3-2006

# Optimizing Mean Mission Duration for Multiple-Payload Satellites

John A. Flory


# OPTIMIZING MEAN MISSION DURATION FOR MULTIPLE-PAYLOAD SATELLITES

THESIS

John A. Flory, Captain, USAF

AFIT/GOR/ENS/06-08

**DEPARTMENT OF THE AIR FORCE**
**AIR UNIVERSITY**

**AIR FORCE INSTITUTE OF TECHNOLOGY**

**Wright-Patterson Air Force Base, Ohio**

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense or the United States Government.

AFIT/GOR/ENS/06-08

# OPTIMIZING MEAN MISSION DURATION FOR MULTIPLE-PAYLOAD SATELLITES

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

John A. Flory, B.S.
Captain, USAF

March 2006

AFIT/GOR/ENS/06-08

# OPTIMIZING MEAN MISSION DURATION FOR MULTIPLE-PAYLOAD SATELLITES

**John A. Flory, B.S.**

**Captain, USAF**

Approved:

| | |
|---|---|
| Dr. Jeffrey P. Kharoufeh<br>Thesis Advisor | Date |
| Dr. Stephen Baumert<br>Committee Member | Date |

# Abstract

This thesis addresses the problem of optimally selecting and specifying satellite payloads for inclusion on a satellite bus to be launched into a constellation. The objective is to select and specify payloads so that the total lifetime utility of the constellation is maximized. The satellite bus is limited by finite power, weight, volume, and cost constraints. This problem is modeled as a classical knapsack problem in one and multiple dimensions, and dynamic programming and binary integer programming formulations are provided to solve the problem. Due to the computational complexity of the problem, the solution techniques include exact methods as well as four heuristic procedures including a greedy heuristic, two norm-based heuristics, and a simulated annealing heuristic. The performance of the exact and heuristic approaches is evaluated on the basis of solution quality and computation time by solving a series of notional and randomly-generated problem instances. The numerical results indicate that, when an exact solution is required for a moderately-sized constellation, the integer programming formulation is most reliable in solving the problem to optimality. However, if the problem size is very large, and near-optimal solutions are acceptable, then the simulated annealing algorithm performs best among the heuristic procedures.

# Acknowledgements

I would like to thank several people for their assistance throughout this research effort. First, I would like to thank my advisor, Dr. Jeffrey Kharoufeh. His feedback and careful attention to the research were integral to its success and to my motivation throughout the entire thesis process. Also, I am indebted to my reader, Dr. Stephen Baumert, for his valuable comments. I would also like to thank Mr. Justin Comstock whose advice helped ensure the applicability of this research. Finally, I would like to thank Mr. Jack Brendmoen for providing some guidance on the notional data.

John A. Flory

# Table of Contents

# List of Figures

# List of Tables

Table Page

OPTIMIZING MEAN MISSION DURATION
FOR MULTIPLE-PAYLOAD SATELLITES

# 1. Introduction

## *1.1 Background*

The U.S. Department of Defense (DoD) spends approximately $18 billion dollars annually on the development, procurement, and operation of satellites and other space-based assets. According to a 2003 General Accounting Office report [9], many of these acquisition programs consistently suffer from cost overruns and delays. Individual satellites and satellite constellations constitute a sizable portion of these space-based systems. However, it is possible that the costs of the satellite acquisition process may be greatly reduced through the application of methodologies that more effectively allocate resources.

Satellites range in complexity from relatively simple $40-million GPS navigation satellites to complex and expensive half-billion dollar MILSTAR tactical communication satellites. Although the current satellite design trend is toward simple and inexpensive satellites due to mitigation of launch and component failures, prevailing circumstances still necessitate the design and use of expensive, complex, multipurpose satellites. Because of the nature of funding, organizations often receive large, irregular monetary allocations to acquire satellite systems, and in order to use funding more effectively, satellites are designed to have as many mission capabilities as possible. Furthermore, multiple satellites often work in concert forming a constellation. As satellites in the constellation begin to degrade and lose functionality, decisions about how to best replace them given limited financial and material resources must be made. Deciding which mission capabilities (i.e. payloads) to in-

clude on new satellites is paramount. Payloads must be selected according to the functions they perform and given both physical and design parameter specifications to ensure they are compatible with, and operate effectively on, the satellite bus.

One of the critical specifications of a satellite payload is its mean mission duration (MMD). MMD is, approximately, a measure of the duration mission planners can expect a payload to be functional. MMD is not a mean in the mathematical sense, nor does it denote how long a payload will actually function. It simply provides mission planners a reasonable expectation of the amount of time the payload will be useful. As shown in Figure 1.1, MMD is usually less than the design life of the payload. Increasing a payload's MMD is analogous to increasing its reliability. This is accomplished by adding redundant systems and components as well as using materials less susceptible to degradation. Such measures increase the payload's cost, weight, volume, and other requirements. These are resources for which a satellite bus has only a finite capacity.

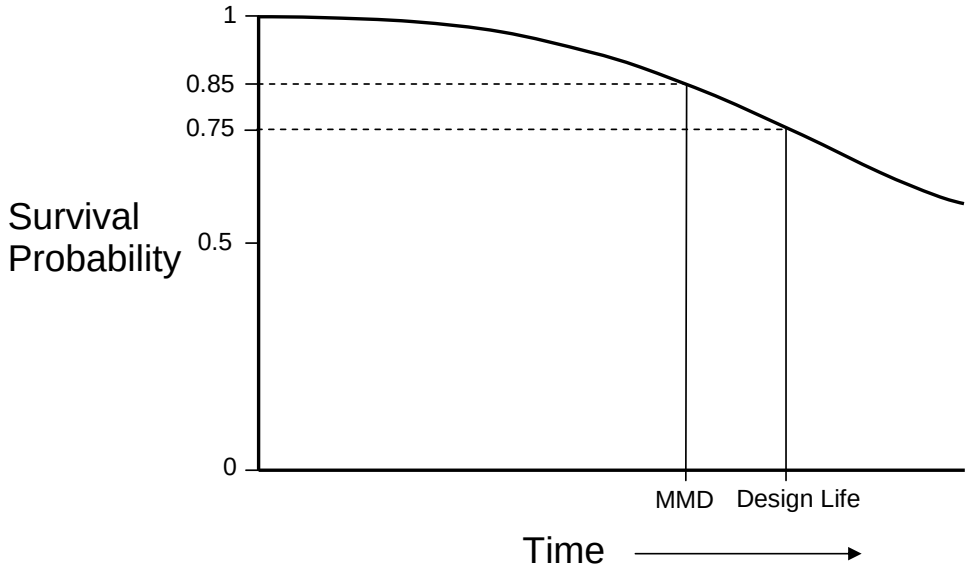


Figure 1.1 Graphical depiction of payload MMD.

Some methodologies currently exist for selecting and specifying satellite payloads. However, they are either very general and qualitative or so specific that their

application to general specification problems is very limited. This research seeks to develop an analytical methodology to select and specify satellite payloads that has sufficient generality for application to virtually any type of satellite constellation. Such a method would take quantifiable characteristics present in every type of satellite and payload and use that information in a methodical approach to make payload selection and specification decisions. The use of operations research techniques can greatly reduce the cost of a satellite constellation by allocating resources more effectively while simultaneously increasing its ability to achieve the overall mission objectives.

## *1.2 Problem Definition and Methodology*

Consider a satellite constellation observed at fixed times. Each satellite has a set of payloads associated with it, and at each observation, the functional status of each payload in the constellation is known. Also known are the MMDs and time in service of payloads initially in the constellation. The survival distributions of similar or dissimilar payloads are assumed to be statistically independent. At predetermined times, single-satellite buses will be equipped with payloads from a fixed set of all available payloads and launched into the constellation. Any selected payload must be assigned a MMD specification from a finite set of MMD specifications available to that payload. Each bus has finite power, cost, weight, and volume constraints. A payload's type denotes its specific function and is independent of its MMD specification. Moreover, each payload has a utility associated with it though utility, as it is used in this thesis, does not satisfy the strict definition of utility. Payload utility is assumed to be a function of the payload's relative importance, MMD specification, and the expected number of functional like payloads in the constellation. The importance of a payload is analogous to its value to the mission. It is only dependent on the payload's type and is not affected by the payload's MMD specification. Utility dependence on the number of like-type, functioning payloads allows a payload's util-

ity function to model diminishing marginal returns. It is assumed that a payload's power consumption is proportional to its utility. For a given payload type and MMD specification, discrete functions are assumed to exist that give the cost, weight, and volume resources consumed by the payload.

The objective is to select and specify payloads for the satellites being launched such that the total overall utility of the constellation is maximized. Payload utility can be static (constant over time) or dynamic (changing over time). Additionally, it can be deterministic (known with certainty) or stochastic (probabilistic). Enumerating each combination, utility can be characterized as one of the following: static and deterministic, dynamic and deterministic, static and stochastic, or dynamic and stochastic. The characterization of utility is integral to the form of the resulting mathematical model for payload selection. The problem of selecting and specifying satellite payloads is similar to a class of mathematical programming problems known as *knapsack problems*. Given a set of items, each having an associated profit and weight, the knapsack problem seeks to place them in a knapsack of finite weight-capacity such that the profit of included items is maximized. A payload selection model for a single satellite will be developed for each of the four characterizations of payload utility and shown to be a relaxation of a type of multidimensional knapsack problem. The dynamic and stochastic case most realistically describes the nature of actual payload utility and is extended to a multi-satellite case. A solution methodology will be developed to solve the multi-satellite problem. Exact solutions to knapsack problems generally require either dynamic programming or integer programming formulations. Therefore, in order to apply knapsack-based solution techniques to the multi-satellite payload selection problem, the multi-satellite problem will be formulated as both a dynamic program and an integer program.

A dynamic program can be solved (at great expense) by completely enumerating the state space, and such a method can be applied to the dynamic programming formulation of the payload selection problem. Integer programs are solved primarily

using branch-and-bound or branch-and-cut algorithms. Commercial IP solvers, like Xpress, apply these algorithms in concert with preprocessing and search heuristics to attain solutions more quickly. To provide a solution to the integer programming formulation, the Xpress solver will be used. Moreover, heuristic solution methods will be developed. An elementary heuristic for the one-dimensional knapsack problem is based on the profit-to-cost ratio of the items. This can be extended to solve the relaxed, multidimensional knapsack problem associated with the payload selection problem. Furthermore, the ability of a more advanced heuristic like simulated annealing to solve the payload selection problem will also be explored. Exact methods guarantee the optimality of the solution but can take considerable time. Heuristics are generally faster; however, they provide no guarantee of convergence to optimality. It is desirable to determine which method can best be applied to the problem of satellite payload selection and specification. The performance of the exact and heuristic solution methods will be compared by applying them to multiple problem instances using notional and randomly-generated payload and satellite data sets. Ultimately, one of the main objectives of this thesis is to provide recommendations for which technique performs best to provide optimal, or near-optimal, solutions in a reasonable amount of time.

### *1.3 Thesis Outline*

The remainder of the thesis is organized as follows. Chapter 2 presents a review of literature associated with payload selection methodologies as well as descriptions and main results for knapsack problems. In chapter 3, the assumptions and mathematical models for both the single-satellite and multi-satellite payload selection problems are discussed followed by dynamic programming and integer programming formulations of the multi-satellite case. Chapter 4 introduces exact and heuristic solution methods for solving the payload selection problem as well as prospective functional forms for the payload utility and survival functions. The solution methods

are subsequently applied to problem instances using both notional and randomly-generated data sets for payload resource requirements and satellite resource capacities. Finally, chapter 5 summarizes conclusions, unresolved issues, and possibilities for future extensions of the research.

# 2. Relevant Literature

This chapter reviews the literature pertaining to the selection and MMD specification of satellite payloads. The chapter begins with a review of current payload design and selection methodologies and proceeds to introduce a closely-related problem known as the *knapsack problem*. Because the knapsack problem provides a framework for payload selection, potential exact and heuristic solution methods for the knapsack problem are also discussed.

## *2.1 Payload Selection Strategies*

The literature on payload selection and specification is relatively sparse. Most literature concerning satellite payloads addresses the intricacies associated with their design and construction. Larson and Wertz [13] present a ten-step methodology for payload selection and specification based on a satellite's mission objectives. The process begins by translating the relatively general mission objectives of a satellite into payload objectives that denote the specific functions payloads are required to perform. Next, subject trades are conducted in which *subjects*, the specific objects with which payloads interact, are identified. Together the subjects and payload objectives allow generation of a *payload operations concept*. The payload operations concept identifies what is necessary to enable payloads to both perform their functions and communicate results.

Next, throughput and performance requirements of the payloads can be identified followed by the actual identification of candidate payloads. Because tasks can often be broken down or shared in many different ways, many possibilities exist for candidate payloads. The characteristics of each candidate payload must be estimated including its performance and resource requirements. Once the characteristics are estimated, a base-line of prospective payloads is selected. This selection is usually based on cost versus performance considerations. Next, both life-cycle cost and op-

erability need to be assessed. Although this step ultimately seeks to define payload utility as a function of cost, it is very complex and involves extensive contact with the satellite's prospective users and possible mission objective tradeoffs. Finally, the requirements imposed by the payloads on the satellite bus, ground operations, and mission control are enumerated. The process is concluded by thoroughly documenting all conclusions and may be repeated multiple times.

Although this selection methodology provides a general framework for payload selection and specification, it does not provide a specific quantitative method for decisions regarding the inclusion, exclusion, or specification of payloads. Bell [2] presents a methodology applied to secondary payloads on GPS satellites. A secondary nuclear detonation detection system (NDS) payload is included on every GPS satellite. However, NDS ground systems can only monitor twenty-four of, at that time, twenty-nine GPS satellites. Determining which satellites to monitor is equivalent to a payload selection problem in which payloads are to be selected for twenty-four satellites. The decision of which satellites to monitor was based on their individual value. A satellite is valued by its *contribution to coverage*; that is, its ability to both observe and detect a nuclear detonation. A satellite's real-time connectivity, optical sensor sensitivity, and orbital location were used to compute a contribution to coverage coefficient. The first step of the solution involves determining an initial coverage that ensures a relatively even distribution of satellites over each orbital plane. A heuristic is applied iteratively that replaces spare satellites in each plane. For each successive solution, the coverage of the entire constellation is calculated using a classified, GPS/MS simulation program.

## *2.2 Potential Optimal Solution Methodologies*

The satellite payload selection problem seeks to maximize the total overall utility, or some other benefit, subject to resource constraints. This is analogous to a class of problems known as *knapsack problems*. This section discusses both the

one- and multidimensional knapsack problems (KP/MKP) as well as exact solution methods for solving them.

### *2.2.1 Overview of Knapsack Problems*

The one-dimensional knapsack problem is one of the oldest and most widely-studied mathematical programming problems. Given a knapsack of finite capacity $b$, which items, each having weight $a_i$ and profit $c_i$, $i = 1, 2, \ldots, l$, should be placed in the knapsack such that total benefit is maximized and the capacity of the knapsack is not exceeded? Let $x_i = 1$ denote the inclusion of item $i$ and $x_i = 0$ denote the exclusion of item $i$, $i = 1, 2, \ldots, l$. Stated mathematically it is:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{l} c_i x_i \\ \text{Subject to} \quad & \sum_{i=1}^{l} a_i x_i \leq b \\ & x_i \in \{0, 1\} \\ & c_i,\, a_i,\, \text{b} \in \mathbb{Z}_+, \quad i = 1, 2, \ldots, l. \end{aligned}$$

The knapsack problem is NP-hard, and no algorithm is known that yields an optimal solution in polynomial-time; however, fully polynomial approximation algorithms do exist [17]. Many variations of the one-dimensional KP have been formulated to include the multiple-choice KP in which the item set $\Theta$ is partitioned into $k$ subsets, $\Theta_1, \Theta_2, \ldots, \Theta_k$, such that only one item in each subset may be selected. Another variation is the bounded KP in which each $x_i \in \mathbb{Z}_+$ and is bounded by $b_i \in \mathbb{Z}_+$ such that $x_i \leq b_i$. Despite their differences, both the multiple-choice and bounded KPs are NP-hard.

Many important results of the one-dimensional KP center around its LP-relaxation. Dantzig's [4] classical method to solve the LP-relaxation is to order

objects by decreasing profit-to-weight ratio, where

$$\frac{c_1}{a_1} > \frac{c_2}{a_2} > \ldots > \frac{c_l}{a_l}. \tag{2.1}$$

The solution proceeds with the insertion of item 1, and continues by inserting items into the knapsack by successively decreasing profit-to-weight ratios. Eventually, item $s$ will be reached which cannot fit into the knapsack. It can be shown [17] that the optimal solution to the LP-relaxation is given by $x_s = \bar{b}/a_s$, where $\bar{b} = b - \sum_{i=1}^{s-1} a_i$. Item $s$ is deemed the *break* or *critical* item. Simply including the first $s-1$ items in the knapsack often provides a good solution and forms the basis of a greedy, profit-to-cost ratio heuristic. Based on integrality of all profits and weights, Dantzig [4] showed an upper-bound of the knapsack solution is

$$\sum_{i=1}^{s-1} c_i + \left\lfloor \bar{b}\frac{c_s}{a_s} \right\rfloor. \tag{2.2}$$

This upper bound plays a role in branch-and-bound algorithms discussed later in this chapter.

Consider a generalization of the knapsack problem to the multidimensional case. The multidimensional knapsack problem assumes that the knapsack has multiple constraints akin to multiple dimensions. For a problem with $m$ constraints, the mathematical program for the MKP is:

$$\begin{aligned} \text{Maximize} \quad & \sum_{j=1}^{l} c_j x_j \\ \text{Subject to} \quad & \sum_{j=1}^{l} a_{ij} x_j \leq b_i, \quad i = 1, 2, \ldots, m \\ & x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, l, \end{aligned}$$

where $a_{ij}$ is the size of object $j$ on the $i$th dimension. The MKP is NP-hard; however, unlike the one-dimensional KP, no fully-polynomial approximation algorithms exist unless $P = NP$ [17]; however, polynomial approximations do exist [8]. Variations for the MKP also exist including the multi-choice, multidimensional knapsack problem (MCMKP). A formulation for the MCMKP is provided for this thesis based on an adaptation of Martello and Toth's [17] formulation for the one-dimensional, multi-choice knapsack problem. Define the following quantities:

$$\begin{aligned}
\Theta &\equiv \text{Item set} \\
\Theta_1, \Theta_2, \ldots, \Theta_k &\equiv \text{Partition of item set } \Theta \\
&\quad \text{where, } \bigcup_{i=1}^{k} \Theta_i = \Theta \text{ and } \Theta_i \cap \Theta_j = \emptyset,\ i \neq j \\
c_j &\equiv \text{Benefit of item } j,\ j = 1, 2, \ldots, l \\
a_{ij} &\equiv \text{Size of item } j \text{ on dimension } i \\
&\quad j = 1, 2, \ldots, l,\ i = 1, 2, \ldots, D \\
b_i &\equiv \text{Limit of knapsack dimension } i,\ i = 1, 2, \ldots, D
\end{aligned}$$

Let

$$x_j = \begin{cases} 1, & \text{if item } j \text{ is included} \\ 0, & \text{if item } j \text{ is not included} \end{cases}.$$

The MCMKP may be formulated as

$$\text{Maximize} \quad \sum_{j=1}^{l} c_j x_j \tag{2.3}$$

$$\text{Subject to} \quad \sum_{j=1}^{l} a_{ij} x_j \leq b_i, \quad i = 1, 2, \ldots, D \tag{2.4}$$

$$\sum_{j \in \Theta_i} x_j = 1, \quad i = 1, 2, \ldots, k \tag{2.5}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, l. \tag{2.6}$$

Many different algorithms have been developed to solve the one-dimensional and, to a lesser extent, the multidimensional knapsack problems exactly. These methods are primarily based on dynamic or integer programming formulations. Each is addressed in turn in the following subsections.

### *2.2.2 Dynamic Programming and KP/MKP*

Dynamic programming provides a method to analyze sequential decision processes [5]. It is based on a recursive relationship known as Bellman's functional equation. In general, dynamic programming requires a state space denoted by $\chi$, where $x \in \chi$ is a possible state of $\chi$, and a decision variable $\nu \in \pi$, where $\pi$ is the set of decisions at each stage. Assume $T$ decision stages and let the variable $t$ denote the index of the current decision stage. For a given stage and decision, a payoff function exists denoted by $a(\boldsymbol{x}, \nu, t)$; and the state $\boldsymbol{x'}$ at stage $t+1$ is given by the function $g(\boldsymbol{x}, \nu, t)$, where

$$\boldsymbol{x'} = g(\boldsymbol{x}, \nu, t). \tag{2.7}$$

Let $f(\boldsymbol{x}, t)$ be defined as the maximum (minimum) value of $f$ attainable given state $\boldsymbol{x}$ at stage $t$. Without loss of generality, assume the objective is to maximize $f(\boldsymbol{x_0}, t_0)$, where $\boldsymbol{x_0}$ and $t_0$ are the initial state and stage, respectively. Bellman's equation can be written either as either a forward or backward recursion. As a forward recursion, Bellman's equation is

$$f(\boldsymbol{x}, t) = \max_{\nu \in \pi} \{\boldsymbol{x} \in \chi : a(\boldsymbol{x}, \nu, t) + f[g(\boldsymbol{x}, \nu, t), t+1]\}. \tag{2.8}$$

A boundary condition is required to specify a terminal value of the recursion. Typically, this is

$$f(\boldsymbol{x}, T) = 0, \quad \forall\, \boldsymbol{x} \in \chi. \tag{2.9}$$

The pairs $(\boldsymbol{x}, t)$ form unique nodes in the dynamic programming state-space. The key principle in dynamic programming is the *principle of optimality*. Consider an optimal path from node $(\boldsymbol{x}_1, t_1)$ to node $(\boldsymbol{x}_4, t_4)$. For any two intermediate nodes $(\boldsymbol{x}_2, t_2)$ and $(\boldsymbol{x}_3, t_3)$ in this path, the principle of optimality states that the optimal path from $(\boldsymbol{x}_2, t_2)$ to $(\boldsymbol{x}_3, t_3)$ is the same as one contained in the optimal path from $(\boldsymbol{x}_1, t_1)$ to $(\boldsymbol{x}_4, t_4)$.

The primary advantage of dynamic programming is its generality as virtually no assumptions are imposed on the underlying functions or state-space. Unfortunately, the computational and storage requirements associated with dynamic programming are often massive. Dynamic programming-based techniques have been applied to both the one- and multidimensional knapsack problems. We first consider one-dimensional knapsack applications.

An elementary application of dynamic programming to the knapsack problem is found in Denardo [5] in which the decision $\nu$ is to include one of the $T$ items at each stage. In addition a *slack item 0* is assumed to exist, where $c_0 = 0$ and $a_0 = 1$. The state-space is scalar $x$, where $x$ is the remaining capacity of the knapsack. Therefore, feasible values of $x$ are $x \in \{0, 1, 2, \ldots, b\}$. Note that the stage index $t$ is unnecessary in this particular formulation and will be omitted. If item $i$ is included; that is, $\nu = i$, the payoff function $a(x, i) = c_i$, and $g(x, i) = x - w_i$, $i = 1, 2, \ldots, l$, $x \leq b$. The objective function $f(x)$ is the maximum value the knapsack can contain given remaining capacity $x$. A backward recursion is written as

$$f(x) = \max_i \{x > 0 : c_i + f(x - w_i)\}. \tag{2.10}$$

Since no items can be inserted into a knapsack with a remaining capacity of zero, the boundary condition is $f(0) = 0$, and the solution to the knapsack problem is $f(b)$. The most straightforward way to solve the dynamic program is by explicit enumeration in which each $f(x)$ is evaluated for all $T + 1$ possible item inclusions.

This means there are $T + 1$ branchings from initial state $b$. Furthermore, each resulting, nonzero state will potentially require successive branchings of size $T + 1$. Therefore, the complexity grows rapidly.

Although intuitive, explicit enumeration is extremely inefficient. Accelerated reaching is a method to eliminate state-space computations. Assume the items are sorted such that $c_1/a_1 \geq c_i/a_i$, $i = 2, 3, \ldots, l$. Let $m(x)$ be defined as the lowest index item that can be included in an optimal packing of knapsack size $x$, where

$$m(x) = \min \{i : f(i) = c_i + f(x - a_i)\}, \tag{2.11}$$

and $m(0) \equiv T$.

It can be shown that there exists $x^* = \max\{x : m(x) > 1\}$; that is, there is a state $x^*$ such that every knapsack of size $x^*$ or larger will include at least one item 1 in its optimal packing. It can also be shown that

$$x^* \leq (a_1 - 1)H = \bar{x^*}, \tag{2.12}$$

where $H = \max_j \{c_j\}$, $j = 1, 2, \ldots, l$. Therefore, when in a state $x > \bar{x^*}$ only the decision involving the insertion of item 1 needs to be evaluated. The solution is accelerated by continuing to insert item 1 until reaching a state $x < \bar{x^*}$. At this point, the remainder of the DP must be evaluated using standard methods. In practice, reaching is difficult to apply for large and non-integer knapsack problems. So more advanced methods have been developed to apply dynamic programming.

Horowitz and Sahni [11] studied a number partitioning problem that is a special case of the one-dimensional KP. They present a dynamic programming algorithm that, seeking to reduce storage requirements, splits the sorted multiset of weights into two multisets having a difference in cardinality of 0 or 1. Each smaller multiset has a profit set associated with it. A profit-maximizing dynamic recursion is applied to each multiset, and the optimal solution is attained by searching the end states of

each recursion and determining an optimal feasible pair. While this approach relies only on dynamic programming, often dynamic programming is used in concert with other algorithms.

Pisinger [20] augments a class of algorithm known as a *core algorithm* with dynamic programming. The *core* of a knapsack consists of those knapsack variables whose optimal values are different between the optimal IP solution and the LP-relaxation. Since solving the core is equivalent to solving the knapsack problem, the core is estimated. Once a core is estimated, Pisinger uses a dynamic, reaching recursion that moves bi-directionally from the break item alternatively inserting and deleting items. Each state is compared to an upper bound and fathomed accordingly. Martello et al. [15] refine Pisinger's algorithm using a core that can consist of non-consecutive items to better fill the knapsack. A similar dynamic program is applied; however, the state space is controlled actively by decreasing knapsack capacity, solving a surrogate relaxation, and deriving an improved lower bound through optimal item to state-space paring.

The complexity of the MKP generally precludes the effectiveness of exact dynamic programming. Bertsimas and Demir [3] mitigate dynamic programming state space requirements by applying approximate dynamic programming techniques to the MKP. Instead of applying the exact dynamic recursion formula, an approximation is used resulting in less storage and computation. Bertsimas and Demir apply three different approximations. The first is a base-heuristic that rounds the successive LP-relaxation solutions; the second involves a parametric approximation that samples the state space; and the third is a nonparametric approximation. The authors conclude that the base-heuristic approximation provides the best solutions.

Although dynamic programming provides a solution method, it does not take advantage of properties associated with the problem's integer programming formulation. In the next section, we consider IP-based methods.

### *2.3 IP Formulation Results for KP/MKP*

Generally, the most efficient algorithms for solving knapsack problems involve the IP formulation. This is because information can be exploited by relaxing the integer constraints in both linear programming (LP), Lagrangian, and surrogate relaxations. An algorithm that successively uses information from LP-relaxations is the branch-and-bound algorithm [25]. Consider the branch-and-bound algorithm applied to a knapsack problem. At node 0, the branch-and-bound algorithm initially solves the LP-relaxation of the KP. The objective value of the relaxation forms an upper bound for the optimal IP solution value. If all variables have integer values, the initial LP-relaxation is optimal; otherwise, a non-integer variable $x_i$ is chosen and two new LP-relaxations are created by imposing an integrality constraint $x_i \leq \lfloor x_i \rfloor$ and $x_i \geq \lceil x_i \rceil$ in each respective relaxation. This is branching, and it forms two additional nodes.

The algorithm chooses one of the relaxations and solves it. The solution must be checked for feasibility in the IP. If it is feasible, and one or more non-integer variables exist, integrality constraints are imposed on another variable and branching occurs again. If the solution is infeasible, further branching is unnecessary as all nodes generated by branching on an infeasible node will, themselves, be infeasible. This is called fathoming a node. Assuming feasibility is maintained, continued branching eventually yields a feasible, integer solution. The objective function value of this initial, integer solution forms a lower bound for the objective function value of the optimal, KP solution. Now another unexplored node is chosen and branching occurs on it. Because each node forms an upper bound for all successor nodes, any node whose objective function value is less than the value of the initial integer solution need not be explored further.

While branch-and-bound improves the lower bound by successively finding better integer solutions, the upper bound given by the initial LP-relaxation can be tightened through the use of cutting plane algorithms such as Gomory cuts. The

two are combined in a branch-and-cut algorithm which is similar to branch-and-bound except that cuts are generated at successive LP-relaxations and stored in a cut pool. The computations required to find an optimal solution may be reduced through application of preprocessing to eliminate variables and search heuristics to determine the best branching nodes.

Faster branch-and-bound algorithms for the one-dimensional KP have been developed by exploiting properties unique to the knapsack problem to attain tighter upper-bounds, faster computation, or better candidate solutions. One of the first advances was the upper-bound developed by Dantzig [14] and was given in equation (2.2).

Horowitz and Sahni [11] modified a previous branch-and-bound algorithm making it a depth-wise search, thereby, reducing computing storage requirements from exponential to linear. Based on the same profit-to-cost sorting of Dantzig's method [4], it consists of forward moves and backtracking. A forward move adds the largest possible set of ordered, non-included items into the current solution; and backtracking removes the last item added. Martello and Toth [16] present a similar algorithm; however, the forward move consists of two phases involving building and saving the current solution. This reduces the number of backtrackings. Of particular significance is their development of a tighter upper-bound than that developed by Dantzig. For particularly hard problems, Martello and Toth [14] develop minimum and maximum cardinality upper bounds computed by using a Lagrangian-relaxation. A branch-and-bound algorithm uses these upper bounds to fathom nodes.

Core algorithms have also been developed that incorporate branch-and-bound. Pisinger [19] initially determines the break item $b$, the core's center, through an efficient partial sorting algorithm. A greedy algorithm finds an initial solution, and a branch-and-bound enumerates items from the break item outward. Each time the branching grows outside the core, a nearby interval of items undergoes variable reduction before being added to the core.

In contrast to one-dimensional KPs, the MKP has very few specialized, IP-based procedures [8]. Soyster et al. [22] developed an algorithm suited for MKPs with many variables and few constraints. Based on partitioning the variables into integer and fractional sets, it successively finds the optimal integer solution of a series of LP-relaxations given a partial fractional solution. Convergence is not guaranteed, nor does it occur quickly for more than 100 variables. Shih [21] presents a modified branch-and-bound in which each variable is relabeled with $m$ subscripts denoting the rank of its profit-to-cost ratio for each of the $m$ constraints. The algorithm proceeds in a greedy manner by attempting to load each constraint to equality. The minimum of the $m$ resulting objective values is chosen as the upper bound, and branching is done on the constraint with the largest upper bound. The algorithm was tested on thirty, five-constraint knapsack problems with less than 100 variables and performed reasonably well.

Fréville and Pleateau [6] present a reduction method, i.e. an algorithm that attempts to reduce the problem size. Surrogate relaxation heuristics determine a lower bound, and tests are applied to eliminate variables and constraints. Fréville and Plateau [7] also develop an effective method to find the exact solution of the bidimensional knapsack problem. Using the surrogate dual they show that an exact optimal solution for the bidimensional case is equivalent to a search on the $[0, 1]$ interval. Furthermore, they modify a previous search method and prove optimality occurs after a finite number of iterations.

Using exact algorithms to solve knapsack problems is often of greater theoretical than practical interest. Even the most efficient algorithms can take an inordinate amount of time to solve larger problems. Heuristic algorithms provide an alternative and are discussed in the next section.

## *2.4 Potential Heuristic Solution Approaches*

Knapsack-type problems are ideal for heuristics because of the time and computation often involved in exact solutions. Heuristics are solution methods that find optimal or near-optimal solutions but are not guaranteed to converge to optimality. There are a number of general types of heuristic methods that can be applied to many different types of problems. These include simulated annealing, tabu search, genetic algorithms, etc. Heuristics begin with some initial point in the solution space and consist of two primary phases: performing a global search and performing a local search. The global search enables the heuristic to explore the solution space in order to avoid settling on the nearest local optimum. The local search allows the heuristic to narrow its scope and seek the best solution in a particular area of the solution space. In general, a heuristic is distinguished by the methods used to execute global and local searches as well as the rules used to govern transitions between each type of search. A widely applied heuristic that searches using probabilistic moves is simulated annealing.

The simulated annealing heuristic is designed to mimic the process of annealing metals. When a metal is annealed, it begins in a molten state. Although metal atoms prefer to be aligned, the high-temperature enables them to have random orientations. Slowly cooling the metal enables the atoms to gradually settle into a lower-energy aligned state; however, a rapid, cooling schedule can result in atoms becoming trapped in unaligned, high-energy states creating imperfections. The simulated annealing algorithm behaves similarly in that, at high temperatures, simulated annealing allows free movement throughout the solution space. It is effectively a random search as suboptimal solutions have a high probability of acceptance. At low temperatures, movement throughout the solution space is more restricted and simulated annealing accepts only superior solutions. Therefore, it effectively becomes a

hill-climbing algorithm [18]. Consider the optimization problem

$$\begin{aligned} \text{Maximize} \quad & f(\boldsymbol{x}) \\ \text{Subject to} \quad & \boldsymbol{x} \in \chi. \end{aligned}$$

A simulated annealing heuristic begins at a high initial *temperature*, denoted by $T_0$. A random solution $\boldsymbol{x}_0 \in \chi$ is chosen as a starting point, and a random move is made to $\boldsymbol{x}_1 \in \chi$, where $\boldsymbol{x}_1$ is an adjacent or neighboring solution of $\boldsymbol{x}_0$. Then the objective values $f(\boldsymbol{x}_0)$ and $f(\boldsymbol{x}_1)$ are compared. Define $\delta = f(\boldsymbol{x}_1) - f(\boldsymbol{x}_0)$. The neighboring solution can be accepted under two scenarios: (i) if $\delta \geq 0$, move to $\boldsymbol{x}_1$, and (ii) if $\delta < 0$, move to $\boldsymbol{x}_1$ if $q < e^{\frac{\delta}{T_0}}$, where $q \sim \mathrm{U}[0, 1]$.

At this point, additional solutions can be explored at a temperature of $T_0$, or the temperature can be lowered to $T_1$ according to some schedule. Let $N \geq 1$ be defined as the number of solutions explored at each fixed temperature. At each temperature decrease, the heuristic continues from its last feasible solution and explores $N - 1$ additional solutions. Simulated annealing terminates when the temperature decreases to a predetermined terminal temperature $T_f < T_0$ and $N$ solutions have been explored at $T_f$. Aside from selection of the parameters themselves, the variations of simulated annealing abound. Differences include temperature schedules, solution selection, termination conditions, and post-processing [18].

When using the simulated annealing heuristic, optimality is guaranteed only under very specific conditions. Hajek [10] provides a simulated annealing algorithm that, under certain conditions, is guaranteed to find a global minimum using temperature schedule $T_k = C/\ln(1 + k)$, where $C$ is the depth of the deepest local minimum. Although of theoretical interest, the temperature schedule is too slow to be practically useful. In general, the more closely a temperature schedule maintains thermal equilibrium throughout, that is, the probability distribution of state transitions remain close to their equilibrium distribution at a given temperature, the

higher the quality of the resulting solution. Triki et al. [23] provide a summary of effective cooling schedules. These include the geometric cooling schedule and cooling schedules that rely on past information called *adaptive cooling schedules*.

The selection of parameter values is integral to the performance of simulated annealing. Ben-Ameur [1] created an algorithm to select $T_f$ based a specified acceptance probability of a suboptimal move. The method requires sampling transitions in the state space and use of a recursive formula. More commonly, parameter selection is done experimentally. Wang and Wu [24] devised a six-step method to experimentally determine parameter methods subject to time constraints using response surface methodology. Although theoretical selection of parameters is not without merit, selection of parameters is often done experimentally by trying a variety of parameter settings and observing satisfactory, near-optimal solutions.

This chapter introduced current satellite payload selection methodologies. A general, ten-step process was discussed to determine payload specifications from a satellite's mission objective. Also reviewed was a more specific, quantitative methodology applied to an analogous problem of monitoring secondary payloads. The knapsack problem was introduced in both one- and multidimensional cases as an approach to formulate the payload selection problem. Analytical results for knapsack problems were presented as well as dynamic programming and integer programming based solution procedures. Finally, the simulated annealing heuristic was introduced as a possible solution method for the payload selection problem. In the next chapter, formal mathematical models are presented for the payload selection and specification problem. Four single-satellite models are presented and shown to be equivalent to a relaxation of the multi-choice, multidimensional knapsack problem. Then one of the single-satellite models is extended to the multi-satellite case. The chapter concludes by formulating the multi-satellite model as both a dynamic program and an integer program.

# 3. Mathematical Model Description

In this chapter, formal mathematical models are developed for the satellite payload selection and MMD specification problem. The problem assumptions and model characteristics are first described followed by four mathematical models describing a single-satellite problem. The final model is an extension of the single-satellite model to the multiple-satellite case. Finally, both dynamic programming and integer programming formulations are provided to solve the multiple-satellite model.

## *3.1 Model Assumptions and Definitions*

Assume a satellite constellation consisting of $S$ satellites is observed at fixed time epochs with equal inter-inspection time $\Delta$. Each satellite bus carries payloads, and upon inspection the binary status (functional/not functional) of each satellite's payloads is known with certainty. It is also assumed that at time $n = 0$ the mean mission duration (MMD) specifications of any payloads on satellites already in the constellation are known as well as their time in service. At $M$ predetermined epochs $n_1, n_2, \ldots, n_M$, single-satellite buses will be loaded with payloads selected from a fixed set of all payloads and launched into the constellation. All selected payloads must have a specified, nonzero MMD. It is assumed that satellite payloads immediately enter service upon launch. Associated with each satellite payload is a total lifetime utility. Utility in the context of this research does not correspond to the strict definition in utility theory. This will be fully discussed later in the chapter. The objective is to maximize the total lifetime utility of the constellation.

Each satellite bus has finite power, cost, weight, and volume constraints. It is assumed that any set of specified payloads requiring more resources than the satellite bus can provide is infeasible. A description of each finite resource is provided in what follows.

The satellite's power subsystem is responsible for generating and storing all electrical energy required by the payloads. For earth-orbiting satellites, photovoltaic cells convert solar radiation into electrical energy which is then stored in the battery. Energy storage is necessary because during certain periods of the satellite's orbit, the sun is eclipsed by the earth and the solar cells are unable to receive solar radiation. During eclipse, the battery is the sole provider of energy to the satellite. Therefore, the battery continuously undergoes charge and discharge cycles as the satellite passes in and out of eclipse. Because the duration and periodicity of eclipses depends on the satellite's orbit, the depth of battery discharge varies from an average of 15-20% for the short, frequent eclipse periods of low earth orbit to 50% for the long, infrequent eclipse periods of geostationary orbit. The battery cycle life (i.e. the number of times it can be charged and discharged) is dependent on the depth of discharge. A greater depth of discharge reduces the cycle life of the battery. Therefore, taking the expected depth of discharge into account, the total lifetime output of the battery can be estimated by multiplying the expected energy discharge per cycle by the cycle life. For all formulations, the power constraint is equivalent to the lifetime output of the satellite's battery and is denoted by $P$ (measured in Watt-Years).

An overall budget is allotted to a satellite mission that usually includes development, construction, launch, and support of the satellite. It is assumed that a known portion of this total allotment is assigned specifically to payload procurement. This quantity, denoted as $C$ (measured in \$), will constitute the total cost constraint.

The weight of a satellite is ultimately constrained by the capability of the selected launch vehicle to place the satellite in its required orbit. The weight of a mission-capable satellite at launch is defined as the *loaded weight* and includes the weights of both propellent and all satellite subsystems. Subtracting the propellent weight from the loaded weight yields the satellite *dry weight*. Payloads typically constitute between 15%-50% of satellite dry weight. In this research, it is assumed

that a launch vehicle is selected and the weights of both propellent and non-payload subsystems are estimated leaving a known allowance for total payload weight denoted by $W$ (measured in lb).

A satellite's volume is constrained by the volume of the launch vehicle's payload compartment. Additionally, the satellite must conform to the compartment's geometry. Because ensuring geometric feasibility drastically increases the problem complexity, it is assumed that a basic satellite bus design is used of known volume that ensures geometric feasibility after payload inclusion. The total payload volume constraint $V$ (measured in $ft^3$) is the difference between the volumes of the launch vehicle's payload compartment and the volume of the satellite bus.

Assume there exist $K$ different satellite payload types. Let $L_1, L_2, \ldots, L_K$ denote the lifetimes of each satellite payload type. It shall be assumed that these lifetimes are mutually statistically independent. For each payload type, assume there are $\xi$ distinct, nonzero MMD specifications. Denote the MMD of payload $i$ on satellite $j$ by $m_i^j$, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$. Let $\Theta_i$ denote the set of nonzero type-$i$ MMD specifications, where $|\Theta_i| = \xi$, $i = 1, 2, \ldots, K$. Additionally, each payload type $i$ can be assigned $m_i^j = 0$ which is equivalent to excluding payload type $i$ from satellite $j$.

A payload is selected if it is assigned a nonzero MMD specification. Therefore, the MMD specification serves as both a selection and specification variable. The values $m_i^j$, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$ will serve as the decision variables throughout. For satellite $j$, define $\boldsymbol{m}^j = [m_1^j, m_2^j, \ldots, m_K^j]$, $j = 1, 2, \ldots, M$, as the row vector of MMD specifications for all payload types on satellite $j$. Associated with each payload is a utility function. Utility in the classical sense is a relative scalar measure that compares the probabilistic outcomes of two consequences. Keeny and Raiffa [12] define utility by considering a set of consequences, $x_1, x_2, \ldots, x_n$, such that their preference order is $x_1 \prec x_2 \prec \ldots \prec x_n$. If $x_j \prec x_i$, then consequence $x_i$ is preferable to $x_j$. A numerical value scaling $\varphi_i$ is assigned to each $x_i$ such that

$\varphi_1 = 0$, $\varphi_n = 1$, and $\varphi_1 < \varphi_2 < \ldots < \varphi_n$, $i = 1, 2, \ldots, n$. Now consider an action $a$ that results in consequence $x_i$ with probabilities $p_i$. The utility $\bar{\varphi}$ of $a$ is calculated as

$$\bar{\varphi} \;=\; \sum_i p_i \varphi_i. \tag{3.1}$$

Although the term payload *utility* will be used throughout this thesis, it is not utility in the sense defined by equation (3.1). Instead, payload utility is analogous to a relative value scaling on $\mathbb{R}_+$. The term *utility* is invoked throughout this research instead of *value* for two reasons: the computation of payload utility is similar to the computation of utility in the classical sense, and payload utility is dependent on a payload's relative importance which is closely analogous to the value scaling used in the utility definition. Utility of a payload type $i$ is a function of three factors:

1. The relative importance of payload type $i$;
2. the MMD specification of payload type $i$;
3. the expected number of functional, type $i$ payloads in the constellation.

The relative importance of a payload is only dependent on its functional type. This can be thought of as a measure of its relative functional importance to the constellation's mission and is analogous to a payload's value. Define $\psi_i$ as the relative importance of payload type $i$, $i = 1, 2, \ldots, K$.

As a payload's MMD is increased, it is expected that the payload will operate longer. However, it is also possible that the shape of its utility curve will change as its construction is more robust. Thus, MMD is included in the utility function. The rationale for utility dependence on the number of functional, like-type payloads is to model diminishing marginal returns. For example, as the constellation contains increasing numbers of a payload type, further additions of that type may add only negligible utility to the constellation. It may eventually be optimal to add a payload

of another type, even if it is of lower importance. The random number of functional type-$i$ payloads at time $n$ is denoted by a random variable $Q_i(n)$ and its expectation by $E[Q_i(n)]$, $i = 1, 2, \ldots, K$, $n \geq 0$.

Utility can be either static (constant over time) or dynamic (varying over time). In the case of dynamic utility, it is assumed that utility is a discrete-time stochastic process changing values only at inspection times $i\Delta$, $i = 0, 1, \ldots$ . The static utility of payload type $i$ on satellite $j$ is denoted by $u_i^j(\psi_i, m_i^j, E[Q_i])$; and $u_i^j(\psi_i, m_i^j, E[Q_i(n)]; n)$ denotes the dynamic utility of payload type $i$ on satellite $j$ at epoch $n$, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$.

Whether the utility is static or dynamic, the total lifetime utility of a payload type $i$ on satellite $j$ is represented by $U_i^j(m_i^j)$, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$. For satellite $j$, define the vector

$$\boldsymbol{u}^j(\boldsymbol{m}^j) = [U_1^j(m_1^j), U_2^j(m_2^j), \ldots, U_K^j(m_K^j)], \quad j = 1, 2, \ldots, M. \tag{3.2}$$

The elements of $\boldsymbol{u}^j(\boldsymbol{m}^j)$ are the total lifetime utilities of payloads on satellite $j$ with MMD specification $\boldsymbol{m^j}$, $j = 1, 2, \ldots, M$. Each payload consumes power, cost, weight, and volume resources. Satellite payloads are, in general, custom-built items and are not mass produced. The nuances associated with different satellites preclude them having a standardized design. Estimates are typically given by engineers for a payload's resource requirements if designed to meet a particular set of MMD specifications. Therefore, these estimates will be modeled as discrete functions. Define the discrete functions $C_i(m_i^j)$, $W_i(m_i^j)$, and $V_i(m_i^j)$ that represent the cost, weight, and volume, respectively, of payload type $i$ with specification $m_i^j$, and since $m_i^j = 0$ corresponds to excluding payload $i$ on satellite $j$, assume that

$$C_i(0) = V_i(0) = W_i(0) = 0, \quad i = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, M. \tag{3.3}$$

It is assumed that the rate of a payload's power consumption is proportional to its utility and independent of the payload's MMD specification. Let $A_i$ be defined as the rate of power consumption of a new type-$i$ payload, $i = 1, 2, \ldots, K$. If utility is static, power consumption can be calculated using $A_i$, $i = 1, 2, \ldots, K$. To allow for dynamic utility, payload power must be calculated by multiplying utility by a scaling factor. Let $A_i^{'j}$ denote the power scaling factor of payload $i$ on satellite $j$ where,

$$A_i^{'j} = \frac{A_i}{u_i(\psi_i, m_i^j, 0; 0)}, \quad i = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, M. \tag{3.4}$$

The scaling factor is calculated by multiplying the initial rate of power consumption by $1/u_i(\psi_i^j, m_i^j, 0; 0)$, which is the utility of payload $i$ on satellite $j$ when it enters service and no utility dependence is present. The resulting constant maintains the direct proportionality of power consumption to utility when utility increases or decreases at future times; therefore, it is power required per unit of utility.

Total lifetime power consumption of payload type $i$ on satellite $j$ is defined by $R_i^j(m_i^j)$, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$. For any satellite $j$, define vector functions $\boldsymbol{p}^j(\boldsymbol{m}^j)$, $\boldsymbol{c}^j(\boldsymbol{m}^j)$, $\boldsymbol{w}^j(\boldsymbol{m}^j)$, and $\boldsymbol{v}^j(\boldsymbol{m}^j)$:

$$\begin{aligned}
\boldsymbol{p}^j(\boldsymbol{m}^j) &= [R_1^j(m_1^j), R_2^j(m_2^j), \ldots, R_K^j(m_K^j)], \quad j = 1, 2, \ldots, M, \\
\boldsymbol{c}^j(\boldsymbol{m}^j) &= [C_1(m_1^j), C_2(m_2^j), \ldots, C_K(m_K^j)], \quad j = 1, 2, \ldots, M, \\
\boldsymbol{w}^j(\boldsymbol{m}^j) &= [W_1(m_1^j), W_2(m_2^j), \ldots, W_K(m_K^j)], \quad j = 1, 2, \ldots, M, \\
\boldsymbol{v}^j(\boldsymbol{m}^j) &= [V_1(m_1^j), V_2(m_2^j), \ldots, V_K(m_K^j)], \quad j = 1, 2, \ldots, M.
\end{aligned}$$

### *3.2 Single-Satellite Model*

Utility can also be characterized as deterministic or stochastic. Any mathematical model formulation is affected by the underlying characterization of payload

utility. Interchanging static or dynamic and deterministic or stochastic leads to four possible types of payload utility:

1. Static and deterministic;
2. Dynamic and deterministic;
3. Static and stochastic;
4. Dynamic and stochastic.

In the next four sections, the payload selection and specification problem is formulated for a single-satellite bus to be launched into an empty constellation. A distinct mathematical programming formulation is constructed for each type of utility. For notational brevity, the satellite subscript $j$ will be suppressed, and the utility dependence variable $E[Q_i(n)]$ is omitted since no utility dependencies exist in this case.

### *3.2.1 Static and Deterministic Utility*

Consider the problem of payload selection and specification for a single-satellite bus to be launched into an empty constellation. Payload utility is assumed to be static and deterministic; therefore, it is constant over time and known with certainty. The total lifetime utility of payload $i$ is

$$U_i(m_i) = u(\psi_i, m_i)m_i, \quad i = 1, 2, \ldots, K, \tag{3.5}$$

and the total lifetime power consumption of payload $i$ is

$$R_i(m_i) = A_i u(\psi_i, m_i)m_i, \quad i = 1, 2, \ldots, K. \tag{3.6}$$

The mathematical programming formulation under static and deterministic utility is as follows:

$$\begin{aligned}
\text{Maximize} \quad & \boldsymbol{u}(\boldsymbol{m}) \cdot \mathbf{1} && && (3.7)\\
\text{Subject to} \quad & \boldsymbol{p}(\boldsymbol{m}) \cdot \mathbf{1} && \leq \quad P && (3.8)\\
& \boldsymbol{c}(\boldsymbol{m}) \cdot \mathbf{1} && \leq \quad C && (3.9)\\
& \boldsymbol{v}(\boldsymbol{m}) \cdot \mathbf{1} && \leq \quad V && (3.10)\\
& \boldsymbol{w}(\boldsymbol{m}) \cdot \mathbf{1} && \leq \quad W && (3.11)\\
& m_i && \in \quad \Theta_i \cup \{0\}, \quad i = 1, 2, \ldots, K && (3.12)
\end{aligned}$$

where $\mathbf{1}$ is a column vector of ones.

The payload selection and specification problem is a relaxation of the MCMKP of equations (2.3)-(2.6). The set of all items $\Theta = \Theta_1 \cup \Theta_2 \cup \ldots \cup \Theta_K$ consists of each unique payload type and nonzero MMD specification combination. The items are partitioned by payload type into subsets $\Theta_i$, $i = 1, 2, \ldots, K$. The problem considered in this thesis is a slight relaxation of the MCMKP because it is not required that one item of each type (or partition) be selected as in the MCMKP. Payloads can be given a MMD of zero so, *at most* one item of each type can be selected. Power, cost, weight, and volume represent multiple dimensions of the knapsack.

Static and deterministic utility results in a relatively straightforward formulation. Because utility is static and deterministic, the computation of total payload utility and power consumption require only one evaluation of the utility function along with two and three multiplications, respectively. Therefore, with the exception of a few minor calculations, the static and deterministic case of payload selection is virtually a relaxed MCMKP. Next dynamic and deterministic utility is considered.

### *3.2.2 Dynamic and Deterministic Utility*

In this section, the case of dynamic and deterministic payload utility is considered. Recall, it is assumed that utility is a discrete-time stochastic process. Thus, the utility of payload $i$ remains constant on all inter-inspection intervals. Therefore, the total lifetime utility of a payload is calculated by summing the total utility on each individual time interval over all time intervals spanned by the payload's MMD. Let $\kappa_i$ be defined as the number of time intervals spanned by payload $i$, where $\kappa_i = \lfloor m_i/\Delta \rfloor$, $i = 1, 2, \ldots, K$. Then, the total lifetime utility and power consumption of payload type $i$ are respectively approximated as

$$U_i(m_i) = \sum_{n=0}^{\kappa_i - 1} u_i(\psi_i, m_i; n)\Delta, \tag{3.13}$$

$$R_i(m_i) = \sum_{n=0}^{\kappa_i - 1} A_i' u_i(\psi_i, m_i; n)\Delta. \tag{3.14}$$

Substituting equations (3.13) and (3.14) into vectors $\boldsymbol{u}(\boldsymbol{m})$ and $\boldsymbol{p}(\boldsymbol{m})$ leads to the same formulation as equations (3.7)–(3.12). Therefore, the dynamic and deterministic utility formulation is also equivalent to a relaxation of the MCMKP. In the next section, static and stochastic utility is considered.

### *3.2.3 Static and Stochastic Utility*

A payload with static and stochastic utility will operate at a fixed-value of utility throughout its lifetime; however, the exact value it operates at is not known with certainty. Instead the possible values are described by a probability distribution. The expression for total lifetime utility of a payload is identical in form to that of the static, deterministic case:

$$U_i(m_i) = u_i(\psi_i, m_i)m_i, \quad i = 1, 2, \ldots, K. \tag{3.15}$$

However, because the utility function is stochastic, it is advantageous to consider the mathematical expectation of total payload lifetime utility:

$$\begin{aligned} E[U_i(m_i)] &= E[u_i(\psi_i, m_i)m_i] \\ &= E[u_i(\psi_i, m_i)]m_i, \quad i = 1, 2, \ldots, K. \end{aligned} \tag{3.16}$$

Similarly, the expected total lifetime power consumption of payload $i$ is given by:

$$E[R_i(m_i)] = A_i E[u_i(\psi_i, m_i)]m_i, \quad i = 1, 2, \ldots, K. \tag{3.17}$$

Now, define vector functions $\boldsymbol{u}_E(\boldsymbol{m})$ and $\boldsymbol{p}_E(\boldsymbol{m})$ as follows:

$$\boldsymbol{u}_E(\boldsymbol{m}) = [E[U_1(m_1)], E[U_2(m_2)], \ldots, E[U_K(m_K)]], \tag{3.18}$$

$$\boldsymbol{p}_E(\boldsymbol{m}) = [E[R_1(m_1)], E[R_2(m_2)], \ldots, E[R_K(m_K)]]. \tag{3.19}$$

The static and stochastic payload selection and specification problem is formulated by inserting vectors $\boldsymbol{u}_E$ and $\boldsymbol{p}_E$ into equations (3.7) and (3.8). The static and stochastic problem formulation is virtually identical to the static and deterministic formulation with the exception of the presence of the expectation operator. Therefore, it is also equivalent to the relaxed MCMKP. Finally, the dynamic and stochastic model is formulated.

### *3.2.4 Dynamic and Stochastic Utility*

Dynamic and stochastic utility changes in value over time; however, its value at a future time is not known with certainty and is described by a probability distribution. Because utility is assumed to be a discrete-time stochastic process, its expected value is constant over all inter-inspection intervals. Therefore, analogous to the derivation of the dynamic and deterministic case, total payload lifetime utility

and power consumption for payload type $i$ are

$$E[U_i(m_i)] = \sum_{n=0}^{\kappa_i - 1} E[u_i(\psi_i, m_i; n)]\Delta, \quad i = 1, 2, \ldots, K, \tag{3.20}$$

$$E[R_i(m_i)] = \sum_{n=0}^{\kappa_i - 1} A_i' E[u_i(\psi_i, m_i; n)]\Delta, \quad i = 1, 2, \ldots, K, \tag{3.21}$$

respectively, where the expected utility and power consumption on each time interval is summed over all intervals spanned by the payload MMD.

Substitution of (3.20) and (3.21) into vectors $\boldsymbol{u}_E(\boldsymbol{m})$ and $\boldsymbol{p}_E(\boldsymbol{m})$ leads to the identical formulation of equations (3.7)–(3.12). Like the other three cases, the dynamic and stochastic formulation is equivalent to a relaxed MCMKP.

The dynamic and stochastic formulation is of the most interest for the satellite payload selection and specification problem because it most accurately reflects the realistic behavior of satellite payloads in space. Payload utility changes over time and typically decreases as payload components fail or degrade in the space environment. Furthermore, utility at any future time may not be known with certainty. A payload may fail immediately and have zero utility upon entering service, or it may operate at near-maximum utility throughout its entire mean mission duration. So, because of the dynamic and stochastic nature of actual payload utility, the dynamic and stochastic model will be the central focus of the remainder of this thesis.

## *3.3 Multi-Satellite Extension*

Consider an extension of the dynamic and stochastic model to the full-constellation problem in which payloads for $M$ satellites are selected and specified. Recall that the payloads are launched sequentially at predetermined epochs into a pre-existing constellation of $S$ satellites. It is assumed that both the launch times of the pre-existing satellites and their payload MMDs are known. Let $\bar{m}_i^j$ be defined as the remaining MMD of payload type $i$ on preexisting satellite $j$ and $\bar{\kappa}_i^j = \lfloor \bar{m}_i^j / \Delta \rfloor$ be

defined as the number of time intervals spanned by the remaining MMD of payload type $i$ on preexisting satellite $j$, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, S$.

Therefore, $\bar{U}_i^j$, the expected total remaining utility of payload type $i$ on preexisting satellite $j$ is

$$E[\bar{U}_i^j(\bar{m}_i^j)] = \sum_{n=0}^{\bar{\kappa}_i^j - 1} E[u_i(\psi_i, m_i^j, E[Q_i(n)]; n)]\Delta, \tag{3.22}$$

for $i = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, S$.

Let $\bar{\boldsymbol{u}}_E^j(\bar{\boldsymbol{m}}^j)$ be a vector function of the expected total remaining utilities of all payloads on a preexisting satellite $j$ defined by

$$\bar{\boldsymbol{u}}_E^j(\bar{\boldsymbol{m}}^j) = [E[\bar{U}_1^j(\bar{m}_1^j)], E[\bar{U}_2^j(\bar{m}_2^j)], \ldots, E[\bar{U}_K^j(\bar{m}_K^j)]], \quad j = 1, 2, \ldots, S. \tag{3.23}$$

Likewise, for payloads on the $M$ satellites to be launched, the expected total utility and power consumption are

$$E[U_i^j(m_i^j)] = \sum_{n=0}^{\kappa_i^j - 1} E[u_i(\psi_i, m_i^j, E[Q_i(n')]; n)]\Delta, \tag{3.24}$$

$$E[R_i^j(m_i^j)] = \sum_{n=0}^{\kappa_i^j - 1} A_i^{'j} E[u_i(\psi_i, m_i^j, E[Q_i(n')]; n)]\Delta, \tag{3.25}$$

where, $n' = n_j + n$ adjusts the number of time intervals a payload has been in service to the actual time interval of the constellation.

For satellites $j = S+1, S+2, \ldots, S+M$, vectors $\boldsymbol{u}_E^j(\boldsymbol{m}^j)$ and $\boldsymbol{p}_E^j(\boldsymbol{m}^j)$ are

$$\boldsymbol{u}_E^j(\boldsymbol{m}^j) = [E[U_1^j(m_1^j)], E[U_2^j(m_2^j)], \ldots, E[U_K^j(m_K^j)]], \quad j = 1, 2, \ldots, M, \tag{3.26}$$

$$\boldsymbol{p}_E^j(\boldsymbol{m}^j) = [E[R_1^j(m_1^j)], E[R_2^j(m_2^j)], \ldots, E[R_K^j(m_K^j)]], \quad j = 1, 2, \ldots, M. \tag{3.27}$$

The specification of $M$ satellites results in $M$ power, cost, weight, and volume constraints denoted by $P_j$, $C_j$, $W_j$, and $V_j$ for satellites $j = S+1, S+2, \ldots, S+M$. The payload selection and specification problem of a satellite constellation with dynamic and stochastic utility is as follows:

$$\text{Maximize} \quad \sum_{j=1}^{S} \bar{\boldsymbol{u}}_E^j(\bar{\boldsymbol{m}}^j) \cdot \mathbf{1} \;+\; \sum_{j=S+1}^{S+M} \boldsymbol{u}_E^j(\boldsymbol{m}^j) \cdot \mathbf{1} \tag{3.28}$$

$$\text{Subject to} \quad \boldsymbol{p}_E^j(\boldsymbol{m}^j) \cdot \mathbf{1} \;\leq\; P_j, \quad j = 1, 2, \ldots, M \tag{3.29}$$

$$\boldsymbol{c}^j(\boldsymbol{m}^j) \cdot \mathbf{1} \;\leq\; C_j, \quad j = 1, 2, \ldots, M \tag{3.30}$$

$$\boldsymbol{v}^j(\boldsymbol{m}^j) \cdot \mathbf{1} \;\leq\; V_j, \quad j = 1, 2, \ldots, M \tag{3.31}$$

$$\boldsymbol{w}^j(\boldsymbol{m}^j) \cdot \mathbf{1} \;\leq\; W_j, \quad j = 1, 2, \ldots, M \tag{3.32}$$

$$m_i^j \;\in\; \Theta_i \cup \{0\}, \tag{3.33}$$

$$i = 1, 2, \ldots, K, \quad j = S+1, S+2, \ldots, S+M.$$

When the single-satellite case is extended to multiple-satellites, the effect of utility dependence must be considered. Recall that utility dependence affects both the total lifetime utility and power requirements of payloads. If payload utilities are assumed to be independent, then it is only necessary to solve the payload selection problem once for each distinct type of satellite and apply each optimal loading to all identical satellites. However, if payload utilities are dependent, the optimal loading of each successive satellite will depend on previous satellite loadings. It is tempting to conclude that an exact solution would consist of solving a succession of MCMKPs, one for each satellite, using the appropriate resource requirements for the payloads available to each respective satellite. Such a solution would maximize the utility of each individual satellite in sequence; however, it would not necessarily maximize the utility of the overall constellation. In maximizing the constellation's utility, it may be necessary to assign a preceding satellite a suboptimal loading such that a

subsequent satellite can be assigned a loading that results in an increased overall constellation utility.

Therefore, due to utility dependence, the payload selection problem cannot be solved as a series of knapsack problems. However, its similarity to knapsack-type problems suggests two potential solution methodologies. First, payload assignment can be thought of as a sequential decision process in which the decision to include a particular payload on a particular satellite is made at each stage. Such a process lends itself nicely to a dynamic programming formulation. Second, it is possible that binary variables can be used to define different payload assignments, and the problem can be formulated as an integer program. In the next section both approaches are presented.

## *3.4 Dynamic and Integer Programming Formulations*

In this section, the general constellation payload specification and selection problem is formulated using both dynamic and integer programming. Both formulations provide a basis for exact solutions to the payload selection problem.

### *3.4.1 Dynamic Programming Formulation*

First considered is a dynamic programming approach in which decisions are made in sequential stages. Such a method agrees with the intuitive way one might solve this problem. For example, one could begin by specifying payload 1 on satellite 1 and then specify payload 2 on satellite 1. Once all satellite 1 payloads are specified, payloads on satellite 2 can be specified. Continuing in this manner, one could eventually conclude by specifying payload $K$ on satellite $M$. If these specifications are made such that the total constellation utility is maximized, the resulting solution is equivalent to a dynamic programming solution.

More formally, a decision stage $v \in \{1, 2, \ldots, \ell\}$ is the specification of a unique satellite and payload type combination, where $\ell = KM$. Therefore, the dynamic programming formulation can be thought of as a problem in which $\ell$ payloads are specified.

Define $\boldsymbol{p}_1 = [P_1, P_2, \ldots, P_M]$, $\boldsymbol{c}_1 = [C_1, C_2, \ldots, C_M]$, $\boldsymbol{v}_1 = [V_1, V_2, \ldots, V_M]$, and $\boldsymbol{w}_1 = [W_1, W_2, \ldots, W_M]$ as vectors of the initial power, cost, weight and volume resources of satellites $1, 2, \ldots, M$. The stage variable is $v$, the number of remaining, unspecified payloads, $v = 1, 2, \ldots, \ell$. As payloads are added, satellite resources are consumed. Let $\boldsymbol{p}_v$, $\boldsymbol{c}_v$, $\boldsymbol{v}_v$, and $\boldsymbol{w}_v$ represent the vectors of remaining power, cost, weight and volume resources of satellites $1, 2, \ldots, M$ at stage $v$, $v = 1, 2, \ldots, \ell$. The objective function $f(v, \boldsymbol{p}_v, \boldsymbol{c}_v, \boldsymbol{v}_v, \boldsymbol{w}_v)$ is defined as the maximum utility achievable given $v$ unspecified payloads and $\boldsymbol{p}_v$, $\boldsymbol{c}_v$, $\boldsymbol{v}_v$, and $\boldsymbol{w}_v$ resources remaining on satellites $1, 2, \ldots, M$, $v = 1, 2, \ldots, \ell$. The payoff function is $U_v(m_v)$ defined as the expected total utility of payload $v$ with specification $m_v$, $v = 1, 2, \ldots, \ell$. The expected total power consumption of payload $v$ and specification $m_v$ is denoted by $R_v(m_v)$, and let $C_v(m_v)$, $V_v(m_v)$, and $W_v(m_v)$ represent the cost, volume, and weight, respectively, of payload $v$ with specification $m_v$, $v = 1, 2, \ldots, \ell$.

Specification of payload $v$ on satellite $j$ consumes power, cost, volume, and weight resources defined by vectors $\boldsymbol{p}_c, \boldsymbol{c}_c, \boldsymbol{w}_c$, and $\boldsymbol{v}_c$, respectively, where

$$\boldsymbol{p}_c = R_v(m_v)\boldsymbol{e}_j, \quad j = 1, 2, \ldots, M, \quad v = 1, 2, \ldots, \ell, \tag{3.34}$$

$$\boldsymbol{c}_c = C_v(m_v)\boldsymbol{e}_j, \quad j = 1, 2, \ldots, M, \quad v = 1, 2, \ldots, \ell, \tag{3.35}$$

$$\boldsymbol{w}_c = W_v(m_v)\boldsymbol{e}_j, \quad j = 1, 2, \ldots, M, \quad v = 1, 2, \ldots, \ell, \tag{3.36}$$

$$\boldsymbol{v}_c = V_v(m_v)\boldsymbol{e}_j, \quad j = 1, 2, \ldots, M, \quad v = 1, 2, \ldots, \ell, \tag{3.37}$$

and $\boldsymbol{e}_j$ is defined as the $j$th elementary vector, $j = 1, 2, \ldots, M$.

Letting $f^*$ represent the optimal cost-to-go function, a forward dynamic programming recursion is written as follows:

$$\begin{aligned} f^*(v, \boldsymbol{p}_v, \boldsymbol{c}_v, \boldsymbol{w}_v, \boldsymbol{v}_v) &= \max_{m_v}\{\ U_v(m_v) \\ &+ f^*(v-1, \boldsymbol{p}_{v-1} - \boldsymbol{p}_c, \boldsymbol{c}_{v-1} - \boldsymbol{c}_c, \boldsymbol{w}_{v-1} - \boldsymbol{w}_c, \boldsymbol{v}_{v-1} - \boldsymbol{v}_c)\}, \end{aligned} \tag{3.38}$$

for $v = 1, 2, \ldots, \ell$. After the last stage, there are no more payloads to select and further increases in utility are not possible. Likewise, if either power, cost, weight, or volume resources are depleted, the addition of payloads is impossible and utility cannot be increased. Therefore, the dynamic program has the following set of boundary conditions:

$$f^*(0, \boldsymbol{p}_0, \boldsymbol{c}_0, \boldsymbol{v}_0, \boldsymbol{w}_0) = 0, \quad \forall\ \boldsymbol{p}_0,\ \boldsymbol{c}_0,\ \boldsymbol{v}_0,\ \boldsymbol{w}_0 \geq \boldsymbol{0}, \tag{3.39}$$

$$f^*(v, \boldsymbol{0}, \boldsymbol{c}_v, \boldsymbol{v}_v, \boldsymbol{w}_v) = 0, \quad \forall\ v \geq 0, \quad \forall\ \boldsymbol{c}_v,\ \boldsymbol{v}_v,\ \boldsymbol{w}_v \geq \boldsymbol{0}, \tag{3.40}$$

$$f^*(v, \boldsymbol{p}_v, \boldsymbol{0}, \boldsymbol{v}_v, \boldsymbol{w}_v) = 0, \quad \forall\ v \geq 0, \quad \forall\ \boldsymbol{p}_v,\ \boldsymbol{v}_v,\ \boldsymbol{w}_v \geq \boldsymbol{0}, \tag{3.41}$$

$$f^*(v, \boldsymbol{p}_v, \boldsymbol{c}_v, \boldsymbol{0}, \boldsymbol{w}_v) = 0, \quad \forall\ v \geq 0, \quad \forall\ \boldsymbol{p}_v,\ \boldsymbol{c}_v,\ \boldsymbol{w}_v \geq \boldsymbol{0}, \tag{3.42}$$

$$f^*(v, \boldsymbol{p}_v, \boldsymbol{c}_v, \boldsymbol{v}_v, \boldsymbol{0}) = 0, \quad \forall\ v \geq 0, \quad \forall\ \boldsymbol{p}_v,\ \boldsymbol{c}_v, ,\ \boldsymbol{v}_v \geq \boldsymbol{0}. \tag{3.43}$$

The optimal solution to the problem is

$$f^*(\ell, \boldsymbol{p}_\ell, \boldsymbol{c}_\ell, \boldsymbol{w}_\ell, \boldsymbol{v}_\ell). \tag{3.44}$$

The state-space of the payload selection problem grows rapidly. In stage $v$, the payload selection and specification problem with $M$ satellites, $K$ payload types, and $\xi$ nonzero MMD specifications per payload has the following state-space size:

$$\text{States in stage } v = (\xi + 1)^v. \tag{3.45}$$

$$\text{States in final stage} = (\xi + 1)^\ell. \tag{3.46}$$

It is clear that for relatively small values of $M$, $K$, and $\xi$, the number of states can be massive. For example, when $M = 2$ satellites and $K = 5$ payloads, each with $\xi = 3$ possible nonzero MMD specifications, the final stage of the dynamic program has $(3 + 1)^{5 \cdot 2} = 4^{10} = 1,048,576$ states to consider. Figure 3.1 graphically depicts the dynamic programming approach for a selection and specification problem in which $\xi = 2$ nonzero MMD specifications are available for each payload.

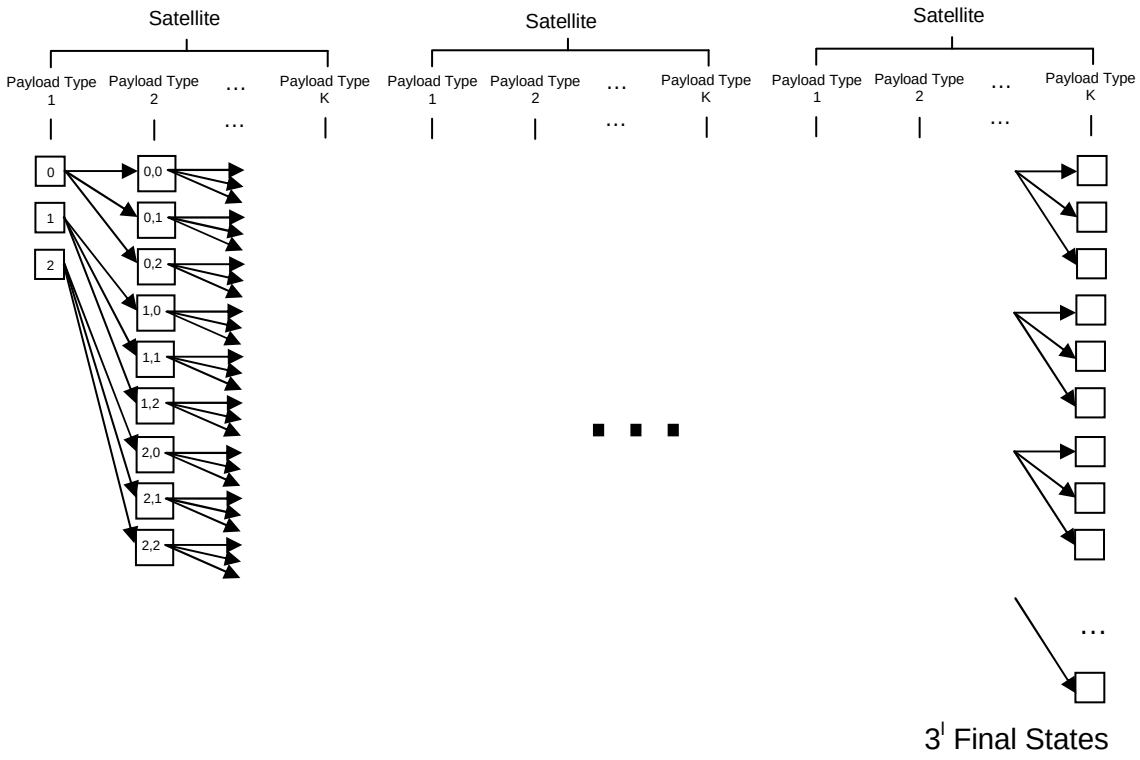


Figure 3.1 Graphical depiction of a dynamic programming solution ($\xi = 2$).

### *3.4.2 Integer Programming Formulation*

Now an integer programming formulation is considered. Let $m'_j$ represent a payload's MMD specification on satellite $j$, $j = 1, 2, \ldots, M$. Define $\boldsymbol{m}' = [m'_1, m'_2, \ldots, m'_M]$

as the vector of a payload's MMD specifications on satellites $1, 2, \ldots, M$. Let,

$$x^i_{\boldsymbol{m}'} = \begin{cases} 1, & \text{if payload type } i \text{ is given MMD } m'_1 \text{ on satellite 1,} \\ & m'_2 \text{ on satellite 2,...,} m'_M \text{ on satellite } M \\ 0, & \text{otherwise} \end{cases}$$

where, the total utility associated with each $x^i_{\boldsymbol{m}'}$ is the sum of the utilities of type $i$ payloads over all $M$ satellites at their respective MMDs. Let $u^i_{\boldsymbol{m}'}$ represent the total utility derived from the vector of specifications $\boldsymbol{m}'$, where

$$u^i_{\boldsymbol{m}} = \sum_{j=1}^{M} U^j_i(m^j_i), \quad i = 1, 2, \ldots, K. \tag{3.47}$$

Furthermore, for $x^i_{\boldsymbol{m}'}$, let the total consumption of power, cost, volume, and weight resources on satellite $j$ be denoted by $p^{ij}_{\boldsymbol{m}'}$, $c^{ij}_{\boldsymbol{m}'}$, $v^{ij}_{\boldsymbol{m}'}$, and $w^{ij}_{\boldsymbol{m}'}$, respectively:

$$p^{ij}_{\boldsymbol{m}'} = R^j_i(m'_j), \quad i = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, M, \tag{3.48}$$

$$c^{ij}_{\boldsymbol{m}'} = C_i(m'_j), \quad i = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, M, \tag{3.49}$$

$$v^{ij}_{\boldsymbol{m}'} = V_i(m'_j), \quad i = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, M, \tag{3.50}$$

$$w^{ij}_{\boldsymbol{m}'} = W_i(m'_j), \quad i = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, M. \tag{3.51}$$

Letting $P_j, C_j, V_j$, and $W_j$ represent the power, cost, volume, and weight resources associated with satellite $j$ the IP formulation is as follows:

$$\text{Maximize} \quad \sum_{i=1}^{K} \sum_{\boldsymbol{m'}} u^i_{\boldsymbol{m'}} x^i_{\boldsymbol{m'}} \tag{3.52}$$

$$\text{Subject to} \quad \sum_{i=1}^{K} p^{ij}_{\boldsymbol{m'}} x^i_{\boldsymbol{m'}} \leq P_j, \quad j = 1, 2, \ldots, M \tag{3.53}$$

$$\sum_{i=1}^{K} c^{ij}_{\boldsymbol{m'}} x^i_{\boldsymbol{m'}} \leq C_j, \quad j = 1, 2, \ldots, M \tag{3.54}$$

$$\sum_{i=1}^{K} v^{ij}_{\boldsymbol{m'}} x^i_{\boldsymbol{m'}} \leq V_j, \quad j = 1, 2, \ldots, M \tag{3.55}$$

$$\sum_{i=1}^{K} w^{ij}_{\boldsymbol{m'}} x^i_{\boldsymbol{m'}} \leq W_j, \quad j = 1, 2, \ldots, M \tag{3.56}$$

$$\sum_{\boldsymbol{m'}} x^i_{\boldsymbol{m'}} = 1, \quad i = 1, 2, \ldots, K \tag{3.57}$$

$$x^i_{\boldsymbol{m'}} \in \{0, 1\}, \quad i = 1, 2, \ldots, K. \tag{3.58}$$

For a $M$-satellite, $K$-payload type specification problem in which each payload has $\xi$ nonzero MMDs available, the number of binary variables and constraints is:

$$\text{Number of binary variables} = K(\xi + 1)^M. \tag{3.59}$$

$$\text{Number of constraints} = K + 4M. \tag{3.60}$$

Associated with each payload type is the special ordered set found in equation (3.57), and associated with each satellite are the four resource constraint equations (3.53)-(3.56). Although the number of binary variables grows quickly as the number of payload types and satellites in a problem increases, it is, in most cases, less than the number of states in the final stage of the problem's corresponding dynamic programming formulation. Also, unlike the dynamic programming state growth, the binary variables do not increase substantially when the number of payload types increases.

Superficially, the IP formulation appears more efficient than the DP formulation. However, the relatively small number of constraints to variables is of some concern. This is because variables increase the dimensionality of the solution space, whereas, constraints, while not affecting the dimensionality, narrow the scope of the solution space. The $M$ equality constraints represented by equation (3.57) allow only one of $K$ variables in each of the $M$ groups to be nonzero, thus, greatly reducing the solution space. The presence of these constraints will offset some of the inefficiencies associated with the formulation having relatively few constraints.

This chapter began with a list of assumptions for the payload selection problem including assumptions for both the payloads and satellite buses. The formulation of the problem is dependent on the four characterizations of payload utility: static and deterministic, dynamic and deterministic, static and stochastic, and dynamic and stochastic. Mathematical models were developed for the single-satellite problem using each characterization of utility. It was shown that each single-satellite model can be transformed into a relaxation of the multi-choice, multidimensional knapsack problem. Since the dynamic and stochastic model most closely represents the realistic nature of payload utility, it was extended to a multiple-satellite model. To enable the multiple-satellite model's solution, both dynamic and integer programming formulations were derived. Solution methods must now be applied to exactly solve these formulations. In the next chapter, such methods are discussed. Additionally, three prospective heuristics are introduced to achieve fast, near-optimal solutions. Finally, the performance of both the exact and heuristic methods is evaluated by solving payload selection problems using both notional and randomly-generated data.

# 4. Numerical Experimentation

In this chapter, the dynamic and stochastic problem is solved for notional and randomly-generated problem instances. In addition to using exact methods to solve the dynamic and integer programming formulations of the payload selection problem, four heuristics are devised. The first two are based on an extension of the classic, profit-to-weight ratio heuristic of the one-dimensional knapsack problem; the third is a greedy heuristic based on payload utility; and the fourth is a simulated annealing routine. Three small, medium, and large problems are solved using each method with a notional data set. Problem size is designated by the number of binary variables in the resulting IP formulation. Finally, the smallest problem instance is solved for payloads and satellite buses whose resource requirements and capacities have been randomly generated. Results include both problem solutions and a comparison of the solution techniques' performance.

## *4.1 Exact Solution Methods*

A dynamic program can be solved using a variety of methods many of which rely, to some degree, on enumeration. Pure enumeration is evaluation of all states in the state space. Although pure enumeration guarantees optimality, it is very costly in terms of computational requirements and storage. For cases with appropriate problem structure, approximate dynamic programming may be used wherein the cost-to-go function is approximated to avoid complete enumeration. The formulation for the payload selection problem assumes no underlying structure for the importance, MMD, and resource requirements of each payload. This requires all payloads to be examined or, more specifically, all stages in the DP to be evaluated. Therefore, it is not possible to find good approximations of the cost-to-go function because no inferences can be made on attributes of unspecified payloads. Additionally, this precludes the development of bounds to eliminate suboptimal states. For this reason,

a pure enumerative routine with pruning was used and carried out via Matlab. At each stage, infeasible states are eliminated resulting in reduced computational and storage requirements. The IP formulation in this research was solved using Dash Optimization's commercial solver Xpress. Xpress, like most industrial solvers, uses a variety of methods to solve an integer program. Preprocessing is applied to the initial problem to eliminate variables, and heuristics are used to determine the best starting point for the branch-and-cut algorithm, which uses both Gomory cuts and lifted cover inequalities at branch nodes. The algorithm is strengthened through the use of *strong branching* in which branching samples are ranked, and the best branch is chosen.

## *4.2 Greedy and Simulated Annealing Heuristics*

In this section, the heuristic methods are introduced. Two norm-based heuristics are presented which are based on extending the one-dimensional knapsack problem's profit-to-cost ratio heuristic to a multidimensional knapsack. A greedy heuristic based on payload utility is briefly discussed. Also outlined is an application of the simulated annealing routine to the payload selection problem.

### *4.2.1 Norm-Based Heuristics*

The motivation for the norm-based heuristics introduced in this section is the classic profit-to-weight ratio heuristic for the one-dimensional KP. As stated earlier, the payload selection problem for a single satellite is a relaxation of the MCMKP. Consider a single satellite $j$ and define the set of objects as all unique pairs $(i, m_i^j)$, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$; that is, every possible payload type and MMD combination. Payload *weights* are power, cost, weight, and volume requirements. A greedy solution to the multiple-satellite payload selection problem is to maximize the utility of each satellite successively, in other words, solving a succession of MCMKPs, each involving one of the $M$ satellite buses. As discussed earlier, such a greedy

solution does not guarantee optimality of the overall constellation's utility; however, all heuristic methods for the payload selection problem take this greedy approach.

For individual satellite buses, the one-dimensional profit-to-weight ratio heuristic cannot be directly applied to the payload selection problem. First, the satellite bus is a four-dimensional knapsack instead of a one-dimensional knapsack. One could choose either power, cost, weight, or volume and apply the greedy profit-to-weight ratio heuristic based on that resource. However, failing to take all resources into account leads to a heuristic of questionable effectiveness. The second reason the classic profit-to-weight heuristic cannot be used is the additional restriction that at most one payload of each type can be loaded onto a single satellite bus. Fortunately, this is easily resolved by tracking which payload types have already been loaded onto the satellite bus.

To resolve the multi-dimensionality, a scalar is required that provides some aggregate measure of a satellite's resource consumption. This is done using a similar motivation as the Toyota heuristic for MKPs. Dividing each payload's utility by an aggregated scalar provides an analogous profit-to-weight ratio. To motivate such a measure, first consider the $p$-norm of a vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$, where

$$\|\boldsymbol{x}\|_p = (x_1^p + x_2^p + \ldots + x_n^p)^{1/p} \tag{4.1}$$

and $p \geq 1$. The $p$-norm provides a generalized measure of distance, and when $p = 2$, it is the Euclidean distance. For each satellite, there are $K(\xi+1)$ payload type/MMD combinations. Define $\hat{U}_i^j$, $\hat{R}_i^j$, $\hat{C}_i^j$, $\hat{W}_i^j$, and $\hat{V}_i^j$ as the utility and power, cost, weight, and volume resources used by payload type/MMD combination $i$ on satellite $j$, respectively, $i = 1, 2, \ldots, K(\xi + 1)$, $j = 1, 2, \ldots, M$. For payload type/MMD combination $i$ on satellite $j$, define vector $\boldsymbol{\omega}_i^j = [\hat{R}_i^j/P_j, \hat{C}_i^j/C_j, \hat{V}_i^j/V_j, \hat{W}_i^j/W_j]$, $i = 1, 2, \ldots, K(\xi + 1)$, $j = 1, 2, \ldots, M$. That is, $\boldsymbol{\omega}_i^j$ is the vector of ratios of the payload's resource requirements to the bus's resource capacities, $i = 1, 2, \ldots, K(\xi + 1)$,

$j = 1, 2, \ldots, M$. Each resource has different units, so scaling them by their respective satellite capacities makes them comparable. Now, for some value of $p$, define constant $\omega_i^j$ as the aggregate payload weight, where

$$\omega_i^j = \|\boldsymbol{\omega}_i^j\|_p,\ i = 1, 2, \ldots, K(\xi + 1),\quad j = 1, 2, \ldots, M. \tag{4.2}$$

For satellite $j$, $\omega_i^j$ and $\hat{U}_i^j$ can be computed for all $K(\xi + 1)$ combinations of payload type and MMD specifications, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$. This forms the set of $K(\xi + 1)$ items. The combinations (or items) for each satellite can be ordered such that

$$\frac{\hat{U}_1^j}{\omega_1^j} \geq \frac{\hat{U}_2^j}{\omega_2^j} \geq \ldots \geq \frac{\hat{U}_{K(\xi+1)}^j}{\omega_{K(\xi+1)}^j}.$$

Payloads can then be inserted into the satellite greedily from largest to smallest ratios. A formal description of the heuristic is as follows:

For satellites $j = 1, 2, \ldots, M$.

1. Select value of $p \in \mathbb{R}_+$.
2. Compute $\hat{U}_i^j$ and $\omega_i^j$ for each payload/MMD combination $i$ on satellite $j$, $i$=$1, 2, \ldots, K(\xi + 1)$, $j = 1, 2, \ldots, M$.
3. Order combinations such that: $\frac{\hat{U}_1^j}{\omega_1^j} \geq \frac{\hat{U}_2^j}{\omega_2^j} \geq \ldots \geq \frac{\hat{U}_{K(\xi+1)}^j}{\omega_{K(\xi+1)}^j}$.
4. For combinations $i = 1, 2, \ldots, K(\xi + 1)$: If combination $i$ can be included, and the payload type in combination $i$ has not been previously loaded, include combination $i$. Otherwise, exclude combination $i$.

Extensive testing of the $p$-norm heuristic for different values of $p$ indicated that the heuristic was more accurate for $2 < p < \infty$. In particular a value of $p = 5$ is chosen for all runs deeming the heuristic the *5-norm* heuristic. It may seem counterintuitive that a non-Euclidean norm yields better results. However, the effect of a larger $p$-value in the norm is that larger vector components have more effect

on the norm's value. Since each component of $\boldsymbol{\omega}_i^j$ represents the fraction of total satellite resources a payload type/MMD combination consumes, the resource that consumes the largest amount of satellite capacity most strongly affects the norm's value. In other words, under the 5-norm, payloads receive a larger aggregate measure of resource consumption if they consume a larger portion of one resource than they would under the 2-norm. This is the motivation for the next heuristic.

Although $\boldsymbol{\omega}_i^j$ represents a measure of resources required by a payload type/MMD combination, it does not take into account the relative scarcity of each resource. For example, if a satellite bus is severely limited in its capacity to produce power, it is intuitive that payloads having a larger relative power requirement should have a greater effect on the norm's value. This is accomplished by weighting the norm elements by the relative scarcity of resources. Let $\zeta_P^j$, $\zeta_C^j$, $\zeta_W^j$, and $\zeta_V^j$ represent the relative scarcity of power, cost, weight, and volume resources on satellite $j$, respectively, where

$$\zeta_P^j = \frac{\sum_{i=1}^{K(\xi+1)} \hat{R}_i^j}{P_j}, \quad j = 1, 2, \ldots, M, \tag{4.3}$$

$$\zeta_C^j = \frac{\sum_{i=1}^{K(\xi+1)} \hat{C}_i^j}{C_j}, \quad j = 1, 2, \ldots, M, \tag{4.4}$$

$$\zeta_W^j = \frac{\sum_{i=1}^{K(\xi+1)} \hat{W}_i^j}{W_j}, \quad j = 1, 2, \ldots, M, \tag{4.5}$$

$$\zeta_V^j = \frac{\sum_{i=1}^{K(\xi+1)} \hat{V}_i^j}{V_j}, \quad j = 1, 2, \ldots, M. \tag{4.6}$$

For payload type/MMD combination $i$ on satellite $j$, define a weighted 2-norm $\omega_i^{'j}$, where

$$\omega_i^{'j} = \sqrt{\zeta_P^j(\hat{R}_i^j/P_j)^2 + \zeta_C^j(\hat{C}_i^j/C_j)^2 + \zeta_W^j(\hat{W}_i^j/W_j)^2 + \zeta_V^j(\hat{V}_i^j/V_j)^2} \tag{4.7}$$

for $i = 1, 2, \ldots, K(\xi + 1)$, $j = 1, 2, \ldots, M$. The scalar $\omega_i^{'j}$ forms the basis for the next heuristic using exactly the same routine as the first with the exception that $\omega_i^{'j}$ is used in place of $\omega_i^j$. This is called the *weighted norm* heuristic.

To determine the effectiveness of the norm-based heuristics, they will be compared to a strictly greedy heuristic that bases selection decisions on the total utility of payload type/MMD combinations. The greedy heuristic first computes and then sorts the utility associated with each payload type/MMD combination in descending order, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$. Starting with the largest utility combination, it attempts to include it, subject to feasibility. It then attempts to include each successive combination and finishes by attempting to include combination $K(\xi + 1)$. Because it does not take payload resource requirements into account, the greedy heuristic is expected to perform worse than the norm-based heuristics.

### *4.2.2 Simulated Annealing Heuristic*

Simulated annealing is applied to each satellite in the payload selection problem in a greedy manner. Like the two previous heuristics, it seeks to maximize the utility of each individual satellite successively, so each satellite is solved as an individual MCMKP. Without loss of generality, the loading of a single satellite is considered in the following discussion. Define $x_i$, $i = 1, 2, \ldots, K(\xi + 1)$ as:

$$x_i = \begin{cases} 1, & \text{if payload type/MMD combination } i \text{ is included} \\ 0, & \text{otherwise} \end{cases}.$$

A solution $\boldsymbol{x} = [x_1, x_2, \ldots, x_{K(\xi+1)}]$ is feasible if $\boldsymbol{x} \in \chi$, where $\chi$ is the set of all payload loadings such that:

1. At most one of each payload type $i$ is included, $i = 1, 2, \ldots, K$.
2. Total power, cost, weight, and volume requirements of included payloads does not exceed $P$, $C$, $W$, and $V$, respectively.

For any $\boldsymbol{x} \in \chi$, a neighboring solution is any $\boldsymbol{x'} \in \chi$ that differs from $\boldsymbol{x}$ by the inclusion or exclusion of one payload type/MMD combination, $i = 1, 2, \ldots, K$. Therefore, a move will consist of adding or removing a payload subject to feasibility. Recall that $T_0$ is the initial temperature, and $T_f$ is the terminal temperature, where $T_f < T_0$. A geometric cooling schedule is used with rate $r \in (0, 1)$. Under a geometric cooling schedule $T_{i+1} = r^i T_i$, $i = 1, 2, \ldots, I$, where $I = \min_j \{j : T_f \geq r^j T_0\}$. The geometric cooling schedule lends itself well to simulated annealing because initially temperature decreases rapidly from $T_0$ avoiding an inordinate amount of time being spent in the random-search phase. The cooling schedule then tapers off as it reaches its terminal temperature $T_f$ allowing full exploration of the local optimum. At each temperature level, $N$ solutions are explored. A detailed outline of the heuristic is presented in Figure 4.1.

Selection of parameters $T_0$, $T_f$, $r$, and $N$ is integral to the performance of simulated annealing. Although theoretical selection of parameters is not without merit, an experimental approach was taken to determine parameter settings. A variety of parameter settings was tested and those selected that gave satisfactory, near-optimal solutions. The simulated annealing parameters used are found in Table 4.1

Table 4.1 Simulated Annealing parameter settings.

| Parameter | $T_0$ | $T_f$ | $r$ | $N$ |
|---|---|---|---|---|
| Value | 500 | 0.01 | 0.05 | 5 |

**Select:** $N \geq 1,\ r \in (0,1),\ T_0, T_f \quad \text{s.t.} \quad 0 < T_0 < T_f$

**Set:** $T = T_0,\ n = 0$

$\boldsymbol{x_1} = \text{random}(\boldsymbol{x}) \in \chi$

**while** $T \geq T_f$ **do**

  **while** $n \leq N$ **do**

    $\boldsymbol{x}_2 = \text{random}(\boldsymbol{x}) \in \chi$

    $\delta = f(\boldsymbol{x}_2) - f(\boldsymbol{x}_1)$

    **if** $\delta > 0$ **then**

      $\boldsymbol{x}_1 = \boldsymbol{x}_2$

    **else if** $\text{random}(q) \sim \text{U}[0,1] < e^{\frac{\delta}{T}}$ **then**

      $\boldsymbol{x}_1 = \boldsymbol{x}_2$

    **end if**

    $n = n + 1$

  **end while**

  $T = rT$

**end while**

Figure 4.1 Simulated annealing algorithm.

In simulated annealing, it is desirable to set $T_0$ large enough to get some free movement throughout the solution space. However, a sufficiently high value may require a large number of iterations before the system *cools* to $T_f$. Therefore, because sufficient, free movement throughout the solution space cannot always be practically ensured, simulated annealing is often dependent on its starting solution. A way to mitigate this is to run multiple replications of the simulated annealing heuristic and select the best solution. A faster, less exact simulated annealing heuristic run over multiple replications is capable of providing better solutions than one lengthy, computationally-expensive routine.

Because simulated annealing, as it pertains to the payload specification problem, shows strong dependence on the initial solution, replications were performed. Loadings for each of the individual $M$ satellites were specified by a single, simulated annealing run. However, the overall $M$ satellite loading problem was run multiple times. As with the selection of parameters, the optimal number of replications was determined through experimentation to balance optimality with run-time. The number of replications chosen was 30.

In order to evaluate the performance of both the norm-based heuristics and simulated annealing, numerical experiments are required to compare the performance of the heuristics against exact solutions. The next three sections describe both the construction and execution of these experiments.

### *4.3 Description of Experiment*

A numerical experiment was conducted to compare the performance of the exact and heuristic solution methods using notional payload data. A method is deemed exact if, in theory, it is guaranteed to yield the optimal solution. Exact methods include the Matlab enumeration of the dynamic programming formulation and the Xpress solver solution to the integer programming formulation. Heuristic methods include simulated annealing, the two norm-based heuristics, and the greedy heuristic.

Table 4.2 Summary of solution methods.

| Exact | Heuristic |
|---|---|
| Pure Enumeration | Simulated Annealing |
| Xpress Solver | Weighted 2-Norm |
| | 5-Norm |
| | Greedy |

Performance is measured by two attributes: solution quality and run-time. Although exact methods are guaranteed to eventually yield optimality, it may not

be achievable in practice due to time or computational limits. The design of the numerical experiments consists of two major parts. The first is selection of an explicit functional form for payload utility. The second is generation of payload and satellite bus data as well as the problem instances. Each is next discussed in turn.

### *4.3.1 Payload Utility Function*

No assumptions are made regarding the functional form of payload utility in the derivation of the models for payload selection. Both the exact and the heuristic methods will work with any arbitrary function. Let indicator function $\phi_i^j(n) \in \{0,1\}$ denote the functional status (up or down) of payload $i$ on satellite $j$ at time $n$, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$. To motivate the functional form of utility, condition payload utility on $\phi_i^j(n)$:

$$E[u_i^j(\psi_i^j, m_i^j; n)] = \sum_{x \in \{0,1\}} E[u_i^j(\psi_i^j, m_i^j; n) | \phi_i^j(n) = x] P\{\phi_i^j(n) = x\}. \tag{4.8}$$

Both $E[u_i^j(\psi_i^j, m_i^j; n) | \phi_i^j(n) = x]$ and $P\{\phi_i^j(n) = x\}$ need to be characterized. Assume the conditional utility expression is characterized by

$$E[u_i^j(\psi_i^j, m_i^j; n) | \phi_i^j(n) = 1] = \frac{D_i^j(\psi_i^j, m_i^j; n)}{q_i(n)^\gamma}, \gamma \geq 0 \tag{4.9}$$

$$E[u_i^j(\psi_i^j, m_i^j; n) | \phi_i^j(n) = 0] = 0 \tag{4.10}$$

where, $D_i^j(\psi_i^j, m_i^j; n)$ is a deterministic function describing the utility decay of payload $i$ on satellite $j$ at epoch $n$, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$. In other words, it is assumed that the utility decline of a *functioning* payload is a deterministic function and that, when a payload fails, it has zero utility.

The constant $\gamma$ is a utility dependence parameter such that when $\gamma = 0$, there is no dependence, and $\gamma \geq 0$ implies dependence. Both the enumerative and heuristic methods require the calculation of a payload's total utility when it enters

a constellation, so total utility cannot be adjusted for future payloads added. It is possible to reformulate the IP in a way that allows the total utility of each payload to be adjusted for the addition of future payloads; however, this is not done to allow direct comparison of the IP method with the other methods. Therefore, if utility dependence exists, only the total utilities of subsequent payloads are diminished. A value of $\gamma = 0.5$ was used throughout the numerical experiments.

In selecting a functional form $D_i^j(\psi_i^j, m_i^j; n)$, $i = 1, 2, \ldots, K$, $j = 1, 2, \ldots, M$, it is assumed that the utility of a functional payload declines exponentially with time, and

$$D_i^j(\psi_i^j, m_i^j; n) = \psi_i^j e^{-\beta n/m_i^j}, \quad \beta > 0. \tag{4.11}$$

The constant $\beta$ is a *tuning parameter* to adjust the shape of the curve. It is assumed that payload survival distributions are independent; however, no additional assumptions are made regarding the distributions, and it is not necessary to have a memoryless distribution. However, because satellite payloads are largely comprised of electronic components, it is assumed that payload survival distributions are exponentially distributed. Denote the survival distribution by

$$P\{\phi_i^j(n) = 1\} = e^{-\alpha n/m_i^j}, \quad \alpha > 0 \tag{4.12}$$

where, $\alpha$ is another tuning parameter. The values $\beta = |\ln 0.5|$ and $\alpha = |\ln 0.9|$ were used throughout the experiment. These values imply that payloads will survive to their MMD with 90% probability, and upon reaching their MMD, they will operate at 50% of their original utility.

### *4.3.2 Notional Data and Problem Instances*

The motivation for using notional data as opposed to actual data is that a notional data set can be tailored to better test the solution methods. For example, if a specific set of actual data were used, it is possible that power is the only limiting

resource. In this case, the payload selection problem would effectively reduce to a one-dimensional knapsack problem; and the solution methods would be evaluated on a special, one-dimensional resource case as opposed to a more general, multi-dimensional case. The notional data set can be constructed to make the scarcity of the resources competitive thereby avoiding such a situation.

Although the data is notional, it is desirable to use realistic values for payload resource requirements. General ranges were obtained from a payload engineer for the power, cost, weight, and volume requirements of the satellite payloads and capacities of the satellite buses. In practice, a payload's cost estimate is directly proportional to its weight; therefore, costs and weights are generally correlated throughout the notional data. It is assumed that each payload can be assigned an MMD of 3 years, 6 years, or 10 years. The same set of MMD choices were used for each payload type to allow an easier comparison of results, but it is not necessary, in general, that each type of payload selects among the same set of MMD values. A payload type constructed to a higher MMD specification will likely include redundant systems or more robust materials requiring additional cost, weight, and volume resources. Therefore, in the notional data, as MMD is increased, the cost, weight, and volume resources consumed by payloads also increases. The notional data used in this research are presented in Table 4.3.

Ranges were also obtained for the resource capacities of the satellite buses. It is assumed throughout that all satellite buses have identical resource capacities, so each satellite's loading can be compared more easily. Satellite bus data is provided in Table 4.4

To fully explore each method's performance, a variety of problem instances are required, both simple and complex. Problem complexity for all methods is dependent on three variables: the number of satellite buses, payload types available, and MMD specifications available to each payload type. Through all runs, each payload type has three MMD specifications available to ensure realism. The number of satellites

Table 4.3 Notional satellite payload data.

| Type | MMD (Yr) | Importance | Power (W) | Cost ($100k) | Weight (lb) | Volume ($ft^3$) |
|---|---|---|---|---|---|---|
| 1 | 3 | 10.0 | 500 | 425 | 450 | 15.0 |
| | 6 | 10.0 | 500 | 460 | 475 | 17.0 |
| | 10 | 10.0 | 500 | 500 | 500 | 20.0 |
| 2 | 3 | 8.5 | 475 | 375 | 400 | 16.0 |
| | 6 | 8.5 | 475 | 405 | 415 | 18.0 |
| | 10 | 8.5 | 475 | 430 | 420 | 19.5 |
| 3 | 3 | 7.5 | 425 | 410 | 430 | 10.0 |
| | 6 | 7.5 | 425 | 460 | 480 | 13.0 |
| | 10 | 7.5 | 425 | 480 | 495 | 14.0 |
| 4 | 3 | 7.0 | 260 | 300 | 230 | 10.0 |
| | 6 | 7.0 | 260 | 350 | 280 | 13.0 |
| | 10 | 7.0 | 260 | 370 | 300 | 14.0 |
| 5 | 3 | 6.0 | 225 | 370 | 380 | 13.0 |
| | 6 | 6.0 | 225 | 400 | 390 | 15.5 |
| | 10 | 6.0 | 225 | 410 | 395 | 17.5 |
| 6 | 3 | 5.5 | 300 | 280 | 240 | 8.0 |
| | 6 | 5.5 | 300 | 320 | 290 | 9.0 |
| | 10 | 5.5 | 300 | 380 | 310 | 12.0 |
| 7 | 3 | 5.0 | 275 | 150 | 280 | 7.0 |
| | 6 | 5.0 | 275 | 190 | 350 | 9.5 |
| | 10 | 5.0 | 275 | 240 | 410 | 14.0 |
| 8 | 3 | 3.0 | 175 | 270 | 225 | 4.0 |
| | 6 | 3.0 | 175 | 310 | 360 | 5.5 |
| | 10 | 3.0 | 175 | 335 | 300 | 8.0 |

and payload types, however, are varied. Problem size is measured by the number of binary variables in the resulting IP formulation. This measure was chosen because, as seen later, the IP formulation was the only exact method that was practical for all problem sizes. Small problems have $\leq 100$ IP variables; medium problems have $101 - 10,000$ IP variables; and large problems have $\geq 10,000$ IP variables. Three instances each of small, medium, and large problems were generated to provide a good sampling over each range of problem size.

The preexistence of satellites in the constellation at time $n = 0$ does not affect problem complexity. In all problems, it is assumed there are no preexisting satel-

Table 4.4 Notional satellite bus data.

| Power Capacity (W-Yr) | Budget ($100k) | Weight (lb) | Volume ($ft^3$) |
|---|---|---|---|
| 10500 | 2500 | 2500 | 100 |

Table 4.5 Payload selection problem instances.

| Problem Size | Factor | Instance 1 | Instance 2 | Instance 3 |
|---|---|---|---|---|
| Small | No. Satellites | 1 | 2 | 2 |
| | No. Payload Types | 8 | 4 | 5 |
| | No. IP Variables | 32 | 64 | 80 |
| Medium | No. Satellites | 3 | 4 | 5 |
| | No. Payload Types | 4 | 6 | 6 |
| | No. IP Variables | 256 | 1536 | 8192 |
| Large | No. Satellites | 6 | 6 | 7 |
| | No. Payload Types | 6 | 8 | 8 |
| | No. IP Variables | 24576 | 32768 | 131072 |

lites; therefore, utility dependence is solely generated by the payloads selected by each method. This prevents confounding the source of utility dependency effects. Satellites are launched at epochs 2, 5, 7, 9, 10, 11, 13, and 15, and with such close launch intervals, utility dependence is expected to occur. All problem instances begin by selecting payloads for epoch 2, the first launch epoch. Additional satellites use each successive launch epoch. Matlab code was written to execute the dynamic programming enumeration and all heuristic methods. The IP formulation was solved using Xpress software. All computations were preformed on a Dell Precision Workstation with a 2.66 GHz Intel Xeon CPU and 2 GB memory. In the next section, numerical results are presented for each instance of small, medium, and large problem sizes.

## *4.4 Numerical Results and Summary*

This section contains results of the notional payload selection problem instances. The results are presented from the smallest problem instance to the largest. For each problem, the comparison of performance measures is presented followed by

the actual payload specifications provided by each method. Simulated annealing is abbreviated by *S.A.*, the greedy heuristic is abbreviated by *G.H.*, and the weighted norm heuristic is abbreviated by *W. Norm*. First presented are the small problems beginning with the 1 satellite, 8 payload type instance in Tables 4.6 and 4.7. Note that this is the only instance lacking utility dependence and is equivalent to solving a pure MCMKP.

Table 4.6 Maximum total utility (small instance 1).

| Method | Total Utility | % Diff. Optimal | Time (s) |
|---|---|---|---|
| Xpress Solver | 228.7 | 0.0 | 0.5 |
| Enumeration | 228.7 | 0.0 | 751.39 |
| Simulated Annealing | 219.1 | 4.2 | 22.25 |
| Greedy Heuristic | 211.0 | 7.7 | 0.06 |
| 5-Norm | 225.5 | 1.4 | 0.09 |
| Weighted Norm | 209.6 | 8.4 | 0.12 |

Table 4.7 Payload specifications (small instance 1).

| Method | Xpress | Enum. | S.A. | G.H. | 5-Norm | W. Norm |
|---|---|---|---|---|---|---|
| Satellite No. | 1 | 1 | 1 | 1 | 1 | 1 |
| Payload 1 MMD | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 0 | 0 | 0 | 10 | 0 | 10 |
| Payload 3 MMD | 0 | 0 | 3 | 0 | 0 | 0 |
| Payload 4 MMD | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 5 MMD | 10 | 10 | 10 | 3 | 10 | 0 |
| Payload 6 MMD | 3 | 3 | 0 | 0 | 10 | 3 |
| Payload 7 MMD | 10 | 10 | 0 | 0 | 0 | 0 |
| Payload 8 MMD | 0 | 0 | 3 | 0 | 3 | 0 |

Although this is a simple instance, the enumerative routine took a comparatively inordinate amount of time to find an exact solution when compared to the IP-based, Xpress solver. In both time and accuracy, the 5-norm performed better overall than the other heuristics coming within 5% of the optimal solution. Now consider the 2-satellite, 4-payload type result in Tables 4.8 and 4.9.

Table 4.8 Maximum total utility (small instance 2).

| Method | Total Utility | % Diff. Optimal | Time (s) |
|---|---|---|---|
| Xpress Solver | 414.1 | 0.0 | 0.5 |
| Enumeration | 414.1 | 0.0 | 1951.44 |
| Simulated Annealing | 414.1 | 0.0 | 22.00 |
| Greedy Heuristic | 385.6 | 6.9 | 0.06 |
| 5-Norm | 382.0 | 7.8 | 0.09 |
| Weighted Norm | 381.0 | 8.0 | 0.12 |

Table 4.9 Payload Specifications (small instance 2).

| Method | Xpress | | Enum. | | S.A. | | G.H. | | 5-Norm | | W. Norm | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |
| Payload 1 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 6 | 10 | 6 | 10 | 6 | 10 | 10 | 10 | 10 | 6 | 10 | 6 |
| Payload 3 MMD | 6 | 10 | 6 | 10 | 6 | 10 | 0 | 10 | 0 | 10 | 0 | 10 |
| Payload 4 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 6 | 10 | 10 | 10 | 10 |

Despite a doubling of IP variables from 32 to 64, the Xpress solver found an optimal solution in the same amount of time as the previous instance. However, the solution time of the enumerative method nearly doubled. Simulated annealing also found the optimal solution in a modest amount of time. The two norm-based heuristics performed worse than the greedy heuristic. Presented next are results for the 2 satellite, 5 payload type problem in Tables 4.10 and 4.11.

Table 4.10 Maximum total utility (small instance 3).

| Method | Total Utility | % Diff. Optimal | Time (s) |
|---|---|---|---|
| Xpress Solver | 437.6 | 0.0 | 0.7 |
| Enumeration* | - | - | >288,0000 |
| Simulated Annealing | 435.1 | 0.6 | 20.76 |
| Greedy Heuristic | 405.2 | 7.4 | 0.07 |
| 5-Norm | 430.1 | 1.7 | 0.14 |
| Weighted Norm | 430.1 | 1.7 | 0.11 |

For this instance and all subsequent instances, complexity of the enumeration precluded it from finding a solution in a reasonable amount of time. Meanwhile, the

Table 4.11 Payload specifications (small instance 3).

| Method | Xpress | | Enum. | | S.A. | | G.H. | | 5-Norm | | W. Norm | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |
| Payload 1 MMD | 10 | 10 | - | - | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 3 | 10 | - | - | 3 | 10 | 10 | 10 | 6 | 10 | 10 | 3 |
| Payload 3 MMD | 3 | 3 | - | - | 6 | 3 | 0 | 10 | 0 | 3 | 0 | 10 |
| Payload 4 MMD | 10 | 10 | - | - | 6 | 10 | 10 | 0 | 10 | 10 | 10 | 10 |
| Payload 5 MMD | 10 | 10 | - | - | 10 | 10 | 3 | 6 | 10 | 10 | 3 | 10 |

Xpress solver took only 0.2 s longer to find the optimal solution. The performance of the two norm-based heuristics was comparable, and they were closer to optimal than the greedy heuristic. Of the heuristics, simulated annealing found solutions that were closest to the optimal solution. Presented next is the smallest, medium-size problem having 3 satellites and 4 payload types in Tables 4.12-4.14.

Table 4.12 Maximum total utility (medium instance 1).

| Method | Total Utility | % Diff. Optimal | Time (s) |
|---|---|---|---|
| Xpress Solver | 590.0 | 0.0 | 0.4 |
| Enumeration | - | - | - |
| Simulated Annealing | 568.6 | 3.6 | 26.77 |
| Greedy Heuristic | 563.8 | 4.4 | 0.06 |
| 5-Norm | 561.0 | 4.9 | 0.09 |
| Weighted Norm | 561.0 | 4.9 | 0.13 |

Table 4.13 Payload specifications (medium instance 1).

| Method | Xpress | | | Enum. | | | S.A. | | |
|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 |
| Payload 1 MMD | 10 | 10 | 10 | - | - | - | 6 | 10 | 10 |
| Payload 2 MMD | 6 | 10 | 10 | - | - | - | 6 | 6 | 10 |
| Payload 3 MMD | 6 | 10 | 10 | - | - | - | 10 | 10 | 10 |
| Payload 4 MMD | 10 | 10 | 10 | - | - | - | 10 | 10 | 10 |

Although the underlying IP has grown to 256 variables, Xpress still obtained a solution quickly. Simulated annealing outperformed the other heuristics by a slight margin. The effects of utility dependence are becoming apparent. In all solutions,

Table 4.14 Payload specifications (medium instance 1).

| Method | G.H. | | | 5-Norm | | | W. Norm | | |
|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 |
| Payload 1 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 10 | 10 | 10 | 10 | 6 | 10 | 10 | 6 | 10 |
| Payload 3 MMD | 0 | 10 | 10 | 0 | 10 | 10 | 0 | 10 | 10 |
| Payload 4 MMD | 10 | 6 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |

satellite 3 is assigned all four payload types with a MMD specification of 10. Note that all payloads on satellite 3 will experience decreased utilities due to the number of payloads already present in the constellation. Power consumption is proportional to utility, so each payload requires less power; hence, both the number of payloads and their MMD specifications can be increased without violating the power constraint. Next, consider the 4 satellite, 6 payload type problem in Tables 4.15-4.17.

Table 4.15 Maximum total utility (medium instance 2).

| Method | Total Utility | % Diff. Optimal | Time (s) |
|---|---|---|---|
| Xpress Solver | 874.5 | 0.0 | 85.0 |
| Enumeration | - | - | - |
| Simulated Annealing | 850.1 | 2.8 | 47.90 |
| Greedy Heuristic | 815.7 | 6.7 | 0.10 |
| 5-Norm | 835.4 | 4.5 | 0.11 |
| Weighted Norm | 822.7 | 5.9 | 0.15 |

Table 4.16 Payload specifications (medium instance 2).

| Method | Xpress | | | | Enum. | | | | S.A. | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| Payload 1 MMD | 6 | 10 | 10 | 10 | - | - | - | - | 10 | 3 | 3 | 10 |
| Payload 2 MMD | 6 | 3 | 10 | 10 | - | - | - | - | 3 | 10 | 10 | 10 |
| Payload 3 MMD | 3 | 6 | 0 | 10 | - | - | - | - | 0 | 6 | 10 | 3 |
| Payload 4 MMD | 10 | 10 | 10 | 10 | - | - | - | - | 10 | 6 | 10 | 10 |
| Payload 5 MMD | 10 | 10 | 10 | 10 | - | - | - | - | 6 | 6 | 10 | 10 |
| Payload 6 MMD | 3 | 6 | 10 | 0 | - | - | - | - | 10 | 10 | 6 | 10 |

Table 4.17 Payload specifications (medium instance 2).

| Method | G.H. | | | | 5-Norm | | | | W. Norm | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| Payload 1 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 10 | 10 | 10 | 10 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 3 MMD | 0 | 10 | 10 | 10 | 0 | 0 | 10 | 10 | 0 | 10 | 0 | 10 |
| Payload 4 MMD | 10 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 10 | 10 |
| Payload 5 MMD | 3 | 6 | 6 | 0 | 10 | 10 | 0 | 0 | 0 | 0 | 10 | 0 |
| Payload 6 MMD | 0 | 0 | 0 | 10 | 10 | 3 | 3 | 10 | 3 | 6 | 10 | 10 |

Finding an exact solution requires substantially more time for the IP solver than in the previous instance. Simulated annealing found a near-optimal solution in roughly half the time. Of the remaining heuristics, only the 5-norm found a solution within 5% of optimal. The largest, medium-sized problem is presented next in Tables 4.18-4.21. This is the first instance in which Xpress fails to find a provable, optimal solution. Therefore, all methods are compared against the best, integer solution obtained as well as the lowest, upper bound on the objective function value. This bound is determined through cuts generated by Xpress.

Table 4.18 Maximum total utility (medium instance 3).

| Method | Total Utility | % Diff. Best Solution | % Diff. Upper Bound | Time (s) |
|---|---|---|---|---|
| Xpress Solver | 1088.8 | 0.0 | 0.2 | 1404.5 |
| Enumeration | - | - | - | - |
| Simulated Annealing | 1039.4 | 4.5 | 4.7 | 109.13 |
| Greedy Heuristic | 1033.0 | 5.1 | 5.3 | 0.11 |
| 5-Norm | 1054.3 | 3.2 | 3.4 | 0.13 |
| Weighted Norm | 1037.1 | 4.7 | 5.0 | 0.17 |

The simulated annealing solution is within 5% of the best integer solution and upper bound; however, the 5-norm came the closest to optimality of all the heuristics. The largest problems are next presented beginning with the 6 satellite, 6 payload type instance found in Tables 4.22-4.25. Xpress managed to find a solution, but it required a substantial amount of time. Although simulated annealing found

Table 4.19 Payload specifications (medium instance 3).

| Method | Xpress | | | | | Enum. | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| Payload 1 MMD | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - |
| Payload 2 MMD | 3 | 0 | 0 | 0 | 10 | - | - | - | - | - |
| Payload 3 MMD | 0 | 3 | 6 | 10 | 0 | - | - | - | - | - |
| Payload 4 MMD | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - |
| Payload 5 MMD | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - |
| Payload 6 MMD | 3 | 10 | 10 | 10 | 10 | - | - | - | - | - |
| Payload 7 MMD | 3 | 6 | 10 | 10 | 10 | - | - | - | - | - |
| Payload 8 MMD | 0 | 0 | 0 | 0 | 0 | - | - | - | - | - |

Table 4.20 Payload specifications (medium instance 3).

| Method | S.A. | | | | | G.H. | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| Payload 1 MMD | 6 | 0 | 10 | 10 | 6 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 0 | 10 | 0 | 6 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 3 MMD | 6 | 6 | 3 | 10 | 10 | 0 | 10 | 10 | 0 | 10 |
| Payload 4 MMD | 10 | 10 | 6 | 10 | 6 | 10 | 0 | 10 | 10 | 10 |
| Payload 5 MMD | 6 | 6 | 10 | 10 | 0 | 3 | 6 | 0 | 10 | 10 |
| Payload 6 MMD | 6 | 3 | 10 | 0 | 10 | 0 | 0 | 0 | 10 | 0 |
| Payload 7 MMD | 0 | 6 | 10 | 3 | 10 | 0 | 0 | 6 | 3 | 6 |
| Payload 8 MMD | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 4.21 Payload specifications (medium instance 3).

| Method | 5-Norm | | | | | W. Norm | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| Payload 1 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 0 | 10 | 0 | 10 | 10 | 10 | 0 | 10 | 10 | 10 |
| Payload 3 MMD | 0 | 0 | 10 | 10 | 0 | 0 | 10 | 0 | 10 | 0 |
| Payload 4 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 10 | 10 |
| Payload 5 MMD | 10 | 0 | 10 | 10 | 0 | 0 | 0 | 10 | 10 | 0 |
| Payload 6 MMD | 10 | 0 | 10 | 0 | 10 | 3 | 10 | 6 | 0 | 10 |
| Payload 7 MMD | 0 | 10 | 0 | 0 | 10 | 0 | 0 | 10 | 0 | 10 |
| Payload 8 MMD | 3 | 0 | 0 | 0 | 10 | 0 | 3 | 0 | 0 | 10 |

a solution closest to optimal, the difference in solution quality of the heuristics is small. Consider the 6 satellite, 8 payload problem in Tables 4.26-4.29. In this instance, simulated annealing performed worse than all other heuristics including the greedy heuristic. The 5-norm performed best. In the final instance involving 7 satellites and 8 payloads in Tables 4.30-4.33, the 5-norm, once again, had the most optimal solution followed by the weighted norm.

Table 4.22 Maximum total utility (large instance 1).

| Method | Total Utility | % Diff. Optimal | Time (s) |
|---|---|---|---|
| Xpress Solver | 1248.0 | 0.0 | 5435.7 |
| Enumeration | - | - | - |
| Simulated Annealing | 1191.4 | 4.5 | 71.02 |
| Greedy Heuristic | 1186.9 | 4.9 | 0.11 |
| 5-Norm | 1188.4 | 4.8 | 0.13 |
| Weighted Norm | 1180.4 | 5.4 | 0.25 |

Table 4.23 Payload specifications (large instance 1).

| Method | Xpress | | | | | | Enum. | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| Payload 1 MMD | 6 | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - | - |
| Payload 2 MMD | 6 | 6 | 10 | 10 | 10 | 10 | - | - | - | - | - | - |
| Payload 3 MMD | 3 | 3 | 3 | 3 | 10 | 10 | - | - | - | - | - | - |
| Payload 4 MMD | 10 | 10 | 10 | 10 | 3 | 10 | - | - | - | - | - | - |
| Payload 5 MMD | 6 | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - | - |
| Payload 6 MMD | 6 | 6 | 6 | 10 | 10 | 3 | - | - | - | - | - | - |

Several conclusions can be drawn from the results of these notional problem instances. Of the exact solution methods, the IP-based Xpress solver clearly worked best. Although it was unable solve three of the larger problems to optimality, it placed relatively tight bounds on the optimal objective function value and provided a near-optimal integer solution. The enumerative method took inordinately long, even for small problems and if terminated early, provided no solution. Of the heuristic methods, the simulated annealing routine and the 5-norm provided solutions

Table 4.24 Payload specifications (large instance 1).

| Method | S.A. | | | | | | G.H. | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| Payload 1 MMD | 6 | 10 | 6 | 10 | 10 | 6 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 3 | 10 | 10 | 6 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 3 MMD | 3 | 6 | 10 | 10 | 10 | 10 | 0 | 10 | 10 | 10 | 10 | 10 |
| Payload 4 MMD | 6 | 6 | 6 | 10 | 6 | 10 | 10 | 0 | 10 | 10 | 10 | 3 |
| Payload 5 MMD | 10 | 3 | 3 | 10 | 10 | 10 | 3 | 6 | 6 | 0 | 10 | 10 |
| Payload 6 MMD | 10 | 0 | 10 | 6 | 3 | 6 | 0 | 0 | 0 | 10 | 3 | 10 |

Table 4.25 Payload specifications (large instance 1).

| Method | 5-Norm | | | | | | W. Norm | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| Payload 1 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 3 MMD | 0 | 0 | 10 | 10 | 3 | 10 | 0 | 10 | 0 | 10 | 10 | 10 |
| Payload 4 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 10 | 10 | 10 | 10 |
| Payload 5 MMD | 10 | 10 | 0 | 0 | 10 | 0 | 0 | 0 | 10 | 0 | 10 | 0 |
| Payload 6 MMD | 10 | 3 | 3 | 10 | 10 | 10 | 3 | 6 | 10 | 10 | 3 | 10 |

Table 4.26 Maximum total utility (large instance 2).

| Method | Total Utility | % Diff. Best Solution | % Diff. Upper Bound | Time (s) |
|---|---|---|---|---|
| Xpress Solver | 1286.3 | 0.0 | 1.0 | 3472.1 |
| Enumeration | - | - | - | - |
| Simulated Annealing | 1219.6 | 5.2 | 6.1 | 131.81 |
| Greedy Heuristic | 1232.3 | 4.2 | 5.1 | 0.12 |
| 5-Norm | 1250.3 | 2.8 | 3.7 | 0.16 |
| Weighted Norm | 1234.6 | 4.0 | 4.9 | 0.18 |

that were consistently close to optimality. Simulated annealing is the most time consuming, yet the most reliable heuristic. On all but one instance, in which the sub-optimality was 5.2%, simulated annealing came within 5% of the optimal or best integer solution.

Table 4.27 Payload specifications (large instance 2).

| Method | Xpress | | | | | | Enum. | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| Payload 1 MMD | 6 | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - | - |
| Payload 2 MMD | 0 | 3 | 3 | 0 | 10 | 10 | - | - | - | - | - | - |
| Payload 3 MMD | 6 | 0 | 10 | 10 | 10 | 0 | - | - | - | - | - | - |
| Payload 4 MMD | 10 | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - | - |
| Payload 5 MMD | 10 | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - | - |
| Payload 6 MMD | 6 | 6 | 6 | 10 | 3 | 10 | - | - | - | - | - | - |
| Payload 7 MMD | 3 | 6 | 0 | 0 | 0 | 10 | - | - | - | - | - | - |
| Payload 8 MMD | 0 | 3 | 0 | 10 | 0 | 0 | - | - | - | - | - | - |

Table 4.28 Payload specifications (large instance 2).

| Method | S.A. | | | | | | G.H. | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| Payload 1 MMD | 3 | 6 | 10 | 10 | 6 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 0 | 6 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 3 MMD | 10 | 0 | 6 | 0 | 10 | 10 | 0 | 10 | 10 | 0 | 10 | 10 |
| Payload 4 MMD | 3 | 6 | 0 | 10 | 10 | 3 | 10 | 0 | 10 | 10 | 10 | 10 |
| Payload 5 MMD | 10 | 10 | 10 | 10 | 0 | 0 | 3 | 6 | 0 | 10 | 10 | 0 |
| Payload 6 MMD | 10 | 3 | 0 | 3 | 0 | 10 | 0 | 0 | 0 | 10 | 0 | 10 |
| Payload 7 MMD | 3 | 10 | 3 | 3 | 10 | 10 | 0 | 0 | 6 | 3 | 6 | 10 |
| Payload 8 MMD | 0 | 3 | 6 | 10 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 4.29 Payload specifications (large instance 2).

| Method | 5-Norm | | | | | | W. Norm | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 5 | 6 |
| Payload 1 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 0 | 10 | 0 | 10 | 10 | 10 | 10 | 0 | 10 | 10 | 10 | 10 |
| Payload 3 MMD | 0 | 0 | 10 | 10 | 0 | 10 | 0 | 10 | 0 | 10 | 0 | 10 |
| Payload 4 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 10 | 10 | 10 |
| Payload 5 MMD | 10 | 0 | 10 | 10 | 0 | 0 | 0 | 0 | 10 | 10 | 0 | 0 |
| Payload 6 MMD | 10 | 0 | 10 | 0 | 10 | 10 | 3 | 10 | 6 | 0 | 10 | 10 |
| Payload 7 MMD | 0 | 10 | 0 | 0 | 10 | 10 | 0 | 0 | 10 | 0 | 10 | 10 |
| Payload 8 MMD | 3 | 0 | 0 | 0 | 10 | 0 | 0 | 3 | 0 | 0 | 10 | 0 |

Table 4.30 Maximum total utility (large instance 3).

| Method | Total Utility | % Diff. Best Solution | % Diff. Upper Bound | Time (s) |
|---|---|---|---|---|
| Xpress Solver | 1469.7 | 0.0 | 2.8 | 7012.7 |
| Enumeration | - | - | - | - |
| Simulated Annealing | 1415.6 | 3.7 | 6.3 | 151.88 |
| Greedy Heuristic | 1420.4 | 3.4 | 6.0 | 0.14 |
| 5-Norm | 1441.3 | 1.9 | 4.6 | 0.16 |
| Weighted Norm | 1426.5 | 2.9 | 5.6 | 0.21 |

Table 4.31 Payload specifications (large instance 3).

| Method | Xpress | | | | | | | Enum. | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Payload 1 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - | - | - |
| Payload 2 MMD | 3 | 3 | 3 | 0 | 10 | 10 | 10 | - | - | - | - | - | - | - |
| Payload 3 MMD | 3 | 0 | 6 | 10 | 0 | 10 | 0 | - | - | - | - | - | - | - |
| Payload 4 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - | - | - |
| Payload 5 MMD | 6 | 10 | 10 | 10 | 10 | 10 | 10 | - | - | - | - | - | - | - |
| Payload 6 MMD | 3 | 3 | 10 | 10 | 10 | 0 | 10 | - | - | - | - | - | - | - |
| Payload 7 MMD | 0 | 6 | 0 | 0 | 0 | 6 | 10 | - | - | - | - | - | - | - |
| Payload 8 MMD | 0 | 6 | 0 | 10 | 3 | 0 | 0 | - | - | - | - | - | - | - |

Table 4.32 Payload specifications (large instance 3).

| Method | S.A. | | | | | | | G.H. | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Payload 1 MMD | 10 | 6 | 0 | 10 | 6 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 0 | 6 | 10 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 3 MMD | 0 | 10 | 10 | 6 | 10 | 0 | 0 | 0 | 10 | 10 | 0 | 10 | 10 | 10 |
| Payload 4 MMD | 10 | 3 | 3 | 6 | 0 | 10 | 10 | 10 | 0 | 10 | 10 | 10 | 10 | 10 |
| Payload 5 MMD | 6 | 0 | 6 | 10 | 10 | 10 | 10 | 3 | 6 | 0 | 10 | 10 | 0 | 10 |
| Payload 6 MMD | 6 | 10 | 10 | 10 | 10 | 0 | 10 | 0 | 0 | 0 | 10 | 0 | 10 | 0 |
| Payload 7 MMD | 10 | 3 | 3 | 10 | 0 | 10 | 10 | 0 | 0 | 6 | 3 | 6 | 10 | 6 |
| Payload 8 MMD | 0 | 0 | 10 | 0 | 10 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

For the heuristic methods it is important to determine whether the observations made regarding each heuristic method hold true in general. For any heuristic, it is likely that it will perform well for certain payload data sets and poorly for oth-

Table 4.33 Payload specifications (large instance 3).

| Method | 5-Norm | | | | | | | W. Norm | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Satellite No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Payload 1 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Payload 2 MMD | 0 | 10 | 0 | 10 | 10 | 10 | 10 | 10 | 0 | 10 | 10 | 10 | 10 | 10 |
| Payload 3 MMD | 0 | 0 | 10 | 10 | 0 | 10 | 10 | 0 | 10 | 0 | 10 | 0 | 10 | 10 |
| Payload 4 MMD | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 10 | 10 | 10 | 10 |
| Payload 5 MMD | 10 | 0 | 10 | 10 | 0 | 0 | 10 | 0 | 0 | 10 | 10 | 0 | 0 | 10 |
| Payload 6 MMD | 10 | 0 | 10 | 0 | 10 | 10 | 0 | 3 | 10 | 6 | 0 | 10 | 10 | 0 |
| Payload 7 MMD | 0 | 10 | 0 | 0 | 10 | 10 | 6 | 0 | 0 | 10 | 0 | 10 | 10 | 6 |
| Payload 8 MMD | 3 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 3 | 0 | 0 | 10 | 0 | 0 |

ers. Therefore, it is desirable to test each heuristic on a large number of randomly generated data sets in order to draw stronger conclusions regarding the performance of each method. This is done in the next section.

## *4.5 Random Problem Instances*

In this section, randomly generated problem instances are solved by each heuristic, and their solutions are compared to optimal solutions. Because the overhead associated with conditioning data for Xpress is time-consuming, the enumeration routine was used to exactly solve each problem instance. In order to keep the time required for an exact solution manageable, only the 1 satellite, 8 payload problem was solved. In addition to payloads having randomly-generated resource requirements, the satellite bus has randomly-generated resource capacities. Two separate methods were used to generate data, and 100 instances of each method's problems were solved.

The motivation of the first method is to generate a variety of random problems within ranges of expected payload resource requirements and satellite bus capacities. The method, denoted *M1*, uses the same values of payload importance as the notional data in Table 4.3. All payload resource requirements and satellite bus capacities were

generated using a uniform distribution. Payloads of the same type are assigned the same power requirement; however, unlike the notional data, there is no correlation between payload MMD and randomly-generated resource consumption. Tables 4.34 and 4.35 show the ranges over which the payload and satellite bus data was generated:

Table 4.34 Ranges of payload requirements for M1 problem instances.

| Resource | Power (W) | Cost ($100k) | Weight (lb) | Volume ($ft^3$) |
|---|---|---|---|---|
| Range | 100-500 | 100-500 | 100-500 | 3-20 |

Table 4.35 Ranges of bus capacities for M1 problem instances.

| Resource | Power (W-Yr) | Cost ($100k) | Weight (lb) | Volume ($ft^3$) |
|---|---|---|---|---|
| Range | 7500-12500 | 2000-3000 | 2000-3000 | 75-125 |

The second method, denoted *M2*, is designed to test the heuristics over a wider-range of problem instances than M1. Instead of using the notional vales, payload importance values are randomly-generated between 0 and 10. Data in M2 is generated using a uniform distribution on ranges specified in Tables 4.36 and 4.37.

Table 4.36 Ranges of payload requirements for M2 problem instances.

| Resource | Power (W) | Cost ($100k) | Weight (lb) | Volume ($ft^3$) |
|---|---|---|---|---|
| Range | 0-500 | 0-500 | 0-500 | 0-20 |

Table 4.37 Ranges of bus capacities for M2 problem instances.

| Resource | Power (W-Yr) | Cost ($100k) | Weight (lb) | Volume ($ft^3$) |
|---|---|---|---|---|
| Range | 0-10500 | 0-2500 | 0-2500 | 0-100 |

To execute the experiment, 100 problem instances of the 1 satellite, 8 payload type problem were exactly solved via DP enumerations using M1 and M2 randomly-generated data. Then each heuristic was applied to the problems, and the resulting solutions were compared to the exact solutions. The heuristic performance on the M1 generated problem instances are in Tables 4.38 and 4.39, and the results of the

M2 generated problem instances are in Tables 4.40 and 4.41. Table 4.42 shows how many M1 and M2 problem solutions were beyond 5% of optimality.

The mean differences from optimality of the heuristics' solutions were relatively close in the M1 problem set. Simulated annealing has the lowest mean followed by the 5-norm with a slightly higher mean. The lower standard deviation of simulated annealing indicates that it provides near-optimal solutions more consistently than do the other heuristics. In the M1 problem instances, the weighted norm performed roughly equivalently to the greedy heuristic, and no clear advantage was attained in using the weighted norm. The performance results of the M2 problem instances provide a starker contrast of solution quality between the different heuristics. Simulated annealing has a much lower mean difference from optimality than any other heuristic. The 5-norm once again performs better than the weighted norm; however, unlike the M1 problem set, the weighted norm performs better than the greedy heuristic.

Table 4.38 Heuristic solution quality (100 replications of M1 random data).

| Method | % Mean Diff. | % Std. Dev. | % Max Diff. | % Median Diff. |
|---|---|---|---|---|
| Simulated Annealing | 2.4931 | 2.0204 | 7.7330 | 2.1342 |
| Greedy Heuristic | 5.0545 | 5.1807 | 24.9068 | 4.3784 |
| 5-Norm | 3.0781 | 3.5238 | 15.6261 | 1.8375 |
| Weighted Norm | 4.9201 | 5.4404 | 26.9841 | 3.3097 |

Table 4.39 Heuristic solution time (100 replications M1 random data).

| Method | Simulated Annealing | Greedy Heuristic | 5-Norm | Weighted Norm |
|---|---|---|---|---|
| Mean Time (s) | 20.87 | 0.02 | 0.02 | 0.02 |

A number of conclusions can be drawn from the results of the notional and random problem instances. Clearly, the exact method with the best performance is applying the Xpress solver to the IP formulation. The DP enumeration routine could only solve the two smallest notional problems in a reasonable amount of time. Xpress

Table 4.40 Heuristic solution quality (100 replications of M2 random data).

| Method | % Mean Diff. | % Std. Dev. | % Max Diff. | % Median Diff. |
|---|---|---|---|---|
| Simulated Annealing | 0.2013 | 0.8337 | 5.3157 | 0.0000 |
| Greedy Heuristic | 10.2515 | 14.1882 | 74.8440 | 4.0379 |
| 5-Norm | 3.4383 | 5.7982 | 26.8590 | 0.0000 |
| Weighted Norm | 4.4508 | 8.5532 | 46.8890 | 0.0000 |

Table 4.41 Heuristic solution time (100 replications M2 random data).

| Method | Simulated Annealing | Greedy Heuristic | 5-Norm | Weighted Norm |
|---|---|---|---|---|
| Mean Time (s) | 24.26 | 0.02 | 0.02 | 0.03 |

Table 4.42 Number of solutions beyond 5% optimality (100 replications M1 and M2).

| Method | Simulated Annealing | Greedy Heuristic | 5-Norm | Weighted Norm |
|---|---|---|---|---|
| M1 | 11 | 44 | 24 | 40 |
| M2 | 1 | 49 | 25 | 25 |

solved all but three of the notional problem instances, and for the three it failed to solve, the best integer solutions were within 3% of optimality. The best heuristic based on solution quality and consistency is simulated annealing. The closeness to optimality of the 5-norm solutions often rivals those of simulated annealing but, the 5-norm fails to attain near-optimal solutions consistently. However, the 5-norm ran on the order of 1,000-10,000 times faster than simulated annealing, and it can be argued that, taking time into consideration, the 5-norm heuristic has the best overall performance. However, practically considering the lengthy time frame associated with satellite construction and the monetary value of the resources at stake, the time required by simulated annealing ($\sim$3 minutes for the largest problem) is negligible compared to the value gained attaining a nearly-optimal solution.

Several factors must be considered when deciding between the use of the Xpress IP solver or simulated annealing. Most importantly is what level of sub-optimality can be tolerated. Although this research provides the means by which to select and specify payloads, the solution is just a starting point in the process of actually con-

structing a satellite. Modifications to the solution will likely be required because of geometric or thermal considerations. Additionally, the mission objectives or priorities of the satellite constellation itself may change. Therefore, a 5% difference in optimality between the exact and simulated annealing solutions may not be very significant with respect to the other design factors. Additionally, because of the size of the IP formulation, a costly commercial solver like Xpress or CPLEX is almost certainly required. In contrast, the simulated annealing routine is well-documented and coding it in virtually any computer language requires very little specialized or proprietary knowledge. In any case, the tradeoff between a solution's quality and its complexity must be assessed in determining which technique to apply.

# 5. Conclusions and Future Research

Nations spend an enormous amount of financial resources acquiring, launching, and operating satellites. Selecting and specifying satellite payloads is a process that can benefit tremendously from the development of methodologies for more effective resource allocation. Payloads are the mission critical components of a satellite and require power, cost, weight, and volume resources, of which, the satellite bus can only provide a limited amount. A critical specification of a satellite payload is its mean mission duration (MMD), which denotes the payload's lifetime for mission planning purposes. Increasing a payload's MMD specification requires additional components and materials increasing the payload's cost, weight, and volume. Current payload selection and specification methodologies are either general and qualitative or only applicable to a specific type of satellite constellation. This research has developed a general, quantitative methodology to select and specify satellite payloads that can be applied to virtually any satellite constellation.

The payload selection and specification problem was shown to be similar to a class of mathematical programming problems known as *knapsack problems*. To motivate a methodology for the payload selection problem, a thorough review of the literature on knapsack problems was presented to include well-known results and a variety of solution techniques. Solution techniques are generally based on two formulations of the knapsack problem: a dynamic programming formulation and an integer programming formulation. Exact solution methods for each formulation were reviewed. Enumerative methods are typically applied to dynamic programs, while integer programs can be relaxed to provide information about their solution. Solution algorithms that exploit the LP-relaxation are the branch-and-bound and branch-and-cut algorithms. Using the ideas associated with knapsack problems, formal mathematical models were developed for the payload selection problem. It was assumed that a satellite constellation observed at fixed intervals will be maintained

or expanded by launching satellites individually at predetermined epochs. At each launch epoch, payloads are selected for the satellite bus to be launched and assigned MMD specifications. Each individual payload has a utility associated with it that, while not satisfying the textbook definition of utility, is a function of the payload's importance, MMD specification, and the random number of like-type functional payloads. The objective of the payload selection and specification problem is to maximize the total lifetime utility of the satellite constellation. Four characterizations of utility were discussed: static and deterministic, dynamic and deterministic, static and stochastic, and dynamic and stochastic.

Mathematical models for the single-satellite payload selection problem were developed for each characterization of utility. It was shown that each was a relaxation of a multi-choice, multidimensional knapsack problem. Because dynamic and stochastic utility most closely describes the actual nature of payload utility, this model was extended to a multi-satellite case which was the central focus of this research. To enable similar solution methods to those used in knapsack problems, both dynamic programming and integer programming formulations were derived for the multi-satellite model. The dynamic programming formulation was solved exactly using a Matlab program to completely enumerate the state space and prune infeasible states. The integer programming formulation was solved through application of Dash Optimization's Xpress solver that employs the branch-and-bound algorithm in concert with preprocessing and search heuristics. In addition to exact methods, four heuristic methods were introduced. Each heuristic solved the payload selection and specification problem as a series of MCMKPs; however, such an approach did not guarantee optimality. Both a *5-norm* and a *weighted norm* heuristic were developed by extending the classical profit-to-weight ratio heuristic of the one-dimensional KP. For comparison purposes, a greedy heuristic that selects payloads based on their utility was introduced. Finally, a simulated annealing routine was developed for the payload selection problem using experimentally determined parameters. Pos-

sible functional forms for payload utility functions and survival distributions were developed, and the performance of the exact and heuristic solution methods was evaluated on a variety of both notional and randomly-generated problem instances. It was observed that the Xpress solver vastly outperformed the dynamic programming enumeration in both solution time and practical ability to attain an optimal solution. Of the heuristics, the solutions provided by simulated annealing and the *5-norm* heuristic were typically closest to optimal. However, the greater consistency of simulated annealing in providing near-optimal solutions led to the conclusion that it was the best heuristic to use for satellite payload selection. Deciding whether to solve the payload selection problem using an exact IP-based solver or a simulated annealing routine is a multifaceted decision, and tradeoffs must be made between solution quality and complexity.

Although a general methodology has been developed for the selection and specification of satellite payloads, there are a number of unresolved issues and areas of future research. The power constraints of satellite buses were quantified in terms of energy measurements, the lifetime discharge of the battery. In reality, before it ceases to produce energy, a battery's *true power*, the rate of its energy delivery, will decline to an insufficient level for the operation of many payloads. This is a very important factor in the operability of many satellites, and the model's realism would be greatly increased by incorporating it. Additionally, the independence of payload survival distributions was assumed. This is somewhat restrictive as payload queueing, in which one payload prompts the operation of another, is relatively common. Therefore, the failure of a payload may cause the effective failure of another. Closely related is extending the assumption of utility dependence to incorporate the utility dependence between different types of payloads. A possible research direction to incorporate dependence in survival distributions and utility functions is attempting to aggregate the dependent payloads into a single payload whose survival distribution and utility function characterize the superposition of the aggregated payloads.

Finally, neither thermal or geometric constraints were considered. In practice it is critical to dissipate the heat generated by payloads. Also, it must be ensured that the payload's geometries allow them to fit properly on the bus. The ability to extend the payload selection and specification model to incorporate these items would greatly enhance both its usefulness and effectiveness in the process of acquiring, launching and operating satellite systems.

# Bibliography


1. W. Ben-Ameur (2004). Computing the initial temperature of simulated annealing. *Computation Optimization and Applications*, **29**, 396-385.
2. Bell, A.J. (2002). *Analysis of GPS Satellite Allocation for the United States Nuclear Detonation Detection System (USNDS)*. M.S. Thesis, Air Force Institute of Technology, Wright-Patterson, OH.
3. Bertsimas, D. and R. Demir (2002). An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science*, **48**, 550-565.
4. Dantzig, G. (1957). Discrete variable extremum problems. *Operations Research*, **5**, 266-277.
5. Denardo, E. (2003). *Dynamic Programming: Models and Applications*. Dover, New York.
6. Fréville, A. and G. Plateau (1986). Heuristics and reduction methods for multiple constraints 0-1 linear programming problems. *European Journal of Operational Research*, **24**, 206-215.
7. Fréville, A. and G. Plateau (1993). An exact search for the solution of the surrogate dual of the 0-1 bidimensioal knapsack problem. *European Journal of Operational Research*, **68**, 413-421.
8. Fréville, A. (2004). The multidimensional 0-1 knapsack problem: An overview. *European Journal of Operational Research*, **155**, 1-21.
9. General Accounting Office (2003). *Military Space Operations: Common Problems and Their Effects on Satellite and Related Acquisitions*. Washington D.C., GPO.
10. Hajek, H. (1988). Cooling schedules for optimal annealing. *Mathematics of Operations Research*, **13**, 311-329.
11. Horowitz, E. and S. Sahni (1974). Computing paritions with applications to the knapsack problem. *Journal of the Association for Computing Machinery*, **2**, 277-292.
12. Keeney, R. and Raiffa, H (1993). *Decisions with Multiple Objectives*. Cambridge University Press, Cambridge, UK.
13. Larson, W. and J. Wertz (2004). *Space Mission Analysis and Design*. Microcosm Press, El Segundo, CA.
14. Martello, S. and D. Pisinger (2000). New trends in exact algorithms for the 0-1 knapsack problem. *European Journal of Operational Research*, **123**, 325-332.

15. Martello, S., D. Pisinger, and P. Toth (1999). Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, **45**, 414-424.

16. Martello, S. and P. Toth (1977). An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, **1**, 169-175.

17. Martello, S. and P. Toth (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York.

18. Michalewicz, Z. and D. Fogel (2000). *How to Solve It: Modern Heuristics*. Springer-Verlag, New York.

19. Pisinger, D. (1995). An expanding core algorithm for the exact 0-1 knapsack problem. *European Journal of Operational Research*, **87**, 175-187.

20. Pisinger, D. (1997). A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, **46**, 758-767.

21. W. Shih (1979). A branch and bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society*, **30**, 369-378.

22. Soyster A., B. Lev, and W. Slivka (1978). Zero-one programming with many variables and few constraints. *European Journal of Operations Research*, **2**, 195-201.

23. Triki, E., Y. Collette, and P. Siarry (2005). A theoretical study on the behavior of siulated annealing leading to a new cooling schedule. *European Journal of Operational Research*, **166**, 77-92.

24. Wang, T. and Wu, K (1999). A parameter set design procedure for the simulated annealing algorithm under the computational time constraint. *Computers and Operations Research*, **26**, 665-678.

25. Wolsey, L.A. (1998). *Integer Programming*. John Wiley & Sons, Inc., New York

# Appendix A. DP Enumeration Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AUTHOR: Capt John Flory
%         AFIT/ENS/GOR-06M
%         March 2006
% This program performs a complete enumeration of the multi-satellite
% payload selection and specification problem.  Pruning is used to
% eliminate infeasible states.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all;

% Begin clock
tic;

% Set alpha and beta, the tuning parameters for the utility decay function
% and survival distribution, respectively
alpha = log(.5);
beta = log(.9);

gamma = .5;

% Number of satellites, payload types and MMD specifications per type
N_Sat = 2;
N_Type = 5;
N_Spec = 3;

% Time horizion of problem
Epochs = 30;

% Set launch time periods
Nl = [2 5 7 9 10 11 13 15];

% Input Payload Data -- [Importance,MMD,Power,Cost,Weight,Volume]
PD(1,:,1)=[10        3        500     425     450     15];
PD(2,:,1)=[10        6        500     460     475     17];
PD(3,:,1)=[10       10        500     500     500     20];
PD(1,:,2)=[8.5      3        475     375     400     16];
PD(2,:,2)=[8.5      6        475     405     415     18];
PD(3,:,2)=[8.5      10        475     430     420     19.5];
PD(1,:,3)=[7.5      3        425     410     430     10];
```

```
PD(2,:,3)=[7.5      6        425     460     480     13];
PD(3,:,3)=[7.5      10        425     480     495     14];
PD(1,:,4)=[7        3        260     300     230     10];
PD(2,:,4)=[7        6        260     350     280     13];
PD(3,:,4)=[7        10        260     370     300     14];
PD(1,:,5)=[6        3        225     370     380     13];
PD(2,:,5)=[6        6        225     400     390     15.5];
PD(3,:,5)=[6        10        225     410     395     17.5];
PD(1,:,6)=[5.5      3        300     280     240     8];
PD(2,:,6)=[5.5      6        300     320     290     9];
PD(3,:,6)=[5.5      10        300     380     310     12];
PD(1,:,7)=[5        3        275     150     280      7];
PD(2,:,7)=[5        6        275     190     350     9.5];
PD(3,:,7)=[5        10        275     240     410     14];
PD(1,:,8)=[3        3        175     270     225         4];
PD(2,:,8)=[3        6        175     310     260     5.5];
PD(3,:,8)=[3        10        175     335     300     8];

% Create array of the initial number of each payload type in constellation
% at time 0
Init_Cons = zeros(N_Type,N_Spec);

% Input the number of each specific payload type/nonzero MMD combination in
% constellation at time 0
Init_Cons = [0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0;0 0 0];

% Create array for expected numbers of payload types at each time period
ENum_Orig = zeros(N_Type,Epochs);

% Given the initial numbers of each payload type/nonzero MMD combination
% initially in constellation, calculate their expected numbers at future
% time periods.
for i = 1:Epochs;
    for j = 1:N_Type;
        Num = 0;
        for k = 1:N_Spec
            if i <= PD(k,2,j)+1;
                Num = Num + Init_Cons(j,k) * Death_Exp(beta,PD(k,2,j),i-1);
            end;
        end;
        ENum_Orig(j,i) = Num;
    end;
end;
```

```
85
86  % Set number of states for stage 1
87  N_States = N_Spec+1;
88  N_States
89
90  % Initialize utility vector for all states in stage 1
91  Utility = zeros(N_States,1);
92
93  % Initialize utility vector for all states in previous stage
94  Utility_Last = zeros(N_States,1);
95
96  % Initialize vector that stores payload contents of stage 1
97  Contents = zeros(N_States,1);
98
99  % Initialize vector of expected number of payload type being specified
100 % in stage 1
101 ENum = zeros(N_States,Epochs);
102
103 % Initialize vector of state designators as feasible or infeasible
104 Feasible = zeros(N_States,1);
105
106 % Initialize vectors of satellites' remaining power, costs, weight, and
107 % volume resources for all states in stage 1
108 Power = zeros(N_States,N_Sat);
109 Cost = zeros(N_States,N_Sat);
110 Weight = zeros(N_States,N_Sat);
111 Volume = zeros(N_States,N_Sat);
112
113 % Set power, cost, weight, and volume capacities of all satellite buses
114 for i = 1:N_States;
115     for j = 1:N_Sat;
116         Power(i,j) = 10500;
117         Cost(i,j) = 2500;
118         Weight(i,j) = 2500;
119         Volume(i,j) = 100;
120     end;
121 end;
122
123 % Create identity matrix to construct elementary vectors
124 Ident = eye(N_Sat,N_Sat);
125
126 % Begin stage 1
127 for i = 1:N_Type;
```

```
128     for j = 1:N_Sat;
129         e_i = Ident(j,:);
130 
131         % Only do for stage 1
132         if i==1 & j==1;
133 
134             % Iterate through all stage 1 states
135             for k = 1:N_States;
136 
137                 % Load expected number of payload type being added
138                 % stage 1
139                 ENum(k,:) = ENum_Orig(i,:);
140 
141                 % Update contents vector to reflect the addition of this
142                 % payload type and its given MMD specification
143                 Contents(k,1) = mod(k,N_Spec+1)*10^(N_Sat*(i-1)+(j-1));
144 
145                 % If specifying a nonzero MMD
146                 if mod(k,N_Spec+1) ~= 0;
147                     MMD = PD(k,2,i);
148                     psi = PD(mod(k,N_Spec+1),1,i);
149 
150                     % Iterate over all epochs payload is in service
151                     for l = Nl(j)+1 : (Nl(j)+1)+MMD
152                         n = l-Nl(j)-1;
153 
154                         % Update number of expected payloads at each
155                         % time
156                         ENum(k,l)= ...
157                             ENum_Orig(i,l)+1*Death_Exp(beta,MMD,n);
158 
159                         % Get total number of like-type payloads in
160                         % the constellation
161                         N = max(ENum(k,l),1);
162 
163                         % Compute the power scaling factor
164                         a = PD(mod(k,N_Spec+1),3,i)/...
165                             Util_Exp(psi,1,gamma,alpha,MMD,0);
166 
167                         % Compute the added utility of a time
168                         % period
169                         z = Util_Exp(psi,N,gamma,alpha,MMD,n)...
170                             *Death_Exp(beta,MMD,n);
```

```
171
172                     % Update total payload utility
173                     % and bus power consumption
174                     Utility(k,1) = Utility(k,1) + z;
175                     Power(k,:) = Power(k,:) - z * a * e_i;
176                 end;
177             end;
178
179             % If payload has nonzero MMD specification subtract
180             % power, cost, weight, and volume resources from bus
181             % capacity
182             if mod(k,N_Spec+1) ~= 0;
183                 Cost(k,:) = Cost(k,:)-PD(mod(k,N_Spec+1),4,i)*e_i;
184                 Weight(k,:) = Weight(k,:)-PD(mod(k,N_Spec+1),5,i)*e_i;
185                 Volume(k,:) = Volume(k,:)-PD(mod(k,N_Spec+1),6,i)*e_i;
186             end;
187         end;
188
189         % Now check feasibility of states in stage and count number of
190         % feasible states
191         N_Feasible = N_States;
192
193         for k = 1:N_States
194             if Power(k,j)<0 | Cost(k,j)<0 | Weight(k,j)<0 | ...
195                     Volume(k,j)<0
196                 Feasible(k,1) = 0;
197                 N_Feasible = N_Feasible-1;
198             else
199                 Feasible(k,1) = 1;
200             end;
201         end;
202
203         % Now eliminate infeasible states
204         t = 1;
205         for k = 1:N_States
206             if Feasible(k,1) == 1;
207                 Utility(t,1) = Utility(k,1);
208                 Contents(t,1) = Contents(k,1);
209                 Power(t,:) = Power(k,:);
210                 Cost(t,:) = Cost(k,:);
211                 Weight(t,:) = Weight(k,:);
212                 Volume(t,:) = Volume(k,:);
213                 ENum(t,:) = ENum(k,:);
```

```
                t = t+1;
            end;
        end;

        % Transfer information about current states into storage to
        % transfer to stage 2
        N_States = N_Feasible * (N_Spec+1);
        Utility_Last = Utility;
        Contents_Last = Contents;
        ENum_Last = ENum;
        Power_Last = Power;
        Cost_Last = Cost;
        Weight_Last = Weight;
        Volume_Last = Volume;
        N_States

    else
        % Iterate through all states in stage and load state values
        % from the previous stage
        for k = 1:N_States;
            Utility(k,1) = Utility_Last(ceil(k/(N_Spec+1)),1);
            Contents(k,1) = Contents_Last(ceil(k/(N_Spec+1)),1)...
                + mod(k,N_Spec+1) * 10^(N_Sat*(i-1)+(j-1));
            Power(k,:) = Power_Last(ceil(k/(N_Spec+1)),:);
            Cost(k,:) = Cost_Last(ceil(k/(N_Spec+1)),:);
            Weight(k,:) = Weight_Last(ceil(k/(N_Spec+1)),:);
            Volume(k,:) = Volume_Last(ceil(k/(N_Spec+1)),:);

            % If you start specifying next payload type, load new
            % expected numbers; otherwise, continue with the old.
            if j ~= 1;
                ENum(k,:) = ENum_Last(ceil(k/(N_Spec+1)),:);
            else;
                ENum(k,:) = ENum_Orig(i,:);
            end;

        end;

        % Iterate through all states in stage
        for k = 1:N_States;

            % If payload has nonzero MMD
            if mod(k,N_Spec+1) ~= 0;
```

```
257
258                 % Load payload data values
259                 MMD = PD(mod(k,N_Spec+1),2,i);
260                 psi = PD(mod(k,N_Spec+1),1,i);
261
262                 % Compute no. time periods payload is in service
263                 for l = Nl(j) + 1 : (Nl(j)+1) + MMD
264                     n = l - Nl(j) - 1;
265
266                     % Update expected number of functional payloads
267                     ENum(k,l) = ENum(k,l) + Death_Exp(beta,MMD,n);
268                     N = max(ENum(k,l),1);
269
270                     % Compute power scaling factor
271                     a = PD(mod(k,N_Spec+1),3,i)/...
272                         Util_Exp(psi,1,gamma,alpha,MMD,0);
273
274                     % Compute stage incremental utility
275                     z = Util_Exp(psi,N,gamma,alpha,MMD,n)...
276                         * Death_Exp(beta,MMD,n);
277
278                     % Update total payload utility and
279                     % bus power consumption
280                     Utility(k,1) = Utility(k,1) + z;
281                     Power(k,:) = Power(k,:) - a * z * e_i;
282                 end;
283             end;
284
285             % Update total bus power, weight, and volume consumption
286             if mod(k,N_Spec+1) ~= 0;
287                 Cost(k,:) = Cost(k,:) -PD(mod(k,N_Spec+1),4,i)*e_i;
288                 Weight(k,:) = Weight(k,:) -PD(mod(k,N_Spec+1),5,i)*e_i;
289                 Volume(k,:) = Volume(k,:) -PD(mod(k,N_Spec+1),6,i)*e_i;
290             end;
291         end;
292
293         % Now check feasibility of states in stage and count number of
294         % feasible states
295         N_Feasible = N_States;
296
297         for k = 1:N_States
298             if Power(k,j)<0 | Cost(k,j)<0 | Weight(k,j)<0...
299                 | Volume(k,j)<0
```

```
                    Feasible(k,1)=0;
                    N_Feasible = N_Feasible-1;
                else
                    Feasible(k,1) = 1;
                end;
            end;

            % Now eliminate infeasible states
            t = 1;
            for k = 1:N_States
                if Feasible(k,1) == 1;
                    Utility(t,1) = Utility(k,1);
                    Contents(t,1) = Contents(k,1);
                    Power(t,:) = Power(k,:);
                    Cost(t,:) = Cost(k,:);
                    Weight(t,:) = Weight(k,:);
                    Volume(t,:) = Volume(k,:);
                    ENum(t,:) = ENum(k,:);
                    t = t+1;
                end;
            end;

            if(i~=N_Type | j~=N_Sat);

                % Set number of states for next stage
                N_States = N_Feasible * (N_Spec+1);
            else
                N_States = N_Feasible;
            end;

            % Store all stage values for use in next stage
            Utility_Last = Utility;
            Contents_Last = Contents;
            ENum_Last = ENum;
            Power_Last = Power;
            Cost_Last = Cost;
            Weight_Last = Weight;
            Volume_Last = Volume;
            N_States
        end;
    end;
end;
disp('END');
```

```
343
344 % Find end state with greatest total utility
345 Best = 0;
346 Best_Index = 1;
347 for k = 1:N_States;
348     if Utility(k,1) > Best
349         Best = Utility(k,1);
350         Best_Index = k;
351     end;
352 end;
353 Sat_Cont = zeros(N_Sat,N_Type);
354 Best_Content = Contents(Best_Index,1);
355
356 % Translate the contents of the best state into actual payload
357 % specifications
358 for i = 1:N_Type;
359     for j = 1:N_Sat;
360         z = floor(Best_Content/10^((N_Sat*(N_Type-1) +...
361             (N_Sat-1)) - N_Sat*(i-1)-(j-1)));
362         Sat_Cont(N_Sat+1-j,N_Type+1-i) = z;
363         Best_Content = Best_Content -...
364             z*10^((N_Sat*(N_Type-1)+(N_Sat-1))-N_Sat*(i-1)-(j-1));
365     end;
366 end;
367
368 % Stop clock
369 toc;
370 time = toc;
371 time
372
373 % Display solution, optimal utility value and remaining power, cost,
374 % weight and volume on all buses
375 Sat_Cont
376 Utility(Best_Index)
377 Power(Best_Index,:)
378 Cost(Best_Index,:)
379 Weight(Best_Index,:)
380 Volume(Best_Index,:)
```

# Appendix B. IP Generation Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AUTHOR: Capt John Flory
%         AFIT/ENS/GOR-06M
%         March 2006
% This program generates the objective functions and constraint matrix for
% the IP formulation of the payload selection and specification problem.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Set survival and utility decay parameters
Death_Coef = log(.9);
Util_Coef = log(.5);

% Set utility dependence parameter
Type_Dep = .5;

% Set number of satellites, payload types, and nonzero MMD specifications
N_Sat = 2;
N_Type = 5;
N_Spec = 3;

% Time horizon of problem
Epochs = 30;

% Set launch time periods
Nl = [2 5 7 9 10 11 13 15];

% Input Payload Data -- [Importance,MMD,Power,Cost,Weight,Volume]
PD(1,:,1)=[10        3        500     425     450     15];
PD(2,:,1)=[10        6        500     460     475     17];
PD(3,:,1)=[10       10        500     500     500     20];
PD(1,:,2)=[8.5      3        475     375     400     16];
PD(2,:,2)=[8.5      6        475     405     415     18];
PD(3,:,2)=[8.5      10        475     430     420     19.5];
PD(1,:,3)=[7.5      3        425     410     430     10];
PD(2,:,3)=[7.5      6        425     460     480     13];
PD(3,:,3)=[7.5      10        425     480     495     14];
PD(1,:,4)=[7        3        260     300     230     10];
PD(2,:,4)=[7        6        260     350     280     13];
PD(3,:,4)=[7        10        260     370     300     14];
PD(1,:,5)=[6        3        225     370     380     13];
PD(2,:,5)=[6        6        225     400     390     15.5];
```

```
PD(3,:,5)=[6        10        225     410     395     17.5];
PD(1,:,6)=[5.5      3        300     280     240     8];
PD(2,:,6)=[5.5      6        300     320     290     9];
PD(3,:,6)=[5.5      10        300     380     310     12];
PD(1,:,7)=[5        3        275     150     280      7];
PD(2,:,7)=[5        6        275     190     350     9.5];
PD(3,:,7)=[5        10        275     240     410     14];
PD(1,:,8)=[3        3        175     270     225         4];
PD(2,:,8)=[3        6        175     310     260     5.5];
PD(3,:,8)=[3        10        175     335     300     8];

% Set satellite bus capacities
Power_Limit = 10500;
Cost_Limit = 2500;
Volume_Limit = 2500;
Weight_Limit = 100;
ENum_Orig = zeros(N_Type,Epochs);

% Set numbers of payload types/MMD specifications initially in the
% constellation
Init_Cons = zeros(N_Type,N_Spec);
Init_Cons = [0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0];

% Calculate expected number of remaining payloads at each time period
for i = 1:Epochs;
    for j = 1:N_Type;
        Num=0;
        for k = 1:N_Spec
            if i <= PD(k,2,j) + 1;
                Num = Num + Init_Cons(j,k) * ...
                    Death_Exp(Death_Coef,PD(k,2,j),i-1);
            end;
        end;
        ENum_Orig(j,i) = Num;
    end;
end;

% Compute total number of variables and number of variables associated with
% each payload type
tot_var = N_Type * (N_Spec + 1)^N_Sat;
type_var = (N_Spec + 1)^N_Sat;

% Create vector of utility objective coefficients, and power, cost, weight,
```

```
% and volume coefficients for the constraints
Utility = zeros(1,tot_var);
Power = zeros(tot_var, N_Sat);
Cost = zeros(tot_var, N_Sat);
Volume = zeros(tot_var, N_Sat);
Weight = zeros(tot_var, N_Sat);

% Create array of all possible MMD specifications over the satellites for
% each payload type
Spec = zeros(tot_var,N_Sat);

for i = 1 : N_Type;
    for j = 1 : type_var;
        % Translate variable index into a set of MMD specifications
        comb = dec2base(j-1,4);
        dec_comb = str2num(comb);
        spec = zeros(1,N_Sat);

        for k = 1 : N_Sat;
            ENum(k,:) = ENum_Orig(i,:);
        end;

        % Generate MMD specifications
        for k = 1 : N_Sat;
            spec(1,k) = floor(dec_comb/10^(N_Sat-k));
            dec_comb = dec_comb - spec(1,k) * 10^(N_Sat-k);
        end;

        Spec(type_var * (i-1) + j,:) = spec;
        u = 0; u1 = 0;

        % Calculate expected number of payloads
        for k = 1 : N_Sat;
            posn = N_Sat - (k-1);
            if spec(1,posn) ~= 0;
                MMD = PD(spec(1,posn),2,i);
                for l = 1 : MMD + 1;
                    ENum(1,Nl(posn)+l) = ENum(1,Nl(posn)+l) + ...
                        Death_Exp(Death_Coef,MMD,l-1);
                end;
            end;
        end;

```

```
128         % Calculate utility, power, cost, weight, volume requirements for
129         % each set of MMD specifications
130         for k = 1 : N_Sat;
131             p = 0; c = 0; w = 0; v = 0; ut = 0;
132             posn = N_Sat - (k-1);
133             if spec(1,posn) ~= 0;
134                 psi = PD(spec(1,posn),1,i);
135                 MMD = PD(spec(1,posn),2,i);
136                 a_1 = PD(spec(1,posn),3,i) / ...
137                     Util_Exp(psi,1,Type_Dep,Util_Coef,MMD,0);
138                 c = c + PD(spec(1,posn),4,i);
139                 w = w + PD(spec(1,posn),5,i);
140                 v = v + PD(spec(1,posn),6,i);
141
142                 % Compute total utility and power requirements
143                 for l = 1 : MMD + 1;
144                     N = max( ENum(1,Nl(posn)+l),1);
145                     u1 = Util_Exp( psi,N,Type_Dep,Util_Coef,MMD,l-1 ) * ...
146                         Death_Exp( Death_Coef,MMD,l-1 );
147                     u = u + u1;
148                     p = p + u1 * a_1;
149                     ENum(1,Nl(posn)+l) = ENum(1,Nl(posn)+l) - ...
150                         Death_Exp(Death_Coef,MMD,l-1);
151                     ut = ut+u1;
152                 end;
153
154                 % Set all coefficients with computed values
155             end;
156             Utility1( type_var*(i-1)+j,posn )=ut;
157             Power( type_var*(i-1)+j, posn ) = p;
158             Cost( type_var*(i-1)+j, posn ) = c;
159             Volume( type_var*(i-1)+j, posn ) = v;
160             Weight( type_var*(i-1)+j, posn ) = w;
161         end;
162         Utility(1, type_var*(i-1)+j ) = u;
163     end;
164 end;
165
166 % Create vector of objective coefficients
167 Obj_Coefs = Utility;
168
169 % Compute number of constraints
170 num_constraint = N_Type + 4 * N_Sat;
```

```
171
172 % Create constraint matrix and RHS constraints
173 Cons_Mat = zeros(num_constraint, tot_var);
174 RHS = zeros(num_constraint,1);
175
176 % Create special ordered set constraints
177 for i = 1 : N_Type;
178     for j = 1 : type_var;
179         Cons_Mat(i, type_var*(i-1) + j) = 1;
180         RHS(i) = 1;
181     end;
182 end;
183
184 % Create power, cost, weight, and volume constraints
185 for i = 1 : N_Type;
186     for j = 1 : type_var;
187         for k = 1 : N_Sat;
188             cons = (N_Type + 1) + 4 * (k-1);
189             indx = type_var * (i-1) + j;
190             Cons_Mat(cons,indx) = Power(indx,N_Sat-(k-1));
191             Cons_Mat(cons+1,indx) = Cost(indx,N_Sat-(k-1));
192             Cons_Mat(cons+2,indx) = Weight(indx,N_Sat-(k-1));
193             Cons_Mat(cons+3,indx) = Volume(indx,N_Sat-(k-1));
194             RHS(cons) = Power_Limit;
195             RHS(cons+1) = Cost_Limit;
196             RHS(cons+2) = Volume_Limit;
197             RHS(cons+3) = Weight_Limit;
198         end;
199     end;
200 end;
201
202 Cons_Mat1 = zeros(num_constraint, tot_var+1);
203
204 for i = 1 : num_constraint;
205     Cons_Mat1(i,:) = [Cons_Mat(i,:),-1];
206 end;
207
208 % Create array with just objective and constraint coefficients
209 System1 = [Utility, -1;Cons_Mat1];
210 System1 = System1';
211
212 % Output array to .csv file
213 csvwrite('System1.txt',System1);
```

# Appendix C. Simulated Annealing Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AUTHOR: Capt John Flory
%         AFIT/ENS/GOR-06M
%         March 2006
% This program applies a simulated annealing routine to the payload
% specification problem.  Simulated annealing is used to maximize the
% utility of each satellite in succession by solving the multi-choice,
% multidimensional knapsack problem associated with each satellite.
% Multiple replications of this proceedure are applied to the constellation
% and the replication with the greatest overall utility is chosen.  A
% geometric-cooling schedule is used.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
tic;

% Set number of simulated annealing replications
N_replicate = 30;

% Set survival and utility function tuning parameters
Death_Coef = log(.9);
Util_Coef = log(.5);

% Set utility dependence parameter
Type_Dep = .5;

% Set number of satellites, payload types, and nonzero MMD specifications
% available to each payload type
N_Sat = 7;
N_Type = 8;
N_Spec = 3;

% Set time horizon of problem
Epochs = 30;

% Set satellite launch time periods
Nl = [2 5 7 9 10 11 13 15];

% Input Payload Data -- [Importance,MMD,Power,Cost,Weight,Volume]
PD(1,:,1)=[10        3        500     425     450     15];
PD(2,:,1)=[10        6        500     460     475     17];
```

```
PD(3,:,1)=[10       10        500     500     500     20];
PD(1,:,2)=[8.5      3         475     375     400     16];
PD(2,:,2)=[8.5      6         475     405     415     18];
PD(3,:,2)=[8.5      10        475     430     420     19.5];
PD(1,:,3)=[7.5      3         425     410     430     10];
PD(2,:,3)=[7.5      6         425     460     480     13];
PD(3,:,3)=[7.5      10        425     480     495     14];
PD(1,:,4)=[7        3         260     300     230     10];
PD(2,:,4)=[7        6         260     350     280     13];
PD(3,:,4)=[7        10        260     370     300     14];
PD(1,:,5)=[6        3         225     370     380     13];
PD(2,:,5)=[6        6         225     400     390     15.5];
PD(3,:,5)=[6        10        225     410     395     17.5];
PD(1,:,6)=[5.5      3         300     280     240     8];
PD(2,:,6)=[5.5      6         300     320     290     9];
PD(3,:,6)=[5.5      10        300     380     310     12];
PD(1,:,7)=[5        3         275     150     280     7];
PD(2,:,7)=[5        6         275     190     350     9.5];
PD(3,:,7)=[5        10        275     240     410     14];
PD(1,:,8)=[3        3         175     270     225       4];
PD(2,:,8)=[3        6         175     310     260     5.5];
PD(3,:,8)=[3        10        175     335     300     8];

% Input satellite bus capacities
for i = 1:N_Sat;
    Power(i) = 10500;
    Cost(i) = 2500;
    Weight(i) = 2500;
    Volume(i) = 100;
end;

% Create empty arrays to store the solution and total utility of each
% replication
rep_soln = zeros(N_Sat,N_Type,N_replicate);
rep_utility = zeros(1,N_replicate);

% For all replications
for replicate = 1 : N_replicate;
    replicate
    ENum_Orig = zeros(N_Type,Epochs);

    % Input numbers of each payload type/MMD specification initially in
    % the constellation
```

```
85      Init_Cons = zeros(N_Type,N_Spec);
86      Init_Cons = [0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0];
87
88      % Calculate expected number of remaining payloads from those
89      % originally in the constellation
90      for i = 1:Epochs;
91          for j = 1:N_Type;
92              Num=0;
93              for k = 1:N_Spec
94                  if i <= PD(k,2,j) + 1;
95                      Num = Num + Init_Cons(j,k) * Death_Exp(Death_Coef,PD(k,2,j),i-1);
96                  end;
97              end;
98              ENum(j,i) = Num;
99          end;
100     end;
101
102     % Create arrays to store total utility of each satellite and the
103     % specifications of its payloads
104     Utility = zeros(1,N_Sat);
105     Soln = zeros(N_Sat,N_Type);
106
107     for i = 1:N_Sat;
108         % Create vectors to store the utility, power, cost, weight, and
109         % volumes of all payload/specification combinations
110         utility = zeros(1, N_Type * N_Spec);
111         power = zeros(1, N_Type * N_Spec);
112         cost = zeros(1, N_Type * N_Spec);
113         weight = zeros(1, N_Type * N_Spec);
114         volume = zeros(1, N_Type * N_Spec);
115
116         for j = 1 : N_Type;
117             for k = 1 : N_Spec;
118                 % Compute cost, weight, and volume of payload/specification
119                 % combination
120                 cost(1, N_Spec * (j-1) + k ) = PD(k,4,j);
121                 weight(1, N_Spec * (j-1) + k ) = PD(k,5,j);
122                 volume(1, N_Spec * (j-1) + k ) = PD(k,6,j);
123
124                 % Load importance and MMD of combination
125                 psi = PD(k,1,j);
126                 MMD = PD(k,2,j);
127
```

```
128                 % Initialize utility and power summation values to zero
129                 u = 0; u1 = 0;
130                 p = 0;
131
132                 % Compute power scaling factor
133                 a_1 = PD(k,3,j) / Util_Exp(psi,1,Type_Dep,Util_Coef,MMD,0);
134
135                 % Compute total utility and power consumption of
136                 % payload/specification combination
137                 for l = 1 : MMD + 1;
138
139                     % Compute expected number of payloads if payload is
140                     % included
141                     N = max( ENum(j,Nl(i)+l) + ...
142                         Death_Exp(Death_Coef, MMD, l-1), 1);
143                     u1 = Util_Exp( psi,N,Type_Dep,Util_Coef,MMD,l-1 ) ...
144                         * Death_Exp( Death_Coef,MMD,l-1 );
145                     u = u + u1;
146                     p = p + u1 * a_1;
147                 end;
148
149                 % Store total utility and power consumption of combination
150                 utility(1, N_Spec * (j-1) + k ) = u;
151                 power(1, N_Spec * (j-1) + k ) = p;
152             end;
153
154             % Create utility/power/cost/weight/volume master array
155             UPCWV = [utility',power',cost',weight',volume'];
156
157         end;
158
159         % Randomly generate a starting slution
160         Vc = zeros(1,N_Type*N_Spec);
161
162         for j = 1 : N_Type;
163             % Decide whether to include (1) or exclude (0) payload type
164             inc = round(rand(1,1));
165             % If including, determine random specification for payload
166             if inc == 1;
167                 inc_spec = round(N_Spec * rand(1,1) - 0.5) + 1;
168                 Vc(1, N_Spec * (j-1) + inc_spec) = 1;
169             end;
170         end;
```

```
171
172         % Randomly remove items until solution is feasible
173         feasible = 0;
174
175         while ~feasible;
176             if (Vc * UPCWV(:,2) <= Power(i)) & ...
177                     (Vc * UPCWV(:,3) <= Cost(i)) & ...
178                     (Vc * UPCWV(:,4) <= Weight(i)) & ...
179                     (Vc * UPCWV(:,5) <= Volume(i));
180                 feasible = 1;
181             end;
182
183             % If starting solution is not feasible, randomly remove objects
184             if feasible == 0;
185                 flipped = 0;
186                 while flipped == 0
187                     rand_indx = round( (N_Type * N_Spec) * rand(1,1) - 0.5) + 1;
188                     if Vc(1,rand_indx) == 1;
189                         Vc(1,rand_indx) = 0;
190                         flipped = 1;
191                     end;
192                 end;
193             end;
194         end;
195         Initial_Soln = Vc;
196
197         % Begin actual simulated annealing heuristic
198         % Set simulated annealing parameters
199         T = 500;  % Initial temperature
200         r = .05;   % Initialize cooling rate
201         T_min = .01; % Terminal temperature
202         N_Max = 5; % No. solutions explored at each temperature
203
204         while T > T_min;
205             n = 0;
206
207             while n <= N_Max;
208                 % Generate random neighbor of current solution
209                 feasible = 0;
210                 loop_count = 0;
211                 while ~feasible & loop_count <= 100;
212                     Vn = Neighbor(Vc,N_Type,N_Spec);
213                     if (Vn * UPCWV(:,2) <= Power(i)) & ...
```

```
                        (Vn * UPCWV(:,3) <= Cost(i)) & ...
                        (Vn * UPCWV(:,4) <= Weight(i)) & ...
                        (Vn * UPCWV(:,5) <= Volume(i));
                    feasible = 1;
                end;
                loop_count = loop_count+1;
            end;

            % Compare current solution to neighboring solution
            Vc_Util = Vc * UPCWV(:,1);
            Vn_Util = Vn * UPCWV(:,1);

            % If neighbor is better, move to it
            if (Vn_Util > Vc_Util) & feasible == 1
                Vc = Vn;

            % If neighbor is worse move to it with a probability
            else
                if rand(1,1) < ...
                        exp((Vn_Util - Vc_Util)/T) & feasible == 1
                    Vc = Vn;
                end;
            end;
            n = n+1;
        end;

        % Decrease temperature
        T = (1-r) * T;
    end;

    % Compute total bus utility
    Utility(1,i) = Vc * UPCWV(:,1);

    % Update numbers of each payload type in constellation
    for j = 1 : N_Type;
        for k = 1 : N_Spec;
            indx = N_Spec * (j-1) + k;
            if Vc(1,indx) == 1;
                Soln(i,j) = k;
                for l = 1 : PD(k,2,j)+1;
                    ENum(j,Nl(i)+l) = ENum(j,Nl(i)+l) ...
                        + Death_Exp(Death_Coef,PD(k,2,j),l-1);
                end;
```

```
                end;
            end;
        end;
    end;

    % Sum utility of all satellites, and store resulting utility along with
    % specificaitons
    tot_util = 0;
    for i = 1 : N_Sat;
        tot_util = tot_util + Utility(1,i);
    end;
    rep_soln(:,:,replicate) = Soln;
    rep_utility(replicate) = tot_util;
end;

% Find replication with greatest utility
indx = 1;
for i = 1 : N_replicate
    if rep_utility(i) >= rep_utility(indx);
        indx = i;
    end;
end;

% Stop clock
t = toc;
t

% Print best utility and specifications
rep_utility(indx)
rep_soln(:,:,indx)
```

# Appendix D. Random Neighbor Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AUTHOR: Capt John Flory
%         AFIT/ENS/GOR-06M
%         March 2006
% This function, used in the simulated annealing code, generates a random
% neighboring solution to a given solution.  A neighbor differes by the
% exclusion or inclusion of a single payload type/MMD combination.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Neighbor = n(Vc,N_Type,N_Spec);
Neighbor = Vc;
length = size(Neighbor,2);

% Determine whether item will be added (move = 1) or removed (move = 0).
% If empty, can only add items
if Neighbor == zeros(1,length);
    move = 1;
else
    % Check to see one payload of each type has been included.  In this
    % case, no more items can be added
    full = 1;
    for i = 1 : N_Type
        type_inc = 0;
        for j = 1 : N_Spec
            indx = N_Spec * (i-1) + j;
            if Neighbor(1,indx) == 1;
                type_inc = 1;
            end;
        end;
        full = full & type_inc;
    end;

    % If full, items can only be removed, not added
    if full == 1;
        move = 0;
    else;
        % If neither empty or full, randomly decide to add/remove item
        move = round(rand(1,1));
    end;
end;

```

```
% If adding an object
if move == 1;
    % Determine random type to include that is not already included
    type_inc = 1;
    while type_inc == 1;
        type = round(N_Type * rand(1,1) - 0.5) + 1;
        type_inc1 = 0;

        for i = 1 : N_Spec;
            if Neighbor(1,N_Spec * (type-1) + i) == 1;
                type_inc1 = 1;
            end;
        end;
        type_inc = type_inc & type_inc1;

        if type_inc == 0;
            spec = round(N_Spec * rand(1,1) - 0.5) + 1;
            Neighbor(1, N_Spec * (type-1) + spec) = 1;
        end;
    end;
    % If removing object
else
    % Determine random type to remove that is not already removed
    type_inc = 0;
    while type_inc == 0;
        type = round(N_Type * rand(1,1) - 0.5) + 1;
        type_inc1 = 0;

        % Determine if type is included
        for i = 1 : N_Spec;
            if Neighbor(1,N_Spec * (type-1) + i) == 1;
                type_inc1 = 1;
                spec = i;
            end;
        end;
        type_inc = type_inc | type_inc1;

        if type_inc == 1;
            Neighbor(1, N_Spec * (type-1) + spec) = 0;
        end;
    end;
end;
```

# Appendix E. *5-Norm* Heuristic Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AUTHOR: Capt John Flory
%         AFIT/ENS/GOR-06M
%         March 2006
% This program applies the 5-norm heuristic to the selection and
% specification of payloads for a multi-satellite constellation.  The
% heuristic is applied to each of the satellites in succession seeking to
% maximize the utility of each.  The heuristic solves the multi-choice,
% multidimensional knapsack problem associated with each bus through an
% extension of the traditional profit-to-cost ratio heuristic for the
% one-dimensional knapsack problem.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

% Start clock
tic;

% Set norm to be the 5-norm
norm = 5;

% Set survival and utility function tuning parameters
Death_Coef = log(.9);
Util_Coef = log(.5);

% Set utility dependence parameter
Type_Dep = .5;

% Set number of satellites, payload types, and nonzero MMD specifications
N_Sat = 7;
N_Type = 8;
N_Spec = 3;

% Set time horizon of problem
Epochs = 30;

% Set launch time periods
Nl = [2 5 7 9 10 11 13 15];

% Input Payload Data -- [Importance,MMD,Power,Cost,Weight,Volume]
PD(1,:,1)=[10        3        500     425     450     15];
```

```
PD(2,:,1)=[10        6        500     460     475     17];
PD(3,:,1)=[10       10        500     500     500     20];
PD(1,:,2)=[8.5      3        475     375     400     16];
PD(2,:,2)=[8.5      6        475     405     415     18];
PD(3,:,2)=[8.5      10        475     430     420     19.5];
PD(1,:,3)=[7.5      3        425     410     430     10];
PD(2,:,3)=[7.5      6        425     460     480     13];
PD(3,:,3)=[7.5      10        425     480     495     14];
PD(1,:,4)=[7        3        260     300     230     10];
PD(2,:,4)=[7        6        260     350     280     13];
PD(3,:,4)=[7        10        260     370     300     14];
PD(1,:,5)=[6        3        225     370     380     13];
PD(2,:,5)=[6        6        225     400     390     15.5];
PD(3,:,5)=[6        10        225     410     395     17.5];
PD(1,:,6)=[5.5      3        300     280     240     8];
PD(2,:,6)=[5.5      6        300     320     290     9];
PD(3,:,6)=[5.5      10        300     380     310     12];
PD(1,:,7)=[5        3        275     150     280     7];
PD(2,:,7)=[5        6        275     190     350     9.5];
PD(3,:,7)=[5        10        275     240     410     14];
PD(1,:,8)=[3        3        175     270     225        4];
PD(2,:,8)=[3        6        175     310     260     5.5];
PD(3,:,8)=[3        10        175     335     300     8];

% Set satellite resource capacities
for i = 1:N_Sat;
    Power(i) = 10500;
    Cost(i) = 2500;
    Weight(i) = 2500;
    Volume(i) = 100;
end;

ENum_Orig = zeros(N_Type,Epochs);

% Set number of payloads of each type and specification initially in
% the constellation
Init_Cons = zeros(N_Type,N_Spec);
Init_Cons = [0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0];

% Calculate expected number of payloads at each time period
for i = 1:Epochs;
    for j = 1:N_Type;
        Num=0;
```

```
        for k = 1:N_Spec
            if i <= PD(k,2,j) + 1;
                Num = Num + Init_Cons(j,k) * ...
                    Death_Exp(Death_Coef,PD(k,2,j),i-1);
            end;
        end;
        ENum(j,i) = Num;
    end;
end;

% Create vectors for total satellite utility and payload specifications
Utility = zeros(1,N_Sat);
Soln = zeros(N_Sat,N_Type);

for i=1:N_Sat;

    % Create vectors to store the utility, power, cost, weight, and volumes
    % of each payload type/MMD specification
    utility = zeros(1, N_Type * N_Spec);
    utility_ratio = zeros(1, N_Type * N_Spec);
    power = zeros(1, N_Type * N_Spec);
    cost = zeros(1, N_Type * N_Spec);
    weight = zeros(1, N_Type * N_Spec);
    volume = zeros(1, N_Type * N_Spec);

    for j = 1 : N_Type;
        for k = 1 : N_Spec;

            % Load cost, weight, volume, importance, and MMD of the payload
            % type/MMD combination
            cost(1, N_Spec * (j-1) + k ) = PD(k,4,j);
            weight(1, N_Spec * (j-1) + k ) = PD(k,5,j);
            volume(1, N_Spec * (j-1) + k ) = PD(k,6,j);
            psi = PD(k,1,j);
            MMD = PD(k,2,j);

            % Initialize total utility and power consumption to zero
            u = 0;
            u1 = 0;
            p = 0;

            % Compute power scaling factor
            a_1 = PD(k,3,j) / Util_Exp(psi,1,Type_Dep,Util_Coef,MMD,0);
```

```
128
129             % Compute total utility and power consumption of a combination
130             for l = 1 : MMD + 1;
131                 % Compute expected number if combination is included
132                 N = max( ENum(j,Nl(i)+l) + ...
133                     Death_Exp(Death_Coef, MMD, l-1), 1);
134                 u1 = Util_Exp( psi,N,Type_Dep,Util_Coef,MMD,l-1 ) ...
135                     * Death_Exp( Death_Coef,MMD,l-1 );
136                 u = u + u1;
137                 p = p + u1 * a_1;
138             end;
139
140             % Update utility and power arrays with computed values
141             utility(1, N_Spec * (j-1) + k ) = u;
142             power(1, N_Spec * (j-1) + k ) = p;
143
144             % Compute 5-norm of resource ratios
145             size = ( (p/Power(i))^norm + (PD(k,4,j)/Cost(i))^norm + ...
146                 (PD(k,5,j)/Weight(i))^norm + ...
147                 (PD(k,6,j)/Volume(i))^norm )^(1/norm);
148
149             % Update vector of utility-to-aggregated weight ratios
150             utility_ratio(1, N_Spec * (j-1) + k ) = u/size;
151
152             % Create utility/power/cost/weight/volume master array
153             UPCWV = [utility_ratio',utility',power',cost',weight',volume'];
154         end;
155     end;
156
157     % After all values for payloads have been computed
158     % determine payloads to include
159
160     % Sort the utility-to-aggregrate weight ratios
161     utility_ratio = sort(utility_ratio);
162
163     % Initialize vector of combinations included
164     Types_Inc = zeros(1,N_Type);
165     for j = 1 : N_Type * N_Spec
166         u = utility_ratio(N_Type*N_Spec+1-j);
167         ind = 0;
168         indx = 0;
169
170         % Determine index of combination in unsorted UPCWV master array
```

```
171         while ind == 0 & indx < N_Type*N_Spec+2;
172             if UPCWV(indx+1,1) == u;
173                 ind = 1;
174             end;
175             indx = indx + 1;
176         end;
177 
178         % Determine payload type of combination
179         Type = ceil(indx/N_Spec);
180 
181         % Determine specification of combination
182         if mod(indx,N_Spec) == 0;
183             Spec = N_Spec;
184         else
185             Spec = mod(indx,N_Spec);
186         end;
187 
188         % Calculate remaining resources if payload is included
189         if Types_Inc(1,Type) == 0;
190             c1 = Power(i) - UPCWV(indx, 3);
191             c2 = Cost(i) - UPCWV(indx, 4);
192             c3 = Weight(i) - UPCWV(indx, 5);
193             c4 = Volume(i) - UPCWV(indx, 6);
194 
195             % If including combination is feasible, include it
196             if c1 >= 0 & c2 >= 0 & c3 >= 0 & c4 >= 0;
197                 Utility(i) = Utility(i) + UPCWV(indx, 2);
198                 Power(i) = Power(i) - UPCWV(indx, 3);
199                 Cost(i) = Cost(i) - UPCWV(indx, 4);
200                 Weight(i) = Weight(i) - UPCWV(indx, 5);
201                 Volume(i) = Volume(i) - UPCWV(indx, 6);
202                 Types_Inc(Type) = PD(Spec,2,Type);
203 
204                 % Update number of payloads in constellation
205                 for l = 1 : Types_Inc(Type)+1
206                     ENum(Type,Nl(i)+l) = ENum(Type,Nl(i)+l) + ...
207                         Death_Exp(Death_Coef, Types_Inc(Type), l-1);
208                 end;
209             end;
210         end;
211     end;
212 
213     % Update payload specifications for satellite
```

```
214     Soln(i,:) = Types_Inc;
215 end;
216 
217 % Display payload specifications on all satellites
218 Soln
219 
220 % Stop clock
221 t = toc;
222 t
223 tot_util = 0;
224 
225 % Compute total constellation utility
226 for i = 1:N_Sat;
227     tot_util = tot_util + Utility(1,i);
228 end;
229 
230 % Display total utility and remaining resources of satellites
231 tot_util
232 Power(1,:)
233 Cost(1,:)
234 Weight(1,:)
235 Volume(1,:)
```

# Appendix F. Weighted Norm Heuristic Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AUTHOR: Capt John Flory
%         AFIT/ENS/GOR-06M
%         March 2006
% This program applies the weighted heuristic to the payload selection and
% specification problem.  The weighted norm heuristic applies a weighted,
% 2-norm to the vector of resource requirement to capacity ratios.  The
% terms of the 2-norm are weighted by the relative scarcity of each
% resource.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;

% Start clock
tic;

% The heuristic uses a 2-norm
norm = 2;

% Set survival and utility decay function tuning parameters
Death_Coef = log(.9); Util_Coef = log(.5);

% Set dependence parameter
Type_Dep = .5;

% Set number of satellites, payload types, and nonzero MMD specifications
N_Sat = 7; N_Type = 8; N_Spec = 3;

% Set time horizion of problem
Epochs = 30;

% Set launch time periods
Nl = [2 5 7 9 10 11 13 15];

% Input Payload Data -- [Importance,MMD,Power,Cost,Weight,Volume]
PD(1,:,1)=[10        3        500     425     450     15];
PD(2,:,1)=[10        6        500     460     475     17];
PD(3,:,1)=[10       10        500     500     500     20];
PD(1,:,2)=[8.5      3        475     375     400     16];
PD(2,:,2)=[8.5      6        475     405     415     18];
PD(3,:,2)=[8.5      10        475     430     420     19.5];
PD(1,:,3)=[7.5      3        425     410     430     10];
```

```
PD(2,:,3)=[7.5      6        425     460     480     13];
PD(3,:,3)=[7.5      10        425     480     495     14];
PD(1,:,4)=[7        3        260     300     230     10];
PD(2,:,4)=[7        6        260     350     280     13];
PD(3,:,4)=[7        10        260     370     300     14];
PD(1,:,5)=[6        3        225     370     380     13];
PD(2,:,5)=[6        6        225     400     390     15.5];
PD(3,:,5)=[6        10        225     410     395     17.5];
PD(1,:,6)=[5.5      3        300     280     240     8];
PD(2,:,6)=[5.5      6        300     320     290     9];
PD(3,:,6)=[5.5      10        300     380     310     12];
PD(1,:,7)=[5        3        275     150     280      7];
PD(2,:,7)=[5        6        275     190     350     9.5];
PD(3,:,7)=[5        10        275     240     410     14];
PD(1,:,8)=[3        3        175     270     225         4];
PD(2,:,8)=[3        6        175     310     260     5.5];
PD(3,:,8)=[3        10        175     335     300     8];

% Set satellite resource capacities
Power_Limit = 10500;
Cost_Limit = 2500;
Weight_Limit = 2500;
Volume_Limit = 100;

ENum_Orig = zeros(N_Type,Epochs);

% Input number of payload type/MMD specification combinations in
% the constellation at time 0
Init_Cons = zeros(N_Type,N_Spec);
Init_Cons = [0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0];

% Calculate expected number of payloads at each time period
for i = 1:Epochs;
    for j = 1:N_Type;
        Num=0;
        for k = 1:N_Spec
            if i <= PD(k,2,j) + 1;
                Num = Num + Init_Cons(j,k) * ...
                    Death_Exp(Death_Coef,PD(k,2,j),i-1);
            end;
        end;
        ENum(j,i) = Num;
    end;
```

```
end;

% Set satellite resource capacities
for i = 1:N_Sat;
    Power(i) = Power_Limit;
    Cost(i) = Cost_Limit;
    Weight(i) = Weight_Limit;
    Volume(i) = Volume_Limit;
end;

% Create vector to store resource scarcity values
PCWV_Sc = zeros(N_Sat,4);

% Create vector to store total utility of each satellite and its payload specifications
Utility = zeros(1,N_Sat);
Soln = zeros(N_Sat,N_Type);

for i = 1:N_Sat;

    % Create vectors to store the utility, power, cost, weight, and volumes
    utility = zeros(1, N_Type * N_Spec);
    utility_ratio = zeros(1, N_Type * N_Spec);
    power = zeros(1, N_Type * N_Spec);
    cost = zeros(1, N_Type * N_Spec);
    weight = zeros(1, N_Type * N_Spec);
    volume = zeros(1, N_Type * N_Spec);

    for j = 1 : N_Type;
        for k = 1 : N_Spec;

            % Load combination resouce requirements, importance, and MMD
            cost(1, N_Spec * (j-1) + k ) = PD(k,4,j);
            weight(1, N_Spec * (j-1) + k ) = PD(k,5,j);
            volume(1, N_Spec * (j-1) + k ) = PD(k,6,j);
            psi = PD(k,1,j);
            MMD = PD(k,2,j);

            % Initialize total utility and power consumption values to zero
            u = 0;
            u1 = 0;
            p = 0;

            % Compute power scaling factor
```

```
128             a_1 = PD(k,3,j) / Util_Exp(psi,1,Type_Dep,Util_Coef,MMD,0);
129
130             % Compute payload type/MMD specification's total utility and power consumption
131             for l = 1 : MMD + 1;
132
133                 % Compute expected number if combination is included
134                 N = max( ENum(j,Nl(i)+l) + ...
135                     Death_Exp(Death_Coef, MMD, l-1), 1);
136                 u1 = Util_Exp( psi,N,Type_Dep,Util_Coef,MMD,l-1 ) * ...
137                     Death_Exp( Death_Coef,MMD,l-1 );
138                 u = u + u1;
139                 p = p + u1 * a_1;
140             end;
141
142             % Update utility and power arrays with computed power and utility
143             utility(1, N_Spec * (j-1) + k ) = u;
144             power(1, N_Spec * (j-1) + k ) = p;
145
146         end;
147     end;
148
149     PCWV_sc = zeros(1,4);
150
151     % Compute total resource requirements of all combinations on a bus
152     for j = 1 : N_Type;
153         for k = 1 : N_Spec;
154             PCWV_sc(1,:) = PCWV_sc(1,:) + ...
155                 [power(1, N_Spec * (j-1) + k ), ...
156                 PD(k,4,j), PD(k,5,j), PD(k,6,j)];
157         end;
158     end;
159
160     % Compute resource scarcities
161     PCWV_sc(1,1) = PCWV_sc(1,1) / Power(i);
162     PCWV_sc(1,2) = PCWV_sc(1,2) / Cost(i);
163     PCWV_sc(1,3) = PCWV_sc(1,3) / Weight(i);
164     PCWV_sc(1,4) = PCWV_sc(1,4) / Volume(i);
165
166     % Compute weighted norm and profit-to-requrements ratios of all combinations
167     for j = 1 : N_Type;
168         for k = 1 : N_Spec;
169             size = ( PCWV_sc(1,1)*(p/Power(i))^norm + ...
170                 PCWV_sc(1,2)*(PD(k,4,j)/Cost(i))^norm + ...
```

```
171                 PCWV_sc(1,3)*(PD(k,5,j)/Weight(i))^norm + ...
172                 PCWV_sc(1,4)*(PD(k,6,j)/Volume(i))^norm )^(1/norm);
173             utility_ratio(1, N_Spec * (j-1) + k ) = ...
174                 utility(1,N_Spec * (j-1) + k ) / size;
175
176             % Create utility/power/cost/weight/volume master array
177             UPCWV = [utility_ratio',utility',power',cost',weight',volume'];
178         end;
179     end;
180
181     % Sort the ratios based on the weighted norm
182     utility_ratio=sort(utility_ratio);
183
184     % Initialize vector of included payload types
185     Types_Inc = zeros(1,N_Type);
186
187     % Iterate over all combinations
188     for j = 1 : N_Type * N_Spec
189         u = utility_ratio(N_Type*N_Spec+1-j);
190         ind = 0;
191         indx = 0;
192
193         % Determine index of combination in unsorted UPCWV master array
194         while ind == 0 & indx < N_Type*N_Spec+2;
195             if UPCWV(indx+1,1) == u;
196                 ind = 1;
197             end;
198             indx = indx + 1;
199         end;
200
201         % Determine payload type and MMD specification
202         Type = ceil(indx/N_Spec);
203         if mod(indx,N_Spec)==0;
204             Spec = N_Spec;
205         else
206             Spec = mod(indx,N_Spec);
207         end;
208
209         % Calculate remaing resources if combination is included
210         if Types_Inc(1,Type) == 0;
211             c1 = Power(i) - UPCWV(indx, 3);
212             c2 = Cost(i) - UPCWV(indx, 4);
213             c3 = Weight(i) - UPCWV(indx, 5);
```

```
            c4 = Volume(i) - UPCWV(indx, 6);

            % If including combination is feasible, include it
            if c1 >= 0 & c2 >= 0 & c3 >= 0 & c4 >= 0;
                Utility(i) = Utility(i) + UPCWV(indx, 2);
                Power(i) = Power(i) - UPCWV(indx, 3);
                Cost(i) = Cost(i) - UPCWV(indx, 4);
                Weight(i) = Weight(i) - UPCWV(indx, 5);
                Volume(i) = Volume(i) - UPCWV(indx, 6);
                Types_Inc(Type) = PD(Spec,2,Type);

                % Update expected number of payloads in constellation at each time period
                for l = 1 : Types_Inc(Type)+1
                    ENum(Type,Nl(i)+l) = ENum(Type,Nl(i)+l) + ...
                        Death_Exp(Death_Coef, Types_Inc(Type), l-1);
                end;
            end;
        end;
    end;

    % Store payload specifications
    Soln(i,:) = Types_Inc;
end;

% Display payload specifications
Soln

% Stop clock
t = toc;
t

% Compute total constellation utility
tot_util = 0;
for i = 1:N_Sat;
    tot_util = tot_util + Utility(1,i);
end;

% Display total constellation utility and remaining satellite resources
tot_util
Power(1,:)
Cost(1,:)
Weight(1,:)
Volume(1,:)
```

# Appendix G. Greedy Heuristic Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AUTHOR: Capt John Flory
%         AFIT/ENS/GOR-06M
%         March 2006
% This program applies a greedy heuristic to each satellite in the payload
% selection and specification problem.  All payload type/MMD specifications
% are sorted by their total utility.  Payloads are included in order of
% decreasing total utility.  If the includsion of a payload is feasible,
% the payload is included.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Start clock
tic;

% Set survival and utility decay function tuning parameters
Death_Coef = log(.9); Util_Coef = log(.5);

% Set utility dependence parameter
Type_Dep = .5;

% Set number of satellites, payload types, and nonzero MMD specifications
N_Sat = 7; N_Type = 8; N_Spec = 3;

% Set problem time horizion
Epochs = 30;

% Input launch time periods
Nl = [2 5 7 9 10 11 13 15];

% Input Payload Data -- [Importance,MMD,Power,Cost,Weight,Volume]
PD(1,:,1)=[10        3        500     425     450     15];
PD(2,:,1)=[10        6        500     460     475     17];
PD(3,:,1)=[10       10        500     500     500     20];
PD(1,:,2)=[8.5      3        475     375     400     16];
PD(2,:,2)=[8.5      6        475     405     415     18];
PD(3,:,2)=[8.5      10        475     430     420     19.5];
PD(1,:,3)=[7.5      3        425     410     430     10];
PD(2,:,3)=[7.5      6        425     460     480     13];
PD(3,:,3)=[7.5      10        425     480     495     14];
PD(1,:,4)=[7        3        260     300     230     10];
PD(2,:,4)=[7        6        260     350     280     13];
```

```
PD(3,:,4)=[7        10        260     370     300     14];
PD(1,:,5)=[6        3        225     370     380     13];
PD(2,:,5)=[6        6        225     400     390     15.5];
PD(3,:,5)=[6        10        225     410     395     17.5];
PD(1,:,6)=[5.5      3        300     280     240     8];
PD(2,:,6)=[5.5      6        300     320     290     9];
PD(3,:,6)=[5.5      10        300     380     310     12];
PD(1,:,7)=[5        3        275     150     280      7];
PD(2,:,7)=[5        6        275     190     350     9.5];
PD(3,:,7)=[5        10        275     240     410     14];
PD(1,:,8)=[3        3        175     270     225         4];
PD(2,:,8)=[3        6        175     310     260     5.5];
PD(3,:,8)=[3        10        175     335     300     8];

% Set satellite bus resource capacities
for i = 1:N_Sat;
    Power(i) = 10500;
    Cost(i) = 2500;
    Weight(i) = 2500;
    Volume(i) = 100;
end;

ENum_Orig = zeros(N_Type,Epochs);

% Set numbers of each payload type/MMD specification in  the constellation
% at time 0
Init_Cons = zeros(N_Type,N_Spec);
Init_Cons = [0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0];

% Calculate expected number of payloads at each time period
for i = 1:Epochs;
    for j = 1:N_Type;
        Num=0;
        for k = 1:N_Spec
            if i <= PD(k,2,j) + 1;
                Num = Num + Init_Cons(j,k) * ...
                    Death_Exp(Death_Coef,PD(k,2,j),i-1);
            end;
        end;
        ENum(j,i) = Num;
    end;
end;

```

```
% Create vectors to store utility and payload specifications
Utility = zeros(1,N_Sat);
Soln = zeros(N_Sat,N_Type);

for i=1:N_Sat;

    % Create vectors to store the utility, power, cost, weight, and volume
    % requirements
    utility = zeros(1, N_Type * N_Spec);
    power = zeros(1, N_Type * N_Spec);
    cost = zeros(1, N_Type * N_Spec);
    weight = zeros(1, N_Type * N_Spec);
    volume = zeros(1, N_Type * N_Spec);

    for j = 1 : N_Type;
        for k = 1 : N_Spec;
            % Load combination's cost, weight, volume, importance, and MMD
            cost(1, N_Spec * (j-1) + k ) = PD(k,4,j);
            weight(1, N_Spec * (j-1) + k ) = PD(k,5,j);
            volume(1, N_Spec * (j-1) + k ) = PD(k,6,j);
            psi = PD(k,1,j);
            MMD = PD(k,2,j);

            % Initialize total utility and power consumption summation
            % values to zero
            u = 0;
            u1 = 0;
            p = 0;

            % Compute power scaling factor
            a_1 = PD(k,3,j) / Util_Exp(psi,1,Type_Dep,Util_Coef,MMD,0);

            % Compute total utility and power consumption of a payload
            % type/MMD specification combination
            for l = 1 : MMD + 1;
                % Compute expected number of payloads if combination is
                % included
                N = max( ENum(j,Nl(i)+l) + ...
                    Death_Exp(Death_Coef, MMD, l-1), 1);
                u1 = Util_Exp( psi,N,Type_Dep,Util_Coef,MMD,l-1 ) * ...
                    Death_Exp( Death_Coef,MMD,l-1 );
                u = u + u1;
                p = p + u1 * a_1;
```

```
            end;
            % Update utility and power vectors with computed values
            utility(1, N_Spec * (j-1) + k ) = u;
            power(1, N_Spec * (j-1) + k ) = p;

            % Create utility/power/cost/weight/volume master array
            UPCWV = [utility',power',cost',weight',volume'];
        end;
    end;

    % Sort total utility values
    utility = sort(utility);

    % Initialize vector of included payload types
    Types_Inc = zeros(1,N_Type);

    % Iterate over all combinations
    for j = 1 : N_Type * N_Spec
        u = utility(N_Type*N_Spec+1-j);
        ind = 0;
        indx = 0;

        % Determine index of utility in unsorted UPCWV master array
        while ind == 0 & indx < N_Type*N_Spec+2;
            if UPCWV(indx+1,1) == u;
                ind = 1;
            end;
            indx = indx + 1;
        end;
        % Determine payload type
        Type = ceil(indx/N_Spec);

        % Determine payload specification
        if mod(indx,N_Spec)==0;
            Spec = N_Spec;
        else
            Spec = mod(indx,N_Spec);
        end;

        % Calculate remaining satellite resources if payload is included
        if Types_Inc(1,Type) == 0;
            c1 = Power(i) - UPCWV(indx, 2);
            c2 = Cost(i) - UPCWV(indx, 3);
```

```
            c3 = Weight(i) - UPCWV(indx, 4);
            c4 = Volume(i) - UPCWV(indx, 5);

            % If payload can be included, include it
            if c1 >= 0 & c2 >= 0 & c3 >= 0 & c4 >= 0;
                Utility(i) = Utility(i) + u;
                Power(i) = Power(i) - UPCWV(indx, 2);
                Cost(i) = Cost(i) - UPCWV(indx, 3);
                Weight(i) = Weight(i) - UPCWV(indx, 4);
                Volume(i) = Volume(i) - UPCWV(indx, 5);
                Types_Inc(Type) = PD(Spec,2,Type);

                % Update expected number of payloads in constellation at
                % each time period
                for l = 1 : Types_Inc(Type)+1
                    ENum(Type,Nl(i)+l) = ENum(Type,Nl(i)+l) + ...
                        Death_Exp(Death_Coef, Types_Inc(Type), l-1);
                end;
            end;
        end;
    end;
    % Store payload specifications
    Soln(i,:) = Types_Inc;
end;

% Stop clock
t = toc; t
% Display payload specifications
Soln

% Compute total constellation utility
tot_util = 0;
for i = 1:N_Sat;
    tot_util = tot_util + Utility(1,i);
end;

% Display total utility and remaining resources on all satellites
tot_util
Power(1,:)
Cost(1,:)
Weight(1,:)
Volume(1,:)
```

# Appendix H. Payload Survival Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AUTHOR: Capt John Flory
%         AFIT/ENS/GOR-06M
%         March 2006
% This function is called by the programs to give the exponential
% survival distribution of satellite payloads.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function uses the following parameters:
% alpha - Tuning parameter > 0
% MMD   - Mean mission duration specification
% n     - Time period

function Death_Exp = F(alpha,MMD,n);

Death_Exp = exp(-abs(alpha)/MMD*n);
```

# Appendix I. Payload Utility Decay Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AUTHOR: Capt John Flory
%         AFIT/ENS/GOR-06M
%         March 2006
% This function is called by the programs to give the exponential
% decay function of payload utility.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function uses the following parameters:
% psi       - Payload importance
% N         - Number of like-type functional payloads
% Type_Dep  - Value of dependence parameter (gamma)
% alpha     - Tuning parameter
% MMD       - Payload mean mission duration
% n         - Time period

function Util_Exp = u(psi,N,Type_Dep,alpha,MMD,n);

% Positive utility dependence
if Type_Dep > 0;
    Util_Exp = psi/N^(Type_Dep)*exp(-abs(alpha)/MMD*n);

    % No utility dependence
elseif Type_Dep == 0;
    Util_Exp = psi*exp(-abs(alpha)/MMD*n);

    % Logarithmic utility dependence
elseif Type_Dep == -1;
    Util_Exp = psi/log(max(N,1))*exp(-abs(alpha)/MMD*n);

end;
```

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From – To)* |
|---|---|---|
| 02-03-2006 | **Masters Thesis** | Mar 2005 – Mar 2006 |

**4. TITLE AND SUBTITLE**

Optimizing Mean Mission Duration for Multiple-Payload Satellites

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**

Flory, John, A., Capt, USAF

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)**
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Street, Building 642
WPAFB OH 45433-7765

**8. PERFORMING ORGANIZATION REPORT NUMBER**

AFIT/GOR/ENS/06-08

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
National Reconnaissance Office
Attn: William J. Comstock, Rm 43D19H
14675 Leed Road
Chantilly, VA 20151-1715 COMM: (703) 808-4436

**10. SPONSOR/MONITOR'S ACRONYM(S)**

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

**12. DISTRIBUTION/AVAILABILITY STATEMENT**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**

This thesis addresses the problem of optimally selecting and specifying satellite payloads for inclusion on a satellite bus to be launched into a constellation. The objective is to select and specify payloads so that the total lifetime utility of the constellation is maximized. The satellite bus is limited by finite power, weight, volume, and cost constraints. This problem is modeled as a classical knapsack problem in one and multiple dimensions, and dynamic programming and binary integer programming formulations are provided to solve the problem. Due to the computational complexity of the problem, the solution techniques include exact methods as well as four heuristic procedures including a greedy heuristic, two norm-based heuristics, and a simulated annealing heuristic. The performance of the exact and heuristic approaches is evaluated on the basis of solution quality and computation time by solving a series of notional and randomly-generated problem instances. The numerical results indicate that, when an exact solution is required for a moderately-sized constellation, the integer programming formulation is most reliable in solving the problem to optimality. However, if the problem size is very large, and near-optimal solutions are acceptable, then the simulated annealing algorithm performs best among the heuristic procedures.

**15. SUBJECT TERMS**

Satellite Constellations, Payload, Selection, Knapsack Problem, Dynamic Programming, Integer Programming, Heuristic Methods.

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Jeffrey P. Kharoufeh, Phd (AFIT/ENS) |
| U | U | U | UU | 131 | 19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636, ext 4603; e-mail: Jeffrey.Kharoufeh@afit.edu |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std. Z39-18