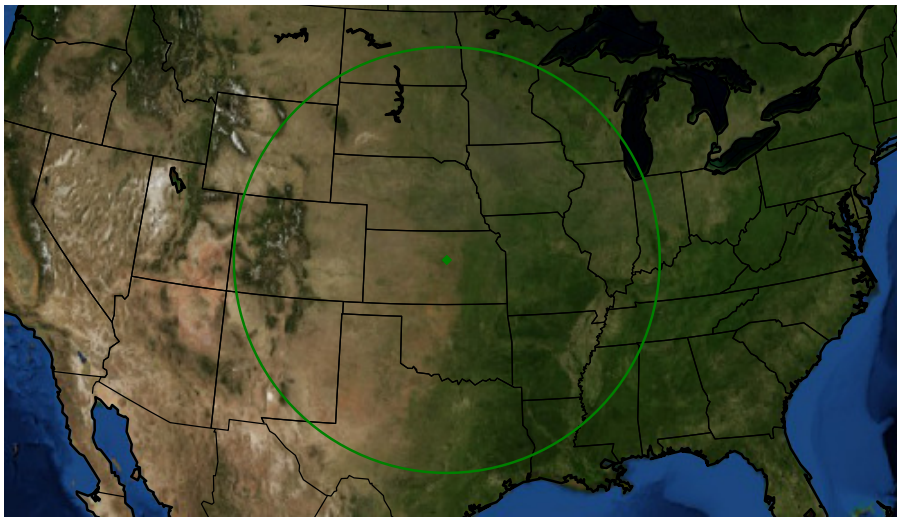


Figure 6. Example Maximum Radius of F-15C mapped

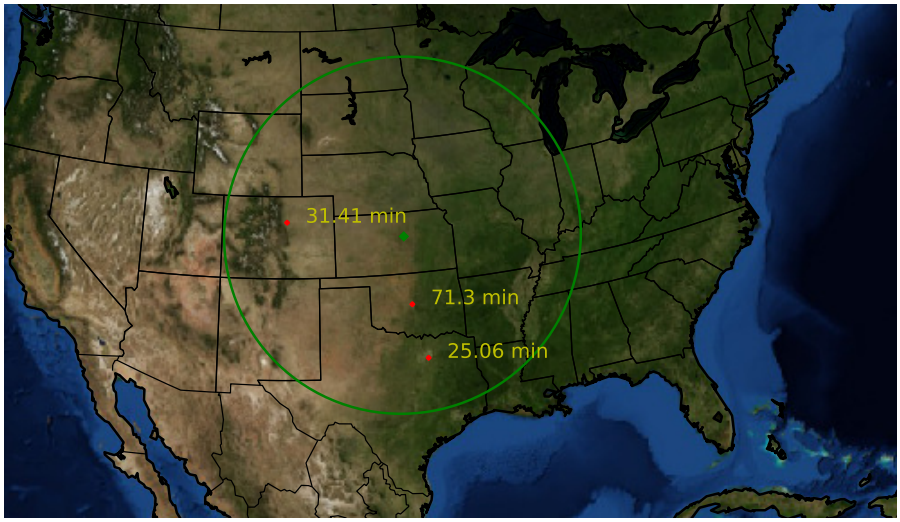


Figure 7. Interest Points around Salina, KS

IV. Analysis

4.1 Introduction

The original model from NASIC calculates a single endurance or range of an aircraft depending on inputs from a user. This method is deemed ±5% accurate based on comparing the output to explicit mission runs from a production program. With the tradeoff frontier of the developed methodology, a series of these outputs are estimated with given flight parameters of the aircraft. To show the differences between the estimated frontier and the original model, several data points were extracted for three different aircraft, with the last aircraft having four differing configurations depending on external fuel and payload. This resulted in six tests for the developed methodology.

4.2 Analysis Aircraft Parameters

From an individual output from the original model, the parameters of the aircraft are derived based on the averages at specific altitudes and weights as discussed in Section 3.6. The following table describes all the parameters for each of the six aircraft used.

Table 1. Aircraft Parameters

	F-15C	F-16C	AC1-000	AC2-000	AC3-000	AC4-000
TSFC	1.159	1.206	1.303	1.232	1.209	1.180
C_L/C_D	8.520	9.880	9.447	7.930	7.655	7.860
Mach	0.732	0.791	0.865	0.830	0.808	0.759
Fuel To Cruise	42738	26474	24006	25471	27780	33809
Distance To Cruise	31.28	24.84	28.635	37.605	44.045	61.64
Combat Fuel	688	402	402	402	402	402
Climb Fuel	523	177	208	204	215	247
Percent Reserve	5	5	5	5	5	5
Minutes Reserve	20	20	20	20	20	20
Ingress Dry Weight	31262	20305	18188	19797	20062	19500
Ingress Full Weight	44710	27470	25155	26764	29211	35616
Egress Dry Weight	30645	20107	17879	17879	17890	17748
Egress Full Weight	44093	27272	24846	24846	24857	24715
Cruise Altitude	30000	30000	30000	30000	30000	30000
Loiter Altitude	30000	30000	30000	30000	30000	30000

The cruise and loiter altitude describe the altitude at which these parameters were derived. These are then estimated for different altitudes using US Standard Atmosphere. The ACX-000 aircraft is a dataset for which the aircraft is unknown to test whether the method is agnostic to a particular aircraft.

4.3 Aircraft Tradeoff Comparison

The tradeoff frontier was compared to the NASIC model using point statistics. Between three and eight points were generated from the NASIC model depending on data availability of the aircraft. These points were generated at ten, twenty, and thirty minutes of loiter time for all aircraft. If an aircraft was able to loiter longer, then 40, 60, 80, and 100 minutes of loiter time were also been generated. The statistics were computed as the average of the absolute value of the difference between the tradeoff frontier, E, and the NASIC model, O.

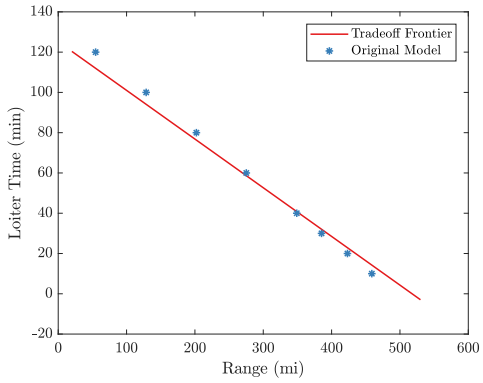
$$D_{avg} = \frac{\sum_{i=1}^{n} |O_i - E_i|}{n} \tag{46}$$

where n is the number of points generated for an aircraft and D_{avg} is reported in minutes. In addition, the maximum difference between an expected value from the tradeoff frontier and the NASIC model were computed for each aircraft at each altitude which is a statistic of the model's variability over different endurances.

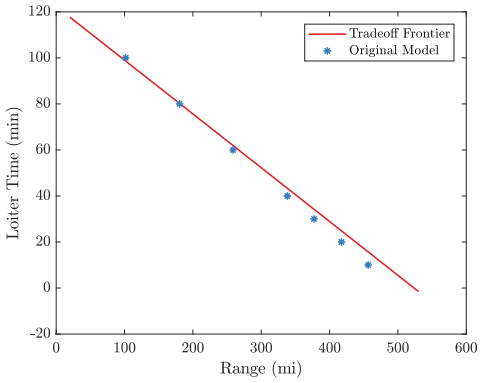
Table 2. Difference Statistics Between NASIC Model and Tradeoff Model

	F-15C	**F-16C**	**AC1-000**	**AC2-000**	**AC3-000**	**AC4-000**
D_{avg} **at 30,000ft**	3.537	5.385	2.614	4.775	2.778	22.742
Max Mins at 30,000ft	5.790	9.817	5.151	4.891	4.154	23.740
D_{avg} **at 20,000ft**	3.083	5.403	2.403	2.628	5.059	23.496
Max Mins at 20,000ft	5.686	9.506	4.174	3.362	5.334	24.055
D_{avg} **at 10,000ft**	4.622	4.838	3.460	3.001	4.566	21.146
Max Mins at 10,000ft	5.872	8.772	5.112	3.568	4.910	22.710

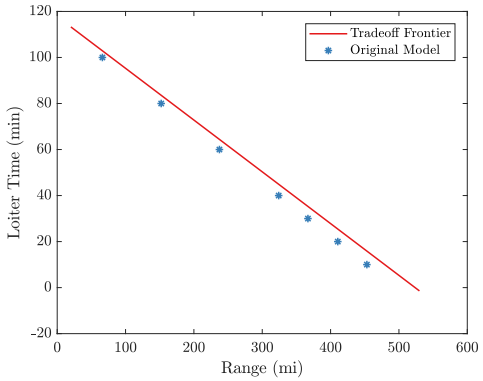
These statistics showed minimal differences between the NASIC model's points and the tradeoff frontier's generated solution in the first five aircraft. There is a larger deviation in the second aircraft, but the trend of the data is the same as the trend-line from the tradeoff frontier. Figure 8 shows the F-15C's tradeoff frontiers mapped with the observed data points from NASIC's model. The mapped tradeoff frontiers for the remaining five aircraft are referenced in Appendix B.



(a) Comparison at 30,000 ft



(b) Comparison at 20,000 ft



(c) Comparison at 10,000 ft

Figure 8. Tradeoff Comparison for F-15C

Overall, the tradeoff comparison relayed results on the same order of magnitude as the observed data points from NASIC's model. The maximum minutes that the model deviated from the observed points for the first two aircraft (for which the most test points were acquired) was approximately ten minutes. This showed that the proposed method was consistent with results from the NASIC model while producing the answers more quickly.

The final comparison of the four different configurations of ACX-000 displayed the main weakness in the analysis. Averaging the coefficient of lift over drag, thrust-specific fuel consumption, and speed of an airframe over the course of an aircraft's weight will effect the results when the difference between an outbound aircraft with

a payload and external fuel tanks and the same inbound aircraft is large. The breakdown occurs in the comparison of AC4-000 where the aircraft is loaded with a large amount of external fuel. As a result, the proposed method is conservative with the loiter time estimated lower than the observed points by about twenty minutes on average. The difference between the observed and expected data from the AC4-000 is large with respect to raw data and the assumption inhibits accurate results when the weight difference is high between an ingress and egress aircraft.

With the tradeoff comparison similar to the results of the NASIC model, a loiter time comparison is determined in seconds with the developed methodology. The tradeoff comparison between loiter time and altitude can also be computed using the adjusted $TSFC_{alt}$ as the changing parameter and setting range to a common mileage. Figure 9 shows an example instance of the tradeoff of loiter time at different altitudes and setting the range to 293 miles.

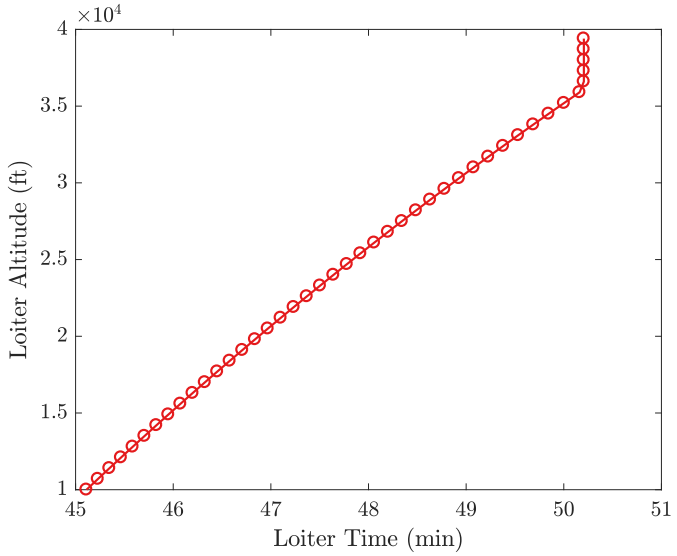


Figure 9. Tradeoff of Loiter Time and Loiter Altitude

Since the model was shown accurate for five of six instances, this estimation can give a user an accurately estimated picture of what a loiter at different altitudes would look like at a given range and given performance characteristics.

V. Assignment Optimization Formulation

5.1 Introduction

The developed methodology accounts for performance characteristics in a concise way that lends itself to optimization formulation without becoming intractable. This chapter discusses the relevance of the aircraft routing problem in literature as well as their shortcomings with respect to the inclusion of aircraft performance. Additionally, the chapter introduces several formulations in which the methodology is used to account for fuel efficiency and loiter availability at interest locations. The first two formulations explore the use of the methodology with multiple aircraft interested in traveling to multiple locations and optimizing total loiter time and priority of locations. The last formulation examines the inclusion of fuel burn on a single aircraft's path to multiple locations while loitering at each location.

5.2 Aircraft Routing Optimization Literature Review

The aircraft assignment problem is an application of the assignment problem to target locations from a starting point. A modified assignment problem can consist of one aircraft and multiple locations, a one-to-one matching of aircraft to target locations, or multiple aircraft to a greater number of locations. An assignment problem is a special case of the transportation problem where supply and demand are equal to one for all locations. The mathematical model for an assignment problem is as

follows:

$$\min \quad \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij} \tag{47}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} x_{ij} = 1, \quad \forall i \in \{1, ..., m\} \tag{48}$$

$$\sum_{i=1} x_{ij} = 1, \quad \forall j \in \{1, ..., m\} \tag{49}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j \in \{1, ..., m\} \tag{50}$$

where c_{ij} is the cost of assigning i to j and m is the number of assignments [15].

A binary integer linear program (BIP) is generally the designation of assignment problems, given that the constraints and objective function are linear and that an assignment decision variable is binary-valued. In their paper on the UAV routing assignment problem, Shetty et al. [16] use a mixed integer linear program wherein the main constraints are service level and weaponry constraints. Their formulation serves as a traveling salesman problem (TSP), where the objective is to minimize cost, but all locations must be visited in the model. There is no upper-level constraint for ability to travel or service for any amount of time. While this formulation showcases a valid assignment problem for UAV waypoints and is similar to the way a formulation for multiple aircraft waypoints are visited, the constraints do not capture the performance characteristics of the aircraft and only seek to minimize cost rather than matching realistic performance capabilities.

In contrast, Alighanbari et al. [6], Schumacher et al. [17], and Taylor et al. [18], use constrained optimization based on range, timing, or fuel ratios. The advantage of this formulation is that it can represent real-world scenarios such as UAV waypoints [6] or flight mapping for commercial airliners [18]. Although Taylor et al. [18] used the TSP formulation in their model, the idea of using fuel as a constraint accounts for

all possible maneuvers in an aircraft. This formulation informs the way performance characteristics of the aircraft is accounted for within constraints.

Vitte [4] discusses the continuous coverage of a target area using a time on station designation and assuming a constant idle time. His formulation allowed for a maximization of time on station for a certain number of aircraft while meeting the constraints for turn-around time and outbound time for any particular aircraft. Integrating this idea while allowing for multiple target locations in an imperfect matching scheme allows for a decision maker to maximize the priority of targets with limited resources. While this is promising with respect to maximizing the priority of targets, the formulation lacks relevance to the performance characteristics of the aircraft used and disregards the fuel efficiency of an aircraft as it burns fuel.

Kannon et. al. [19] examines the incorporation of fuel over a series of time steps in an aircraft routing formulation. At each time step, fuel can change depending on a previous aerial refueling. Using an MILP, the formulation solves for the optimal optimal route through a series of intermediary nodes and the refueling intermediary node between a source and sink. The concept of a time step can follow the use of fuel along a route and involve how fuel efficiency changes over fuel burn. This formulation informs the way fuel is initialized as a decision variable and constrained over the course of the solution process.

5.3 Initial Formulation

Let x_{ij} be a binary variable where $x_{ij} = 1$ if plane i travels to location j. Additionally let d_{ij} be the number of miles from a starting, i point to an interest location,j, and E_{ij} be the maximum loiter time, given that aircraft i has traveled to location j where is calculated using Eq. 44 with d_{ij} used as the range. A negative number in the E_{ij} represents an inability for the aircraft to reach location j from location i and

results in a penalty to the overall objective function. The sets I and J are represented as $I = \{1, 2, \ldots, n\}$ and $J = \{1, 2, \ldots, m\}$ where n is the number of aircraft and m is the number of locations. The formulation for maximizing endurance with n aircraft and m locations follows.

$$\max \sum_{i \in I} \sum_{j \in J} E_{ij} x_{ij} \tag{51}$$

$$\text{s.t} \sum_{i \in I} x_{ij} \leq 1 \qquad \forall j \in J \tag{52}$$

$$\sum_{j \in J} x_{ij} \leq 1 \qquad \forall i \in I \tag{53}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I, j \in J \tag{54}$$

The constraints represent a typical transportation problem where each aircraft can travel to at most one location and each location can be visited once. Using the Haversine formula and the methodology described in Section 3.7, the endurance matrix is determined for an aircraft's performance parameters.

5.4 Priority Assignment Formulation

The priority assignment formulation is similar to the initial formulation and allows prioritizing locations. The decision for an interests point's priority is defined by the vector $\mathbf{v}$. The formulation for maximizing endurance with n aircraft and m locations

follows.

$$\begin{aligned} \max & \sum_{i \in I} \sum_{j \in J} E_{ij} v_j x_{ij} \\ \text{s.t} & \sum_{i \in I} x_{ij} \leq 1, \qquad \forall j \in J, \\ & \sum_{j \in J} x_{ij} \leq 1, \qquad \forall i \in I, \\ & x_{ij} \in \{0, 1\}, \qquad \forall i \in I, j \in J. \end{aligned} \tag{55}$$

Though v_j in this case is represented as a discrete vector of priorities, there can also be a distribution with the endurance associated with an interest location. This broadens the ability of this formulation to utilize the developed methodology for incorporating the dynamic flight characteristics of aircraft into an assignment decision rather than a static constraint used in other literature [6, 17, 18].

5.5 Assignment Problem Tests

The assignment problem was tested using the aircraft parameters from AC4-000. Although the AC4-000 stressed the proposed method and incurred the most error and had the most conservative estimate from the current method, this aircraft had the longest range and, with the given list of interest points, found non-trivial solutions. The combat fuel and climb fuel were set to zero to assume that the aircraft was only interested in loiter at its respective location. The interest points were a random sequence of 100 cities in the US. All tests were run on the same 100 cities.

The sample problem assigned an arbitrary number of twenty aircraft to visit twenty nodes to meet the objective of maximizing total endurance. Figure 10 shows the results of the optimization problem run using the SCIP solver [20].

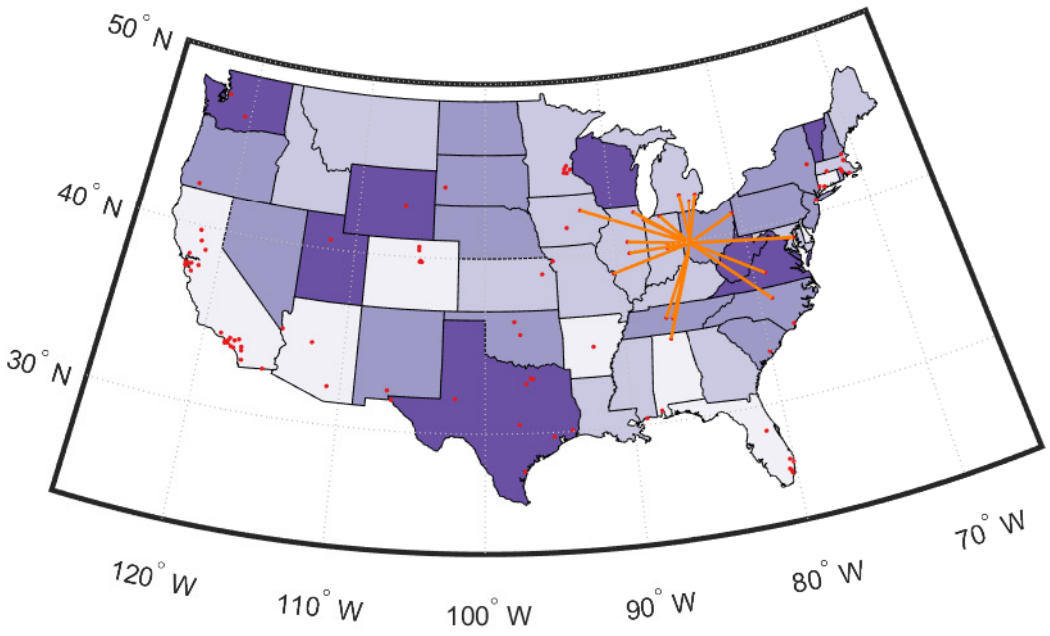


Figure 10. Initial Assignment Problem Example

As hypothesized, the results show a relatively trivial solution since all aircraft are the same and endurance at all locations is weighted equally. The twenty aircraft were assigned to the twenty nearest locations.

The priority assignment problem assigned a random priority to each location between zero and twenty. This test was in an attempt to determine whether a non-trivial solution can be found when certain locations are more desirable to endure over than others. The same aircraft and locations were used in this problem as in the initial assignment problem example. Figure 11 shows the results of the optimization problem run using the SCIP solver [20].

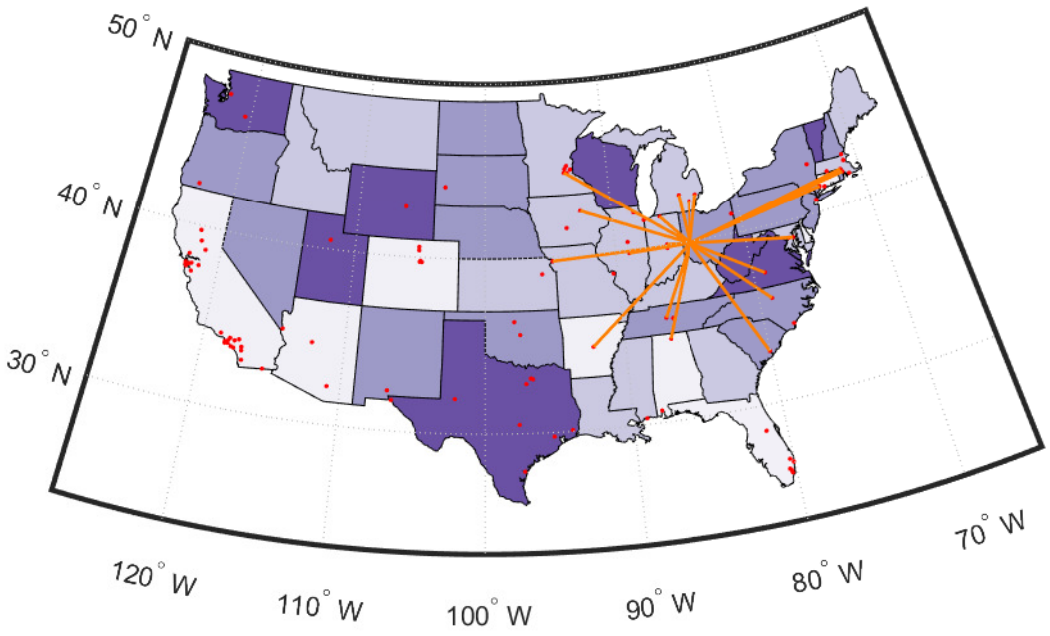


Figure 11. Priority Assignment Problem Example

The optimization problem solution verified that certain locations were a higher priority than others despite the decreased available loiter time. The aircraft traveled to locations past the previous solutions interest locations and chose to loiter at these interest points due to their higher interest.

Using the same optimization formulation as the priority assignment formulation, the last test was using four different aircraft, the F-15C, AC2-000, AC3-000, and AC4-000. Since these three have different endurance capabilities at their given parameters in Table 1, the problem seeks to maximize priority while balancing the dynamic characteristics of the different aircraft. Twenty aircraft were used with five aircraft from each aircraft class. Figure 12 shows the results of the optimization problem run using the SCIP solver [20].

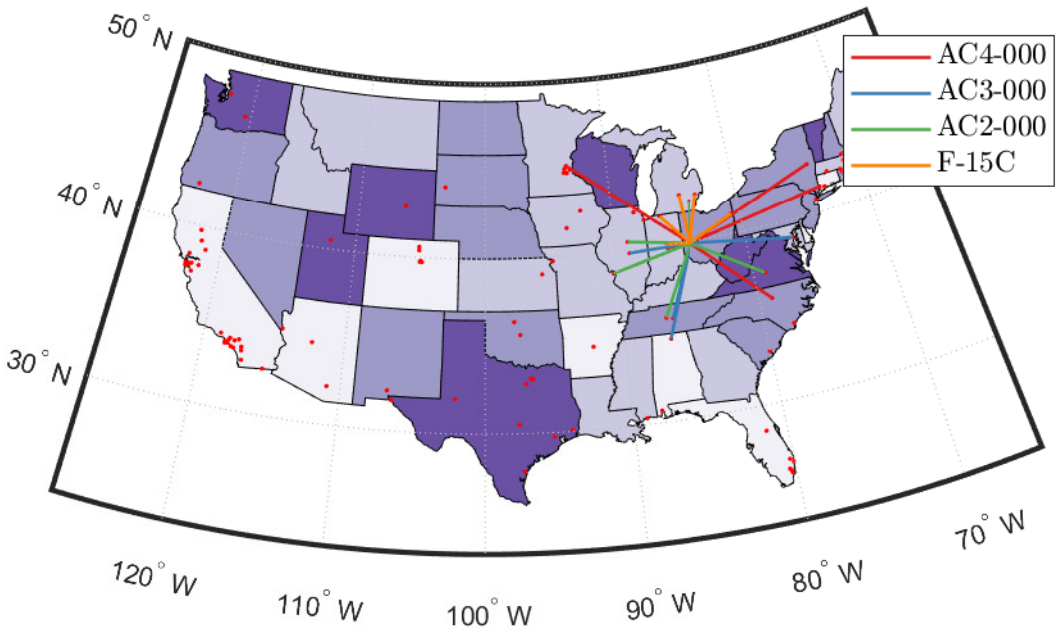


Figure 12. Different Structures Example (with Random Priority)

The solution showed that the aircraft with the longest range were assigned to the interest locations having both higher priorities and farther locations (AC4-000) and the shorter range aircraft were assigned to loiter over the closer distances.

The assignment problem formulations successfully leveraged the methodology to quickly find a maximum loiter time over any interest location using the flight characteristics of each aircraft and in turn solve an assignment problem that maximized the interest location preferences. The solution to the priority assignment formulations were non-trivial and partitioned priority with loiter time and aircraft abilities.

5.6 Decision Dependent Formulation

This sub-problem seeks to route an aircraft with the given flight characteristics to multiple locations while incorporating the fuel efficiency of the nonlinear Bréguet range equation. The aircraft must always start from a given source node and may visit any connected node, given that the move is feasible according to its flight characteristics. The following set of parameters, decision variables, sets, and constraints define the backbone of the mixed integer nonlinear program (MINLP).

Parameters.

- m is the number of locations available to travel
- D is a matrix of the distances between each point $D \in \mathbf{R}^{mxm}$
- E is a matrix of efficiency factors calculated using Bréguet's range equation such that $E_{ij} = \exp\left[\frac{-D_{ij}}{V}\frac{TSFC}{C_L/C_D}\right]$ and $E \in \mathbf{R}^{mxm}$
- s is both the source and sink of the model

Decision Variables.

- x_{ij}^{τ} equals one if the aircraft travels from location i to location j at time decision step τ
- F^{τ} is the amount of fuel used at decision step τ

Sets.

- $T = \{1, 2, ..., m\}$
- $I = \{1, 2, ..., m\}$
- $J = \{1, 2, ..., m\}$

Constraints.

$$\sum_{j\in J} x_{sj}^{\tau} = \begin{cases} 1, & \tau = 1, \\ 0, & \forall \tau \in T - \{1\}, \end{cases} \tag{56}$$

$$\sum_{j\in J} x_{ij}^{\tau+1} - \sum_{j\in J} x_{ji}^{\tau} = 0, \quad \forall i \in N - \{s\}, \tau \in T, \tag{57}$$

$$\sum_{i\in I} x_{ii}^{\tau} = 0, \quad \forall \tau \in T, \tag{58}$$

$$\sum_{i\in I}\sum_{\tau\in T} x_{ij}^{\tau} \leq 1, \quad \forall j \in J, \tag{59}$$

$$\sum_{\tau\in T}\sum_{i\in I} x_{is}^{\tau} = 1, \tag{60}$$

$$W_F - \left(\sum_{i\in I}\sum_{j\in J} E_{ij} W_F x_{ij}^{1}\right) \leq F^{1}, \tag{61}$$

$$W_F \sum_{i\in I}\sum_{j\in J} x_{ij} - \left(\sum_{i\in I}\sum_{j\in J} E_{ij}(W_F - \sum_{k=1}^{\tau} F^{k-1}) x_{ij}^{\tau}\right) \leq F^{\tau}, \quad \forall \tau \in T - \{1\}, \tag{62}$$

$$\sum_{\tau\in T}\sum_{i\in I}\sum_{j\in J} F^{\tau} \leq F_{total}, \tag{63}$$

$$x_{ij}^{\tau} \in \{0, 1\}, \quad \forall i \in I, j \in J, \tau \in T. \tag{64}$$

Constraint (56) accounts for the aircraft originating from the source node at the first decision step. Constraint (57) only allows the aircraft to depart from the node to which it arrived in the last decision step for all nodes except the source node since the return decision step is not known. Constraint (58) does not allow the aircraft to travel to the same node at any decision step while constraint (59) assures that an aircraft never departs the same node more than once in its route. Constraint (60) requires the aircraft to return to the source (once only). Constraints (61) and (62) incorporate the fuel burn at the start of the maneuver and for each decision step

until returning to the source. Decision variable, F^{τ}, is the fuel needed for the range and the additional fuel used for loiter. The variable is constrained by the aircraft's fuel burn from previous steps and needed fuel for the range given its current weight. Constraint (63) ensures that the aircraft does not take a route that requires more fuel than it has in it's tank. Lastly, constraint (64) restricts the model to binary routing decisions. The nonlinearity of constraint (62) differs from the typical MILP in that it requires specialized software or customized algorithms to assure that a global optimal solution is attained.

The objective of the decision dependent formulation is malleable in that it depends on the importance of visiting a number interest locations versus the loiter time over each interest location. Either objective can be constrained in an ε-constrained formulation to require a certain number of visited locations and/or a certain amount of loiter time. The former bound is the simplest constraint to implement; where the sum total of all dimensions of x may be required to be greater than or equal to the number of locations. The latter bound is a harder constraint to capture in a linear form without estimating fuel burn for a certain loiter time without accounting for change in weight. A nonlinear formulation would be the most appropriate constraint for a realistic, small model, but a similar result can be obtained by constraining the minimum amount of loiter time spent at a certain location. In the next formulation, the minimum amount of fuel used for loiter time is specified by f_{min}.

$$\max \quad \sum_{\tau\in T}\sum_{i\in I}\sum_{j\in J} x_{ij}^{\tau} \tag{65}$$

$$\text{s.t.} \quad \sum_{j\in J} x_{sj}^{\tau} = \begin{cases} 1, & \tau = 1, \\ 0, & \forall \tau \in T - \{1\}, \end{cases} \tag{66}$$

$$\sum_{j\in J} x_{ij}^{\tau+1} - \sum_{j\in J} x_{ji}^{\tau} = 0, \quad \forall i \in N - \{s\}, \tau \in T, \tag{67}$$

$$\sum_{i\in I} x_{ii}^{\tau} = 0, \quad \forall \tau \in T, \tag{68}$$

$$\sum_{i\in I}\sum_{\tau\in T} x_{ij}^{\tau} \leq 1, \quad \forall j \in J, \tag{69}$$

$$\sum_{\tau\in T}\sum_{i\in I} x_{is}^{\tau} = 1, \tag{70}$$

$$W_F - \left(\sum_{i\in I}\sum_{j\in J} E_{ij} W_F x_{ij}^{1}\right) - F^{1} \leq -f_{min}, \tag{71}$$

$$W_F \sum_{i\in I}\sum_{j\in J} x_{ij} - \left(\sum_{i\in I}\sum_{j\in J} E_{ij}(W_F - \sum_{k=1}^{\tau} F^{k-1}) x_{ij}^{\tau}\right) - F^{\tau} \leq - f_{min} * \sum_{i\in I}\sum_{j\in J} x_{ij}^{\tau}, \quad \forall \tau \in T - \{1\}, \tag{72}$$

$$\sum_{\tau\in T}\sum_{i\in I}\sum_{j\in J} F^{\tau} \leq F_{total}, \tag{73}$$

$$x_{ij}^{\tau} \in \{0, 1\}, \quad \forall i \in I, j \in J, \tau \in T. \tag{74}$$

Alhough the constraints are nonlinear, the solution is a realistic picture of where an aircraft should travel, given that a cruise at a lighter weight will result in a lower fuel burn than at a heavier weight.

The formulation of this optimization problem is tested on an arbitrary problem of five cities in the state of Ohio, Cleveland, Cincinnati, Columbus, Youngstown, and Akron. The starting node is arbitrarily chosen as Dayton, Ohio. The aircraft used

in the model is the AC4-000 due to its fuel capacity. There is no combat or climb fuel used in the model to simplify the results, but the inclusion of these would be a change to the lower bound on constraint (61) and constraint (62). Additionally, the aircraft does not drop its external fuel tanks or payload for model simplicity as it did in previous models.

The number of combinations available for values of x becomes intractable as the number of locations increase. The five cities available to visit with a differing f_{min} creates interesting results. Specifically, the route the aircraft takes follows paths that depend on how much fuel the aircraft burns at each location. Starting with a f_{min} set at 1000 lbs and maximizing the number of locations visited, the path follows the aircraft from Dayton to Cincinnati to Columbus to Akron to Cleveland to Youngstown and returned to Dayton. All optimization formulations were run using the SCIP solver [20]. The results are shown in Figure 13. There are several equivalent optimal solutions as the summation of every F^{τ} is less than the total fuel available in the aircraft. The addition of a summation over the total fuel consumption while weighting the optimization function to also consider the number of locations visited at each time step will further incentivize the model to seek the most fuel efficient route while loitering for at least 1000 lbs of fuel which is equivalent to loitering for at least ten minutes (varying slightly due to initial weight at each location).

Constraining the model further to allow for at least 2200 lbs of fuel burn at each visited location gives a result where not all cities are visited. The path follows the aircraft from Dayton to Cincinnati to Columbus to Youngstown to Akron and back to Dayton. The results are shown in Figure 14. The formulation requires a fuel burn of 2200 lbs at each location which is equivalent to at least 25 minutes of loiter time at each location. The path follows a circular pattern that looks similar to a traveling salesman problem while excluding Cleveland. While Cleveland is closer

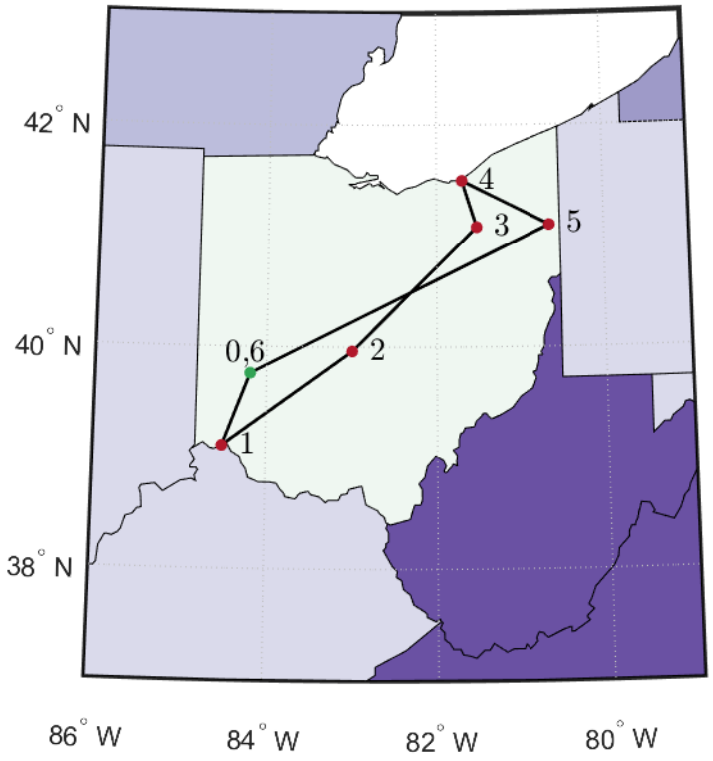


Figure 13. Loiter Constrained to at least 1000 lbs of Fuel

to the starting location, Youngstown is closer to other interest locations and so the optimization chose a path to Youngstown.

Lastly, constraining the model by 3000 lbs of fuel burn at each location demands a loiter time of at least 35 minutes. The results are shown in Figure 15. The path shows a more fuel efficient technique where the aircraft's path burns more fuel on its second decision and travels to Cleveland after traveling to Cincinnati in the first decision step. The path is conservative in its third and fourth decision step, traveling to Columbus and then returning to Dayton. Traveling to Akron and Youngstown is considered infeasible with the required loiter time. There are multiple optimal solutions in this scenario as there is still fuel available. Using a weighted optimization technique with an f_{min} at 3000 lbs to find the most fuel efficient route while visiting

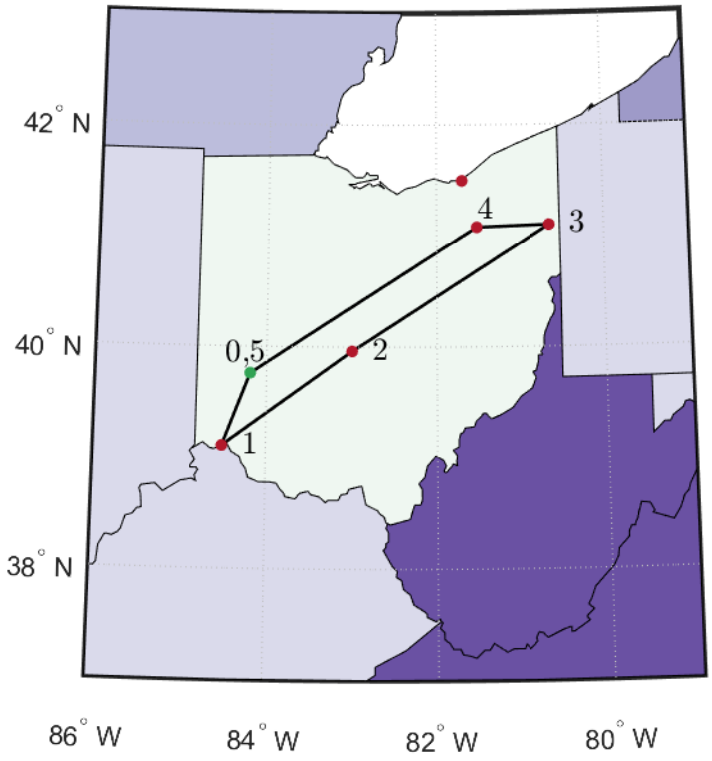


Figure 14. Loiter Constrained to at least 2200 lbs of Fuel

the most locations, the objective function becomes

$$\max \quad w_1 \sum_{\tau \in T} \sum_{i \in I} \sum_{j \in J} x_{ij}^{\tau} - w_2 \sum_{\tau \in T} F^{\tau} \tag{75}$$

where w_1 and w_2 are arbitrary weights to scale the fuel with the binary location variable. In this example case, w_1, w_2 were set to 5000 and 1, respectively. The results are shown in Figure 16. The example shows a path from Dayton to Cleveland to Akron to Youngstown. The path chooses a cluster of the closest cities in optimizing the fuel efficiency coupled with visiting the same number of cities as in the previous scenario. Interestingly, the aircraft does not visit Columbus at the end of its path, most likely due to the efficiency of being lighter and cruising farther in the last leg.

The use of this technique depends on the objective function and what is desired

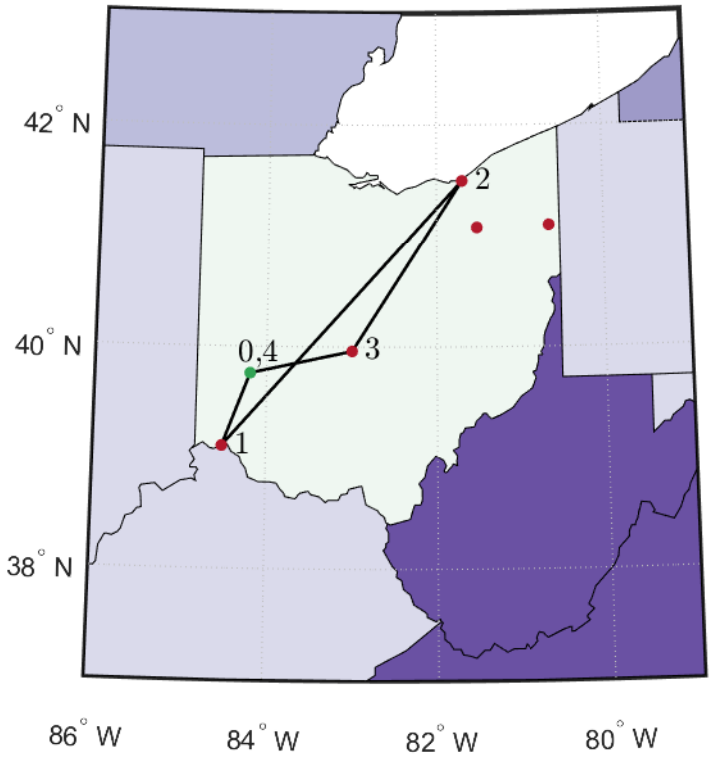


Figure 15. Loiter Constrained to at least 3000 lbs of Fuel

from the MINLP model and the linear program developed in the previous subsection. The incorporation of aircraft characteristics and parameters allows for a more realistic picture of what path an aircraft could take while meeting certain requirements.

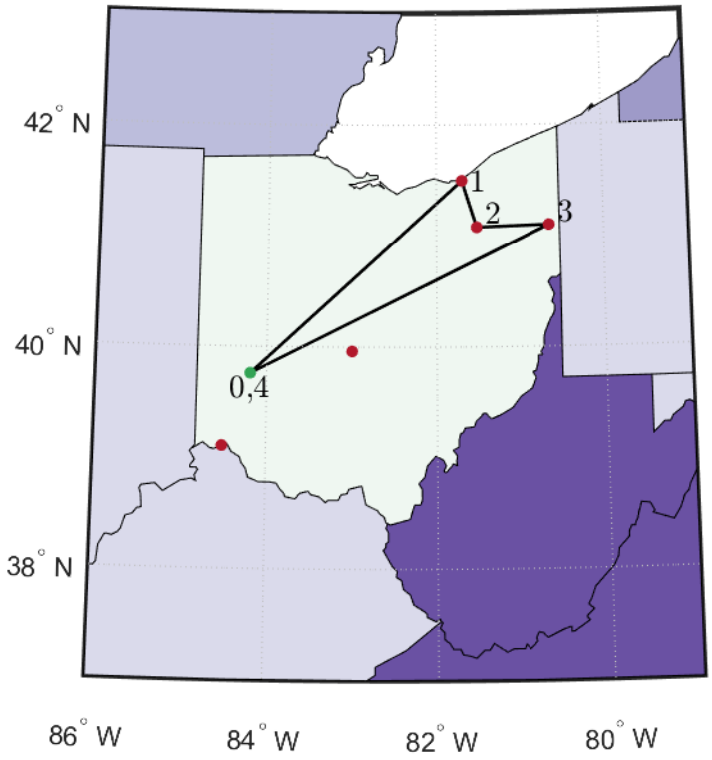


Figure 16. Weighted Objective Function Result

VI. Conclusion and Future Research

6.1 Conclusion

The proposed estimation methodology uses aircraft performance characteristics. The average difference between the tested NASIC model and the developed methodology showed that the estimation technique is effective and the tradeoff between performance metrics can be quickly estimated with one run of the model for a combination of any two of the metrics, range, loiter time, or loiter altitude, with the third held constant. This efficiency in runtime and estimation of tradeoffs is far superior to the original NASIC method which involved running individual points and linearly interpolating between results. The new method can be used to account for flight performance characteristics with a much more efficient use of resources and time.

The developed methodology also benefited the optimization of realistic assignment and routing problems. The nonlinear range equation was used to estimate fuel efficiency and accurately estimate a maximum range and loiter time at a given altitude and specific performance characteristics. The use of this equation in optimization is a novel approach to involving performance characteristics within the assignment and routing formulation. Incorporating the nonlinear range equation in a simple priority assignment problem allowed for a more realistic view of how an aircraft might be assigned to interest locations for loiter depending on their performance characteristics and the priority assigned to each location. Additionally, the involvement of the nonlinear range equation in accounting for fuel efficiency in an a routing formulation for a lighter aircraft indicated that a realistic approach to aircraft performance may be better served with a dynamic constraint rather than using a constant for fuel burn in optimization formulations.

6.2 Future Research

This research is heavily constrained by the estimation of aircraft performance characteristics. Since velocity, lift over drag, and thrust-specific fuel consumption are dynamic as the aircraft burns fuels, accuracy could be increased with a table lookup for each major stage of the aircraft's weight change. The nonlinear range equation estimates the dynamic performance of the aircraft, but accuracy suffers as the weight changes drastically in the cases with a payload or external fuel tank drop. Another dynamic aspect of aircraft performance is the inclusion of flight conditions. While the NASIC model also does not include these factors, the inclusion of weather, maintenance, lethal envelopes, terrain constraints and/or uncertainty would elevate the real-world relevance of the developed methodology.

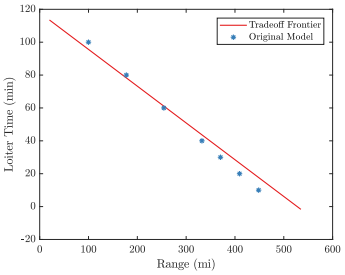
The research's optimization section focused on small example problems for validation of the formulations. Testing the formulated optimization models against relevant literature instances would increase the validity of the model and better examine the impact of adding the dynamic fuel constraint. Efficiency of the optimization model is decreased in its nonlinear form and would also benefit from a dynamic programming approach.

The inclusion of several different jet aircraft allowed for additional adjustments of the model. Extending the methodology to different types of aircraft to include propeller-driven aircraft and unmanned aerial systems would be a valid extension of this model. These are referenced by different range equations and parameters. The routing assignment problem might also better serve an unmanned aerial system versus a jet aircraft due to its mission capabilities and requirements such as range and loiter time over target areas.

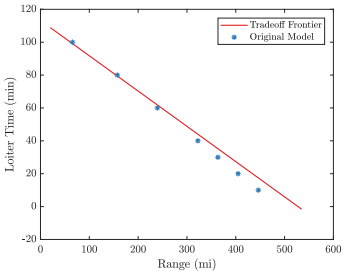
Appendix A. F-15C Parameter Extraction Data

		C_L														
		0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.4	1.6
C_D	0	0.0238	0.0228	0.0258	0.0318	0.0408	0.0548	0.0788	0.1168	0.1628	0.2188	0.2788	0.3568	0.4728	0.8908	1.332
	0.4	0.0238	0.0228	0.0258	0.0318	0.0408	0.0548	0.0788	0.1168	0.1651	0.2238	0.2848	0.3618	0.4788	0.9008	1.346
	0.6	0.0237	0.0227	0.0257	0.0317	0.0407	0.0547	0.0777	0.1167	0.1667	0.2283	0.2957	0.3767	0.5007	0.9237	1.369
	0.7	0.0237	0.0237	0.0257	0.0317	0.0407	0.0547	0.0767	0.1171	0.1688	0.232	0.3017	0.3937	0.5217	0.9347	1.375
	0.8	0.0238	0.0238	0.0268	0.0318	0.0408	0.0547	0.0774	0.1181	0.1728	0.2378	0.3108	0.4108	0.5438	0.9458	1.381
	0.85	0.024	0.024	0.0273	0.0322	0.0408	0.0548	0.0791	0.1191	0.175	0.242	0.318	0.421	0.555	0.95	1.384
	0.87	0.0244	0.0241	0.0275	0.0326	0.0415	0.0552	0.0803	0.12	0.1764	0.2445	0.3224	0.4254	0.5604	0.9524	1.386
	0.9	0.0258	0.0248	0.0282	0.0335	0.0428	0.0572	0.0848	0.1238	0.1788	0.2488	0.3278	0.4318	0.5658	0.9548	1.388
	0.92	0.0271	0.0261	0.0291	0.0345	0.0443	0.0601	0.0911	0.1311	0.1831	0.2541	0.3341	0.4391	0.5671	0.9551	1.39
	0.95	0.0296	0.0276	0.0308	0.0364	0.0469	0.0656	0.0996	0.1376	0.1907	0.2606	0.3416	0.4466	0.5696	0.9536	1.392
	1	0.0371	0.0361	0.0401	0.0511	0.0661	0.0891	0.1171	0.1551	0.2011	0.2651	0.3471	0.4531	0.5751	0.9321	1.375
	1.05	0.0446	0.0426	0.0466	0.0576	0.0766	0.1016	0.1326	0.1706	0.2156	0.2766	0.3536	0.4605	0.5766	0.8856	1.241
	1.1	0.0497	0.0467	0.0507	0.0647	0.0847	0.1127	0.1457	0.1847	0.2317	0.2937	0.3747	0.4737	0.5767	0.7657	0.9597
	1.15	0.0527	0.0487	0.0527	0.0677	0.0897	0.1187	0.1537	0.1977	0.2467	0.3087	0.3847	0.4767	0.5747	0.7377	0.8987
	1.2	0.0547	0.0501	0.0551	0.0701	0.0931	0.1241	0.1621	0.2081	0.2611	0.3241	0.4041	0.4891	0.5751	0.7381	0.9101
	1.25	0.0556	0.0508	0.0556	0.0716	0.0946	0.1276	0.1676	0.2156	0.2736	0.3376	0.4106	0.4946	0.5816	0.7406	0.9136
	1.3	0.0565	0.0513	0.0558	0.0725	0.0965	0.1305	0.1725	0.2255	0.2895	0.3535	0.4265	0.5055	0.5905	0.7495	0.9195
	1.4	0.0574	0.0516	0.0564	0.0734	0.0994	0.1364	0.1844	0.2444	0.3164	0.3854	0.4564	0.5304	0.6044	0.7664	0.9244
	1.5	0.0574	0.0517	0.057	0.0754	0.1024	0.1434	0.1954	0.2634	0.3354	0.4064	0.4784	0.5524	0.6234	0.7794	0.9274
	1.6	0.0566	0.0516	0.0576	0.0756	0.1056	0.1496	0.2066	0.2816	0.3576	0.4286	0.5016	0.5746	0.6416	0.7886	0.9256
	1.8	0.0532	0.0494	0.0587	0.0792	0.1132	0.1612	0.2292	0.3002	0.3732	0.4402	0.5102	0.6152	0.6492	0.7782	0.9042
	2	0.0501	0.0484	0.0606	0.0841	0.1231	0.1761	0.2371	0.3021	0.3701	0.4341	0.4971	0.5661	0.6331	0.7651	0.8831
	2.2	0.0481	0.0481	0.0631	0.0931	0.1371	0.1931	0.2531	0.3141	0.3751	0.4401	0.5021	0.5671	0.6331	0.7581	0.8791
	2.4	0.0475	0.0481	0.0661	0.1031	0.1531	0.2141	0.2731	0.3341	0.3931	0.4551	0.5161	0.5781	0.6411	0.7571	0.8781
	2.5	0.0473	0.0481	0.0681	0.1081	0.1621	0.2241	0.2841	0.3451	0.4051	0.4651	0.5261	0.5861	0.6471	0.7671	0.8881

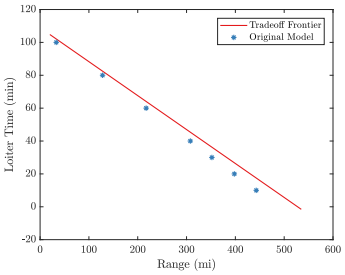
Appendix B. Tradeoff Comparisons



(a) Comparison at 30,000 ft

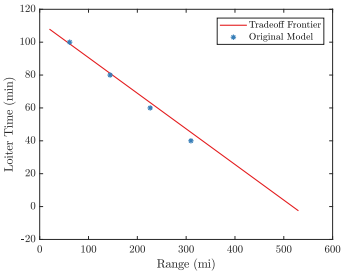


(b) Comparison at 20,000 ft

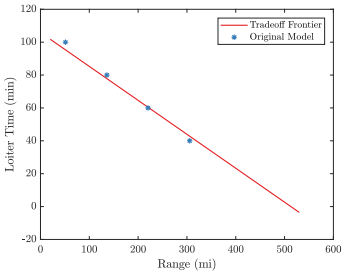


(c) Comparison at 10,000 ft

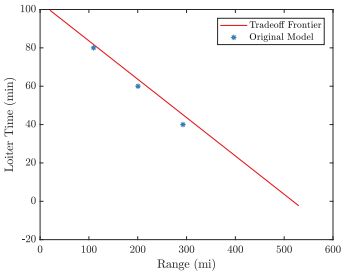
Figure 17. Tradeoff Comparison for F-16C



(a) Comparison at 30,000 ft

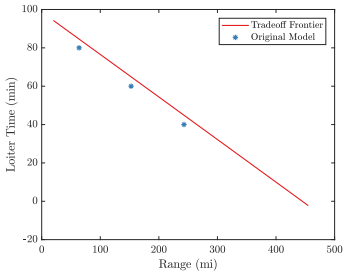


(b) Comparison at 20,000 ft

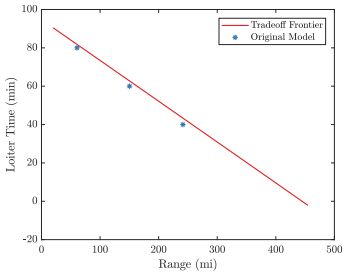


(c) Comparison at 10,000 ft

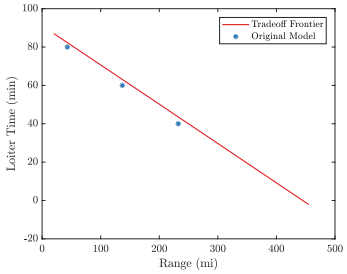
Figure 18. Tradeoff Comparison for AC1-000



(a) Comparison at 30,000 ft

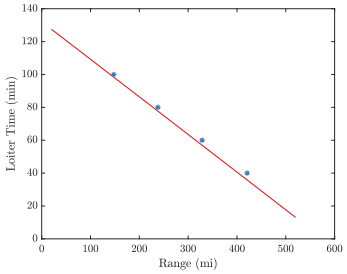


(b) Comparison at 20,000 ft

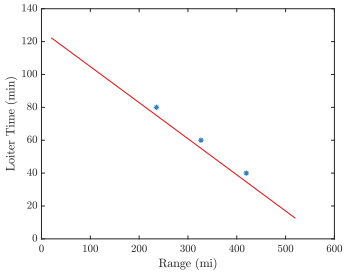


(c) Comparison at 10,000 ft

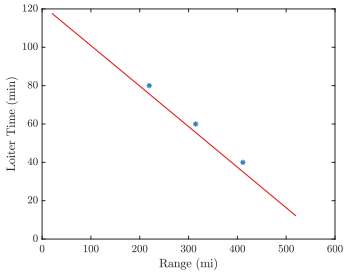
Figure 19. Tradeoff Comparison for AC2-000



(a) Comparison at 30,000 ft

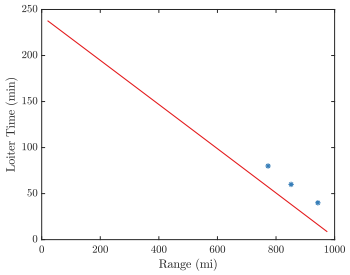


(b) Comparison at 20,000 ft

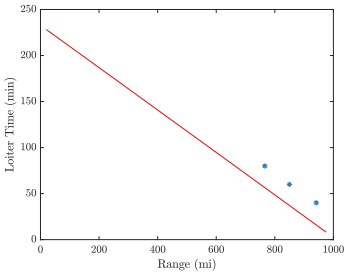


(c) Comparison at 10,000 ft

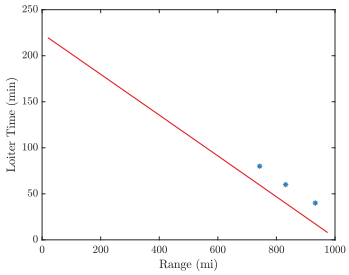
Figure 20. Tradeoff Comparison for AC3-000



(a) Comparison at 30,000 ft



(b) Comparison at 20,000 ft



(c) Comparison at 10,000 ft

Figure 21. Tradeoff Comparison for AC4-000

Appendix C. MATLAB Code for Structure Tradeoffs

A Linear Tradeoff Code

```
function endurance = simpleplan(TSFCsl,LDmax, Range, fuelToCruise,
    distToCruise, maxRange, dryWeight, maxTOW, fuelReserve, Altitude)
%range:        user defined range to calculate endurance (miles)
%fuelToCruise:  weight after takeoff, warmup, and climb to cruise (lbs)
%distToCruise:  distance traveled after takeoff and climb to cruise
%maxRange:     total range for fuel left after climb to cruise (change to
%              calculate by itself after a while) (in miles)
%dryWeight:    empty weight of a/c
%maxTOW:       max take off weight of aircraft
%fuelReserve: user defined fuel reserve left in aircraft for return
%              (percent)
%altitude:     flight plan altitude (in meters)
%TSFCsl:       thrust specific fuel consumption for aircraft at sea level
%endurance:    length of time for loiter (in hours)
%
    ------------------------------------------------------------------------


%Example: T-37 Cessna at 20000ft traveling to 300 miles
%endure = simpleplan(.9,14.8,300,5698,1.5,1391.2,3869,6598,0.2,20000)
%Output: endure = 2.6378 hours
initFuelPerc = fuelToCruise/maxTOW; %Fuel reserve calculation
slopeEgress = (0-initFuelPerc)/(maxRange-distToCruise);
slopeIngress = -slopeEgress;
bEgress = slopeEgress*distToCruise + initFuelPerc; %Intercept for egress
    slope
fuelAvail = maxTOW - dryWeight;
Altitude = convlength(Altitude,'ft','m');


%Intersection point
```

```
inter = (bEgress-fuelReserve)/(2*slopeIngress);

if Range>inter
    endurance ='Wanted range exceeds maximum range';
    print(endurance)
    return
end

weightInitial = (slopeEgress*Range + bEgress)*fuelAvail + dryWeight;
weightFinal = (slopeIngress*Range + fuelReserve)*fuelAvail + dryWeight;

[T,~,~,~] = atmosisa(Altitude); %Matlab function for temperature (Kelvin
    ) at altitude
tempK = convtemp(518.7,'R','K');
theta = T/tempK; %TSFC correction ratio

TSFC = TSFCsl*sqrt(theta); %Corrected TSFC

endurance = 1/TSFC*(LDmax)*log(weightInitial/weightFinal);
x = 0:.1:(inter+300);
y = 1.5:.1:(inter+300);
eqEgress = slopeEgress*y+bEgress;
eqIngress = slopeIngress*x+fuelReserve;
placement = (weightFinal+weightInitial-2*dryWeight)/(2*fuelAvail);
plot(y,eqEgress, x, eqIngress, [Range Range],...
    [(weightFinal-dryWeight)/fuelAvail (weightInitial-dryWeight)/
    fuelAvail],...
    [0 distToCruise],[1 (maxTOW-fuelToCruise)/fuelAvail])
txt = {'Fuel Available', 'for Loiter \rightarrow'};
text(Range,placement,txt,'HorizontalAlignment','right')
title('Flight Plan Mapping')
ylabel('Fuel Reserve (Reserve/Total Fuel)')
```

```
xlabel('Range (mi)')
legend('Egress','Ingress','Endurance Fuel @ Range')

end
```

B Nonlinear Tradeoff Code

```
function [weightInitial, weightFinal, eqIngress, eqEgress, intersect,
    endurance] = FixForWeight(ac)

Altitude = convlength(ac.Altitude,'ft','m');
loiterAlt = convlength(ac.loiterAlt,'ft','m');
[T1,a1,~,~] = atmosisa(Altitude); %Matlab function for temperature (
    Kelvin) at altitude
[T2,~,~,~] = atmosisa(loiterAlt);
[T3,a3,~,~] = atmosisa(3048);
tempK = convtemp(518.7,'R','K');
theta1 = T1/tempK; %TSFC correction ratio
theta2 = T2/tempK;
TSFC_cruise = ac.TSFCsl*sqrt(theta1)/3600; %Corrected TSFC
TSFC_loiter = ac.TSFCsl*sqrt(theta2)/3600;
TSFC_drop = ac.TSFCsl*sqrt(T3/tempK)/3600;
speedMax_cruise = ac.speedMax* convvel(a1,'m/s','ft/s'); %mach to ft/s
    at altitude

% maxRange = (speedMax_cruise/5280)/(TSFC_cruise)*ac.LDmaxLog*log(ac.
    maxTOW/ac.dryWeight);
%Fuel Reserve Conversion
percFuel = (ac.eMaxTOW-ac.eDryWeight)*ac.fuelReserve;
solInitialWeight = exp(ac.TimeReserve*(TSFC_cruise*60)/ac.LDmax)*ac.
    eDryWeight;
```

```
percFuel = (percFuel +(solInitialWeight-ac.eDryWeight))/(ac.eMaxTOW-ac.
    eDryWeight);
%
divider = 5280;
initialReserve = 1-(ac.maxTOW-ac.fuelToCruise)/ac.maxTOW;
translateI = -(log((initialReserve*(ac.maxTOW-ac.dryWeight)+ac.dryWeight
    )/ac.maxTOW)*...
     speedMax_cruise/5280*ac.LDmaxLog/TSFC_cruise+ac.distToCruise);
translateE = (-(ac.LDmaxLog*speedMax_cruise/5280*log((ac.eDryWeight...
     + ac.eMaxTOW*percFuel-ac.eDryWeight*percFuel)/(ac.eMaxTOW)))/
    TSFC_cruise);
intersect = (ac.LDmaxLog*speedMax_cruise*log(-(exp(-(TSFC_cruise*divider
    *...
     translateI)/(2*ac.LDmaxLog*speedMax_cruise))*(exp((TSFC_cruise*
    divider*...
     translateE)/(2*ac.LDmaxLog*speedMax_cruise))*(4*ac.eMaxTOW^2*ac.
    maxTOW^2 + ...
     ac.dryWeight^2*ac.eMaxTOW^2*exp((TSFC_cruise*divider*(translateI +
    ...
     translateE)))/(ac.LDmaxLog*speedMax_cruise)) + ac.eDryWeight^2*ac.
    maxTOW^2*...
     exp((TSFC_cruise*divider*(translateI + translateE))/(ac.LDmaxLog*...
     speedMax_cruise)) - 4*ac.dryWeight*ac.eMaxTOW^2*ac.maxTOW - 4*ac.
    eDryWeight*...
     ac.eMaxTOW*ac.maxTOW^2 + 4*ac.dryWeight*ac.eDryWeight*ac.eMaxTOW*ac.
    maxTOW - ...
     2*ac.dryWeight*ac.eDryWeight*ac.eMaxTOW*ac.maxTOW*exp((TSFC_cruise*
    divider*...
     (translateI + translateE))/(ac.LDmaxLog*speedMax_cruise)))^(1/2) -
    ...
     ac.dryWeight*ac.eMaxTOW*exp((TSFC_cruise*divider*(translateI + 2*...
```

```
39      translateE))/(2*ac.LDmaxLog*speedMax_cruise)) + ac.eDryWeight*ac.
        maxTOW*...
        exp((TSFC_cruise*divider*(translateI + 2*translateE))/(2*...
41      ac.LDmaxLog*speedMax_cruise))))/(2*ac.eMaxTOW*(ac.dryWeight - ac.
        maxTOW))))/(TSFC_cruise*divider);


43

% if ac.Range>intersect
45 %      endurance ='Wanted range exceeds maximum range\n';
%       fprintf(endurance)
47 %      eqEgress = 0;
%       eqIngress = 0;
49 %      weightInitial = 0;
%       weightFinal = 0;
51 %      return
% end

53

weightInitial = (exp(-(ac.Range+translateI)*TSFC_cruise/(speedMax_cruise
    /5280*ac.LDmaxLog))*ac.maxTOW-...
55      ac.dryWeight)/(ac.maxTOW-ac.dryWeight);
weightFinal = (exp((ac.Range-translateE)*TSFC_cruise/(speedMax_cruise
    /5280*ac.LDmaxLog))*...
57      ac.eMaxTOW-ac.eDryWeight)/(ac.eMaxTOW-ac.eDryWeight);
%Change from fraction of reserve to aircraft weight
59 weightInitial = weightInitial*(ac.maxTOW-ac.dryWeight)+ac.dryWeight-(ac.
    dryWeight-ac.eDryWeight);
weightFinal = weightFinal*(ac.eMaxTOW-ac.eDryWeight)+ac.eDryWeight;

61

minlostCombat = 1/(TSFC_loiter*60)*(ac.LDmax)*log((ac.eDryWeight+ac.
    Combat)/ac.eDryWeight);
63 minlostClimb = 1/(TSFC_loiter*60)*(ac.LDmax)*log((ac.eDryWeight+ac.
    climbFuel)/ac.eDryWeight);
```

```
endurance = 1/(TSFC_loiter*60)*(ac.LDmax)*log((weightInitial)/
    weightFinal) -...
     minlostCombat-minlostClimb;
% x = 0:.1:(intersect+300);
% y = 1.5:.1:(intersect+300);
% eqEgress = (exp(-(x+translateE)*TSFC_cruise/(speedMax_cruise/5280*ac.
    LDmaxLog))*ac.maxTOW-...
%       ac.dryWeight)/(ac.maxTOW-ac.dryWeight);
% eqIngress = (exp((y-translateI)*TSFC_cruise/(speedMax_cruise/5280*ac.
    LDmaxLog))*...
%       ac.maxTOW-ac.dryWeight)/(ac.maxTOW-ac.dryWeight);

eqEgress = 0;
eqIngress = 0;

end
```

C Example Structure Code

```
% Sample F-15C
ac.TSFCsl = 1.1590; %Thrust specific fuel consumption per hour at sea
    level
ac.LDmax = 8.52; %lift over drag max (Maybe LDmaxLog)
%ac.LsqrDmax = 17.4013; %C_L^(1/2)/C_D max
ac.speedMax = .7318; %mach point where C_L^(1/2)/C_D is max
ac.LDmaxLog = 8.52; %L/D where C_L^(1/2)/C_D is max
ac.fuelToCruise = 42738; %a/c weight after climb to cruise
ac.distToCruise = 27.2*1.15; %Miles from base when climbing to cruise
ac.Combat = 688;
ac.dryWeight = 31262; %No fuel weight of a/c w/ payload
ac.maxTOW = 44710; %Max takeoff weight of a/c
```

```
ac.eDryWeight = 30645;
ac.eMaxTOW = 44093;
ac.S = 608; %Wing planform area
%% User-Defined Inputs
ac.Range = 10; %Miles
ac.fuelReserve = 0.05; %%left of fuel
ac.TimeReserve = 20; % minutes of endurance available for aircraft
ac.Altitude = 30000; %Feet from sea level
ac.loiterAlt = 30000;
ac.climbFuel = 523; % (Uses fuel burn estimate)


%% Pareto Front
xValue = 20:5:intercept-20;
figure('Name', ' Nonlinear Pareto Frontier')
for j = 1:8
    ac.loiterAlt = 5000*j;
    pFront = [];
    for i = 20:5:intercept-20
        ac.Range = i;
        [~,~,~,~,~,endure] = FixForWeight(ac);
        pFront = [pFront endure];
    end

    plot(xValue,pFront)
    hold on
end

title('Pareto Frontier for Nonlinear Range vs. Time')
ylabel('Loiter Time (min)')
xlabel('Range (mi)')
```

```
legend('5000ft','10000ft','15000ft','20000ft','25000ft','30000ft','35000
    ft','40000ft')

%% Plot Points (30000ft)
ac.loiterAlt = 30000;
pFront = [];
for i = 20:5:intercept-20
    ac.Range = i;
    [~,~,~,~,~,endure] = FixForWeight(ac);
    pFront = [pFront endure];
end
figure('Name',' Pareto Front Comparison (Altitude = 30000ft)')
plot(xValue, pFront,'MarkerSize',7)
hold on
plot([399*1.15 368*1.15 335*1.15 303.5*1.15 239.3*1.15 175.8*1.15
    111.9*1.15 47.6*1.15], [10 20 30 40 60 80 100 120],'*')
ylabel('Loiter Time (min)')
xlabel('Range (mi)')
legend('Tradeoff Frontier','Original Model')
hold off

% Difference between observations
distVect = [399*1.15 368*1.15 335*1.15 303.5*1.15 239.3*1.15 175.8*1.15
    111.9*1.15 47.6*1.15];
endureVect = [10 20 30 40 60 80 100 120];
avgDiff = 0;
maxVal = 0;
for i = 1:length(distVect)
    ac.Range = distVect(i);
    [~,~,~,~,~,endureVal] = FixForWeight(ac);
    avgDiff = avgDiff + abs(endureVal-endureVect(i));
    if abs(endureVal-endureVect(i))>maxVal
```

```
72          maxVal = abs(endureVal-endureVect(i));
        end
74  end
    avgDiff = avgDiff/length(distVect);
76
    %% Plot Points (20000ft)
78  ac.loiterAlt = 20000;
    pFront = [];
80  for i = 20:5:intercept-20
        ac.Range = i;
82      [~,~,~,~,~,endure] = FixForWeight(ac);
        pFront = [pFront endure];
84  end
    figure('Name',' Pareto Front Comparison (Altitude = 20000ft)')
86  plot(xValue, pFront,'MarkerSize',7)
    hold on
88  plot([397*1.15 363*1.15 328*1.15 294.1*1.15 225*1.15 157*1.15
        88.4*1.15], [10 20 30 40 60 80 100],'*')
    ylabel('Loiter Time (min)')
90  xlabel('Range (mi)')
    legend('Tradeoff Frontier','Original Model')
92  hold off

94  % Difference between observations
    distVect = [397*1.15 363*1.15 328*1.15 294.1*1.15 225*1.15 157*1.15
        88.4*1.15];
96  endureVect = [10 20 30 40 60 80 100];
    avgDiff = 0;
98  maxVal = 0;
    for i = 1:length(distVect)
100     ac.Range = distVect(i);
        [~,~,~,~,~,endureVal] = FixForWeight(ac);
```

```
102     avgDiff = avgDiff + abs(endureVal-endureVect(i));
        if abs(endureVal-endureVect(i))>maxVal
104         maxVal = abs(endureVal-endureVect(i));
        end
106 end
    avgDiff = avgDiff/length(distVect);
108
    %% Plot Points (10000ft)
110 ac.loiterAlt = 10000;
    pFront = [];
112 for i = 20:5:intercept-20
        ac.Range = i;
114     [~,~,~,~,~,endure] = FixForWeight(ac);
        pFront = [pFront endure];
116 end
    figure('Name',' Pareto Front Comparison (Altitude = 10000ft)')
118 plot(xValue, pFront,'MarkerSize',7)
    hold on
120 plot([394*1.15 357*1.15 319*1.15 281.8*1.15 206.4*1.15 132.3*1.15
        57.5*1.15], [10 20 30 40 60 80 100],'*')
    ylabel('Loiter Time (min)')
122 xlabel('Range (mi)')
    legend('Tradeoff Frontier','Original Model')
124 hold off

126 % Difference between observations
    distVect = [394*1.15 357*1.15 319*1.15 281.8*1.15 206.4*1.15 132.3*1.15
        57.5*1.15];
128 endureVect = [10 20 30 40 60 80 100];
    avgDiff = 0;
130 maxVal = 0;
    for i = 1:length(distVect)
```

```
132     ac.Range = distVect(i);
        [~,~,~,~,~,endureVal] = FixForWeight(ac);
134     avgDiff = avgDiff + abs(endureVal-endureVect(i));
        if abs(endureVal-endureVect(i))>maxVal
136         maxVal = abs(endureVal-endureVect(i));
        end
138 end
    avgDiff = avgDiff/length(distVect);
```

Appendix D. MATLAB Code for Optimization Formulations

A Assignment Problem

```
data = load('c4_AC000');
locales  = xlsread('uscities(1000largest)');
locales = locales(1:100,:);
data.ac.Combat = 0;
data.ac.climbFuel = 0;
n = 20;
m = length(locales);
startLat = 39.9025;
startLong = -84.2218;
x = binvar(n,m);
d = zeros(m,1);
E = zeros(n,m);

for j = 1:m
    d(j) = dist2pts(startLat,startLong,locales(j,1),locales(j,2));
    for i = 1:n
        E(i,j) = findEnduranceGivenRange(data.ac,d(j));
        if E(i,j) <0
            E(i,j) = 0;
        end
    end
end

const = [];

const = [const sum(x,1)<=1];

const = [const sum(x,2)<=1];

```

```
obj = sum(sum(E.*x));

%Options
options = sdpsettings('solver', 'scip');
% Solve the problem
sol = optimize(const,-obj,options);

% Analyze error flags
if sol.problem == 0
 % Extract and display value
else
 display('Hmm, something went wrong!');
 sol.info
 yalmiperror(sol.problem)
end

%% Plot map
%For each plane if row,col>0 then which = city and plot coordinates
    using
%plotm.
vectLatLong = plotAircraftPaths(value(x),locales);

figure()
ax = usamap('conus');
states = shaperead('usastatelo', 'UseGeoCoords', true,...
   'Selector',...
   {@(name) ~any(strcmp(name,{'Alaska','Hawaii'})), 'Name'});

colormap = cbrewer('seq','Purples',4);
% faceColors = makesymbolspec('Polygon',...
%      {'INDEX', [1 numel(states)], 'FaceColor', ...
%      polcmap(numel(states))}); %NOTE - colors are random
```

```
faceColors = makesymbolspec('Polygon',...
    {'INDEX', [1 numel(states)], 'FaceColor', ...
    colormap}); %NOTE - colors are random
geoshow(ax, states, 'DisplayType', 'polygon','SymbolSpec', faceColors)
set(gcf, 'Renderer', 'opengl') % remove grids for contourf and the
   dashed line in colorbar when saving pdf/eps. in matlab2016a.
hold on

scatterm(locales(:,1),locales(:,2),3,1/255*[228 26 28],'filled')
hold on
for i = 1:length(vectLatLong)
plotm([startLat startLong; vectLatLong(i,1) vectLatLong(i,2)],'Color'
    ,1/255*[255 127 0])
hold on
end
```

B Random Priority Assignment Problem

```
data = load('c4_AC000');
locales  = xlsread('uscities(1000largest)');
locales = locales(1:100,:);
data.ac.Combat = 0;
data.ac.climbFuel = 0;
n = 20;
m = length(locales);
startLat = 39.9025;
startLong = -84.2218;
x = binvar(n,m);
d = zeros(m,1);
E = zeros(n,m);
priority = randi([1 20],1,length(locales));
```

```
14
for j = 1:m
16      d(j) = dist2pts(startLat,startLong,locales(j,1),locales(j,2));
        for i = 1:n
18          E(i,j) = findEnduranceGivenRange(data.ac,d(j));
            if E(i,j) <0
20              E(i,j) = 0;
            end
22      end
end
24
const = [];
26
const = [const sum(x,1)<=1];
28
const = [const sum(x,2)<=1];
30
obj = sum((E.*x)*priority');
32
%Options
34 options = sdpsettings('solver', 'scip');
% Solve the problem
36 sol = optimize(const,-obj,options);

38 % Analyze error flags
if sol.problem == 0
40  % Extract and display value
else
42  sol.info
  yalmiperror(sol.problem)
44 end
```

```
%% Plot map
%For each plane if row,col>0 then which = city and plot coordinates using
%plotm.
vectLatLong = plotAircraftPaths(value(x),locales);
figure()
ax = usamap('conus');
states = shaperead('usastatelo', 'UseGeoCoords', true,...
  'Selector',...
  {@(name) ~any(strcmp(name,{'Alaska','Hawaii'})), 'Name'});
colormap = cbrewer('seq','Purples',4);
faceColors = makesymbolspec('Polygon',...
    {'INDEX', [1 numel(states)], 'FaceColor', ...
    colormap}); %NOTE - colors are random
geoshow(ax, states, 'DisplayType', 'polygon', ...
   'SymbolSpec', faceColors)
set(gcf, 'Renderer', 'opengl') % remove grids for contourf and the dashed line in colorbar when saving pdf/eps. in matlab2016a.
hold on

scatterm(locales(:,1),locales(:,2),3,[1 0 0],'filled')
hold on
for i = 1:length(vectLatLong)
plotm([startLat startLong; vectLatLong(i,1) vectLatLong(i,2)],'Color',1/255*[255 127 0])
hold on
end
```

C Random Priority Assignment Problem with Multiple Structures

```
% Initialize data
```

```
data1 = load('c4_AC000');
data2 = load('c3_AC000');
data3 = load('c2_AC000');
data4 = load('c1_F15');
locales  = xlsread('uscities(1000largest)');
locales = locales(1:100,:);
data1.ac.Combat = 0; data1.ac.climbFuel = 0;
data2.ac.Combat = 0; data2.ac.climbFuel = 0;
data3.ac.Combat = 0; data3.ac.climbFuel = 0;
data4.ac.Combat = 0; data4.ac.climbFuel = 0;
data(1) = data1; data(2) = data2; data(3) = data3; data(4) = data4;
n = 20;
m = length(locales);

%Parameters
startLat = 39.9025;
startLong = -84.2218;
x = binvar(n,m);
d = zeros(m,1);
E = zeros(n,m);
priority = randi([1  20],1,length(locales));



for j = 1:m
    d(j) = dist2pts(startLat,startLong,locales(j,1),locales(j,2));
    k = 1;
    for i = 1:n
        E(i,j) = findEnduranceGivenRange(data(k).ac,d(j));
        if mod(i,5)==0
            k = k + 1;
        end
    end
```

```
34 end

36 % Constraints
   const = [];
38
   const = [const sum(x,1)<=1];
40
   const = [const sum(x,2)<=1];
42
   obj = sum((E.*x)*priority');
44
   %Options
46 options = sdpsettings('solver', 'scip');
   % Solve the problem
48 sol = optimize(const,-obj,options);

50 % Analyze error flags
   if sol.problem == 0
52  % Extract and display value
   else
54  sol.info
    yalmiperror(sol.problem)
56 end

58 %% Plot map
   %For each plane if row,col>0 then which = city and plot coordinates
       using
60 %plotm.
   vectLatLong = plotAircraftPaths(value(x),locales);
62 figure()
   ax = usamap('conus');
64 states = shaperead('usastatelo', 'UseGeoCoords', true,...
```

```
    'Selector',...
    {@(name) ~any(strcmp(name,{'Alaska','Hawaii'})), 'Name'});

colormap = cbrewer('seq','Purples',4);
faceColors = makesymbolspec('Polygon',...
    {'INDEX', [1 numel(states)], 'FaceColor', ...
    colormap});
geoshow(ax, states, 'DisplayType', 'polygon', ...
    'SymbolSpec',faceColors)
set(gcf, 'Renderer', 'opengl') % remove grids for contourf and the
    dashed line in colorbar when saving pdf/eps. in matlab2016a.
hold on


scatterm(locales(:,1),locales(:,2),3,[1 0 0],'filled')
hold on
for i = 1:5
    plotm([startLat startLong; vectLatLong(i,1) vectLatLong(i,2)],...
        'Color',1/255*[228 26 28])
    hold on
end
for i = 6:10
    plotm([startLat startLong; vectLatLong(i,1) vectLatLong(i,2)],...
        'Color',1/255*[55 126 184])
    hold on
end
for i = 11:15
    plotm([startLat startLong; vectLatLong(i,1) vectLatLong(i,2)],...
        'Color',1/255*[77 175 74])
    hold on
end
for i = 16:20
```

```
96      plotm([startLat startLong; vectLatLong(i,1) vectLatLong(i,2)],...
            'Color',1/255*[255 127 0])
98      hold on
    end
100
    h = zeros(4, 1);
102 h(1) = plot(NaN,NaN,'Color',1/255*[228 26 28],'DisplayName','AC4-000');
    h(2) = plot(NaN,NaN,'Color',1/255*[55 126 184],'DisplayName','AC3-000');
104 h(3) = plot(NaN,NaN,'Color',1/255*[77 175 74],'DisplayName','AC2-000');
    h(4) = plot(NaN,NaN,'Color',1/255*[255 127 0],'DisplayName','F15-C');
106 legend(h,'AC4-000','AC3-000','AC2-000','F-15C')
```

D Decision Dependent Formulation

```
1  %Data
   data = load('c4_AC000');
3  locales  = xlsread('uscities(1000largest)');
   locales = locales([798 119 239 173 10],:);
5  data.ac.Combat = 0;
   data.ac.climbFuel = 0;
7  m = length(locales)+1;
   startLat = 39.9025;
9  startLong = -84.2218;
   totFuel = data.ac.maxTOW - data.ac.dryWeight;
11
   %Decision Variables
13 x = binvar(m,m,m,'full');
   F = sdpvar(m,1,'full');
15
   %Initialize
17 allLatLong = zeros(m,2);
```

```
allLatLong(2:m,:) = locales;
allLatLong(1,:) = [startLat startLong];
Altitude = convlength(data.ac.Altitude,'ft','m');
[T1,a1,~,~] = atmosisa(Altitude); %Matlab function for temperature (
    Kelvin) at altitude
tempK = convtemp(518.7,'R','K');
theta1 = T1/tempK; %TSFC correction ratio
TSFC_cruise = data.ac.TSFCsl*sqrt(theta1)/3600; %Corrected TSFC
speedMax_cruise = data.ac.speedMax* convvel(a1,'m/s','ft/s')/5280; %mach
     to mi/s at altitude

D = zeros(m,m); %Distance matrix
C = zeros(m,m); %Fuel efficiency factor
for i = 1:m
    for j = 1:m
        D(i,j) = dist2pts(allLatLong(i,1),allLatLong(i,2),allLatLong(j
    ,1),allLatLong(j,2));
        C(i,j) = exp(-D(i,j)/speedMax_cruise*TSFC_cruise/data.ac.LDmax);
    end
end

%Constraints

constr = [];

constr = [constr sum(x(1,2:m,1)) == 1];

for k = 2:m
    constr = [constr sum(x(1,:,k)) == 0];
end

for k = 1:m-1
```

```
47      for i = 2:m
            constr = [constr sum(x(i,:,k+1))-sum(x(:,i,k)) == 0];
49      end
end

51
constr = [constr sum(sum(x(:,1,:))) == 1];

53
constr = [constr data.ac.maxTOW-sum(sum(C.*x(:,:,1)))*data.ac.maxTOW<=F
    (1)];

55
for k = 2:m
57      constr = [constr (data.ac.maxTOW-sum(F(1:k-1)))*sum(sum(x(:,:,k)))-(
    sum(sum(C.*x(:,:,k)))*(data.ac.maxTOW-sum(F(1:k-1))))<=F(k)];
end

59
for k = 1:m
61      constr = [constr sum(sum(x(:,:,k)))<=1];
end

63
for k = 1:m
65      for i = 1:m
            constr = [constr x(i,i,k) == 0];
67      end
end

69
for j = 1:m
71      constr = [constr sum(sum(x(:,j,:)))<=1]; %Okay
end

73
constr = [constr sum(F)<=totFuel];

75
obj = 500*sum(sum(sum(x)))+sum(F(m-2:m));
```

```
77
%Options
79 options = sdpsettings('solver','scip','debug','on');
% Solve the problem
81 sol = optimize(constr,-obj,options);

83 %% Plot Result
%For each plane if row,col>0 then which = city and plot coordinates
    using
85 %plotm.
vectLatLong = plotAircraftPaths3D(value(x),locales,startLat,startLong);
87
figure()
89 ax = usamap('ohio');
states = shaperead('usastatelo', 'UseGeoCoords', true,...
91   'Selector',...
  {@(name) ~any(strcmp(name,{'Alaska','Hawaii'})), 'Name'});
93
colormap = cbrewer('seq','Purples',8);
95 faceColors = makesymbolspec('Polygon',...
    {'INDEX', [1 numel(states)], 'FaceColor', ...
97    colormap}); %NOTE - colors are random
geoshow(ax, states, 'DisplayType', 'polygon','SymbolSpec', faceColors)
99 set(gcf, 'Renderer', 'opengl') % remove grids for contourf and the
    dashed line in colorbar when saving pdf/eps. in matlab2016a.
hold on
101
scatterm(startLat,startLong,12,1/255*[228 26 28],'filled')
103 hold on
scatterm(locales(:,1),locales(:,2),12,1/255*[228 26 28],'filled')
105 hold on
vectLatLong = [startLat startLong; vectLatLong];
```

```
107
for i = 1:length(vectLatLong)-1
109     deltLat = -vectLatLong(i,1)+vectLatLong(i+1,1);
        deltLong = -vectLatLong(i,2)+vectLatLong(i+1,2);
111     r = quiverm(vectLatLong(i,1),vectLatLong(i,2),deltLat,deltLong);
        %[1 0.498 0];
113     hold on
end
```

E Decision Dependent Formulation with Constrained Loiter Fuel

```
%Data
2 data = load('c4_AC000');
locales  = xlsread('uscities(1000largest)');
4 locales = locales([169 798 173 10 119],:);
data.ac.Combat = 0;
6 data.ac.climbFuel = 0;
m = length(locales)+1;
8 startLat = 39.7589;
startLong = -84.1916;
10 totFuel = data.ac.maxTOW - data.ac.dryWeight;
fmin = 3000;
12


14 %Decision Variables
x = binvar(m,m,m,'full');
16 F = sdpvar(m,1,'full');

18 %Initialize
allLatLong = zeros(m,2);
20 allLatLong(2:m,:) = locales;
```

```
allLatLong(1,:) = [startLat startLong];
Altitude = convlength(data.ac.Altitude,'ft','m');
[T1,a1,~,~] = atmosisa(Altitude); %Matlab function for temperature (
    Kelvin) at altitude
tempK = convtemp(518.7,'R','K');
theta1 = T1/tempK; %TSFC correction ratio
TSFC_cruise = data.ac.TSFCsl*sqrt(theta1)/3600; %Corrected TSFC
speedMax_cruise = data.ac.speedMax* convvel(a1,'m/s','ft/s')/5280; %mach
     to mi/s at altitude

D = zeros(m,m); %Distance matrix
C = zeros(m,m); %Fuel efficiency factor
for i = 1:m
    for j = 1:m
        D(i,j) = dist2pts(allLatLong(i,1),allLatLong(i,2),allLatLong(j
    ,1),allLatLong(j,2));
        C(i,j) = exp(-D(i,j)/speedMax_cruise*TSFC_cruise/data.ac.LDmax);
    end
end

%Constraints

constr = [];

constr = [constr sum(x(1,2:m,1)) == 1];

for k = 2:m
    constr = [constr sum(x(1,:,k)) == 0];
end

for k = 1:m-1
    for i = 2:m
```

```
50          constr = [constr sum(x(i,:,k+1))-sum(x(:,i,k)) == 0];
    end
52 end

54 constr = [constr sum(sum(x(:,1,:))) == 1];

56 constr = [constr data.ac.maxTOW-sum(sum(C.*x(:,:,1)))*data.ac.maxTOW-F
   (1)<=-fmin];

58 for k = 2:m
    constr = [constr (data.ac.maxTOW-sum(F(1:k-1)))*sum(sum(x(:,:,k)))
    -...
60          (sum(sum(C.*x(:,:,k)))*(data.ac.maxTOW-sum(F(1:k-1))))-F(k)<=-
   fmin*sum(sum(x(:,:,k)))];
end
62

for k = 1:m
64     constr = [constr sum(sum(x(:,:,k)))<=1];
end
66

for k = 1:m
68     for i = 1:m
           constr = [constr x(i,i,k) == 0];
70     end
end
72

for j = 1:m
74     constr = [constr sum(sum(x(:,j,:)))<=1]; %Okay
end
76

constr = [constr sum(F)<=totFuel];
78
```

```
obj = 5000*sum(sum(sum(x)))-sum(F);

%Options
options = sdpsettings('solver','scip','debug','on');
% Solve the problem
sol = optimize(constr,-obj,options);

%% Plot Result
%For each plane if row,col>0 then which = city and plot coordinates
    using
%plotm.
vectLatLong = plotAircraftPaths3D(value(x),locales,startLat,startLong);

figure()
ax = usamap('Ohio');
states = shaperead('usastatelo', 'UseGeoCoords', true,...
  'Selector',...
  {@(name) ~any(strcmp(name,{'Alaska','Hawaii'})), 'Name'});
names = {states.Name};
ohioindex = strcmp('Ohio',names);
pennindex = strcmp('Pennsylvania',names);

colormap = cbrewer('seq','Purples',8);
faceColors = makesymbolspec('Polygon',...
    {'INDEX', [1 50], 'FaceColor', ...
    colormap});
stateColor = 1/255*[239 247 241];
pennColor = 1/255*[218 218 235];
geoshow(ax, states, 'DisplayType', 'polygon','SymbolSpec', faceColors)
geoshow(ax, states(ohioindex), 'FaceColor', stateColor)
geoshow(ax, states(pennindex), 'FaceColor', pennColor)
```

```
set(gcf, 'Renderer', 'opengl') % remove grids for contourf and the
    dashed line in colorbar when saving pdf/eps. in matlab2016a.
hold on

vectLatLong = [startLat startLong; vectLatLong];

plotm(vectLatLong,'Color',[0 0 0])

scatterm(startLat,startLong,20,1/255*[49 163 84],'filled')
hold on
scatterm(locales(:,1),locales(:,2),20,1/255*[178 24 43],'filled')
hold on
```

Appendix E. Python Class Method

A Aircraft Class Structure

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  4 18:53:46 2018

@author: laure
"""
import math as m
import numpy as np
from skaero.atmosphere import coesa

class Aircraft(object):

    def __init__(self, alist):
        self.name = alist[0]
        self.TSFCsl = alist[1] # Thrust specific fuel consumption (lbs/sec)
        self.LDmax = alist[2] # Max lift over drag (L/D)
        self.speedMax = alist[3] # Mach speed max when C_L^(1/2)/C_D is maximized
        self.fuelToCruise = alist[4] # Fuel used during climb to cruise
        self.distToCruise = alist[5] # Miles traveled during climb to cruise
        self.Combat = alist[6] #Fuel burned during combat
        self.climbFuel = alist[7] #Fuel burned on climb from combat
        self.fuelReserve = alist[8] # Percent of fuel in reserve
        self.timeReserve = alist[9] # Minutes of endurance in reserve
        self.dryWeight = alist[10] # Weight including payload (excluding fuel) (lbs)
```

```
25          self.maxTOW = alist[11] # Weight including payload and fuel (lbs
    )
            self.eDryWeight = alist[12] # Dry weight without payload/tanks (
    lbs)
27          self.eMaxTOW = alist[13] #Weight with fuel without tanks and
    payload (lbs)
            self.cruiseAltitude = alist[14] # Altitude of cruise in ft
29          self.loiterAltitude = alist[15] #Altitude of loiter in ft
            self.userRange = alist[16] # Miles desired
31
     def EnduranceTradeoffs(self,*args):
33          R = 286.9; "Gas constant"
            Altitude = self.cruiseAltitude/3.2808;
35          loiterAlt = self.loiterAltitude/3.2808;
            _, T1, _, _ = coesa.table(Altitude); "Use International Standard
     Atmosphere data"
37          _, T2, _, _ = coesa.table(loiterAlt)
            tempK = 518.7*5/9;
39          TSFC = self.TSFCsl*m.sqrt(T1/tempK)/3600; "TSFC at altitude (per
     hour)"
            TSFCloiter = self.TSFCsl*m.sqrt(T2/tempK)/3600
41          speedMax = self.speedMax*m.sqrt(T1*R*1.4)*3.2808; "Conversion to
     account for altitude (ft/s)"


43          "Fuel Reserve Conversion"
            percentFuel = (self.eMaxTOW - self.eDryWeight)*self.fuelReserve
    /100;
45          solInitialWeight = m.exp(self.timeReserve*(TSFC*60)/self.LDmax)*
    self.eDryWeight;
            percentFuel = (percentFuel + (solInitialWeight-self.eDryWeight))
    /(self.eMaxTOW-self.eDryWeight);
47
```

```
        initialReserve = 1-(self.maxTOW - self.fuelToCruise)/self.maxTOW
    ;
49      translateI = -(m.log((initialReserve*(self.maxTOW-self.dryWeight
    )+self.dryWeight)/self.maxTOW)*\
                        speedMax/5280*self.LDmax/TSFC+self.distToCruise);
51      translateE = (-(self.LDmax*speedMax/5280*\
                        m.log((self.eDryWeight + self.eMaxTOW*
    percentFuel-\
53                          self.eDryWeight*percentFuel)/(self.
    eMaxTOW)))/TSFC);
        intersect = (self.LDmax*speedMax*m.log(-(m.exp(-(TSFC*5280*\
55          translateI)/(2*self.LDmax*speedMax))*(m.exp((TSFC*5280*\
            translateE)/(2*self.LDmax*speedMax))*(4*self.eMaxTOW**2*self
    .maxTOW**2 +\
57          self.dryWeight**2*self.eMaxTOW**2*m.exp((TSFC*5280*(
    translateI + \
            translateE))/(self.LDmax*speedMax)) + self.eDryWeight**2*
    self.maxTOW**2*\
59          m.exp((TSFC*5280*(translateI + translateE))/(self.LDmax*\
            speedMax)) - 4*self.dryWeight*self.eMaxTOW**2*self.maxTOW -
    4*self.eDryWeight*\
61          self.eMaxTOW*self.maxTOW**2 + 4*self.dryWeight*self.
    eDryWeight*self.eMaxTOW*self.maxTOW - \
            2*self.dryWeight*self.eDryWeight*self.eMaxTOW*self.maxTOW*m.
    exp((TSFC*5280*\
63          (translateI + translateE))/(self.LDmax*speedMax)))**(1/2) -
    \
            self.dryWeight*self.eMaxTOW*m.exp((TSFC*5280*(translateI +
    2*\
65          translateE))/(2*self.LDmax*speedMax)) + self.eDryWeight*self
    .maxTOW*\
            m.exp((TSFC*5280*(translateI + 2*translateE))/(2*\
```

```
67              self.LDmax*speedMax))))/(2*self.eMaxTOW*(self.dryWeight -
    self.maxTOW))))/(TSFC*5280);

69 #         if self.userRange>intersect:
   #             endurance = 'Wanted range exceeds max range';
71 #             return endurance;

73          weightInitialFrac = (m.exp(-(self.userRange + translateI)*TSFC/(
    speedMax/5280*self.LDmax))*\
                                    self.maxTOW - self.dryWeight)/(self.maxTOW-
    self.dryWeight);
75          weightFinalFrac = (m.exp((self.userRange - translateE)*TSFC/(
    speedMax/5280*self.LDmax))*\
                                self.eMaxTOW - self.eDryWeight)/(self.eMaxTOW
    -self.eDryWeight);
77
          "Change from fraction of reserve to aircraft weight"
79          weightInitial = weightInitialFrac*(self.maxTOW - self.dryWeight)
     + \
          self.dryWeight-(self.dryWeight-self.eDryWeight)
81          weightFinal = weightFinalFrac*(self.eMaxTOW - self.eDryWeight) +
     self.eDryWeight;

83          minLostCombat = 1/(TSFCloiter*60)*self.LDmax*m.log((self.
    eDryWeight+self.Combat)/self.eDryWeight)
          minLostClimb = 1/(TSFCloiter*60)*self.LDmax*m.log((self.
    eDryWeight+self.climbFuel)/self.eDryWeight)
85          endurance = 1/(TSFCloiter*60)*self.LDmax*m.log((weightInitial)/
    weightFinal)\
          -minLostCombat - minLostClimb
87
          "Potentially make vectors outputted that make pareto front."
```

```
89
        xRange = np.arange(0,intersect+300,.1)
91      yRange = np.arange(1.5,intersect+300,.1)

93      IngressEQ = [(m.exp(-(i+translateI)*TSFC/(speedMax/5280*self.
    LDmax))*self.maxTOW-\
                    self.dryWeight)/(self.maxTOW-self.dryWeight) for i
    in xRange]
95      EgressEQ = [(m.exp((j-translateE)*TSFC/(speedMax/5280*self.LDmax
    ))*self.maxTOW-\
                     self.dryWeight)/(self.maxTOW-self.dryWeight) for j
     in yRange]
97
        return intersect,weightInitialFrac,weightFinalFrac,EgressEQ,
    IngressEQ,endurance;
```

B Radius and Distance Method

```
# -*- coding: utf-8 -*-
2 """
Created on Sat Nov  3 11:14:09 2018
4
@author: laure
6 """

8 import math as m
import pandas as pd
10 import numpy as np

12 def createCircleAroundWithRadius(lat, lon, radiusMiles):
    latArray = []
```

```
14      lonArray = []

16      for brng in range(0,360):
                lat2, lon2 = getLocation(lat,lon,brng,radiusMiles)
18              latArray.append(lat2)
                lonArray.append(lon2)
20
        return lonArray,latArray
22


24  def getLocation(lat1, lon1, brng, distanceMiles):
        lat1 = lat1 * m.pi/ 180.0
26      lon1 = lon1 * m.pi / 180.0
         #earth radius
28       #R = 6378.1Km
         #R = ~ 3959 Miles
30      R = 3959
        distanceMiles = distanceMiles/R
32
        brng = (brng / 90)* m.pi / 2
34
        lat2 = m.asin(m.sin(lat1) * m.cos(distanceMiles) \
36                      + m.cos(lat1) * m.sin(distanceMiles) * m.cos(brng))

38      lon2 = lon1 + m.atan2(m.sin(brng)*m.sin(distanceMiles)\
                              * m.cos(lat1),m.cos(distanceMiles)-m.sin(lat1)
    *m.sin(lat2))
40
        lon2 = 180.0 * lon2/ m.pi
42      lat2 = 180.0 * lat2/ m.pi


44
```

```
    return lat2, lon2

def mapInterestPts(file,startLat,startLong,distanceMiles):
    R = 3959
    locates = pd.read_excel(file)
    allDist = np.zeros([1,2])
    radii = []
    startLatRad = m.radians(startLat)
    count = 0
    for i in locates.index:
        long = locates["Longitude"][i]
        lat = locates["Latitude"][i]
        latRad = m.radians(lat)
        delLat = m.radians(startLat-lat)
        delLong = m.radians(startLong-long)

        a = (m.sin(delLat/2))**2+m.cos(latRad)*m.cos(startLatRad)*(m.sin
    (delLong/2))**2
        c = 2*m.atan2(m.sqrt(a),m.sqrt(1-a))
        D = R*c
        if D<=distanceMiles:
            coords = np.array([lat,long])
            allDist=np.vstack((allDist,coords))
            radii.append(D)
        if count == 0:
                count = 1
                allDist = np.delete(allDist,(0),axis = 0)

    return allDist,radii;
```

C Model Inputs Text Parser

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov  8 12:49:03 2018

@author: laure
"""

import pandas as pd
import math as m
from skaero.atmosphere import coesa


def inputs(file):
    inputslist = []
    data = pd.read_table(file, header = None, index_col = False, \
                         delim_whitespace =True , skiprows = [0,1],\
                         usecols = range(0,12))
    name = data.iloc[14,0]
    ## Find Aircraft Avgs and Inputs
    lines = tuple(open(file, 'r'))
    for i in range(len(lines)):
        current = lines[i]
        if current.find("0English Weight")==0:
            weightRef = i+1
            break
    weights = lines[weightRef]
    weights = weights.split()
    fullWeight = float(weights[2])
    dryWeight = fullWeight - float(weights[5])
    payload = float(weights[8])
```

```
31      tank = float(weights[15])
        intFuel = float(weights[20])
33      fullWeightWOPayload = dryWeight - payload- tank + intFuel
        dryWeightWOPayload = fullWeightWOPayload - intFuel
35
        for i in range(len(data)):
37          if data.iloc[i,0]=='TOTAL':
                avgRef1 = i + 2
39              break
        for i in range(avgRef1,len(data)):
41          if data.iloc[i,0]=='TOTAL':
                avgRef2 = i-3
43              break

45      avgTable = data.iloc[range(avgRef1,avgRef2),:]


47
        hold = []
49      for i in range(len(avgTable)):
            if fullWeight>=float(avgTable.iloc[i,0]):
51              hold.append(i)


53
        cruiseAlt = float(avgTable.iloc[hold[0],2])
55      avgMach = 0
        avgTSFC = 0
57      avgLD = 0

59      for i in hold:
            avgMach = avgMach + float(avgTable.iloc[i,4])
61          avgTSFC = avgTSFC + float(avgTable.iloc[i,7])
            avgLD = avgLD + float(avgTable.iloc[i,6])
```

```
63
        avgMach = avgMach/len(hold)
65      avgTSFC = avgTSFC/len(hold)
        avgLD = avgLD/len(hold)
67
#       R = 286.9 #"Gas constant"
69      Altitude = cruiseAlt/3.2808;
        h, T, p, rho = coesa.table(Altitude)
71      tempK = 518.7*5/9
        theta = T/tempK #"TSFC correction"
73
#       avgMach = avgMach/(m.sqrt(T*R*1.4)) #Sea level computation
75      avgTSFC = avgTSFC/(m.sqrt(theta)) #Sea level computation
        ## Find Fuel Burn Values
77
        for i in range(len(data)): ## PROBABLY WILL HAVE TO DO THIS
    DIFFERENT
79          if data.iloc[i,0]=='WARMUP':
                fuelRef1 = i
81              break
        for i in range(len(data)):
83          if data.iloc[i,0]=='NRRANG':
                fuelRef2 = i
85              break

87      fuelMini = data.iloc[range(fuelRef1,fuelRef2),:]

89      if fuelMini.iloc[3,0]=='DROP':
            reserveTime = data.iloc[fuelRef2-3,1]
91          reserveTime = float(reserveTime.split('min')[0])
            reserveFuel = data.iloc[fuelRef2-3,3]
93          reserveFuel = float(reserveFuel.split('%Fuel')[0])
```

```
        distToCruise = float(data.iloc[fuelRef1+1,7])*1.15
        fuelToCruise = float(data.iloc[fuelRef1+1,8])
        combatFuel = float(fuelMini.iloc[5,5])-float(fuelMini.iloc[6,6])
        climbFuel = float(fuelMini.iloc[7,5])-float(fuelMini.iloc[8,6])
        loiterAlt = float(fuelMini.iloc[2,7])
    else:
        reserveTime = data.iloc[fuelRef2-3,1]
        reserveTime = float(reserveTime.split('min')[0])
        reserveFuel = data.iloc[fuelRef2-3,3]
        reserveFuel = float(reserveFuel.split('%Fuel')[0])
        distToCruise = float(data.iloc[fuelRef1+1,7])*1.15
        fuelToCruise = float(data.iloc[fuelRef1+1,8])
        combatFuel = float(fuelMini.iloc[4,7])-float(fuelMini.iloc[3,6])
        climbFuel = float(fuelMini.iloc[6,7])-float(fuelMini.iloc[4,7])
        loiterAlt = float(fuelMini.iloc[3,7])

    userRange = 10 #This is a placeholder

    inputslist = [name, avgTSFC, avgLD, avgMach, fuelToCruise,
    distToCruise,\
                  combatFuel, climbFuel, reserveFuel, reserveTime,
    dryWeight, \
                  fullWeight, dryWeightWOPayload, fullWeightWOPayload, \
                  cruiseAlt, loiterAlt, userRange]

    return inputslist;
```

D Example Usage

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Thu Nov  8 13:11:36 2018

@author: laure
"""
#EXAMPLE USAGE
import model_inputs as mod
import AircraftClass as ac
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import RadiusCode as rc

samplefile = 'C:/Users/laure/Documents/Thesis/NASIC Thesis/Notes/
    scrubbeddata/loiterc1_000_output.txt'
interestPts = 'C:/Users/laure/Documents/Thesis/NASIC Thesis/Code/Python/
    Locations.xlsx'
inputslist = mod.inputs(samplefile)
F15C = ac.Aircraft(inputslist)
intersect,_,_,inEQ, outEQ, endurance = ac.Aircraft.EnduranceTradeoffs(
    F15C)

#%% Radius

Lat = 38.820485
Lon = -97.705588
distanceInMiles = intersect

#orthographic basemap, center on specified lat and lon
mp = Basemap(projection='ortho', lat_0 = Lat, lon_0 = Lon, resolution =
    'l')

mp.bluemarble()
mp.drawstates()
```

```
mp.drawcountries()
mp.drawcoastlines()

X,Y = rc.createCircleAroundWithRadius(Lat,Lon,distanceInMiles)

X,Y = mp(X,Y)
mp.plot(X,Y,marker=None,color='g',linewidth=1)

x,y = mp(Lon,Lat)
mp.plot(x,y  ,marker='D',color='g',markersize=2)

## Added points
dist,ranges = rc.mapInterestPts(interestPts, Lat, Lon, intersect)
A = dist[:,0];
B = dist[:,1];
A,B = mp(B,A)
mp.scatter(A,B  ,1,marker = 'D',color = 'r')

for i in range(len(ranges)):
    F15C.userRange = ranges[i]
    _,_,_,_, _, endurance = ac.Aircraft.EnduranceTradeoffs(F15C)
    endurance = str(round(endurance,2))
    plt.text(A[i]+100000,B[i],'%s min' %endurance,fontsize = 8,color = '
    y')



##%% PARETO FRONT ALTITUDES
#xValue = np.arange(20,intersect-20,20)
#pFront = np.zeros((8,len(xValue)))
#
#
```

```
#for j in range(0,8):
#    F15C.loiterAltitude = 5000+5000*j;
#    for i in xValue:
#        F15C.userRange = i
#        _,_,_,_,_,_,endure = ac.Aircraft.EnduranceTradeoffs(F15C)
#        idx = int((i-20)/20)
#        pFront[j,idx] = endure
#
#plt.figure()
#for i in range(0,8):
#    alts = 5000+5000*i
#    plt.plot(xValue,pFront[i,], label = 'Loiter Altitude = %d' % alts)
#    plt.pause(1)
#
#plt.xlabel(r'\textbf{Range} (mi)')
#plt.ylabel(r'\textbf{Loiter Time} (min)')
#plt.title(r"Pareto Front for Nonlinear Range vs. Time ", fontsize=16)
#plt.legend()
#plt.show()
```

Bibliography

1. E. Torenbeek, "Cruise Performance and Range Prediction Reconsidered," *Progress in Aerospace Sciences*, vol. 33, no. 96, pp. 285–321, 1997.

2. C. A. Snyder, "Range and Endurance Tradeoffs on Personal Rotorcraft Design," in *AHS 72nd Annual Forum*, pp. 1–8, 2016.

3. M. Cavcar, "Engineering Notes Bréguet Range Equation?," vol. 43, no. 5, pp. 1542–1544, 2006.

4. T. Vitte, "Optimization of the Range for Continuous Target Coverage," *Journal of Aircraft*, vol. 52, no. 3, pp. 896–902, 2015.

5. D. P. Raymer, "An Approximate Method Of Deriving Loiter Time From Range," *Journal of Aircraft*, vol. 41, no. 4, pp. 938–940, 2004.

6. M. Alighanbari, Y. Kuwata, and J. How, "Coordination and Control of Multiple UAVs with Timing Constraints and Loitering," in *Proceedings of the 2003 American Control Conference*, vol. 6, pp. 5311–5316, 2003.

7. C. J. Chuck and J. Donnelly, "The Compatibility of Potential Bioderived Fuels with Jet A-1 Aviation Kerosene," *Applied Energy*, vol. 118, pp. 83–91, 2014.

8. J. Jonas, "Jet Airplane Range Considerations," *Journal of the Aeronautical Sciences*, vol. 14, no. 1, pp. 124–128, 1947.

9. S. A. Brandt, *Introduction to Aeronautics: A Design Perspective*. Reston: American Institute of Aeronautics and Astronautics, Inc., 3rd ed., 2015.

10. V. Pareto, *Manual of Political Economy: A Variorum Translation and Critical Edition*. OUP Oxford, 2014.

11. R. T. Marler and J. S. Arora, "Survey of Multi-Objective Optimization Methods for Engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004.

12. G. Agrawal, C. Bloebaum, K. Lewis, K. Chugh, C.-H. Huang, and S. Parashar, "Intuitive Visualization of Pareto Frontier for Multiobjective Optimization in n-Dimensional Performance Space," *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, pp. 1–11, 2004.

13. P. Korhonen and J. Wallenius, *Visualization in the Multiple Objective Decision-Making Framework*. 2008.

14. R. Yechout, Thomas, *Introduction to Aircraft Flight Mechanics*. Blacksburg: American Institute of Aeronautics and Astronautics, Inc., 2nd *ed.*, 2014.

15. M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, *Linear Programming and Network Flows*. Hoboken: John Wiley & Sons, 4th ed., 2010.

16. V. K. Shetty, M. Sudit, and R. Nagi, "Priority-Based Assignment and Routing of a Fleet of Unmanned Combat Aerial Vehicles," *Computers and Operations Research*, vol. 35, no. 6, pp. 1813–1828, 2008.

17. C. Schumacher, P. Chandler, M. Pachter, and L. Pachter, "Constrained Optimization for UAV Task Assignment," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, pp. 1–14, 2004.

18. C. Taylor and O. D. Weck, "Coupled Vehicle Design and Network Flow Optimization for Air Transportation Systems," *Journal of Aircraft*, vol. 44, no. 5, pp. 1479–1486, 2007.

19. T. E. Kannon, S. G. Nurre, B. J. Lunday, and R. R. Hill, "The Aircraft Routing Problem with Refueling," *Optimization Letters*, vol. 9, no. 8, pp. 1609–1624, 2015.

20. A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig, "The SCIP Optimization Suite 6.0," ZIB-Report 18-26, Zuse Institute Berlin, 2018.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
21–03–2019	Master's Thesis	Jun 2018 — Mar 2019

4. TITLE AND SUBTITLE

Turbojet Range, Loiter, and Altitude Tradeoff Estimations in Efficient Modeling and Optimization Formulation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Bramblett, Lauren M, 2Lt

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT–ENS–MS–19–M–102

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Air and Space Intelligence Center
1940 Allbrook Drive
WPAFB OH 45433-7765

10. SPONSOR/MONITOR'S ACRONYM(S)

NASIC

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Identifying accurate tradeoffs between loiter time, range, and loiter altitude for jet aircraft are critical for mission planning, but current methods are cumbersome and time and labor intensive. This research introduces a new method for determining tradeoffs between loiter time, range, and loiter altitude for jet aircraft utilizing specific aircraft performance characteristics and the Bréguet range and endurance equations. Results from this new methodology are compared to the existing model used by the National Air and Space Intelligence Center for flight planning. This method is then extended and applied to several routing assignment problems, taking advantage of the incorporation of performance characteristics in their formulation. Example problems are generated using binary integer programming models and nonlinear mixed integer programming models.

15. SUBJECT TERMS

aircraf loiter estimation, range equations, aircraft routing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. L. E. Champagne, AFIT/ENS
U	U	U	UU	115	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4555; lance.champagne@afit.edu