NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

MATHEMATICAL MODELING AND OPTIMAL CONTROL OF BATTLEFIELD INFORMATION FLOW

by

Donovan D. Phillips

June 2008

Dissertation Supervisor:	Wei Kang
Co-Advisor:	Kyle Lin

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2008	**3. REPORT TYPE AND DATES COVERED** Dissertation	
4. TITLE AND SUBTITLE: Mathematical Modeling and Optimal Control of Battlefield Information Flow		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Donovan D. Phillips			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211		**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		**12b. DISTRIBUTION CODE** A	

13. ABSTRACT (maximum 200 words)

The U.S. Army's Future Force requires information dominance to succeed, yet finds itself with an ever-increasing gap between its capacity to collect information and its information processing capacity—with little understanding of how to efficiently utilize scarce processing resources. In this investigation, a model is proposed to adequately represent the flow of information within a command and control context toward the end of optimally controlling this flow. The model is conjectured to be NP-hard in general. Closed-form optimal solutions are derived for special cases of the model, while other cases are shown to be NP-hard. A case of the model is shown to equate to a special case of the quadratic assignment problem not previously known to have a closed-form solution, and such a solution is derived. Upper and lower bounds are derived for more general cases of the model, and heuristic strategies are proposed and tested in discrete event simulation. Strong empirical evidence is produced to demonstrate the effectiveness and robustness of one heuristic.

14. SUBJECT TERMS Dynamic Information Flow, Sequencing and Scheduling, Quadratic Assignment Problem, Complexity			**15. NUMBER OF PAGES** 161
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

MATHEMATICAL MODELING AND OPTIMAL CONTROL OF BATTLEFIELD INFORMATION FLOW

Donovan D. Phillips
Lieutenant Colonel, United States Army
B.S., United States Military Academy, 1989
M.S., Naval Postgraduate School, 1998

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2008

Author: Donovan D. Phillips

Approved by:

Wei Kang Professor of Mathematics Dissertation Supervisor	Kyle Lin Professor of Opns. Research Co-Advisor
Carlos Borges Professor of Mathematics	Hong Zhou Professor of Mathematics
Darryl Ahner Lieutenant Colonel United States Army	

Approved by: _____
Clyde Scandrett, Chair, Department of Applied Mathematics

Approved by: _____
Douglas Moses, Associate Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The U.S. Army's Future Force requires information dominance to succeed, yet finds itself with an ever-increasing gap between its capacity to collect information and its information processing capacity—with little understanding of how to efficiently utilize scarce processing resources. In this investigation, a model is proposed to adequately represent the flow of information within a command and control context toward the end of optimally controlling this flow. The model is conjectured to be NP-hard in general. Closed-form optimal solutions are derived for special cases of the model, while other cases are shown to be NP-hard. A case of the model is shown to equate to a special case of the quadratic assignment problem not previously known to have a closed-form solution, and such a solution is derived. Upper and lower bounds are derived for more general cases of the model, and heuristic strategies are proposed and tested in discrete event simulation. Strong empirical evidence is produced to demonstrate the effectiveness and robustness of one heuristic.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I gratefully acknowledge my advisor, Prof. Wei Kang. I am thankful for his patience, guidance, and unwavering support during this research. I thank my co-advisor, Prof. Kyle Lin, for his thoroughness and clarity. I thank my committee members, Prof. Carlos Borges, Prof. Hong Zhou, and LTC Darryl Ahner, for their guidance and support.

I am deeply thankful for the unquestioned support of my family. My wife, Cate, has mastered the art of juggling, and I am grateful for her unconditional love, prayers, and support. I am indebted to her for the sacrifices she has made on my behalf and that of our family. I thank my children—Emma, Sarah, Grace, and Eli—for their daily encouragement and prayers. I thank my mother, Sheryl Phillips, for showing me by her example how to live by faith, and my father, Donovan L. Phillips, for helping me develop a curiosity for the undiscovered.

I thank the U.S. Military Academy Department of Mathematical Sciences for allowing me this opportunity, and I thank the Army Research Office and the TRADOC Analysis Center for sponsoring this research.

Finally, I acknowledge that none of this could have occurred without the grace and mercy of my Lord Jesus Christ, who provides all wisdom and understanding. I humbly thank Him.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND AND MOTIVATION

The U.S. Army's Future Force is envisioned to "see first, understand first, act first, and finish decisively" [1]. "Seeing first" is accomplished, in part, through the deployment of multiple manned and unmanned systems throughout the operational environment. Future Army commanders are expected to have access to information collected from an enormous array of Intelligence, Surveillance, and Reconnaissance (ISR) assets. Brigade level commanders—such as those commanding the Army's Future Combat System Brigade Combat Teams (FCS BCT)—will directly control hundreds of individual manned and unmanned systems and will have access to information collected by ISR assets belonging to sister organizations and higher echelons, to include Joint and strategic assets. The FCS BCT is programmed to have a total of 8 different ISR systems (air, ground, manned, and unmanned) in addition to the planned combat systems—which come with their own onboard ISR capabilities [2]. Together, these assets will accomplish a variety of collection missions in support of the commander's critical information requirements (CCIR) and intelligence collection plan, to include wide area surveillance (WAS) and reconnaissance, surveillance and target acquisition (RSTA). In addition to the ISR assets currently planned for the FCS BCT, new technologies will undoubtedly be introduced that will increase even further the amount of information available to the commander. Some of these technologies, to include autonomous UAV swarms [3] and self-forming networks of tiny "smart dust" sensors [4], are presently emerging. The data and information made available to the commander combine to form an immense collection of information packages taking many possible forms, including voice transmissions, text messages, video, still imagery, and radar tracking data, as well as data from magnetic, acoustic, seismic, chemical/biological, and weather sensors. The "see first" component of the Army's vision appears well on its way to being realized. "Understanding first," however, specifically the Army's capacity (or relative lack thereof) to process and prioritize the vast amounts of intelligence collected, will remain an enormous challenge into the foreseeable future.

The problem of finding a small subset of relevant information in a very large, rapidly changing data set is not new. Data mining and sensor data fusion approaches have been used recently to address this issue (see [5] for example). This research, however, focuses on a separate—yet related—problem: total information flow in support of decision making. This research emphasizes the dynamic nature of information flow and the interaction between information processing and the commander's decision making process. Here we introduce a dynamical battlefield information flow model. The model is designed so that it is able to accommodate a variety of important factors such as information volume and value, efficiency of information handling, commander's feedback, and random factors. We first introduce an existing descriptive model, the Dynamic Model of Situated Cognition, as a lead-in to the dynamic information flow modeling that follows.

The Dynamic Model of Situated Cognition (DMSC) [6] was first introduced in 2003 and has quickly gained acceptance in the Department of Defense community as a descriptive model of information processing and information flow on the battlefield. Shown in Figure 1, this model illustrates the relationship between technological systems and human perceptual and cognitive processes and provides an effective framework within which to view information flow in a complex system. While not an analytical model, the DMSC does provide a means of viewing the information flow problem in a way that is accessible to analysts and decision makers alike.

The technological portion (the left side) of the model characterizes information as being present in one or more successively filtered "ovals." Oval 1 is Ground Truth, consisting of all available information at a given point in time. Oval 1 is a completely accurate description of ground truth and is constantly updated (and therefore dynamic). Oval 1 contains information about friendly and enemy forces, to include locations and sizes of forces and available weapons systems, as well as less tangible information such as commanders' intent and troop morale. The shapes shown in Oval 1 represent individual information "packages" pertaining to friendly and enemy entities, their disposition, and commanders' intent as well as data on noncombatants, weather, terrain and other environmental conditions.

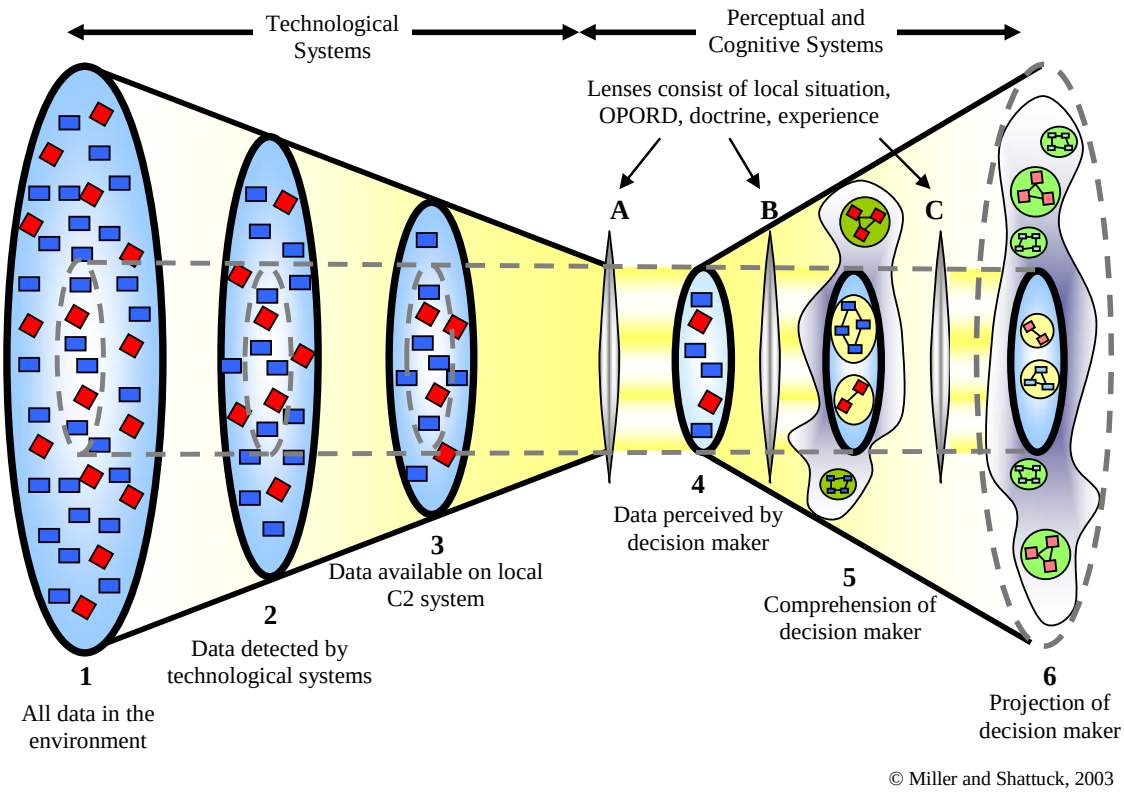


Figure 1 Dynamic Model of Situated Cognition (from [6])

Oval 2 is, in general, an error-prone subset of the information contained in Oval 1 and represents the complete set of information that has been detected or collected by technological systems (i.e., sensors). Two types of error are found in Oval 2—information can be *missed* entirely, or it can be *misrepresented*. Information can generally be misrepresented in two ways: an entity can be represented as a threat when it is not (e.g., a decoy or a noncombatant), or, conversely, a true threat can be represented as a non-threatening entity. Accurate and inaccurate representations are characterized by the various shapes contained in Oval 2 in Figure 1. As with Oval 1, Oval 2 is dynamic—the information contained in it changes with time.

Oval 3 contains all information shown on a decision maker's command and control (C2) display. It contains a filtered, fused subset of the information contained in Oval 2. The future commander is expected to have the ability to tailor his own display by accepting various levels of autonomous configuration in combination with manual

configuration controls (zoom, filtering, field of view, etc.). Clearly, any error present in Oval 2 can propagate to Oval 3 (and beyond). Furthermore, additional error is introduced through inaccurate fusion algorithms and improper filtering techniques. Again, the representation of information in Oval 3 is dynamic.

The three remaining ovals comprise the cognitive portion of the model. Oval 4 contains all information *perceived* by the decision maker. Information contained in Oval 4 is a function of that in Oval 3, but not (usually) a one-to-one correspondence. Oval 5 represents the *comprehension* of the decision maker based on perceived information and his own individual *lens*. Finally, Oval 6 represents the *projection* or *prediction* of the decision maker.

Preceding each of the last three ovals is a lens signifying individual *filters* through which information passes as a decision maker moves from seeing to perceiving, then to comprehending and finally projecting. These lenses are based on the individual's experience level, training, operational orders (OPORDs), guidance and policy, and localized factors such as fatigue or fear. Lenses can be *skewed*—resulting in errors in perceiving, comprehending, and/or predicting—based on errors that have propagated from Ovals 1-3 and lack of experience and/or training, or fear, fatigue and other factors [6].

The dynamic nature of the DMSC is seen in Figure 2. Information can be thought of as flowing from one oval to the next, and decisions are reflected as feedback within the system, dynamically altering the contents of each oval. Feedback can take the form of actions in the real world (Oval 1), redistribution of sensors (Oval 2), and information filtering on local C2 systems (Oval 3). For example, based on what he comprehends about the enemy situation, disposition, and intent, the commander may decide to engage the threat. This will cause a change in ground truth (Oval 1)—due, perhaps, to enemy or friendly casualties, or enemy displacement—and cascading changes in all subsequent ovals [7]. Another, less obvious, example of feedback in the system takes place when the commander modifies existing information processing priorities. This will result in a change to the information he sees on his C2 display (Oval 3), which in turn affects his perception, comprehension and projection (Ovals 4-6).

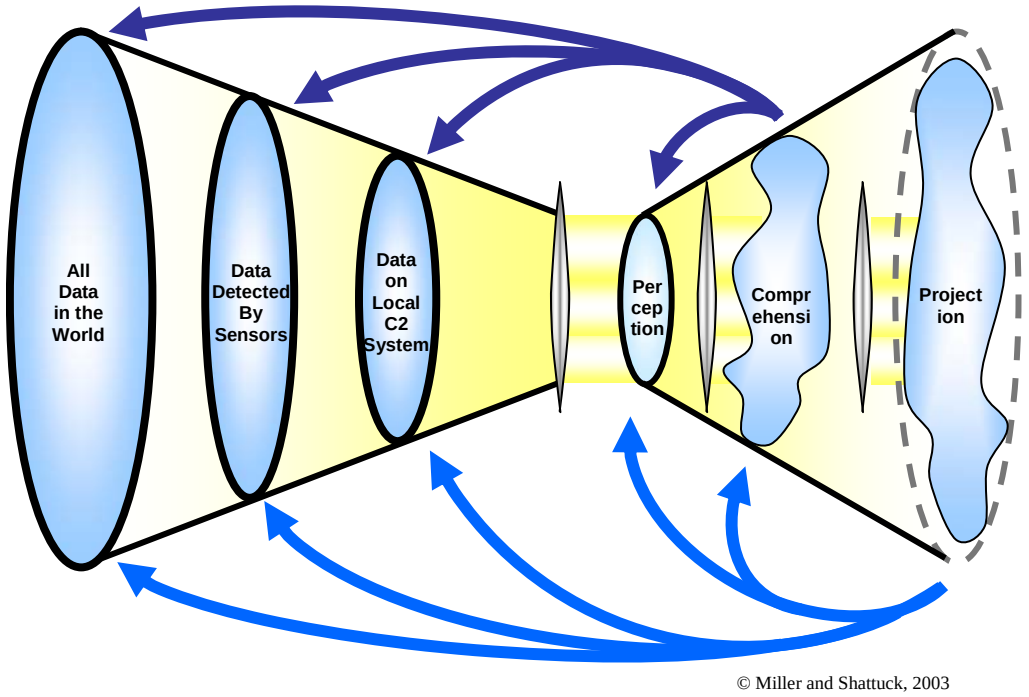


Figure 2 Representation of feedback in the Dynamic Model of Situated Cognition (from [8])

To this point, we have described a system within which information on the battlefield dynamically flows and changes based on controllable factors (friendly decisions and/or actions) as well as uncontrollable factors (enemy decisions/actions, random factors). In the following section, we describe a family of mathematical models that demonstrate the potential to 1) accurately portray this information flow system, and 2) inform the process by which information flow is controlled and prioritized.

B. DEFINING AND SCOPING THE PROBLEM

Figure 3 illustrates several potential mathematical models and the portions of the DMSC they align with. First, we have a dynamic sensor allocation/reallocation model (corresponding to Ovals 1 and 2 of the DMSC). This model could potentially produce an allocation or reallocation plan for organic sensor platforms based on the following inputs, among others.

- Information on organic sensor platforms—location, disposition, capabilities, range, maximum speed, cruising speed, etc.
- CCIR—information elements that directly contribute to successful mission accomplishment [9]
- Likely threat disposition and intent
- Friendly mission

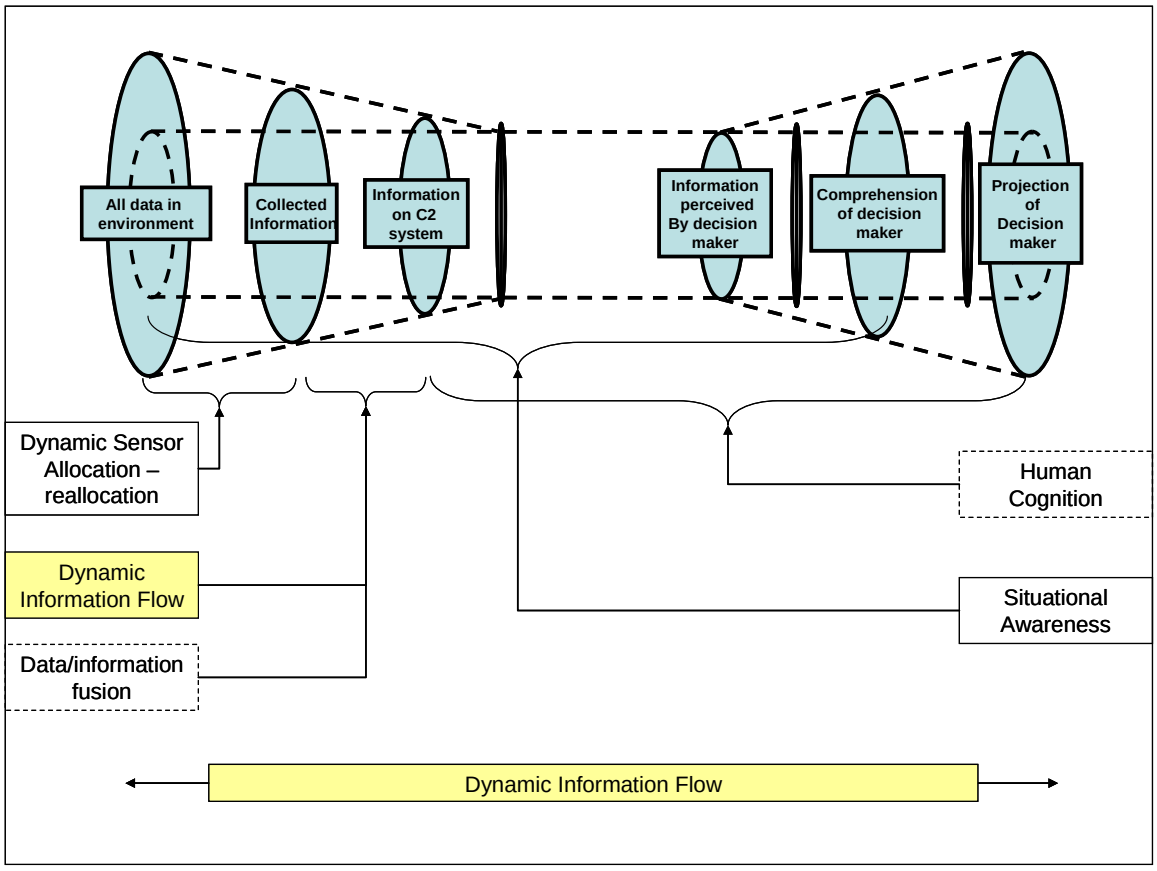


Figure 3 Family of Dynamical System Models

Data and information fusion algorithms and models can also be related to the DMSC. They reside in the space between Ovals 2 and 3, taking raw, unprocessed data and producing processed, fused and filtered information for display on a C2 system. Additionally, human cognition models (Ovals 3-6) representing naturalistic decision making and other approaches would be appropriate within the context of the DMSC, as would models that represent the dynamics of a commander's situational awareness

(Ovals 1-5). These four modeling areas (sensor allocation, fusion, human cognition, and situational awareness) are well-developed in the literature (see [5, 10-13] for examples) and are mentioned for contextual purposes only. They are beyond the scope of this research.

This research, then, focuses on the modeling, analysis, and simulation of dynamic information flow within the command and control system (the physical system) as shown in Figure 3. There are two references to this modeling area in Figure 3: modeling the dynamics of information flow within the entire system remains a long-term research goal; the true focus of this research, however, is on modeling the dynamical flow of information between Ovals 2 and 3. Herein lies the bridge between a commander's ability to see first and understand first. This modeling area will be developed in detail in subsequent chapters.

C. REVIEW OF PREVIOUS RESEARCH

Two main areas in the literature are touched upon in this research: the family of Quadratic Assignment Problems (QAPs) and the area of sequencing and scheduling. We shall address each of these in this literature review.

1. Quadratic Assignment Problem

Koopmans and Beckman [14] first introduced the Quadratic Assignment Problem (QAP) in 1957 as a mathematical model for the assignment of *n* "indivisible economic activities" (i.e., plants) to *n* locations. The general QAP is known to be NP-complete [15, 16]. Burkard, *et al*. [17] identify special cases of the QAP having polynomial-time solutions. Additionally, Burkard, *et al*. [17-20], Cela [21, 22], and most recently Demidenko [23] in 2006 prove a small number of special cases of the QAP having not only polynomial-time solutions, but solutions that can be expressed in closed-form (so called "easy" cases). See Chapter III for a detailed summary of their findings.

2. Sequencing and Scheduling

The body of literature on sequencing and scheduling problems is rich, encompassing, among others, the areas of *requirements generation* ("open shop"—where customer demands create requirements directly, or "closed shop"—where requirements are in the form of inventory replenishment decisions), *processing complexity* (one stage or multi stage; single or parallel processors; flow shop or job shop), and *scheduling criteria* (schedule cost, schedule performance). Baker [24] provides a nice introduction to the topic. Graves [25] provides a fairly comprehensive review of the production scheduling literature with an emphasis on these three areas. Here we focus on the requirements generation portion of the problem, particularly what is known as the *sequential assignment problem.*

Derman, *et al*. [26], derive a recursively defined optimal policy for assigning *n* jobs (arriving in sequential order) to *n* servers, where associated with each job *j* is a random reward, X_j, and associated with each server *i* and job *j* is a known constant p_{ij}, $0 \le p_{ij} \le 1$ measuring the quality of the server ($p_{ij} = 1$ indicates a *perfect* server). The optimal policy assigns the *n* jobs to the *n* servers so as to maximize the total expected reward. Total reward is defined to be $\sum_{j=1}^{n} p_{ij} X_j$. Albright [27] extends the results in [26] to account for discounted rewards, where the discount is a function of time. Agrawala, *et al*. [28], derive a result that minimizes expected total flow time (the sum of job completion times) by non-preemptively sequencing *n* identical jobs (with exponential service times) to be serviced by *m* non-uniform processors. Coffman, et al. [29], address the problem of minimizing expected makespan (maximum job completion time) by assigning *n* jobs to *m* uniform processors with exponential service times. Optimal scheduling rules are derived for two- and three-processor systems. Righter [30] derives results that minimize a weighted function of job completion times for *n* jobs assigned to *m* processors non-preemptively. Service time depends on the server (not the job) and has increasing failure rate (known as IFR). Ross [31] states an optimal policy that maximizes the expected total reward obtained from sequencing *n* jobs for assignment to a single

processor. Each job *i* has an associated independent random service time, X_i, and reward $\alpha^t R_i$ for completion at time *t*, where α $(0 < \alpha < 1)$ is a discount factor. Voutsinas and Pappis [32] derive a similar result for power function rewards (i.e., functions of the form $V_i(t) = K_i t^{a_i}$, for $a_i < 0$). Some of the analytical results we derive in Chapter III are in the area of single-processor sequential assignment problems, and are directly related to—and in some cases extensions of—the work done by Righter [30], Ross [31], and Voutsinas and Pappis [32].

THIS PAGE INTENTIONALLY LEFT BLANK

II. MODEL DEVELOPMENT

In this section, we construct two models that relate to the physical system described in Chapter I. The main model we focus on is a queuing/job sequencing model (section C); a companion model—a discrete time information flow model—is introduced in section E. We begin by stating the necessary definitions and assumptions.

A. DEFINITIONS

Some key terms are defined below.

1. Information Package

An information package is defined as a discrete unit of information in one of many forms (e.g., a single text message, a single still image, or a single video clip). Information packages can be differentiated by processing time required, size (e.g., megabytes), value or priority, reliability "decay" rate, source (type sensor, configuration), and other relevant factors. Information packages are represented as "jobs" to be serviced (or processed) in the queuing/job sequencing model, and are chosen for processing (individually) based on their value. In the discrete time model, information packages are represented in aggregate, with total remaining processing time (workload) in the system the metric of interest. Some examples of information packages include:

- A two-minute video clip received from a high-altitude unmanned aerial vehicle (UAV) showing suspected militants arriving at a meeting location
- A 30-second voice transmission comprising a situation report on the location and activity of a threat air defense artillery battery
- A three-megabyte satellite image of a two-by-two kilometer zone within a city of interest

Multiple information packages can describe the same target. Each distinct piece of information, whether individual reports from different sources or multiple reports from the same source, is treated as an individual information package.

2. Information Package Value

We assume throughout this analysis that each information package can be assigned a value function that describes how its value changes over time. We further assume that the value function associated with an information package is a non-increasing function of time. The value function types considered throughout this analysis consist of the following:

- Exponential: $Val_j(t) = \beta_j e^{-\alpha_j t}$
- Linear: $Val_j(t) = \beta_j - \alpha_j t$
- Step: $Val_j(t) = \begin{cases} \beta_j & 0 \le t \le \tau_j \\ 0 & \text{otherwise} \end{cases}$

Initial value and the rate at which value decays for each information package are defined by the function parameters β, α, and τ. For example, an information package containing the location and velocity of a high value, time sensitive target (e.g., a sedan carrying the head of Al Qaeda in Iraq) could have a value function that is exponential with relatively high β (initial value) and α (value decay rate). We say that value is *realized* once an information package has been processed (see Figure 5 below).

B. ASSUMPTIONS

We make the following assumptions throughout this analysis. Other assumptions will be specified as appropriate.

- Realized value is linearly additive.
- Information package value is nonnegative.
- Information package arrival and processing rates are constant with respect to time (i.e., there is no "seasonality" present in these rates).
- Effects on the system due to bandwidth constraints are not considered.
- Service will not be preempted; once an information package is assigned to a processor, it will not exit until processing is complete.
- Processors are equally-capable (i.e., "uniform").

C. QUEUING/JOB SEQUENCING MODEL FOR INFORMATION VALUE

1. General Model

We represent the battlefield information flow system as a multi-class queuing system with general inter-arrival time and service time distributions and multiple servers/processors (denoted a *G*/*G*/*m* multi-class queue), with the following defined characteristics:

- $j = 1, 2, \ldots, J$ information package (IP) index.
- $k = 1, 2, \ldots, K$ class[1] index.
- S_j service time length for IP *j* (random variable).
- λ_j arrival rate for IP *j*.[2]
- μ_j service rate for IP *j*.[2]
- A_j arrival time of IP *j* (random variable)
- χ_j the time IP *j* is assigned to a processor (this is our *control*). We assume that processing begins immediately upon assignment.
- D_j service completion (system departure) time for IP *j*. D_j is function of both χ_j and S_j (specifically, $D_j = \chi_j + S_j$)
- $Val_j(t)$ value of IP *j* at time *t*

Often in queuing applications we assume exponential inter-arrival and service times; in this case, λ_j and μ_j represent the parameters of these distributions. The models described in this section, however, do not require the assumption of exponential inter-arrival and service times; we therefore consider λ_j and μ_j as rates in the general sense.

---

[1] The number of classes in the physical system is small relative to the number of information packages present; common examples of information package classes include video, still imagery, voice, and text.

[2] Arrival and service distributions (and, hence, rates) are thought of as functions of IP class (thus the system has at most *K* unique arrival and service time distributions); however, for notational simplicity, we assign arrival and service rates to each IP (thus the subscript *j* rather than *k*).

Prior to entering the queue, information packages (IPs) are assumed to undergo a "pre-processing" stage, where value functions and class are assigned to each information package; see Figure 4. Value functions are piecewise continuous functions of time, and are described in detail in Section A.

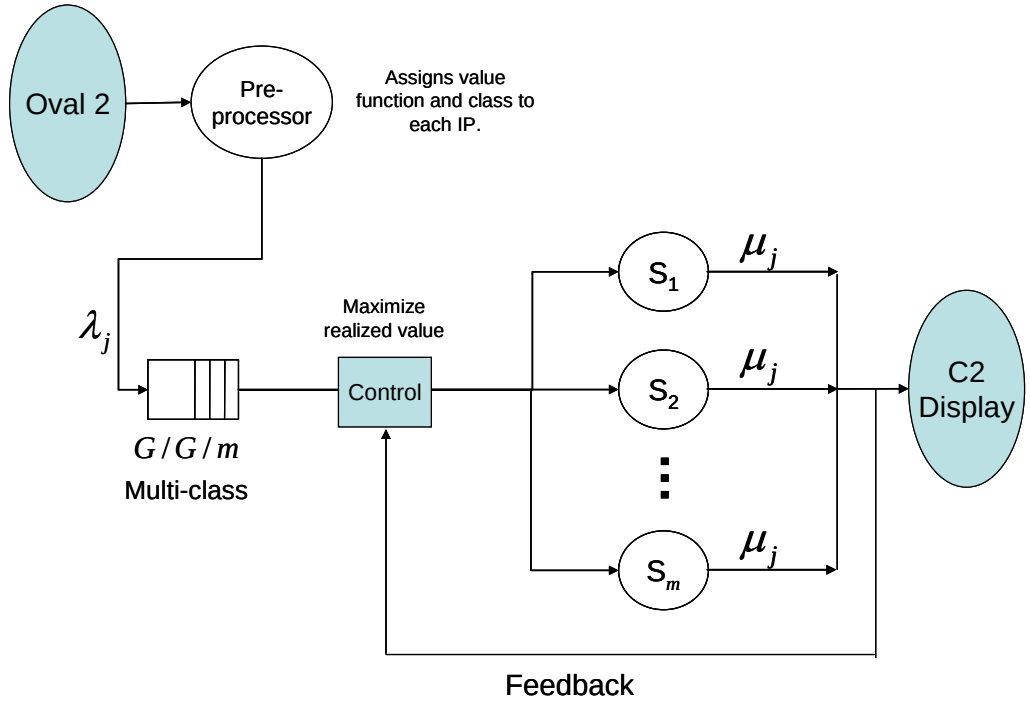


Figure 4 *G*/*G*/*m* Multi-class Queuing System Diagram

Information packages arrive into the queue at rate λ_j and wait to be selected by a controller for processing by one of *m* servers, assumed to be equally capable. We assume that information package arrival rates far outpace the system's processing capacity—rendering the system inherently unstable (queue length grows without bound, along with expected wait times)—which mirrors the actual command and control system under study (which we will refer to as the "physical system").

The role of the controller is to establish a dynamic service policy that results in the maximum value "realized" by the system. To be more precise, we define the realized value associated with package *j* as follows:

$$V_j^r = Val_j\left(d_j\right) = Val_j\left(\chi_j + s_j\right) \tag{2.1}$$[3]

As this definition implies, realized value is computed upon *completion* of service—see Figure 5. The departure time of a package is not deterministic, however, so we define expected realized value associated with package *j* in (2.2).

$$E\left[V_j^r\right] = E\left[Val_j\left(D_j\right)\right] = E\left[Val_j\left(\chi_j + S_j\right)\right] \tag{2.2}$$

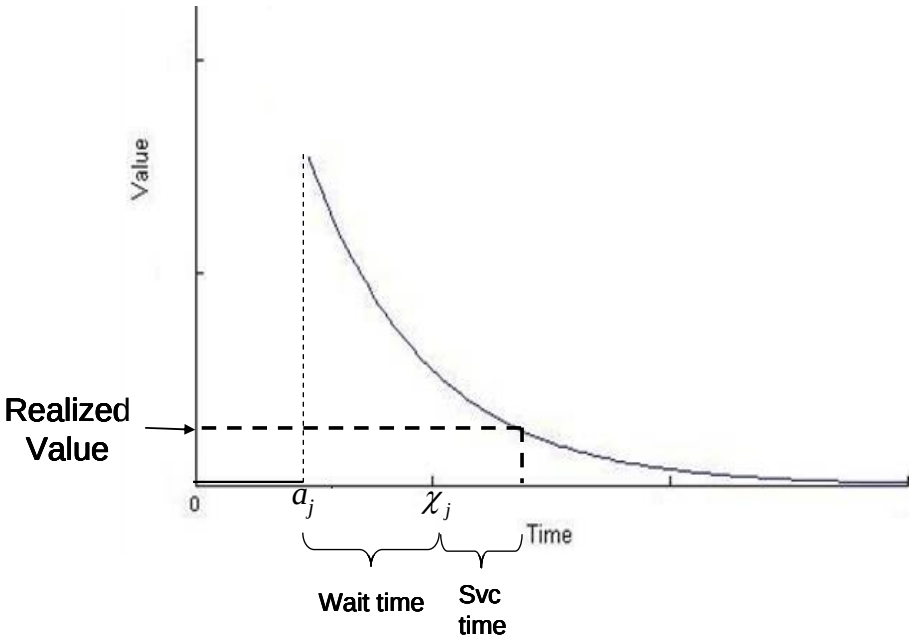


Figure 5 Measuring Realized Value

Finally, we define the *cumulative* expected realized value obtained from processing a set of *Q* information packages as follows:

$$E\left[V^r\right] = E\left[\sum_{j=1}^{Q} Val_j\left(D_j\right)\right] = \sum_{j=1}^{Q} E\left[Val_j\left(\chi_j + S_j\right)\right] \tag{2.3}$$

[3] The lower-case d_j and s_j are used here to denote realizations of the random variables D_j and S_j, respectively.

2. Simplified Model and Variations

We start with a simplified model of the queuing system in order to obtain analytical results. To begin, we assume a *G*/*G*/*m* queue with time independent arrival and service times and no service preemption. We also assume zero time delays due to pre-processing (assigning information package class and value function prior to entrance into the queue), server assignment decisions, and bandwidth constraints. Finally, we assume that value is additive; in other words, value realized from processing various information packages can be linearly aggregated.

The decision process starts when a processor becomes available, a time which we will call t_d (or decision time). Let *Q* be the number of information packages present in Oval 2 at time t_d.[4] An iterative approach is employed as follows.

- Sequence the *Q* information packages in such a way as to maximize cumulative expected realized value obtained, assuming all *Q* information packages will be processed. This sequence becomes the *initial* processing "plan" or strategy, and may be updated as necessary.
- Begin executing this processing strategy by processing the first information package in sequence.
- When a processor next becomes available, decide to either continue with the original strategy, or modify this strategy by updating the sequence. The choice to update the sequence is made if:
  - ◊ New information packages have arrived, or
  - ◊ Actual service time of the first information package differs significantly from the expected service time.
- If updating is warranted, redefine *Q* as appropriate and re-sequence.
- This new sequence becomes the "current" plan or strategy, and execution is begun on it by sending the first package in sequence to the available processor.
- Continue iterating in this fashion until stopping criteria are met.

In this approach, we update (or at least consider updating) the current strategy *every time a processor becomes available*. Of course, other update alternatives exist;

[4] *Q* may instead represent a subset of the information packages present in Oval 2 at time t_d if it makes sense operationally to omit some information packages from consideration.

e.g., we could choose to abide by the original plan for a set amount of time or for a set number of information packages before updating the plan/sequence. We choose the updating scheme outlined above because it most closely matches the strategy that would be employed in the physical system.

We now state the optimal control problem for any set of *Q* information packages. The solution to this problem is the processing sequence that maximizes cumulative expected realized value.

Let *Q* be the number of information packages in the set, let $q = 1, 2, \ldots, Q$ be the identifying index for each information package,[5] and let $g = 1, 2, \ldots, m$ represent the servers present in the system. Additionally, define the following terms:

- n_g number of IPs assigned to server *g* (note that $\sum_g n_g = Q$)
- $x_{gh} \in \{1, 2, \ldots, Q\}$ IP to be processed h^{th} by server *g* (this is the "decision variable")
- φ_Q the set of all possible permutations of the sequence $\langle 1, 2, \ldots, Q \rangle$

Then, the optimal control problem for the queuing/job sequencing model is as follows:

$$\max_{\{x_{gh}\} \in \varphi_Q} E\left[V^r\right] = \sum_{g=1}^{m} \sum_{h=1}^{n_g} E\left[Val_{x_{gh}}\left(D_{x_{gh}}\right)\right] \tag{2.4}$$

This model seeks the processing sequence of information packages (represented by x_{gh}) that results in the maximum cumulative expected realized value, and will be developed in detail in Chapter III. Figure 6 below depicts a typical information package assignment outcome.

[5] *q* is a "temporary" package index; each package still maintains its permanent index, *j*.

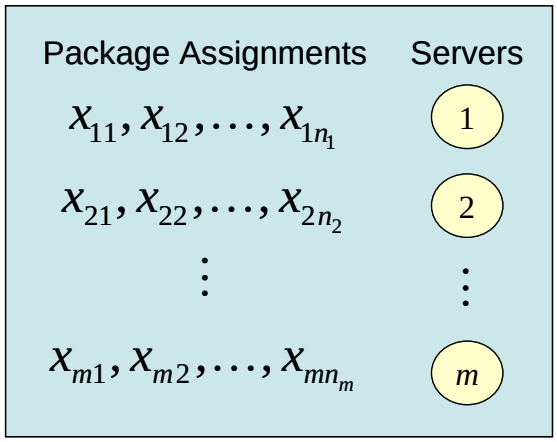


Figure 6 Information Package assignment to servers

As we shall see in Chapter III, (2.4) is, in general, NP-hard.[6] We develop solutions for some special cases in Chapter III, and resort to heuristic strategies for more complex cases in Chapter IV.

D. DATA

For the simulation experiments described in Chapter IV, we require reasonable estimates of the distributional parameters for the random variables defined in the previous section. We turn to a set of experimental data obtained during a recent series of C2 experiments conducted by the Defense Advanced Research Projects Agency (DARPA) to at least partially satisfy this requirement. See [33] for details on this experiment. Table 1 depicts an excerpt of the data collected during this effort. Each row represents an information package, with associated information about the target (enemy element), friendly spotter, arrival times to Ovals 2 and 4 of the Dynamic Model of Situated Cognition, and the total time it took the package to go from Oval 2 to Oval 4.

[6] A problem Π is NP-complete if $\Pi \in \text{NP}$ and, for all other problems $\Pi' \in \text{NP}$, Π' transforms to Π. A problem Π is NP-hard if there exists some NP-complete problem Π' that Turing-reduces to Π [16].

Info Pkg #	Enemy Element	Friendly Source	First Spotted Time (**Arrived Oval 2**)	Friendly Spotter	Time HTR Classified (**Arrived Oval 4**)	Time from O2-O4 (min)
12	DRA1-2	IntelDump	13:27:08	ARVCAU2	15:27:39	120.5
13	PUR5-5	IntelDump	13:27:08	UAV2CAU21	15:32:44	125.6
14	GAR3-14	IntelDump	13:27:08	HHQ0U4	15:41:30	134.4
15	DAR6-2	IntelDump	13:27:08	UAV43	15:49:19	142.2
16	DRA1-12	IntelDump	13:27:08	UAV42	15:58:00	150.9
17	DRA1-5	IntelDump	13:27:08	UAV42	15:59:14	152.1
18	SA15-6-1	HHQ0U5	13:38:27	HHQ0U5	14:02:28	24.0
19	RIC5-1	UAV41	13:49:47	UAV41	13:55:45	6.0
20	NON4-3	UAV41	13:50:30	UAV41	14:00:22	9.9
21	BRD6-1	UAV2CAU22	13:51:21	UAV2CAU22	13:51:42	0.3
22	DRA1-D6	UAV2CAU22	13:53:30	UAV2CAU22	13:53:44	0.2
23	DRA1-D5	UAV2CAU22	13:53:30	UAV2CAU22	13:54:17	0.8
24	FT4-33	UAV1CAU12	13:54:38	UAV1CAU12	13:54:56	0.3
25	SA13-6-1	UAV2CAU22	13:56:00	UAV2CAU22	13:56:17	0.3
26	URA4-1	UAV41	13:57:05	UAV41	14:27:17	30.2
27	URA8-2	ARVCAU1	13:57:29	ARVCAU1	13:57:36	0.1

Table 1 Excerpt from DARPA C2 Experiment Data (from [34])

Data in the column titled “First Spotted Time” are used to estimate inter-arrival time distributions, and data in the column titled “Time from O2-O4” are used to estimate service time distributions. See [34] for the assumptions inherent in this data set and other details.

Figure 7 below depicts a histogram of inter-arrival time data from the DARPA C2 experiment along with the best-fit exponential distribution (with parameter $\lambda = 0.63$); Figure 8 shows the same for service time data (with parameter $\mu = 0.134$). We conclude that it is reasonable to model inter-arrival and service times exponentially.

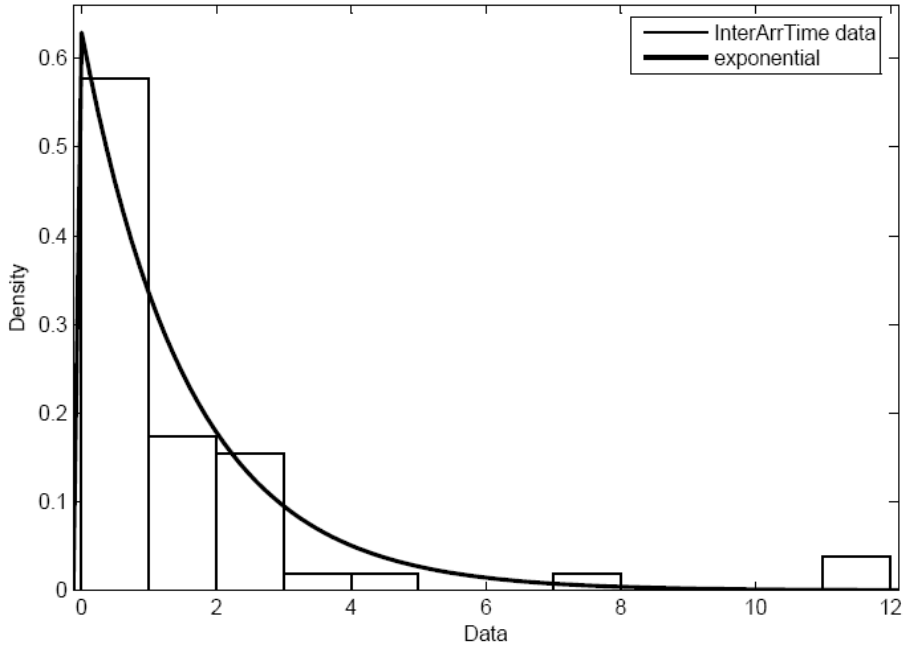


Figure 7 Inter-arrival Time Data with Best-fit Exponential Distribution

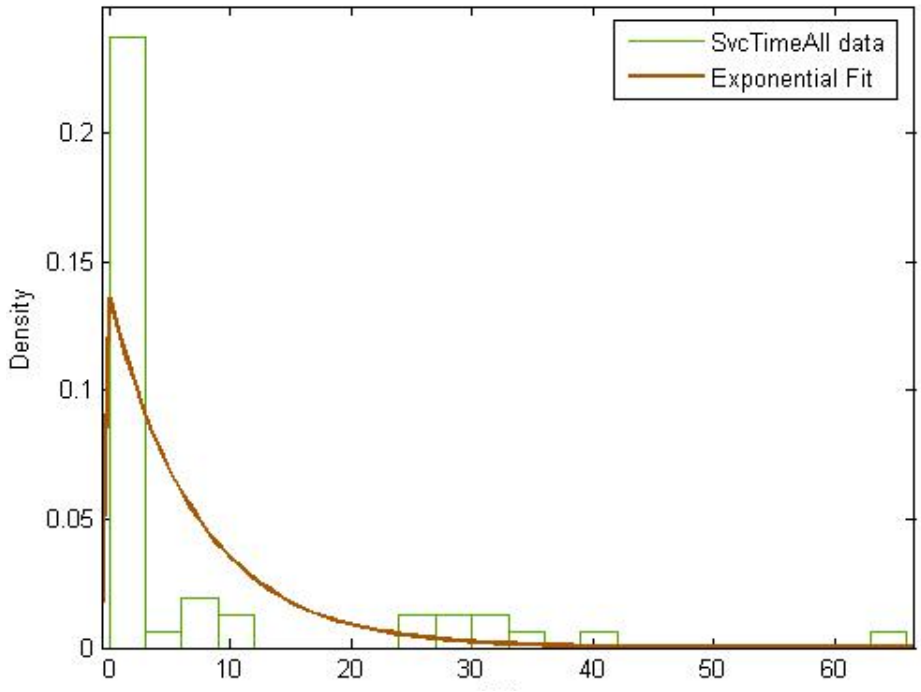


Figure 8 Service Time Data with Best-fit Exponential Distribution

E. DISCRETE STOCHASTIC INFORMATION FLOW MODEL

We now develop a companion model to the queuing/job sequencing model that represents the dynamics of information *volume* (or *workload*) in the system. Figure 9 below illustrates how information passing from Ovals 2 to 3 of the Dynamic Model of Situated Cognition first must pass through an information processing stage. This is the specific context within which the dynamical system model of information flow resides.

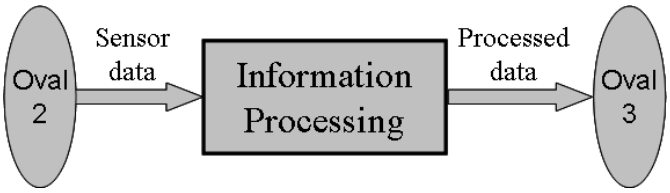


Figure 9 Information Processing Layer in the DMSC

1. Definitions and Assumptions

To develop the model, we first require some definitions and assumptions:

- *required processing time*—a pre-assigned length of time associated with each information package (e.g., a particular still image requires 10 time units to process).
- $x_i(k)$—the number of information packages in Oval 2 at the start of time period *k* for which the *remaining processing time required* is *i* time units.
- $u_i(k)$—the number of information packages arriving in Oval 2 during time period *k* which require a *total*[7] of *i* time units to process.
- Δt—time step size.
- $V(k)$—total volume; a time-dependent measure of the workload present in the system (obtained by summing the remaining processing time values of each information package present in the system (Oval 2) for each time period *k*).

[7] Note the difference between *total* and *remaining* processing time required in the definitions of $x_i(k)$ and $u_i(k)$. When an information package enters the system, it arrives with an assigned total required processing time—and the clock starts. As time progresses in the model, the required processing time is decremented appropriately—yielding the remaining processing time required tracked by $x_i(k)$.

2. Model Development

We now define the Discrete Information Flow Model as follows:

$$x(k+1) = Ax(k) + Bu(k) \tag{2.5}$$

where matrices *A* and *B* are parameters of the system, with *B* initially defined as the identity matrix, and *A* (known as the "upshift" matrix) initially defined as follows:

$$A = \left[a_{ij} \right],\ i = 1 \ldots m,\ j = 1 \ldots n, \text{where } a_{ij} = \begin{cases} 1, & \text{if j} = i+1 \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

The vectors *x* and *u* contain *n* elements—corresponding to the *n* discrete processing time "bins" of size Δt. The *m* rows of *A* correspond to the *m* time steps of size Δt represented in the model (i.e., $m \cdot \Delta t$ is the total time modeled).[8] *A* is a matrix containing all zeros except for the off-diagonal immediately above the main diagonal, which contains all ones. *A* is necessarily square, meaning we must restrict ourselves to cases where *m* = *n*. This defines the completely deterministic (and most simplistic) case, where the required processing time for each information package in the system is decremented Δt time units every time step, and the number of packages arriving each time step (*u*) is fully known.

We now address a more complex case, where the input vector *u* is a function of two random processes—one each for information package *arrival* and *processing* times. We can think of this as analogous to a birth-death process [35], where information packages arrive (birth) and depart (death) from Oval 2 at known (or estimated) rates.

[8] Processing time bin size and time step size may be chosen so as to not be equal; in general, we will keep them of equal size.

Define the following random variables:

- Y_j - time between arrivals of packages $j-1$ and j (referred to as inter-arrival times).
- $T_j = \sum_{q=1}^{j} Y_q$ - arrival time of package *j*.
- P_k - number of packages arriving in Oval 2 in time period *k*.
- R_j - required processing time for package *j*.

Then,

$$P_k = \sum_j \eta\left(\left\lceil \frac{T_j}{\Delta t} \right\rceil - k\right), \text{ where } \eta(x-a) = \begin{cases} 1, & x = a \\ 0, & x \neq a \end{cases}. \tag{2.7}$$

The stochastic version of the discrete information flow model now becomes

$$x(k+1) = Ax(k) + Bu(k)$$

where

$$u_i(k) = \sum_{q=1}^{P_k} \eta\left(\left\lceil \frac{R_q}{\Delta t} \right\rceil - i\right) \tag{2.8}$$

Finally, we can formally define total volume as

$$V(k) = \sum_{i=1}^{n} i \cdot x_i(k). \tag{2.9}$$

3. Model Validation

To test how well the stochastic version of the discrete information flow model fits the actual data, a simple Monte Carlo simulation is developed (see Appendix B for MATLAB code). We utilize the metric $V(k)$ (volume) to facilitate this comparison.

Each run of the simulation proceeds in the following manner:

- Randomly draw inter-arrival times and service times from the appropriate distributions[9]
- Choose Δt ($\Delta t = 2$ min for this assessment)
- Compute $u(k),\ k = 1 \ldots m$ from (2.7) and (2.8).
- Compute $x(k),\ k = 1 \ldots m$ from (2.5) and (2.6).
- Compute $V(k),\ k = 1 \ldots m$ from (2.9).

Figure 10 displays the results of 1000 runs of this simulation plotted with the original experiment data.

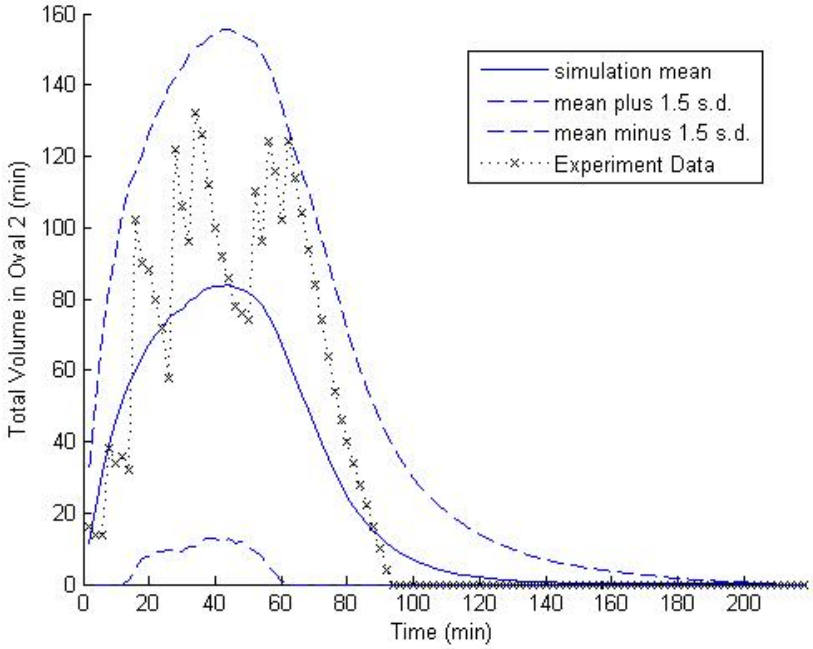


Figure 10 Simulation Results

[9] In this case, we used Gamma distributions (with parameters estimated from the DARPA C2 Experiment data) to generate inter-arrival and service times. It turns out that the Gamma distribution produced a slightly better fit to the DARPA data than did the exponential distribution for both inter-arrival and service times.

This graph plots total volume (workload) in the system over time for simulated results and actual experiment data. The solid line represents the mean total volume for the 1000 simulation runs; dashed lines are $\pm$ 1.5 standard deviations from the mean. The "x's" represent the original DARPA experiment data. From this, we observe that the stochastic model consisting of (2.5) and (2.8) is a reasonable representation of the dynamics and random processes present in the actual data.

THIS PAGE INTENTIONALLY LEFT BLANK

III. ANALYSIS AND COMPLEXITY RESULTS

We now develop the model (2.4) in some detail. All cases in this chapter assume a single processor ($m=1$). We redefine the decision variable to reflect this assumption as follows: x_q is the information package to be processed q^{th}. All results presented are of the "static"[10] type; i.e., we assume a set of information packages of size Q and seek to process them in the sequence that results in maximum realized value attained. We assume no new arrivals and no service preemption. Relative to the most realistic cases of the physical system, this problem is very much oversimplified. However, this simplification is necessary for optimality and complexity analysis, which provides fundamental insights for the overall problem and supports further exploration of practical solutions.

A. STEP VALUE FUNCTIONS

Consider the case where information package value functions take on the form of step functions; that is, consider value functions of the form

$$\begin{aligned} Val_q(t) &= \beta_q H\left(t-a_q\right) H\left(a_q+\tau_q-t\right) \\ &= \begin{cases} \beta_q & a_q \le t \le a_q+\tau_q \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.1}$$

where $H(t)$ is the unit (or Heaviside) step function, a_q is the arrival time of information package q and $\beta_q, \tau_q > 0$. See Figure 11.

[10] See Chapter IV for details on static and dynamic modeling.

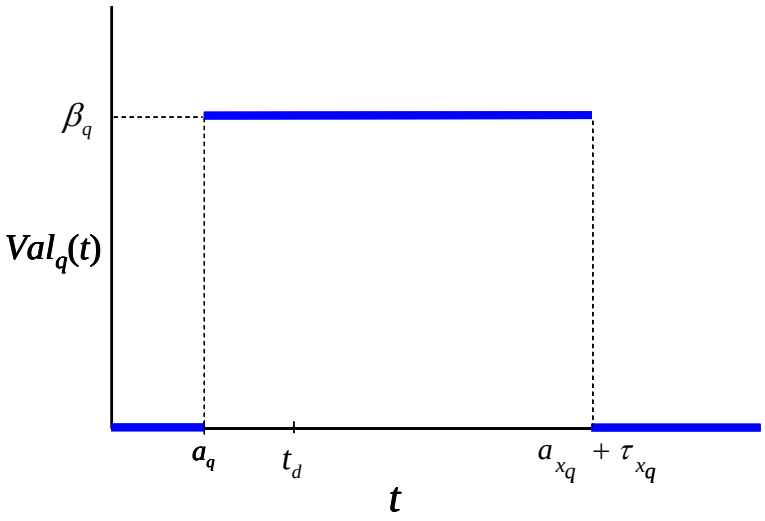


Figure 11 Step Value Function

We assume that $a_q < t_d < a_q + \tau_q$, $\forall q$. Suppose also that service times s_q are deterministic and that $m = 1$. Then maximizing V^r is equivalent to

$$\max_{\{x_q\} \in \varphi_Q} \sum_{q=1}^{Q} Val_{x_q} \left(t_d + \sum_{h=1}^{q} s_{x_h} \right) \tag{3.2}$$

$$= \max_{\{x_q\} \in \varphi_Q} \sum_{\left\{ q : \sum_{h=1}^{q} s_{x_h} \le a_{x_q} + \tau_{x_q} - t_d \right\}} \beta_{x_q} \tag{3.3}$$

$$= \min_{\{x_q\} \in \varphi_Q} \sum_{\left\{ q : \sum_{h=1}^{q} s_{x_h} > a_{x_q} + \tau_{x_q} - t_d \right\}} \beta_{x_q} \tag{3.4}$$

Expression (3.3) equates to maximizing the combined realized value of the information packages that can be processed prior to their value becoming zero (at time $t = a_q + \tau_q$), while, equivalently, (3.4) seeks to minimize the combined realized value of information packages that cannot be processed "in time." We will refer to (3.4) as Problem *I*.

Theorem 3.1: Problem *I* is NP-hard.

Proof. We first define the *decision problem* corresponding to the optimization problem *I*: Let $K > 0$. Does there exist a permutation $\langle x_1, x_2, \ldots, x_Q \rangle \in \varphi_Q$ such that

$$\sum_{\left\{q:\sum_{h=1}^{q} s_{x_h} > a_{x_q} + \tau_{x_q} - t_d\right\}} \beta_{x_q} \leq K \, ? \tag{3.5}$$

We will refer to decision problem (3.5) as I'.

Consider now the known NP-complete problem "Sequencing to Minimize Tardy Task Weight" (short name SS3) in [16], defined as follows:

Let $q = 1, 2, \ldots, Q$ tasks, each having length $l(q) \in \mathbb{Z}^+$, weight $w(q) \in \mathbb{Z}^+$, and a deadline $d(q) \in \mathbb{Z}^+$. Let K be a positive integer. Is there a permutation $\langle x_1, x_2, \ldots, x_Q \rangle \in \varphi_Q$ such that

$$\sum_{\left\{q:\sum_{h=1}^{q} l(x_h) > d(x_q)\right\}} w(x_q) \leq K \, ? \tag{3.6}$$

Problem SS3 (3.6) is clearly a special case of problem I', where $l(q)$, $w(q)$, and $d(q)$ (all positive integers) are analogous to s_{x_h}, β_q, and $a_q + t_q$, respectively. The listed quantities for Problem I', however, are not restricted to integer quantities. It follows, then, that problem I' is NP-hard (the general problem is at least as hard as a special case). Furthermore, we conclude that Problem *I* is at least as hard[11] as Problem I', and, hence, NP-hard as well. □

[11] If the solution to Problem *I* is known, then the solution to Problem *I'* is known for any *K*. The converse is not true, in general.

Consider now the case where information package value functions take on the form of "multi-step" functions, shown in Figure 12 below. Corollary 3.2 follows immediately from the fact that the value function in (3.1) is a special case of the one in Figure 12.

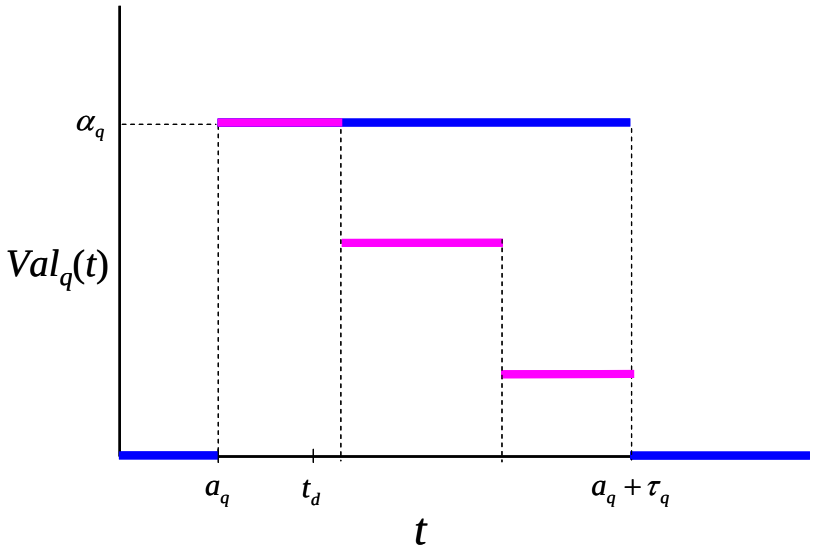


Figure 12 Multi-step Value Function

Corollary 3.2: The optimal control problem involving information packages with multi-step value functions is NP-hard.

B. LINEAR VALUE FUNCTIONS

We now present a set of results specifying the optimal processing strategy for a single server ($m=1$), where all information packages have piecewise linear value functions of the form

$$Val_q(t) = H(t-a_q)H(\tau_q - t)\left[\beta_q - \alpha_q(t-a_q)\right] \tag{3.7}$$

$$= \begin{cases} \beta_q - \alpha_q(t-a_q), & a_q \le t \le \tau_q \\ 0, & \text{otherwise} \end{cases}$$

$H(t)$ is the unit (Heaviside) step function, $\alpha_q > 0$ is the rate at which value decays, and $\beta_q > 0$ is the initial value for each information package. Note that $Val_q(t) = 0$ whenever $t < a_q$ or $t \geq \tau_q = \frac{\beta_q}{\alpha_q} - a_q$. See Figure 13 below.

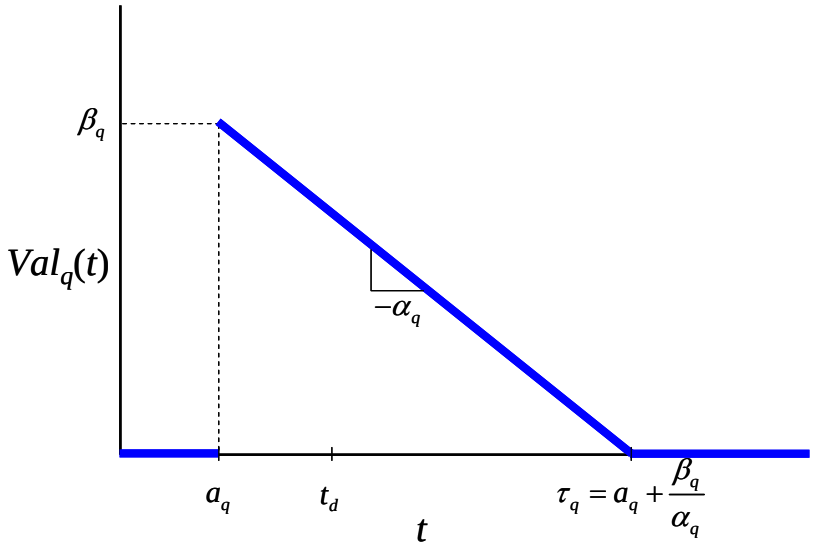


Figure 13 Linear Value Function

Based on Theorem 3.1 and Corollary 3.2, we conjecture that the optimal control problem involving value functions of the type specified in (3.7) is NP-hard. However, as we shall see below, this problem is rendered solvable when one additional condition is met.

1. Main Results

We assume that the *Q* information packages have all arrived by time t_d. If we also assume that *Q* is chosen such that all *Q* information packages can be processed prior to time $t = \tau_q, \ \forall q \in \{1, 2, \ldots, Q\}$,[12] we can simplify the value function as follows.

$$Val_q(t) = \beta_q - \alpha_q(t - a_q) \tag{3.8}$$

[12] Not an unreasonable assumption, it turns out. This assumption can hold for combinations of small *Q*, small α_q, and large β_q. We will see cases in Chapter IV where simulation results affirm the plausibility of this assumption.

Note that, without loss of generality, we can shift the coordinate axes such that $t_d = 0$. With β_q redefined appropriately, the value function for each information package can be simplified as follows.

$$Val_q(t) = \beta_q - \alpha_q t \tag{3.9}$$

The total expected realized value obtained from processing all *Q* information packages is then

$$E\left[V^r\right] = E\left[\sum_{q=1}^{Q}\left(\beta_{x_q} - \alpha_{x_q}\sum_{h=1}^{q} S_{x_h}\right)\right], \tag{3.10}$$

which is the quantity we wish to maximize. It follows that

$$E\left[V^r\right] = \sum_{q=1}^{Q}\left[\beta_{x_q} - \alpha_{x_q}\sum_{h=1}^{q} E\left[S_{x_h}\right]\right] \tag{3.11}$$

$$= \underbrace{\left[\sum_{q=1}^{Q}\beta_q\right]}_{\text{constant}} - \sum_{q=1}^{Q}\sum_{h=1}^{q}\alpha_{x_q} E\left[S_{x_h}\right] \tag{3.12}$$

where the first term in (3.12) represents the total initial value for all information packages and is constant with respect to *x*. So, maximizing $E\left[V^r\right]$ is equivalent to

$$\min_{\{x_h\}\in\varphi_Q} \sum_{q=1}^{Q}\sum_{h=1}^{q}\alpha_{x_q} E\left[S_{x_h}\right]. \tag{3.13}$$

This is the optimal control problem. We now present two results based on (3.13): Lemma 3.3, an intermediate result, assumes only two information packages to be sequenced. Lemma 3.4, the main result in this section, assumes any number of information packages.

Lemma 3.3. Let $Q = 2$ and $m = 1$, and suppose that Q is chosen such that all Q information packages can be processed prior to time $t = \tau_q,\ \forall q \in \{1, 2, \ldots, Q\}$. Let the value function for information package q be $Val_q(t) = \beta_q - \alpha_q t,\ q = 1, 2$. Then processing information package 1 prior to 2 is optimal if and only if

$$\frac{\alpha_1}{E[S_1]} > \frac{\alpha_2}{E[S_2]}. \tag{3.14}$$

Proof. The expected realized value from processing information package 1 then 2 is

$$\begin{aligned} E\left[V^r_{\langle 1,2\rangle}\right] &= E\left[\beta_1 - \alpha_1 S_1 + \beta_2 - \alpha_2 (S_1 + S_2)\right] \\ &= \beta_1 - \alpha_1 E[S_1] + \beta_2 - \alpha_2 E[S_1 + S_2] \end{aligned} \tag{3.15}$$

Processing the information packages in the reverse order yields the following expected realized value.

$$\begin{aligned} E\left[V^r_{\langle 2,1\rangle}\right] &= E\left[\beta_2 - \alpha_2 S_2 + \beta_1 - \alpha_1 (S_1 + S_2)\right] \\ &= \beta_2 - \alpha_2 E[S_2] + \beta_1 - \alpha_1 E[S_1 + S_2] \end{aligned} \tag{3.16}$$

Then

$$E\left[V^r_{\langle 1,2\rangle}\right] > E\left[V^r_{\langle 2,1\rangle}\right]$$

$$\Leftrightarrow \beta_1 - \alpha_1 E[S_1] + \beta_2 - \alpha_2 E[S_1 + S_2] > \beta_2 - \alpha_2 E[S_2] + \beta_1 - \alpha_1 E[S_1 + S_2]$$

$$\Leftrightarrow \alpha_1 E[S_2] > \alpha_2 E[S_1]$$

$$\Leftrightarrow \frac{\alpha_1}{E[S_1]} > \frac{\alpha_2}{E[S_2]} \quad \square$$

Lemma 3.4. Let $Q > 2$ and suppose the remaining assumptions in Lemma 3.3 hold. Then processing the information packages in decreasing order of $\alpha_q / E\left[S_q\right]$ is optimal.

Proof. Suppose the processing sequence $\left\langle q_1, q_2, \ldots, q_j, q_i, \ldots, q_Q \right\rangle$ is optimal, and that $\alpha_{q_i} / E\left[S_{q_i}\right] > \alpha_{q_j} / E\left[S_{q_j}\right]$. Then the expected realized value obtained from processing the information packages in this sequence, $E\left[V^r_{\left\langle q_1, q_2, \ldots, q_j, q_i, \ldots, q_Q \right\rangle}\right]$, is equal to

$$E\left[\beta_{q_1} - \alpha_{q_1} S_{q_1}\right] + E\left[\beta_{q_2} - \alpha_{q_2}\left(S_{q_1} + S_{q_2}\right)\right] + \cdots + E\left[\beta_{q_j} - \alpha_{q_j}\left(S_{q_1} + \cdots + S_{q_j}\right)\right]$$
$$+ E\left[\beta_{q_i} - \alpha_{q_i}\left(S_{q_1} + \cdots + S_{q_j} + S_{q_i}\right)\right] + \cdots + E\left[\beta_{q_Q} - \alpha_{q_Q}\left(S_{q_1} + \cdots + S_{q_Q}\right)\right].$$

Similarly, if we were to switch the processing order of information packages q_j and q_i, (and leave all others unchanged) the expected realized value, $E\left[V^r_{\left\langle q_1, q_2, \ldots, q_i, q_j, \ldots, q_Q \right\rangle}\right]$, would be

$$E\left[\beta_{q_1} - \alpha_{q_1} S_{q_1}\right] + E\left[\beta_{q_2} - \alpha_{q_2}\left(S_{q_1} + S_{q_2}\right)\right] + \cdots + E\left[\beta_{q_i} - \alpha_{q_i}\left(S_{q_1} + \cdots + S_{q_i}\right)\right]$$
$$+ E\left[\beta_{q_j} - \alpha_{q_j}\left(S_{q_1} + \cdots + S_{q_i} + S_{q_j}\right)\right] + \cdots + E\left[\beta_{q_Q} - \alpha_{q_Q}\left(S_{q_1} + \cdots + S_{q_Q}\right)\right].$$

The optimality assumption yields,

$$E\left[V^r_{\langle q_1,q_2,\ldots,q_j,q_i,\ldots,q_Q\rangle}\right] > E\left[V^r_{\langle q_1,q_2,\ldots,q_i,q_j,\ldots,q_Q\rangle}\right]$$

$$\begin{aligned}
\Rightarrow\ & E\left[\beta_{q_1}-\alpha_{q_1}S_{q_1}\right]+E\left[\beta_{q_2}-\alpha_{q_2}\left(S_{q_1}+S_{q_2}\right)\right]+\cdots+E\left[\beta_{q_j}-\alpha_{q_j}\left(S_{q_1}+\cdots+S_{q_j}\right)\right] \\
& +E\left[\beta_{q_i}-\alpha_{q_i}\left(S_{q_1}+\cdots+S_{q_j}+S_{q_i}\right)\right]+\cdots+E\left[\beta_{q_Q}-\alpha_{q_Q}\left(S_{q_1}+\cdots+S_{q_Q}\right)\right]> \\
& E\left[\beta_{q_1}-\alpha_{q_1}S_{q_1}\right]+E\left[\beta_{q_2}-\alpha_{q_2}\left(S_{q_1}+S_{q_2}\right)\right]+\cdots+E\left[\beta_{q_i}-\alpha_{q_i}\left(S_{q_1}+\cdots+S_{q_i}\right)\right] \\
& +E\left[\beta_{q_j}-\alpha_{q_j}\left(S_{q_1}+\cdots+S_{q_i}+S_{q_j}\right)\right]+\cdots+E\left[\beta_{q_Q}-\alpha_{q_Q}\left(S_{q_1}+\cdots+S_{q_Q}\right)\right]
\end{aligned}$$

$$\begin{aligned}
\Rightarrow\ & E\left[\beta_{q_j}-\alpha_{q_j}\left(S_{q_1}+\cdots+S_{q_j}\right)\right]+E\left[\beta_{q_i}-\alpha_{q_i}\left(S_{q_1}+\cdots+S_{q_j}+S_{q_i}\right)\right]> \\
& E\left[\beta_{q_i}-\alpha_{q_i}\left(S_{q_1}+\cdots+S_{q_i}\right)\right]+E\left[\beta_{q_j}-\alpha_{q_j}\left(S_{q_1}+\cdots+S_{q_i}+S_{q_j}\right)\right]
\end{aligned}$$

$$\begin{aligned}
\Rightarrow\ & E\left[-\alpha_{q_j}S_{q_1}-\cdots-\alpha_{q_j}S_{q_j}\right]-E\left[-\alpha_{q_j}S_{q_1}-\cdots-\alpha_{q_j}S_{q_i}-\alpha_{q_j}S_{q_j}\right]> \\
& E\left[-\alpha_{q_i}S_{q_1}-\cdots-\alpha_{q_i}S_{q_i}\right]-E\left[-\alpha_{q_i}S_{q_1}-\cdots-\alpha_{q_i}S_{q_j}-\alpha_{q_i}S_{q_i}\right]
\end{aligned}$$

$$\Rightarrow \alpha_{q_j}S_{q_i} > \alpha_{q_i}S_{q_j}$$

$$\Rightarrow \frac{\alpha_{q_j}}{S_{q_j}} > \frac{\alpha_{q_i}}{S_{q_i}},$$

which is a contradiction. □

The proof above uses a technique known as "adjacent pairwise interchange methods," found in [24] and [31], among others.

2. Quadratic Assignment Problem Formulation and Result

We can reformulate the optimal control problem for linear value functions, (3.13), as an equivalent quadratic assignment problem (QAP) as follows. Let y_{ij} be an indicator variable that equals one if information package *i* is processed j^{th} and zero otherwise. From (3.13) we have

$$\sum_{q=1}^{Q}\sum_{h=1}^{q}\alpha_{x_q}E\left[S_{x_h}\right]=\sum_{h=1}^{Q}\sum_{q=h}^{Q}\alpha_{x_q}E\left[S_{x_h}\right]$$

$$\begin{aligned}
&=E\left[S_{x_1}\right]\left(\alpha_{x_1}+\alpha_{x_2}+\cdots+\alpha_{x_Q}\right)\\
&+E\left[S_{x_2}\right]\left(\alpha_{x_2}+\alpha_{x_3}+\cdots+\alpha_{x_Q}\right)\\
&+\cdots\\
&+E\left[S_{x_{Q-1}}\right]\left(\alpha_{x_{Q-1}}+\alpha_{x_Q}\right)\\
&+E\left[S_{x_Q}\right]\left(\alpha_{x_Q}\right)
\end{aligned}$$

$$=\begin{bmatrix}E[S_1] & E[S_2] & \cdots & E[S_Q]\end{bmatrix}\begin{bmatrix}y_{11}\\y_{21}\\\vdots\\y_{Q1}\end{bmatrix}\left(\alpha_{x_1}+\alpha_{x_2}+\cdots+\alpha_{x_Q}\right)$$

$$+\begin{bmatrix}E[S_1] & E[S_2] & \cdots & E[S_Q]\end{bmatrix}\begin{bmatrix}y_{12}\\y_{22}\\\vdots\\y_{Q2}\end{bmatrix}\left(\alpha_{x_2}+\alpha_{x_3}+\cdots+\alpha_{x_Q}\right)$$

$$+\cdots$$

$$+\begin{bmatrix}E[S_1] & E[S_2] & \cdots & E[S_Q]\end{bmatrix}\begin{bmatrix}y_{1,Q-1}\\y_{2,Q-1}\\\vdots\\y_{Q,Q-1}\end{bmatrix}\left(\alpha_{x_{Q-1}}+\alpha_{x_Q}\right)$$

$$+\begin{bmatrix}E[S_1] & E[S_2] & \cdots & E[S_Q]\end{bmatrix}\begin{bmatrix}y_{1Q}\\y_{2Q}\\\vdots\\y_{QQ}\end{bmatrix}\left(\alpha_{x_Q}\right)$$

$$= \begin{bmatrix} E[S_1] & E[S_2] & E[S_3] & \cdots & E[S_Q] \end{bmatrix} \mathbf{Y} \begin{bmatrix} \alpha_{x_1} + \alpha_{x_2} + \alpha_{x_3} + \cdots + \alpha_{x_Q} \\ \alpha_{x_2} + \alpha_{x_3} + \cdots + \alpha_{x_Q} \\ \alpha_{x_3} + \cdots + \alpha_{x_Q} \\ \vdots \\ \alpha_{x_Q} \end{bmatrix} \qquad (3.17)$$

where $\mathbf{Y} = [y_{ij}]$

$$= \begin{bmatrix} E[S_1] & E[S_2] & \cdots & E[S_Q] \end{bmatrix} \mathbf{Y} \begin{bmatrix} \begin{bmatrix} \alpha_1 & \alpha_2 & \ldots & \alpha_Q \end{bmatrix} \mathbf{YL} \end{bmatrix}^T, \text{ where } \mathbf{L} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

$$= \begin{bmatrix} E[S_1] & E[S_2] & \cdots & E[S_Q] \end{bmatrix} \mathbf{Y} \mathbf{L}^T \mathbf{Y}^T \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_Q \end{bmatrix}^T$$

$$= \left(y_{11}E[S_1] + y_{21}E[S_2] + \cdots + y_{Q1}E[S_Q]\right)\left[\alpha_1\left(y_{11} + \cdots + y_{1Q}\right) + \cdots + \alpha_Q\left(y_{Q1} + \cdots + y_{QQ}\right)\right]$$

$$+ \left(y_{12}E[S_1] + y_{22}E[S_2] + \cdots + y_{Q2}E[S_Q]\right)\left[\alpha_1\left(y_{12} + \cdots + y_{1Q}\right) + \cdots + \alpha_Q\left(y_{Q2} + \cdots + y_{QQ}\right)\right]$$

$$+ \cdots$$

$$+ \left(y_{1Q}E[S_1] + y_{2Q}E[S_2] + \cdots + y_{QQ}E[S_Q]\right)\left[\alpha_1\left(y_{1Q}\right) + \cdots + \alpha_Q\left(y_{QQ}\right)\right].$$

Let $\mathbf{G} = \mathbf{L}^T\mathbf{Y}^T = [g_{ik}] = \left[\sum_{l=k}^{Q} y_{il}\right]$. Then (3.17) is equivalent to

$$\sum_{k=1}^{Q}\left[\left(\sum_{i=1}^{Q} y_{ik} E\left[S_i\right]\right)\left(\sum_{i=1}^{Q} \alpha_i g_{ik}\right)\right]$$

$$= \sum_{k=1}^{Q}\sum_{i=1}^{Q}\sum_{j=1}^{Q} y_{ik} E\left[S_i\right] \alpha_j g_{jk}$$

$$= \sum_{k=1}^{Q}\sum_{i=1}^{Q}\sum_{j=1}^{Q} \alpha_j E\left[S_i\right] y_{ik} \sum_{l=k}^{Q} y_{jl}$$

$$= \sum_{i=1}^{Q}\sum_{j=1}^{Q}\sum_{k=1}^{Q}\sum_{l=k}^{Q} \alpha_j E\left[S_i\right] y_{ik} y_{jl}$$

$$= \sum_{i=1}^{Q}\sum_{j=1}^{Q}\sum_{k=1}^{Q}\sum_{l=1}^{Q} c_{ij} d_{kl} y_{ik} y_{jl}, \text{ where } c_{ij} = \alpha_j E\left[S_i\right] \text{ and } d_{kl} = \begin{cases} 1, & \text{if } k \le l \\ 0, & \text{otherwise} \end{cases} \quad (3.18)$$

Hence, an equivalent optimal control problem (to (3.13)) is

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{Q}\sum_{j=1}^{Q}\sum_{k=1}^{Q}\sum_{l=1}^{Q} c_{ij} d_{kl} y_{ik} y_{jl} \\ \text{subject to} \quad & \sum_{i=1}^{Q} y_{ij} = 1, \ \forall j \\ & \sum_{j=1}^{Q} y_{ij} = 1, \ \forall i \\ & y_{ij} \in \{0,1\}, \ \forall i, j \end{aligned} \quad (3.19)$$

This is a special case of the well-known quadratic assignment problem (QAP) [14]. The general QAP is known to be NP-complete [15, 16]. We show here that not only is (3.19) *not* NP-complete, its solution can be expressed in closed-form.

Theorem 3.5. Consider the quadratic assignment problem (3.19), where $\mathbf{C} = \left[c_{ij}\right] = \left[\alpha_j E\left[S_i\right]\right]$. If the Q information packages are indexed such that

$$\frac{\alpha_1}{E\left[S_1\right]} \ge \frac{\alpha_2}{E\left[S_2\right]} \ge \cdots \ge \frac{\alpha_Q}{E\left[S_Q\right]},$$

(which results in no loss of generality) then the optimal solution is $\mathbf{Y}^* = \left[y_{ij}^* \right] = \mathbf{I}$.

The proof follows immediately from the equivalence of QAP formulation (3.19) and that of (3.13). □

Theorem 3.5 is significant in that it proves the existence of an analytical solution for a case of the quadratic assignment problem not previously known to have an analytical solution. A small number of special cases have previously been shown to have analytical solutions (see [18-21, 23]), but (3.19) is not among them. In general, these special cases are defined in terms of the matrices $\mathbf{C} = \left[c_{ij} \right]$ and $\mathbf{D} = \left[d_{kl} \right]$, and some well-known ones are summarized as follows.

1. If either of the following conditions holds, then the identity permutation is optimal [20]:

$$c_{is} \le c_{js},\ c_{si} \le c_{sj},\ \mathrm{d}_{is} \ge d_{js},\ \text{and } \mathrm{d}_{si} \ge d_{sj},\ \forall\ 1 \le i < j \le Q \text{ and } 1 \le s \le Q \tag{3.20}$$

$$\begin{aligned} &c_{11} \le \ldots \le c_{QQ},\ \mathrm{d}_{11} \ge \ldots \ge d_{QQ},\ \mathrm{c}_{ks} + \mathrm{c}_{sk} = \mathrm{c}_{is} + \mathrm{c}_{si}, \text{ and } d_{st} = d_{ts} \\ &\forall\ 1 \le i < k \le Q \text{ and } 1 \le s, t \le Q \end{aligned} \tag{3.21}$$

2. If **C** is monotone anti-Monge (i.e., –**C** is Monge)[13] and **D** is a symmetric *Toeplitz* matrix generated by a *benevolent function*, then the permutation $\pi^* = \langle 1, 3, 5, 7, 9, \ldots, 10, 8, 6, 4, 2 \rangle$ is optimal [17, 19, 21]. An $n \times n$ matrix $\mathbf{D} = \left[d_{ij} \right]$ is a *Toeplitz matrix* if there exists a function $f : \{-n+1, \ldots, n-1\} \to \mathbb{R}$ such that $d_{ij} = f(i-j)$ for $1 \le i, j \le Q$. The Toeplitz matrix D is said to be *generated by* the function *f*. A function $f : \{-n+1, \ldots, n-1\} \to \mathbb{R}$ is called *benevolent* if it fulfills the following three properties: 1) $f(-i) = f(i),\ \forall\ 1 \le i \le Q-1$; 2) $f(i) \le f(i+1),\ \forall\ 1 \le i \le \left\lfloor \frac{Q}{2} \right\rfloor - 1$, and 3) $f(i) \le f(Q-1),\ \forall\ 1 \le i \le \left\lceil \frac{Q}{2} \right\rceil - 1$ [19].

3. If **C** is a monotone anti-Monge matrix and the elements of **D** satisfy the inequalities below, then π^* is the optimal permutation [23]:

[13] See (3.36) for the definition of a Monge matrix.

$$d_{Q+1-i,Q+1-j} - d_{i,Q+1-j} \le 0,\ d_{Q+1-i,Q+1-j} - d_{Q+1-i,j} \le 0,\ \forall\ 1 \le i \ne j \le \left\lceil \frac{Q}{2} \right\rceil \qquad (3.22)$$

$$d_{Q+1-i,Q+1-j} + d_{Q+1-i,j} - d_{i,Q+1-j} - d_{ij} \le 0,\ d_{i,Q+1-j} - d_{Q+1-i,j} \le 0,\ \forall\ 1 \le i, j \le \left\lceil \frac{Q}{2} \right\rceil \quad (3.23)$$

$$d_{i,Q+2-j} - d_{ij} \ge 0,\ d_{Q+2-j,i} - d_{ji} \ge 0,\ \forall\ 1 \le i \le \left\lceil \frac{Q+1}{2} \right\rceil,\ 2 \le i \le \left\lceil \frac{Q+1}{2} \right\rceil,\ \text{i} \ne \text{j} \quad (3.24)$$

$$d_{Q+2-i,Q+2-j} + d_{Q+2-i,j} - d_{ij} - d_{i,Q+2-j} \ge 0,\ d_{i,Q+2-j} - d_{Q+2-i,j} \ge 0,\ \forall\ 2 \le i, j \le \left\lfloor \frac{Q}{2} \right\rfloor \quad (3.25)$$

It can be easily shown that matrices **C** and **D** defined in (3.18) and (3.19) do not satisfy any of cases 1–3 above (namely because **C** is not anti-Monge). If we assume that the service times (S_i) are identically distributed, we could construct **C** so as to be anti-Monge; but **D** would still not meet any of the conditions outlined above.

3. Linear Assignment Problem Formulation and Result

The result outlined below represents a special case of Theorem 3.5 which, ordinarily, would not warrant special consideration. However, as an application example of the rich literature on Monge matrices[14] and the linear assignment problem (see [14]), we deem it worthy of inclusion here. This result specifies the optimal processing strategy for a single server, with all information packages having linear value functions of the form

$$\begin{aligned} Val_q(t) &= H(t-a_q)H(\tau_q - t)\left[\beta_q - \alpha(t-a_q)\right] \\ &= \begin{cases} \beta_q - \alpha(t-a_q), & a_q \le t \le \tau_q \\ 0, & \text{otherwise} \end{cases} \end{aligned} \qquad (3.26)$$

[14] An $m \times n$ matrix **C** is called *Monge* if it satisfies the so-called *Monge property* $c_{ij} + c_{rs} \le c_{is} + c_{rj}$ for all $1 \le i \le r \le m, 1 \le j \le s \le n$ [37, 38].

where $H(t)$ is the unit step function, $\alpha \geq 0$ is the rate at which value decays (assumed equal for each information package in this case), and $\beta_q > 0$ is the (unique) initial value for each information package.

Corollary 3.6: Suppose at time t_d, Q information packages are chosen for processing, all having linear value functions of the form given in (3.26). Suppose also that $m = 1$ and that Q is chosen such that all Q information packages can be processed prior to time $t = \tau_q$, $\forall q \in \{1, 2, \ldots, Q\}$. Then processing the Q information packages in order of increasing expected service time is the optimal policy.

Proof: By assumption, the Q information packages have all arrived by time t_d and will all be processed by time τ_q, so we can simplify the value function as follows:

$$Val_q(t) = \beta_q - \alpha(t - a_q) \tag{3.27}$$

Note that we can again shift the coordinate axes such that $t_d = 0$. With β_q redefined appropriately, the value function for each information package can be simplified as follows.

$$Val_q(t) = \beta_q - \alpha t \tag{3.28}$$

Define $x_h \in \{1, 2, \ldots, Q\}$ to be the information package processed h^{th} (a modification of our original decision variable to account for the presence of only a single server). Then the total expected realized value obtained from processing all Q information packages is

$$E\left[V^r\right] = \sum_{q=1}^{Q}\left[\beta_{x_q} - \alpha \sum_{h=1}^{q} E\left[S_{x_h}\right]\right] \tag{3.29}$$

which is the quantity we wish to maximize. From (3.29) it follows that

$$E\left[V^r\right] = \sum_{q=1}^{Q} \beta_{x_q} - \alpha \sum_{q=1}^{Q} \sum_{h=1}^{q} E\left[S_{x_h}\right] \tag{3.30}$$

$$= \sum_{q=1}^{Q} \beta_q - \alpha \sum_{q=1}^{Q} \sum_{h=1}^{q} E\left[S_{x_h}\right] \tag{3.31}$$

where the first term in (3.31) represents the total initial value for all information packages (value at time $t = t_d$) and is constant with respect to *x*. So, maximizing $E\left[V^r\right]$ is equivalent to

$$\min_{\{x_h\} \in \sigma_Q} \sum_{q=1}^{Q} \sum_{h=1}^{q} \alpha E\left[S_{x_h}\right] \tag{3.32}$$

We can transform (3.32) into a more recognizable form by introducing a slightly different decision variable. For $i, j = 1, 2, \ldots, Q$, let y_{ij} be an indicator variable that equals one if information package *i* is processed j^{th} and zero otherwise. We have

$$\sum_{q=1}^{Q} \sum_{h=1}^{q} \alpha E\left[S_{x_h}\right] = \begin{bmatrix} Q & Q-1 & \ldots & 1 \end{bmatrix} \mathbf{Y} \begin{bmatrix} \alpha E\left[S_1\right] \\ \alpha E\left[S_2\right] \\ \vdots \\ \alpha E\left[S_Q\right] \end{bmatrix}$$

where $\mathbf{Y} = \left[y_{ij}\right]$. Therefore, the optimal control problem can be written:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{Q} \sum_{j=1}^{Q} c_{ij} y_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{Q} y_{ij} = 1, \ \forall j \\ & \sum_{j=1}^{Q} y_{ij} = 1, \ \forall i \\ & y_{ij} \in \{0,1\}, \ \forall i, j \end{aligned} \tag{3.33}$$

where

$$c_{ij} = \alpha\left(Q - i + 1\right) E\left[S_j\right] \tag{3.34}$$

and

$$\mathbf{C} = \left[c_{ij}\right] = \alpha \begin{bmatrix} QE\left[S_1\right] & QE\left[S_2\right] & \cdots & QE\left[S_Q\right] \\ (Q-1)E\left[S_1\right] & (Q-1)E\left[S_2\right] & \cdots & (Q-1)E\left[S_Q\right] \\ \vdots & & \ddots & \vdots \\ E\left[S_1\right] & E\left[S_2\right] & \cdots & E\left[S_Q\right] \end{bmatrix}. \tag{3.35}$$

This is the classical linear assignment problem (LAP), with cost matrix **C** defined in (3.35). While polynomial-time algorithms exist for solving the general LAP (see [36] for a recent survey), there are certain special cases whereby an analytical solution may be obtained (see [18, 37, 38]).

Assuming service time distributions are known, we can index the Q information packages (without loss of generality) such that $E\left[S_1\right] \le E\left[S_2\right] \le \cdots \le E\left[S_Q\right]$, thereby causing the matrix **C** to possess special properties. The terms in each row are nondecreasing from left to right due to our indexing of the information packages, and the terms in each column are strictly decreasing from top to bottom.

Hoffman [38] and Burkard *et al.* [37] define the following: an $m \times n$ matrix **C** is called *Monge* if it satisfies the so-called *Monge property*

$$c_{ij} + c_{rs} \le c_{is} + c_{rj} \quad \text{for all} \quad 1 \le i \le r \le m, 1 \le j \le s \le n. \tag{3.36}$$

Lemma 3.7 ([18, 37]): a matrix **C** is Monge if and only if its elements satisfy the following inequality:

$$c_{ij} + c_{i+1,j+1} \le c_{i,j+1} + c_{i+1,j} \quad \text{for all} \quad 1 \le i < m, 1 \le j < n. \tag{3.37}$$

Lemma 3.8 [18]: if the cost matrix of a linear assignment problem satisfies the Monge property, then the optimal solution is the "identity permutation" (i.e., $\mathbf{Y}^* = \left[y_{ij}^* \right] = \mathbf{I}$).

We now show that the matrix **C** in (3.35) is Monge.

$$c_{ij} + c_{i+1,j+1} - c_{i,j+1} - c_{i+1,j}$$

$$= (Q - i + 1) E\left[S_j \right] + \left(Q - (i+1) + 1 \right) E\left[S_{j+1} \right] - (Q - i + 1) E\left[S_{j+1} \right] - \left(Q - (i+1) + 1 \right) E\left[S_j \right]$$

$$= E\left[S_j \right] \left(Q - i + 1 - \left(Q - (i+1) + 1 \right) \right) + E\left[S_{j+1} \right] \left(Q - (i+1) + 1 - (Q - i + 1) \right)$$

$$= E\left[S_j \right] - E\left[S_{j+1} \right]$$

$$\leq 0$$

Hence, **C** is Monge by Lemma 3.7. By Lemma 3.8, the optimal solution to (3.33) is $\mathbf{Y}^* = \left[y_{ij}^* \right] = \mathbf{I}$. The interpretation of this solution is to process the Q information packages in order of increasing expected service time, and this completes the proof. □

C. EXPONENTIAL VALUE FUNCTIONS

Suppose all information packages have value functions of the form

$$Val_q(t) = \beta_q e^{-\alpha_q t} \tag{3.38}$$

Suppose that $Q = 2$ and $m = 1$. Suppose also that $S_q \sim \text{Exponential}\left(\mu_q \right)$. Then, for $\langle i, j \rangle \in \varphi_2$,

$$E\left[V^r_{\langle i,j\rangle}\right] = E\left[\beta_i e^{-\alpha_i S_i}\right] + E\left[\beta_j e^{-\alpha_j\left(S_i+S_j\right)}\right]$$

$$= \beta_i\mu_i\int_0^\infty e^{-\alpha_i s_i}e^{-\mu_i s_i}ds_i + \beta_j\mu_i\mu_j\int_0^\infty\int_0^\infty e^{-\alpha_j\left(s_i+s_j\right)}e^{-\mu_i s_i}e^{-\mu_j s_j}ds_i ds_j$$

$$= \frac{\beta_i\mu_i}{\alpha_i+\mu_i} + \frac{\beta_j\mu_i\mu_j}{\left(\alpha_j+\mu_i\right)\left(\alpha_j+\mu_j\right)}$$

Then,

$$E\left[V^r_{\langle 1,2\rangle}\right] > E\left[V^r_{\langle 2,1\rangle}\right] \Leftrightarrow \frac{\beta_1\mu_1}{\alpha_1+\mu_1} + \frac{\beta_2\mu_1\mu_2}{\left(\alpha_2+\mu_1\right)\left(\alpha_2+\mu_2\right)} > \frac{\beta_2\mu_2}{\alpha_2+\mu_2} + \frac{\beta_1\mu_2\mu_1}{\left(\alpha_1+\mu_2\right)\left(\alpha_1+\mu_1\right)}$$

$$\Leftrightarrow \frac{\beta_1\mu_1}{\alpha_1+\mu_1}\left(1-\frac{\mu_2}{\alpha_1+\mu_2}\right) > \frac{\beta_2\mu_2}{\alpha_2+\mu_2}\left(1-\frac{\mu_1}{\alpha_2+\mu_1}\right)$$

$$\Leftrightarrow \frac{\beta_1\mu_1\alpha_1}{\left(\alpha_1+\mu_1\right)\left(\alpha_1+\mu_2\right)} > \frac{\beta_2\mu_2\alpha_2}{\left(\alpha_2+\mu_2\right)\left(\alpha_2+\mu_1\right)} \tag{3.39}$$

Unfortunately, this expression does not lend itself to a distinct ordering as was the case for linear value functions.

Let us now consider the case where $Q > 2$. It is easily shown that for $\langle i, j, k\rangle \in \varphi_3$,

$$E\left[V^r_{\langle i,j,k\rangle}\right] = \frac{\beta_i\mu_i}{\alpha_i+\mu_i} + \frac{\beta_j\mu_i\mu_j}{\left(\alpha_j+\mu_i\right)\left(\alpha_j+\mu_j\right)} + \frac{\beta_k\mu_i\mu_j\mu_k}{\left(\alpha_k+\mu_i\right)\left(\alpha_k+\mu_j\right)\left(\alpha_k+\mu_k\right)}$$

For arbitrary Q and for $\langle q_1, q_2, \ldots, q_Q\rangle \in \varphi_Q$,

$$E\left[V^r_{\langle q_1,q_2,\ldots,q_Q\rangle}\right] = \frac{\beta_{q_1}\mu_{q_1}}{\alpha_{q_1}+\mu_{q_1}} + \frac{\beta_{q_2}\mu_{q_1}\mu_{q_2}}{\left(\alpha_{q_2}+\mu_{q_1}\right)\left(\alpha_{q_2}+\mu_{q_2}\right)} + \cdots + \frac{\beta_{q_Q}\mu_{q_1}\mu_{q_2}\ldots\mu_{q_Q}}{\left(\alpha_{q_Q}+\mu_{q_1}\right)\left(\alpha_{q_Q}+\mu_{q_2}\right)\cdots\left(\alpha_{q_Q}+\mu_{q_Q}\right)}$$

$$= \sum_{m=1}^{Q} \beta_{q_m} \prod_{n=1}^{m} \frac{\mu_{q_n}}{\alpha_{q_m} + \mu_{q_n}} \tag{3.40}$$

From (3.39) and (3.40) we present the following results for $Q > 2$. Lemma 3.9 applies a known result to the problem at hand; to our knowledge, the results presented in Lemmas 3.10 and 3.11 are new.

Lemma 3.9: If $\alpha_q = \alpha, \ \forall q$ then the optimal policy is to process information packages in decreasing order of $\beta_q \mu_q$ (a version of this result is presented in [31]).

Lemma 3.10: If $\mu_q = \mu, \ \forall q$ (i.e., service times are identically distributed) then the optimal policy is to process information packages in decreasing order of $\dfrac{\beta_q \alpha_q}{\left(\alpha_q + \mu\right)^2}$.

Suppose now that $S_q \sim \text{Uniform}\left(a_q, b_q\right)$, for $0 \le a_q < b_q$. Then, for $\langle i, j \rangle \in \varphi_2$,

$$E\left[V^r_{\langle i,j \rangle}\right] = E\left[\beta_i e^{-\alpha_i S_i}\right] + E\left[\beta_j e^{-\alpha_j \left(S_i + S_j\right)}\right]$$

$$= \beta_i \int_{a_i}^{b_i} e^{-\alpha_i s_i} \left(\frac{1}{b_i - a_i}\right) ds_i + \beta_j \int_{a_i}^{b_i} \int_{a_j}^{b_j} e^{-\left(s_i + s_j\right)} \left(\frac{1}{b_i - a_i}\right)\left(\frac{1}{b_j - a_j}\right) ds_i ds_j$$

$$= \frac{\beta_i \left(e^{-\alpha_i a_i} - e^{-\alpha_i b_i}\right)}{\alpha_i \left(b_i - a_i\right)} + \frac{\beta_j \left(e^{-\alpha_j a_j} - e^{-\alpha_j b_j}\right)\left(e^{-\alpha_j a_i} - e^{-\alpha_j b_i}\right)}{\left(\alpha_j\right)^2 \left(b_i - a_i\right)\left(b_j - a_j\right)}$$

After some algebra, one can show that $E\left[V^r_{\langle 1,2 \rangle}\right] > E\left[V^r_{\langle 2,1 \rangle}\right]$ if and only if

$$\frac{\beta_1 \left(e^{-\alpha_1 a_1} - e^{-\alpha_1 b_1}\right)}{\alpha_1 \left(b_1 - a_1\right)} \left[1 - \frac{e^{-\alpha_1 a_2} - e^{-\alpha_1 b_2}}{\alpha_1 \left(b_2 - a_2\right)}\right] > \frac{\beta_2 \left(e^{-\alpha_2 a_2} - e^{-\alpha_2 b_2}\right)}{\alpha_2 \left(b_2 - a_2\right)} \left[1 - \frac{e^{-\alpha_2 a_1} - e^{-\alpha_2 b_1}}{\alpha_2 \left(b_1 - a_1\right)}\right] \tag{3.41}$$

Lemma 3.11 follows directly from (3.41). The proof, not shown, is constructed exactly as that for Lemma 3.4, employing an adjacent pairwise interchange argument.

Lemma 3.11: Suppose that $S_q \sim \text{Uniform}(a,b)$ (i.e., service times are identically distributed) and each information package has a value function of the form (3.38). Then the optimal policy is to process information packages in decreasing order of $\beta_q \left(\frac{e^{-\alpha_q a} - e^{-\alpha_q b}}{\alpha_q} \right)^2$.

D. AN UPPER BOUND FOR MIXED VALUE FUNCTION TYPES

Let $Val_i(t)$ be the (arbitrary) value function for information package i, $i = 1,2,\ldots Q$, let $m = 1$ server, and define the permutation mapping $\varphi(j)$ to be the information package processed j^{th}. Then, from (2.4), the optimal control problem is

$$\underset{\varphi}{\text{maximize}} \; E\left[V^r\right] = \sum_{j=1}^{Q} E\left[Val_{\varphi(j)}\left(D_{\varphi(j)}\right)\right] \tag{3.42}$$

which is equivalent to

$$\underset{\varphi}{\text{maximize}} \; E\left[V^r\right] = \sum_{j=1}^{Q} E\left[Val_{\varphi(j)}\left(\chi_{\varphi(j)} + S_{\varphi(j)}\right)\right]. \tag{3.43}$$

This problem is conjectured to be NP-hard.

However, if we let s_i be the (deterministic) service time for information package i, then we can formulate an upper bound for the solution to (3.43)—which is to say, an upper bound on maximum cumulative realized value. With deterministic service times, the optimal control problem becomes

$$\underset{\varphi}{\text{maximize}} \; V^r = \sum_{j=1}^{Q} Val_{\varphi(j)}\left(\chi_{\varphi(j)} + s_{\varphi(j)}\right). \tag{3.44}$$

Let $\varphi^*(j)$ denote the permutation (or sequence) that solves (3.44), which is the sequence that produces the optimal solution to (3.44), denoted V^*. Define $T_i(t) = \dfrac{Val_i(t)}{s_i}$, a scaled version of the value function for information package *i*, and let $t_i = s_{\varphi^*(1)} + s_{\varphi^*(2)} + \cdots + s_{\varphi^*(i)}$ (the i^{th} service completion time). Then an equivalent expression for V^* is

$$V^* = \sum_{j=1}^{Q} s_{\varphi^*(j)} T_{\varphi^*(j)}(t_j). \tag{3.45}$$

The geometric interpretation of (3.45) is the sum of rectangles with width $s_{\varphi^*(j)}$ and height $T_{\varphi^*(j)}(t_j)$. Figure 14 below shows an example of this for $Q = 4$. The curves in the figure (exponential in this case, but they could be any value function type) represent the functions $T_i(t)$, $i = 1, 2, \ldots, 4$.

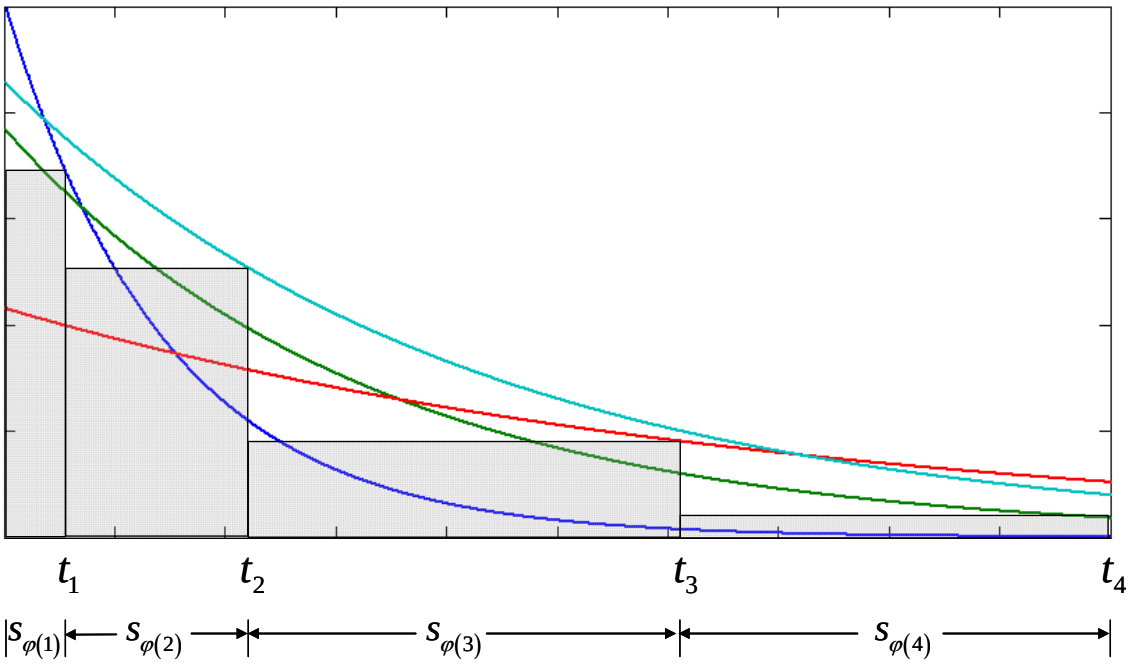


Figure 14 Computing V^* Geometrically

Define the envelope function, $T_{env}(t) = \max\{T_1(t), T_2(t), \ldots, T_Q(t)\}$. This brings us to the following proposition regarding an upper bound on V^*.

Proposition 3.12: $V^* \le \int_0^{t_Q} T_{env}(t)dt$.

Figure 15 below gives the geometric argument for this proposition, where the envelope function $T_{env}(t)$ is shown in black.

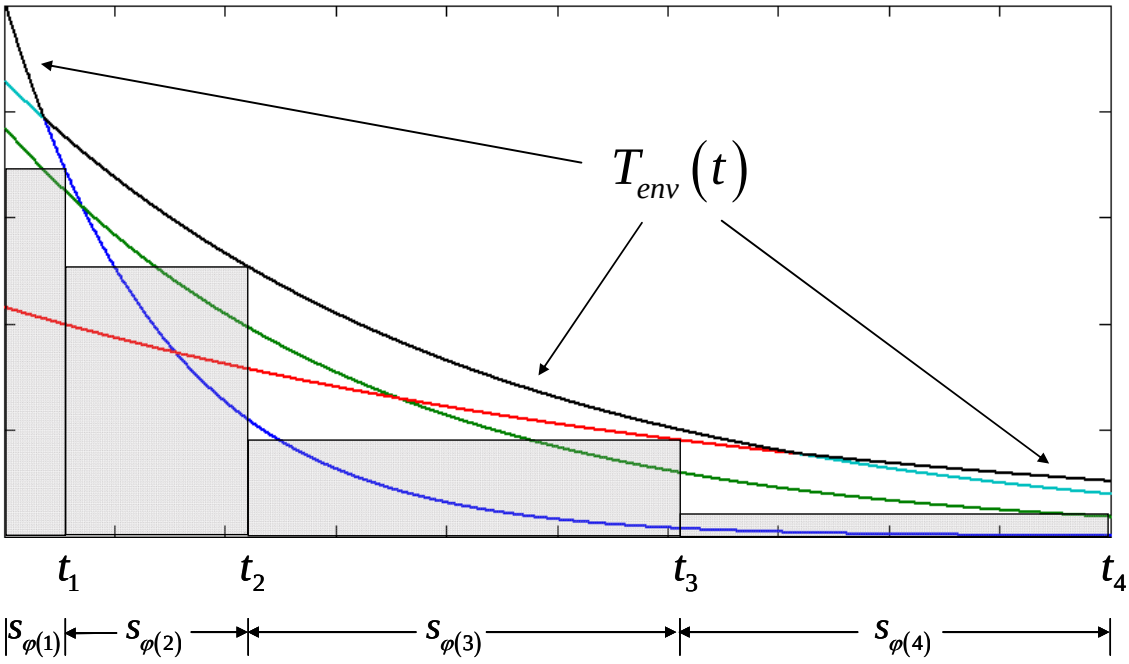


Figure 15 Comparison of V^* to $\int T_{env}(t)\,dt$

Here we have constructed the function $T_{env}(t)$ from the functions $T_i(t) = \frac{Val_i(t)}{s_i}$ and show how it can be used to construct an upper bound on V^*. We will see in the next chapter (Section E) that the function $T_i(t)$ itself yields a good lower bound on V^*.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. INVESTIGATION OF CONTROL STRATEGIES

A. INTRODUCTION

In Chapter III, we introduced some provably optimal controlled queuing strategies for certain special cases of the system in question. We also noted that, even for several seemingly oversimplified cases, the optimal control problem is either provably NP-hard, or conjectured to be so. To address more complex cases (i.e., cases that more closely mirror those present in the physical system), we depart from the search for optimal control policies and introduce a set of heuristic control strategies. We design two simulation experiments to address two related, yet distinct, problems: static scheduling of a set of information packages, and dynamic scheduling with information packages arriving in real time. The static scheduling problem is actually a sub-problem of the dynamic scheduling problem; see Figure 16.

In the first experiment, we assume that a finite set of information packages are available at time zero for scheduling, and that no new information packages arrive. This experiment, known hereinafter as the static scheduling simulation, is used to measure the affect of sequencing these information packages according to various strategies. The purpose of this static simulation is to narrow down the set of heuristic strategy candidates to be considered in the second simulation experiment by measuring their performance (in terms of the metric defined in (4.1)) and robustness to variations in the experiment factors. This second experiment, known as the dynamic scheduling simulation, is meant to be a credible emulation of the physical system: random arrival and service times; an "unstable" queue (i.e., the number of information packages awaiting service grows faster than they can be processed); and a realistic mix of value function types. Its purpose is to compare promising sequencing strategies to each other and, ultimately, to a representation of the "status quo"—how these processing decisions are currently made.

Figure 16 below, depicts the basic process governing the physical system and the dynamic scheduling simulation. Upon the availability of a processor (at time t_d), a set of

size Q of the information packages in Oval 2 is selected for consideration for assignment to the available processor. For our purposes, this set is considered to be the entire contents of Oval 2. The Q information packages are then sequenced either optimally (for small Q) or according to some alternative sequencing strategy. This sequencing step defines the static scheduling problem, which can be thought of as a sub-problem of the dynamic scheduling problem. The first information package in the sequence is assigned to the available processor, and the process repeats when the next processor becomes available. Newly arriving information packages are considered for subsequent processor assignment. The process repeats itself until some pre-defined stopping criteria are met.

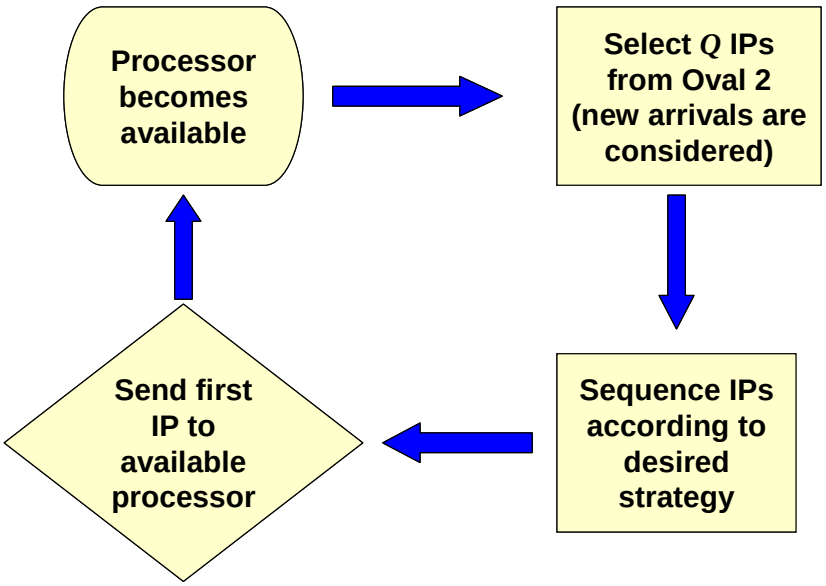


Figure 16 Physical System Process Represented in the Dynamic Simulation

B. HEURISTIC CANDIDATES

The construct of each simulation described in this chapter centers on the decision to assign a specific information package to the next available processor, and the *strategy* to employ when making this decision. Suppose a processor becomes available at time t_d, $d = 1, 2, \ldots, Q$. The following heuristics define some potential assignment strategies; in each case, a unique processing *sequence* of the Q information packages is achieved.

- H1: Assign information packages to processor in descending order of $\frac{\alpha_q}{E\left[S_q\right]}$.
- H2: Assign information packages to processor in descending order of β_q (initial value).
- H3: For each assignment decision *d*, choose the information package with the highest current value; i.e., the highest value of $Val_q\left(t_d\right)$.
- H4: For each assignment decision *d*, choose the information package with the highest value at the end of expected service time; i.e., the highest value of $Val_q\left(E\left[D_q \middle| \chi_q = t_d\right]\right)$.
- H5: For each assignment decision *d*, choose the information package with the highest value for the expression $\frac{Val_q\left(E\left[D_q \middle| \chi_q = t_d\right]\right)}{E\left[S_q\right]}$.
- H6: Assign information packages to processor in descending order of linear combinations of terms as follows.
  - ◊ H6E: $\eta\frac{\alpha_q\beta_q}{E\left[S_q\right]}+\frac{1-\eta}{\beta_q}$, $0\le\eta\le1$ (for exponential value functions only).
  - ◊ H6L: $\alpha_q\left(\frac{\eta}{E\left[S_q\right]}+\frac{1-\eta}{\beta_q}\right)$, $0\le\eta\le1$ (for linear value functions only).
- LIFO: Last in, first out.
- FIFO: First in, first out.
- Random: randomly (uniformly) select the next information package to send to processor.

Some notes and further explanation: 1) H1 is obtained directly from the optimal solution for the linear value function case (see Chapter III, Section B); 2) LIFO and FIFO are only suitable for study in the dynamic simulation case, since the static simulation does not consider the arrival time of each information package; 3) H6E and H6L are not

suitable for study in simulations where value functions are of mixed type; 4) H2–H4 represent varying types of "greedy" strategies, ranging from the "naïve" H2 to the more sophisticated H4.

C. STATIC SCHEDULING SIMULATION

The static scheduling simulation incorporates the following assumptions: single processor; Q information packages that have all arrived by the start of the simulation; no new arrivals; various mixes of value function types; general processing time distributions; and no service preemption. We define the metric $\mathbb{V}^r$, a surrogate for expected total realized value, as follows:

$$\mathbb{V}^r = \sum_{j=1}^{Q} Val_j\left(E\left[D_j \middle| \chi_j\right]\right) = \sum_{j=1}^{Q} Val_j\left(\chi_j + E\left[S_j\right]\right) \tag{4.1}$$

This metric is a function of χ_j (the time information package j is assigned to a processor), which is the control. It is preferable to a metric that incorporates a rigorous notion of expected value—say, $\sum_{j=1}^{Q} E\left[Val_j\left(\chi_j + S_j\right)\right]$—because it can be applied directly regardless of value function type and service time distribution. The simulation is static in the sense that it represents a "snapshot" of the physical system (or the dynamic scheduling problem) at a moment in time—in particular, the time a processor becomes available—and replicates various decision strategies regarding which information package to send to the available processor. See Appendix A for the static simulation MATLAB code.

The static scheduling simulation is not meant to replicate the physical system. The purpose of the static simulation is to provide a mechanism with which to evaluate service policy heuristic candidates by computing $\mathbb{V}^r$ for the information package processing sequence suggested by each heuristic. The result will be a set of promising heuristics to use in the dynamic simulation.

1. Comparison Study

The first component of the static simulation involves a comparison of $\mathbb{V}^r$ for the processing sequence suggested by each heuristic with that for an optimal sequence. Due to the combinatorial nature of the optimization problem—i.e., determining which of the $Q!$ possible sequences maximizes $\mathbb{V}^r$—optimal solutions can only be found for relatively small Q. We begin by examining randomly generated sets of information packages that vary according to the following parameters:

- Value function type: exponential or linear
- Value function parameters: $\alpha \sim \text{Uniform}(0.1,1)$ and $\beta \sim \text{Uniform}(5,10)$, where $Val_q(t) = \beta_q - \alpha_q t$ and $Val_q(t) = \beta_q e^{-\alpha_q t}$ for linear and exponential value functions, respectively. These distributions were chosen to provide a reasonable spread of values for α, the decay rate, and β, the initial value for each information package.
- Expected service times: "disparate" and "similar." For disparate, we use the values $(0.1,\ 3,\ 7)$ time units[15] as expected service times for the three information package classes, respectively, and $(2,\ 3,\ 4)$ for similar.[16]

Additionally, the following factors are held constant:

- Number of information package classes: 3
- Number of information packages generated per *replication*: 9
- Number of replications per *run* or *scenario*:[17] 1000

Figure 17 through Figure 21 represent the results of four scenarios (runs) of the static simulation—one for each combination of value function type (linear, exponential)[18]

[15] Minutes will be used as the unit of time in the simulations.

[16] These expected service time values are based in part on the DARPA C2 Experiment data. See Figure 8.

[17] A run (or scenario) is defined by the levels of the factors used. For example, value function type = "Exponential" and expected service times = "Similar" define Run 1.

and expected service times (similar or disparate). For each run, 1000 replications were performed. For each replication, 9 information packages of the appropriate type were generated through random draw of α and β from their respective distributions. One of three classes was (uniformly) randomly assigned to each information package, along with its associated expected service time. During the simulation, an optimal processing sequence is determined for each replication (optimal with respect to $\mathbb{V}^r$), and the value of $\mathbb{V}^r$ obtained by processing the nine information packages in this sequence is compared with that obtained by processing the information packages in sequences governed by each of the heuristics, H1 through H6.[19] The goal is to find one or more heuristics that perform in near-optimal fashion with respect to $\mathbb{V}^r$. The results are presented below.

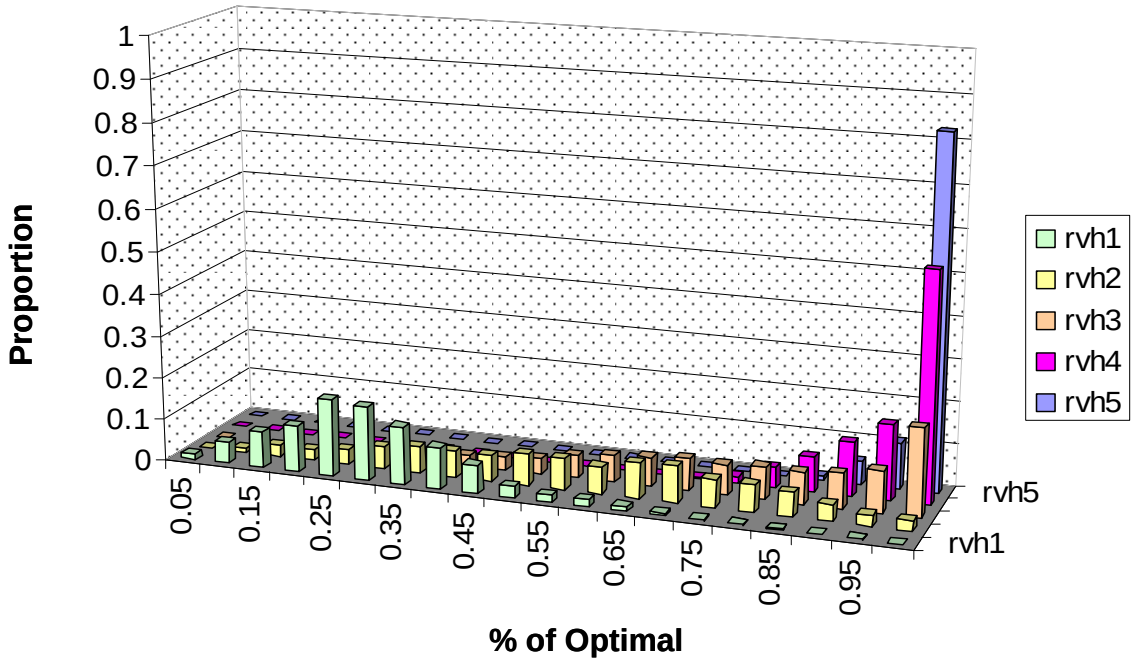


Figure 17 Scenario 1A—Exponential Value Functions, Similar Service Times

[18] For a particular run in the comparison study, only a single type of value function is used (exponential or linear). Runs where value function types are "mixed" are discussed in Section 3 (see Table 5).

[19] The remaining heuristics will be examined in the dynamic simulation.

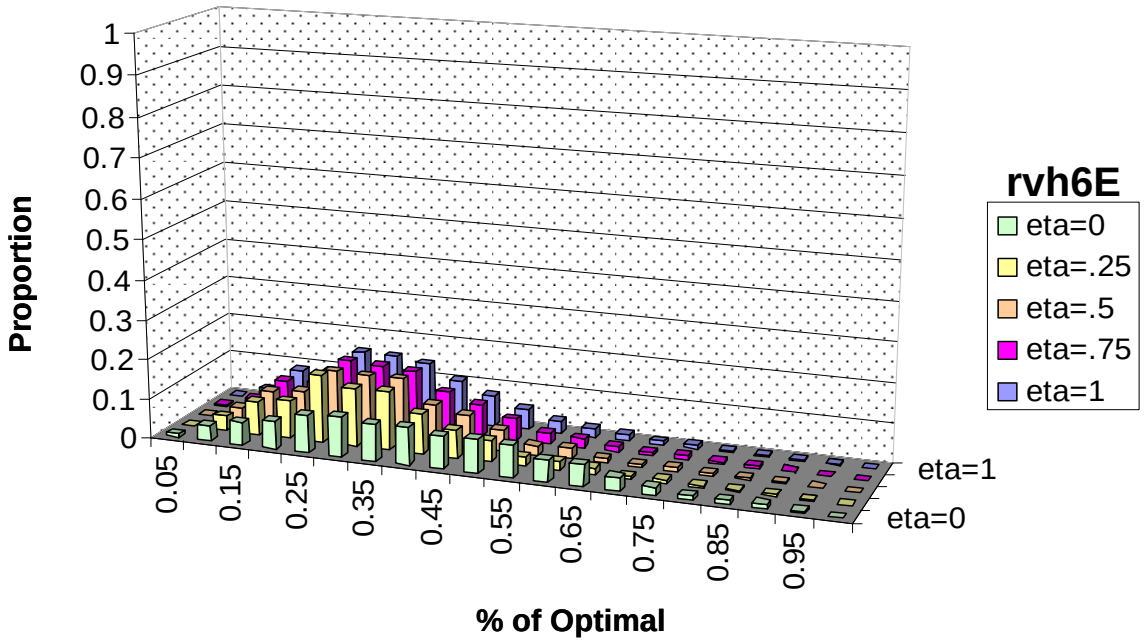


Figure 18 Scenario 1B—Exponential Value Functions, Similar Service Times

Figure 17 and Figure 18 together show the results from Scenario 1, defined by exponential value function type and similar expected service times. These results are explained as follows: "rvh*" in the chart legend stands for the realized value histogram obtained by processing the information packages in the sequence governed by strategy H*. For each heuristic, a histogram is constructed that represents how well the heuristic performed with respect to the optimal sequence. The horizontal axis represents percentage of optimal; the vertical axis represents the proportion of the total number of replications that performed at the given level (or "bin" along the horizontal axis). For example, Figure 17 shows a distribution for heuristic H1 that is centered at 0.25 with a fairly large variance, while H5 is centered at a value greater than 0.95 with a considerably smaller variance. Hence, in the vast majority of the 1000 replications, H5 performed nearly as well as the exact optimal solution. Heuristic H6E (Figure 18) did not perform well for any value of η. It turns out that H6E and H6L do not perform well for any combination of the factors considered, and, hence, will be excluded from subsequent discussion. Results for remaining runs are now presented graphically in Figure 19 through Figure 21. Results are summarized numerically in Table 2.

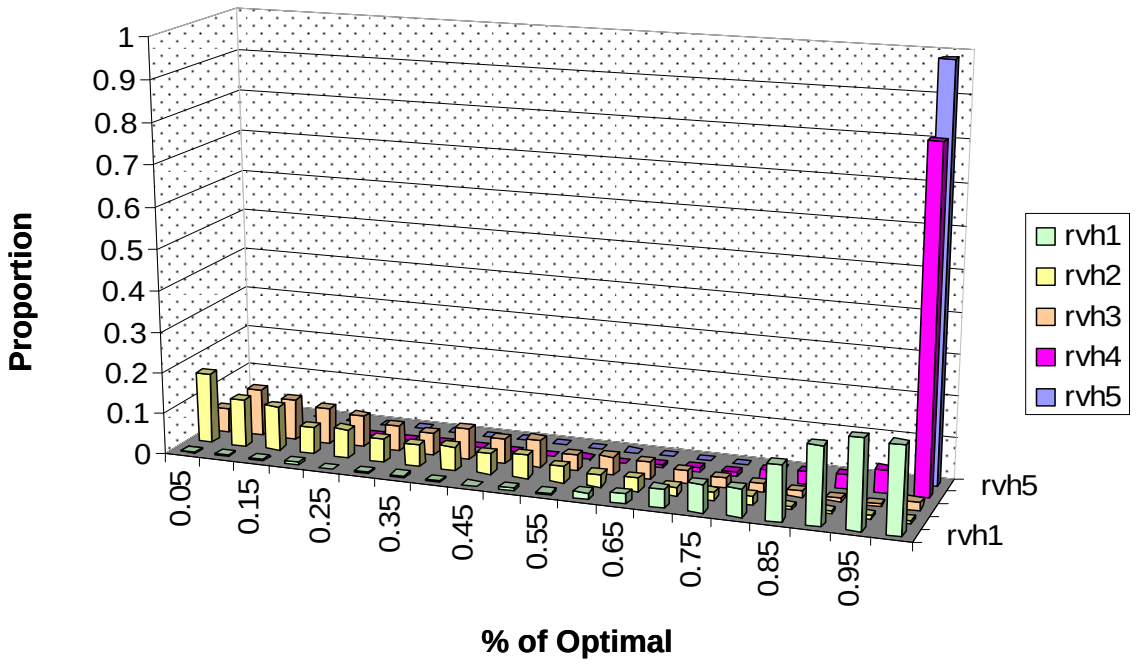


Figure 19 Scenario 2—Exponential Value Functions, Disparate Service Times

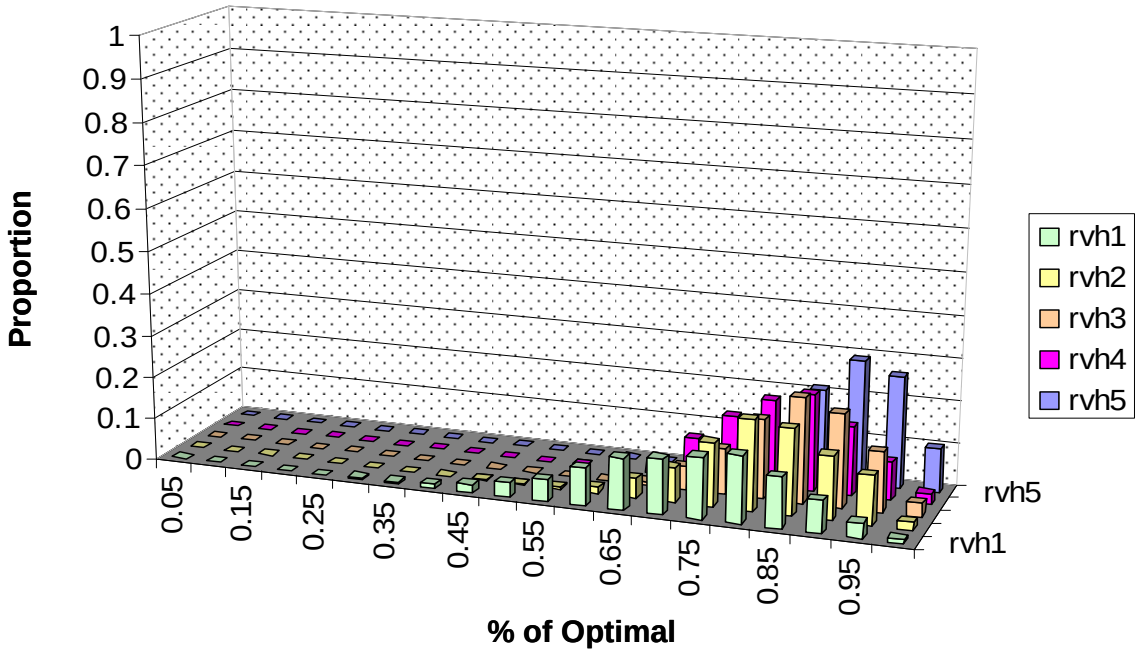


Figure 20 Scenario 3—Linear Value Functions, Similar Service Times

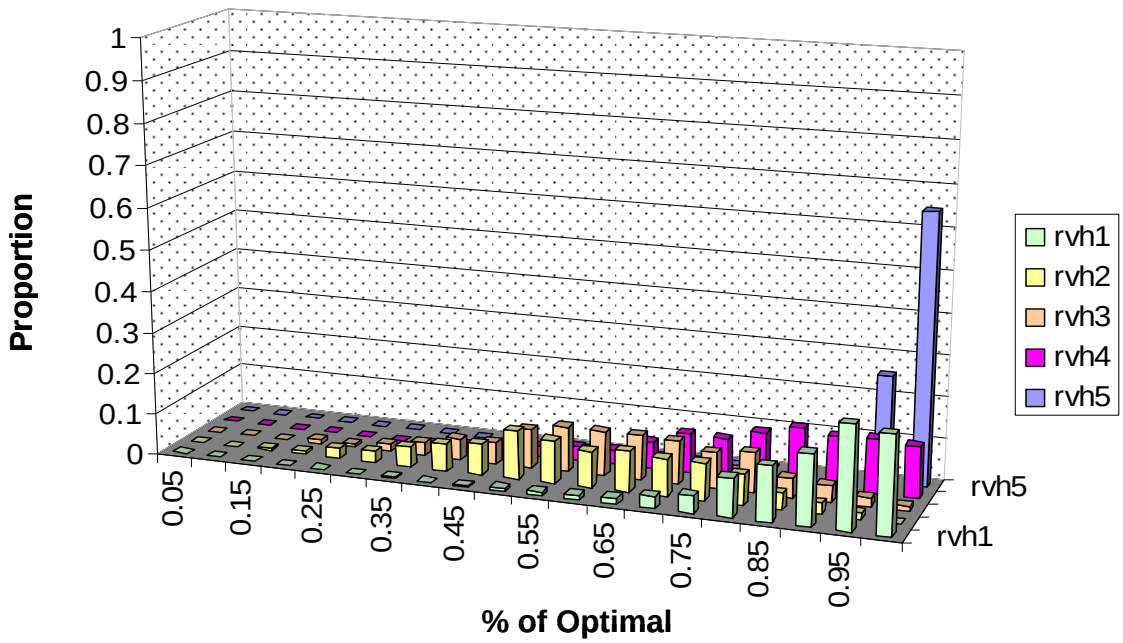


Figure 21 Scenario 4—Linear Value Functions, Disparate Service Times

		Mean Results by Heuristic				
		H1	**H2**	**H3**	**H4**	**H5**
Run	**1**	0.263	0.570	0.772	0.934	0.976
	2	0.876	0.305	0.355	0.955	0.986
	3	0.717	0.798	0.821	0.793	0.871
	4	0.886	0.574	0.603	0.787	0.958

Table 2 Comparison Study Summary Results

The results in Table 2 are computed in a slightly different fashion than those in Figure 17 through Figure 21. The results in these figures are computed by normalizing each heuristic result against the optimal result for each replication; this yields 1000 observations for each heuristic for each run, from which the respective histograms are generated. In Table 2, however, we compute the mean in the more traditional sense by dividing mean $\mathbb{V}^r$ for each heuristic by that for the optimal sequence. Because of the large number of replications, however, and due to the nature of the problem, the mean values reported in Table 2 are very close to their respective empirical distribution means as reflected in the histograms in Figure 17 through Figure 21.

Based on the results in this limited examination, we conclude that H5 performs quite well when compared to optimal sequencing and other heuristic strategies. Additionally, with the exception of H4, the remaining policies are quite sensitive to the input factors, making them less effective as information package processing strategies. One result of note, however, is the effectiveness of H1 in Run 4 (linear value functions, disparate service times); this is to be expected, though, as the conditions in Run 4 nearly match the assumptions in Theorem 3.5. A more thorough examination is required prior to drawing any further conclusions. This is accomplished in the following sections.

2. Validation of Comparison Study

We continue our investigation into the static scheduling problem. One issue that must be resolved is whether the results above will hold in cases where there are a larger number of information packages to be sequenced for processing. Here, we address this issue with a more comprehensive set of simulation runs. This set of runs is described below in Table 3. As in the first set of runs, there are two value function types considered, exponential (E) and linear (L), and two sets of expected service times—similar (S) and disparate (D). Numerical values for expected service times are identical to those used previously. The number of information packages (IPs) alternates between 9 and 100, and we allow two levels each for α and β—high (H) and low (L). As before, each run (scenario) is replicated 1000 times. Recall that for exponential value functions, $Val_j(t) = \beta_j e^{-\alpha_j t}$, and $Val_j(t) = \beta_j - \alpha_j t$ for linear value functions. Furthermore, α and β are drawn from Gamma distributions (unlike the previous set of runs) as follows:

- High $\alpha \sim \Gamma(5,\ 0.15)$
- Low $\alpha \sim \Gamma(1,\ 0.2)$
- High $\beta \sim \Gamma(1.5,\ 7)$
- Low $\beta \sim \Gamma(1,\ 4)$

Run	Val Fct	# IPs	α	β	E[S]
1a	E	9	L	H	S
1b	E	100	L	H	S
2a	E	9	L	H	D
2b	E	100	L	H	D
3a	E	9	L	L	S
3b	E	100	L	L	S
4a	E	9	L	L	D
4b	E	100	L	L	D
5a	E	9	H	H	S
5b	E	100	H	H	S
6a	E	9	H	H	D
6b	E	100	H	H	D
7a	E	9	H	L	S
7b	E	100	H	L	S
8a	E	9	H	L	D
8b	E	100	H	L	D
9a	L	9	L	H	S
9b	L	100	L	H	S
10a	L	9	L	H	D
10b	L	100	L	H	D
11a	L	9	L	L	S
11b	L	100	L	L	S
12a	L	9	L	L	D
12b	L	100	L	L	D
13a	L	9	H	H	S
13b	L	100	H	H	S
14a	L	9	H	H	D
14b	L	100	H	H	D
15a	L	9	H	L	S
15b	L	100	H	L	S
16a	L	9	H	L	D
16b	L	100	H	L	D

Table 3 Validation Run Matrix

Using these distributions for α and β allow for a more realistic representation of the types of information packages present in the physical system. For example, an information package with a high α (value function decay rate) might refer to a Time Sensitive Target (TST); one with a high β (initial value) might refer to a High Value Target (HVT). An information package with high α and β could refer to a target that is both a TST and HVT (see [39] for target type descriptions and definitions). Additionally, the two-parameter Gamma distribution allows for tailoring of the "tails" of the distribution in addition to the mean, as seen in the high values for α and β (see Figure 22 below). The "fatter" tails on the right result in the desired effect of a reasonable likelihood of very high values (relatively) for these cases.

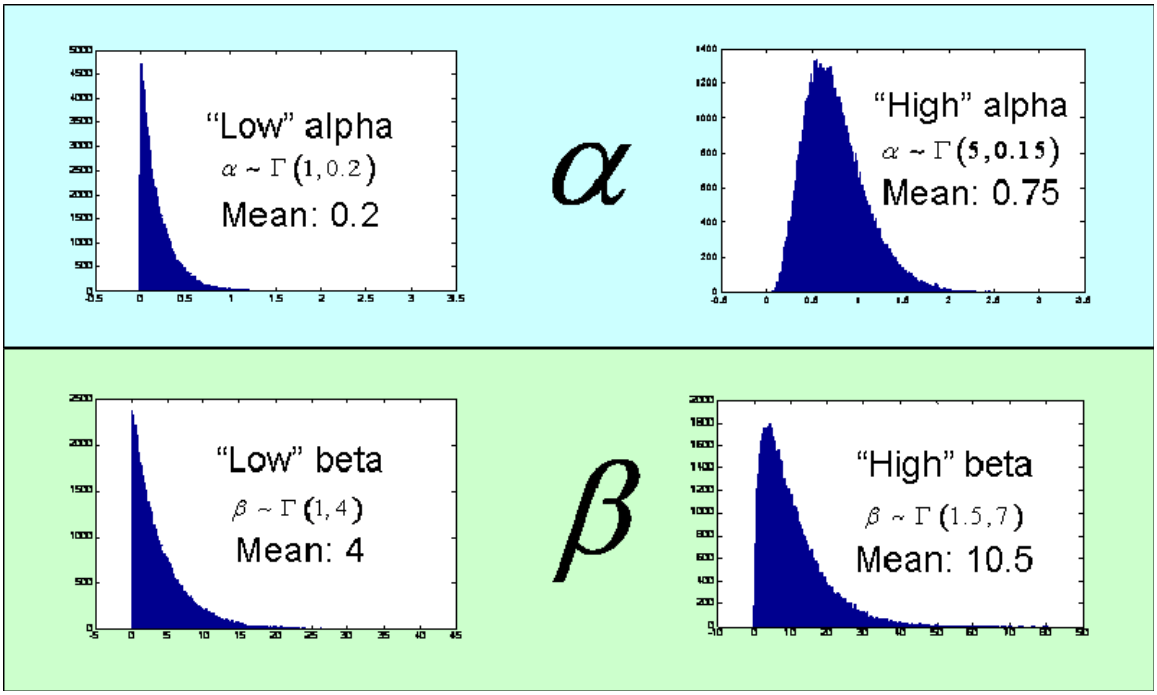


Figure 22 Value Function Parameters

As before, an optimal solution is computed for the runs having 9 information packages, and results for each heuristic are compared to it. For runs involving 100 information packages, exact optimal solutions cannot be determined. H3, seen as a "reasonable" greedy policy, is used as a basis of comparison in these cases. Figure 23 below depicts an example of the detailed results obtained during validation (see Appendix A for the complete set of detailed results for all 16 runs), and consolidated results are shown in Table 4.

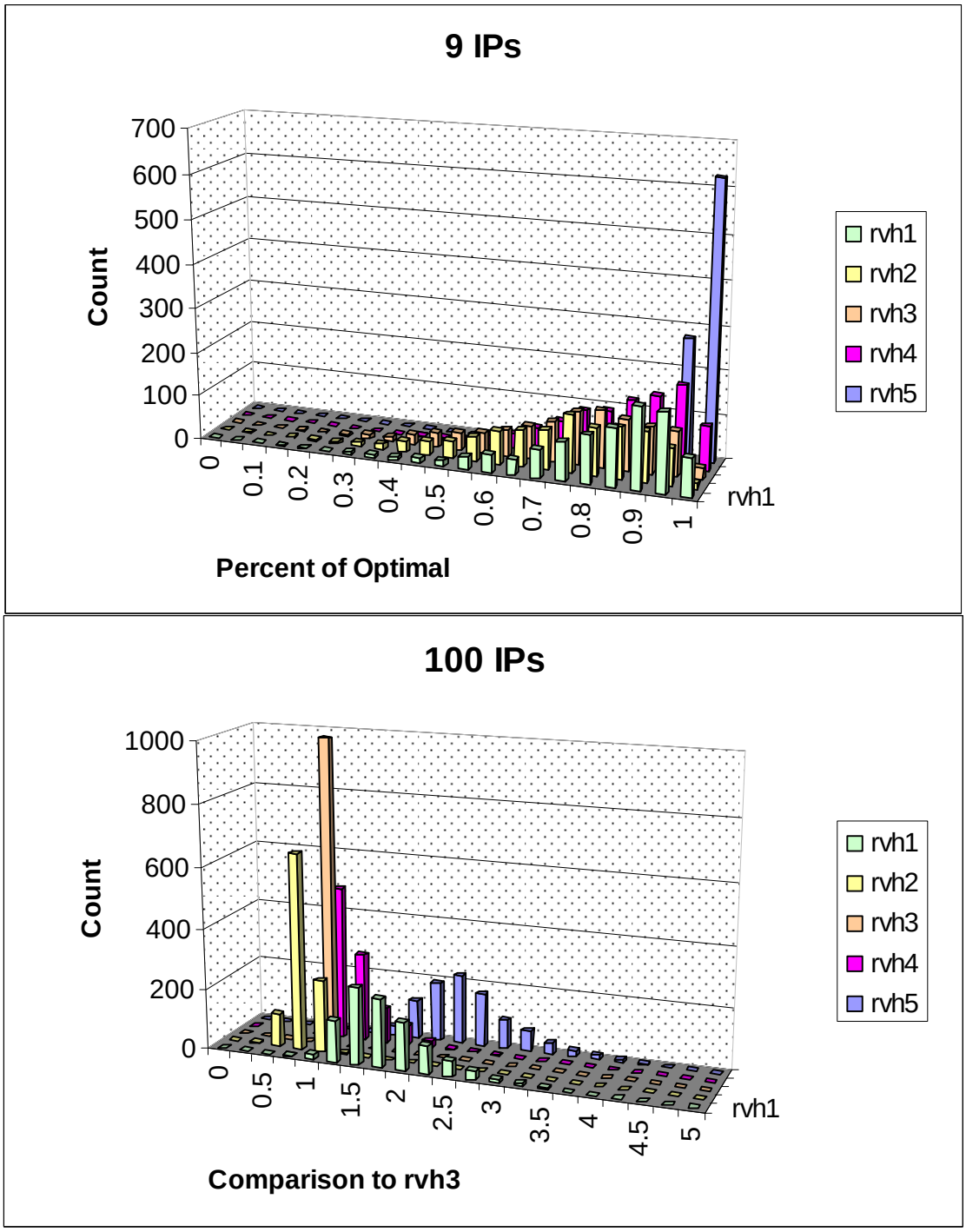


Figure 23 Comparison Study Validation Sample Results (Run 2)[20]

[20] Note that the vertical axes here reflect "count" (out of 1000 replications) rather than "proportion." The interpretation of these graphs is the same as before, however.

		Heuristic Mean Values											
		H1		H2		H3		H4		H5		Best 9 IP	Best 100 IP
	# IPs	9	100	9	100	9	100	9	100	9	100		
Run	1	0.608	0.119	0.910	0.784	0.926	1.000	0.905	0.978	0.925	1.025	H3	H5
	2	0.819	1.791	0.724	0.788	0.739	1.000	0.833	1.213	0.972	2.363	H5	H5
	3	0.585	0.108	0.919	0.810	0.933	1.000	0.916	0.977	0.930	1.022	H3	H5
	4	0.804	1.654	0.734	0.810	0.749	1.000	0.851	1.220	0.975	2.267	H5	H5
	5	0.206	0.042	0.657	0.557	0.752	1.000	0.977	1.709	0.989	1.763	H5	H5
	6	0.859	22.063	0.339	0.675	0.358	1.000	0.949	25.610	0.996	30.720	H5	H5
	7	0.195	0.037	0.686	0.616	0.763	1.000	0.977	1.676	0.988	1.727	H5	H5
	8	0.837	21.522	0.371	0.716	0.388	1.000	0.938	27.134	0.996	32.312	H5	H5
	9	0.975	0.595	0.938	1.025	0.929	1.000	0.924	0.992	0.932	1.025	H1	H5
	10	0.981	1.149	0.894	1.015	0.889	1.000	0.891	1.001	0.961	1.433	H1	H5
	11	0.863	0.293	0.919	0.992	0.917	1.000	0.904	0.985	0.915	1.021	H2	H5
	12	0.916	1.092	0.845	0.978	0.846	1.000	0.858	1.007	0.958	1.557	H5	H5
	13	0.712	0.165	0.918	0.949	0.924	1.000	0.920	0.994	0.941	1.058	H5	H5
	14	0.854	1.777	0.756	0.940	0.764	1.000	0.812	1.100	0.976	2.338	H5	H5
	15	0.390	0.069	0.901	0.916	0.919	1.000	0.932	0.994	0.951	1.060	H5	H5
	16	0.771	2.164	0.649	0.922	0.659	1.000	0.818	1.263	0.985	2.843	H5	H5

Table 4 Comparison Study Validation Consolidated Results

Recall that the exact optimal solution is the basis of comparison for runs involving 9 information packages, while H3 serves this role for runs having 100 information packages (H3 is seen as a "reasonable" greedy policy, making it a reasonable basis of comparison). Therefore, the numerical results shown in Table 4 can only be compared against one another for cases where the number of information packages is the same. While results for the 9-IP cases are strictly less than one (heuristics are inherently sub-optimal), results for the 100-IP cases can be greater than one (indicating that the heuristic in question performed better than H3) or not. The columns labeled "Best 9 IP" and "Best 100 IP" indicate the best performing policy for the 9-IP and 100-IP cases, respectively, for each run.

We observe the following based on these validation runs:

- If a set of factors results in H5 outperforming all other heuristics for the 9-IP case, then H5 outperforms the others in the 100-IP case for the same set of factors (in fact, H5 outperforms the others in all 100-IP cases).
- The five cases where H5 did not perform the best can be explained as follows:

- ◊ The factor levels for runs 10 and 11 nearly match the conditions required by Theorem 3.5, so it is no surprise that H1 performs the best.
- ◊ For runs 1, 3, and 11, the differences in performance between H5 and the highest performing heuristic are essentially negligible. Additionally, these are all cases having similar expected service times. The physical system is assumed (with justification) to have information package classes with *disparate* expected service times only.[21]

- The bottom line is that there is enough consistency between the 9-IP and 100-IP runs to warrant further investigation of H5 as a promising information package processing assignment sequencing heuristic.

3. Robustness of Policy H5

We now complete our investigation of the static scheduling problem by presenting results for cases where information packages are no longer homogeneous with respect to value function type; we allow them to be mixed, as one would expect to encounter in the physical system. Additionally, we now include a third value function type, the step function, defined as follows.

$$Val_q(t) = \beta_q H(t) H(\tau_q - t) = \begin{cases} \beta_q & 0 \le t \le \tau_q \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Note that $H(t)$ is the unit (or Heaviside) step function. This set of simulation runs demonstrates the robustness of heuristic H5 to variations in five significant factors, including three used in previous simulation runs—α, β, and expected service time (all defined as before)—as well as two new factors: value function mix, which describes the expected proportions of information packages having exponential (E), linear (L), and step

[21] For example, typical information package classes in the physical system include, among others, video, still imagery, text, and voice. Clearly, the processing time distributions for each of these classes will differ significantly.

(S) value function types present in the run, and τ, which is the time at which step value functions drop to zero. As with α and β, we define two levels, high and low, for the factor τ as follows:

- High $\tau \sim \Gamma(2,\ 20)$
- Low $\tau \sim \Gamma(1,\ 10)$

The run matrix is shown in Table 5 below.

Run	Reps	# IPs	ValFctMix (E/L/S)	α	β	τ	E[S]
1	500	8	75/15/10	L	L	H	D
2	500	8	75/15/10	L	L	H	S
3	500	8	75/15/10	L	L	L	D
4	500	8	75/15/10	L	L	L	S
5	500	8	75/15/10	L	H	H	D
6	500	8	75/15/10	L	H	H	S
7	500	8	75/15/10	L	H	L	D
8	500	8	75/15/10	L	H	L	S
9	500	8	75/15/10	H	L	H	D
10	500	8	75/15/10	H	L	H	S
11	500	8	75/15/10	H	L	L	D
12	500	8	75/15/10	H	L	L	S
13	500	8	75/15/10	H	H	H	D
14	500	8	75/15/10	H	H	H	S
15	500	8	75/15/10	H	H	L	D
16	500	8	75/15/10	H	H	L	S
17	500	8	33/33/33	L	L	H	D
18	500	8	33/33/33	L	L	H	S
19	500	8	33/33/33	L	L	L	D
20	500	8	33/33/33	L	L	L	S
21	500	8	33/33/33	L	H	H	D
22	500	8	33/33/33	L	H	H	S
23	500	8	33/33/33	L	H	L	D
24	500	8	33/33/33	L	H	L	S
25	500	8	33/33/33	H	L	H	D
26	500	8	33/33/33	H	L	H	S
27	500	8	33/33/33	H	L	L	D
28	500	8	33/33/33	H	L	L	S
29	500	8	33/33/33	H	H	H	D
30	500	8	33/33/33	H	H	H	S
31	500	8	33/33/33	H	H	L	D
32	500	8	33/33/33	H	H	L	S

Table 5 Robust Static Scheduling Simulation Run Matrix

Note that β represents the initial value for all three types of value functions, and α represents the decay rate for linear and exponential value functions. Note also that, unlike previous runs, 500 replications are executed per run, with 8 information packages per replication. This decrease (from 1000 and 9, respectively) is due to run time considerations only, and does not significantly affect the results. As in previous runs, the result for each heuristic is compared to the exact optimal solution. The results for these runs are shown in Table 6. Each result represents an average over the 500 replications for each run.

Run	H2	H3	H4	H5	Spread
1	0.815	0.818	0.861	0.964	0.150
2	0.924	0.924	0.892	0.907	0.032
3	0.783	0.792	0.853	0.967	0.183
4	0.909	0.921	0.907	0.917	0.014
5	0.802	0.809	0.840	0.955	0.153
6	0.915	0.915	0.882	0.894	0.032
7	0.754	0.765	0.819	0.958	0.204
8	0.891	0.903	0.889	0.908	0.019
9	0.652	0.669	0.872	0.975	0.323
10	0.835	0.880	0.894	0.910	0.074
11	0.577	0.603	0.882	0.980	0.403
12	0.752	0.837	0.916	0.929	0.177
13	0.629	0.657	0.827	0.971	0.342
14	0.828	0.882	0.878	0.897	0.069
15	0.575	0.606	0.861	0.976	0.400
16	0.765	0.857	0.901	0.913	0.148
17	0.863	0.863	0.871	0.952	0.090
18	0.910	0.909	0.892	0.896	0.018
19	0.791	0.806	0.855	0.956	0.165
20	0.855	0.873	0.890	0.891	0.036
21	0.872	0.870	0.874	0.949	0.079
22	0.907	0.902	0.883	0.893	0.024
23	0.791	0.802	0.848	0.944	0.153
24	0.847	0.862	0.869	0.882	0.035
25	0.774	0.784	0.840	0.953	0.178
26	0.883	0.897	0.875	0.886	0.021
27	0.651	0.689	0.833	0.961	0.310
28	0.774	0.838	0.880	0.887	0.112
29	0.769	0.786	0.819	0.945	0.176
30	0.868	0.891	0.882	0.892	0.024
31	0.655	0.695	0.808	0.955	0.300
32	0.773	0.841	0.879	0.886	0.114
mean	0.793	0.817	0.868	0.930	0.142
std dev	0.099	0.092	0.027	0.033	0.117

Table 6 Robust Static Simulation Results—Average Percentage of Optimal

Note that H1 is not included in the above results; it is not defined for step functions (there is no decay rate for step functions). The values in this table represent the mean realized value obtained (over the 500 replications) for each policy for each run, expressed as a proportion of the optimal result (for each run). The column titled "Spread" records the difference between the largest and smallest mean result for each run. These results indicate that H5 is quite robust to variations in the significant factors. On average, sequencing information packages according to H5 results in a $\mathbb{V}^r$ that is only 7% less than that obtained from optimal sequencing. The six runs for which H5 does not perform best all have very small spreads, indicating that the four heuristics are nearly indistinguishable in terms of their performance. These results clearly suggest that H5 is robust to variations in the significant factors.

D. DYNAMIC SCHEDULING SIMULATION

The dynamic scheduling simulation is designed to be a credible emulation of the physical system, with random arrival and service times and a realistic mix of value function types. Its purpose is to compare promising heuristic sequencing strategies to each other and, ultimately, to a representation of the "status quo"—how these processing decisions are currently made. Unlike the static scheduling simulation, which represents only a snapshot in time, the dynamic simulation represents the entire duration of an operation. We introduce a new heuristic called "Status Quo" as an estimate of how information package assignment decisions are currently[22] made. Status Quo is defined to be a weighted combination of three previously defined heuristics as follows: 60% LIFO, 30% H3, and 10% Random [40]. Status Quo is executed in the simulation by randomly choosing one of its three component heuristics (according to their respective weights) to use each time an assignment decision is made. The goal is to measure how the policies previously defined (specifically H5) compare with Status Quo using Total Realized Value as the metric.

[22] As observed in the 2006 DARPA Command and Control experiments.

The dynamic simulation starts with zero information packages in Oval 2. Information packages begin arriving according to a process assumed to be Poisson with rate $\lambda_i,\ i = 1, 2, \ldots, NumClass$ and are randomly assigned a value function type and associated parameters (α, β, and/or τ). Information packages are assigned, without preemption, to available processors as per the method described in Figure 16, with service times assumed to follow a Poisson process with rate $\mu_i,\ i = 1, 2, \ldots, NumClass$.[23] This process is repeated in parallel, using identical sets of information packages, for each heuristic (H3, H4, H5, LIFO, FIFO, Random, and StatusQuo). The simulation ends when the last information package arrives, leaving many information packages still unprocessed—a realistic outcome mirroring the physical system. Realized value is computed for each information package processed using the expression

$$V_j^r = Val_j\left(\chi_j + s_j\right), \tag{4.3}$$

and summed to determine Total Realized Value. This value is recorded for each heuristic. MATLAB code for the dynamic simulation can be found in Appendix A. Figure 24 shows a typical outcome of a single replication of the dynamic simulation, revealing how realized value grows over the duration of the simulation (operation) according to the processing strategy utilized.

[23] The Poisson assumptions are reasonable based on data obtained from the DARPA Command and Control experiments. See Figure 7 and Figure 8.

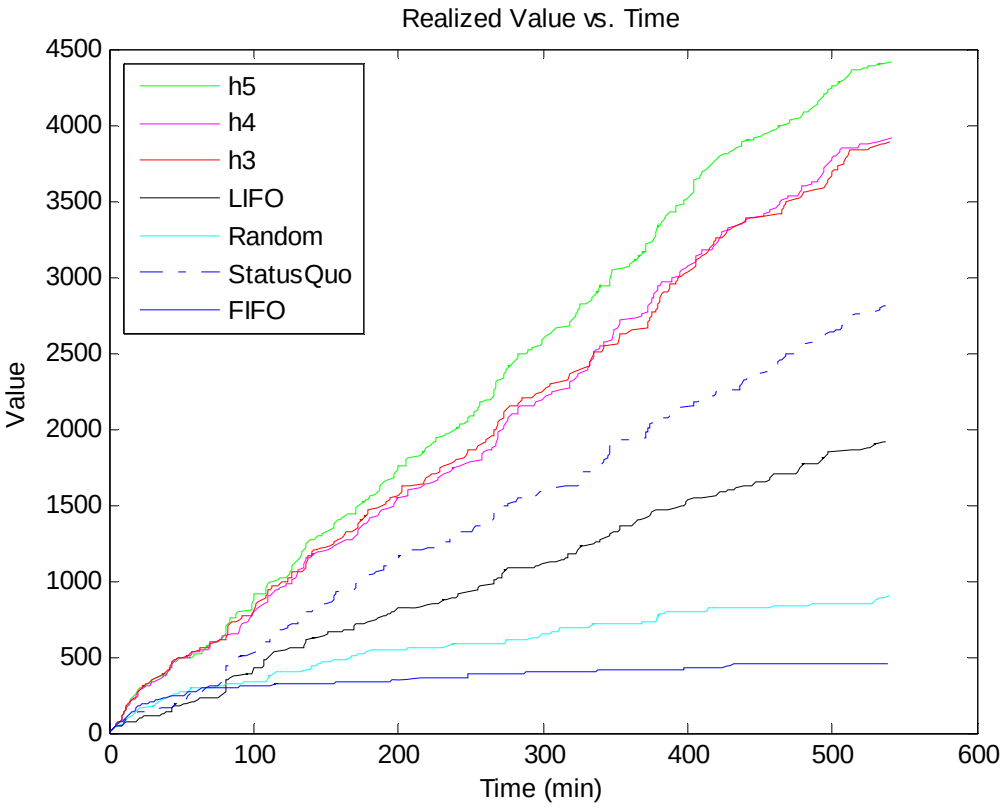


Figure 24 Typical Dynamic Simulation Replication Results

A full-factorial design is executed using the following seven factors, each having two levels (for a total of $2^7 = 128$ runs):

- Number of servers (processors)—3 or 5.
- Value Function Mix—33/33/33 or 75/15/10 for exponential, linear, and step value function types, respectively.
- α (linear and exponential value functions)—low (L) and high (H), as in the static scheduling simulation runs.
- β (linear and exponential value functions)—low (L) and high (H), as in the static scheduling simulation runs.
- β (step functions)—unlike the static case, we allow the initial value for step functions to differ from that for linear and exponential value functions. Two levels are used as follows.
  - ◊ High $\beta \sim \Gamma(1,\ 4)$
  - ◊ Low $\beta \sim \Gamma(0.5,\ 2)$

- τ (step functions)—low (L) and high (H), as in the static simulation runs.
- λ (arrival rate)—two sets of arrival rates are used: [0.631, 0.5, 0.8][24] and [1.2, 0.2, 0.6] arrivals per minute for information packages of class 1, 2, and 3, respectively.

In addition to these varying factors, the following factors are held constant for each run.

- Number of replications—250
- Number of information packages—1000
- Number of information package classes—3
- μ (processing rates)—one set of processing rates is used: [0.134, 0.1, 1][25] processing completions per minute for information packages of class 1, 2, and 3, respectively.

For each of the 128 runs, Total Realized Value is computed for each policy for each replication, and then averaged over all replications. The detailed run matrix can be found in Appendix A, and the results are summarized in Table 7 below.

[24] This set of arrival rates is based in part on DARPA C2 Experiment data.

[25] Service rates are also based in part on DARPA C2 Experiment data

	Mean Values for each heuristic for each run							
Run	H5	H4	H3	LIFO	Random	StatusQuo	FIFO	h5/StatusQuo
1	1556	1424	1357	608	166	887	92	1.75
2	1434	1334	1280	533	151	815	82	1.76
3	1629	1476	1407	632	186	915	98	1.78
4	1493	1368	1324	557	164	837	85	1.78
5	1935	1750	1622	781	192	1013	98	1.91
6	1765	1616	1523	696	174	932	91	1.89
7	2288	2009	1923	894	274	1245	131	1.84
8	2093	1874	1807	810	239	1147	117	1.82
9	4094	3691	3553	1567	561	2368	340	1.73
10	3798	3481	3385	1395	512	2241	312	1.69
11	4161	3716	3567	1605	582	2411	354	1.73
12	3847	3492	3403	1439	527	2235	316	1.72
13	4415	3879	3657	1751	594	2490	355	1.77
14	4028	3610	3449	1553	530	2310	313	1.74
15	4673	4014	3821	1876	672	2629	389	1.78
16	4253	3744	3601	1686	606	2457	345	1.73
17	821	795	730	357	37	372	15	2.21
18	714	693	656	304	31	336	13	2.12
19	907	852	806	384	57	438	22	2.07
20	823	782	757	333	51	386	21	2.13
21	1232	1183	1090	541	63	557	25	2.21
22	1099	1051	1014	460	54	499	22	2.20
23	1664	1540	1482	652	137	878	55	1.89
24	1547	1442	1396	576	120	810	47	1.91
25	2512	2379	2163	1022	144	1276	57	1.97
26	2267	2160	2032	874	125	1160	43	1.95
27	2580	2405	2215	1028	165	1283	66	2.01
28	2387	2227	2111	912	146	1214	58	1.97
29	2885	2700	2418	1201	171	1405	66	2.05
30	2612	2446	2260	1046	146	1309	54	1.99
31	3224	2908	2692	1323	236	1587	90	2.03
32	2970	2714	2551	1162	214	1505	82	1.97
33	1648	1477	1362	680	165	896	88	1.84
34	1486	1348	1279	595	145	802	75	1.85
35	1640	1468	1349	665	159	878	85	1.87
36	1487	1355	1279	592	143	811	76	1.83
37	1853	1641	1528	757	196	991	99	1.87
38	1691	1523	1443	669	170	925	89	1.83
39	1745	1560	1420	728	171	921	89	1.89
40	1577	1435	1337	630	147	850	76	1.85
41	4286	3729	3425	1773	499	2286	272	1.88
42	3824	3391	3189	1557	432	2086	242	1.83
43	4237	3715	3403	1774	472	2282	266	1.86
44	3830	3403	3193	1564	424	2109	240	1.82
45	4427	3830	3510	1858	509	2352	284	1.88
46	3958	3479	3245	1653	453	2161	248	1.83
47	4354	3774	3448	1814	481	2336	269	1.86
48	3878	3442	3192	1597	429	2127	238	1.82
49	692	664	557	312	31	311	13	2.22
50	597	571	512	275	27	279	12	2.14

	Mean Values for each heuristic for each run							
Run	H5	H4	H3	LIFO	Random	StatusQuo	FIFO	h5/StatusQuo
51	673	656	536	317	24	300	10	2.24
52	569	556	481	266	22	268	10	2.12
53	943	896	806	402	55	490	22	1.93
54	832	792	741	348	47	439	21	1.90
55	813	788	673	372	35	365	16	2.23
56	684	662	600	309	27	317	11	2.16
57	1980	1882	1547	863	92	954	37	2.08
58	1697	1622	1408	741	75	831	34	2.04
59	1985	1869	1573	879	98	925	43	2.15
60	1686	1608	1409	740	82	852	37	1.98
61	2081	1977	1624	926	102	994	45	2.09
62	1792	1701	1479	781	83	896	34	2.00
63	2202	2056	1774	969	123	1083	52	2.03
64	1927	1791	1614	825	102	968	42	1.99
65	1757	1702	1662	983	312	1243	183	1.41
66	1679	1631	1615	880	276	1180	167	1.42
67	1866	1796	1768	1032	347	1318	198	1.42
68	1764	1706	1689	928	310	1240	178	1.42
69	2217	2136	2093	1282	369	1519	207	1.46
70	2071	2000	1980	1146	316	1402	177	1.48
71	2644	2510	2463	1476	515	1848	276	1.43
72	2503	2389	2358	1332	453	1720	236	1.46
73	4704	4518	4443	2582	1047	3414	701	1.38
74	4451	4291	4246	2281	909	3173	595	1.40
75	4749	4532	4454	2618	1052	3428	692	1.39
76	4491	4315	4279	2346	958	3236	615	1.39
77	5076	4821	4657	2883	1104	3608	714	1.41
78	4808	4583	4471	2581	969	3373	618	1.43
79	5412	5075	4940	3056	1217	3797	763	1.43
80	5084	4798	4712	2768	1116	3572	696	1.42
81	937	926	917	571	76	577	31	1.62
82	839	829	844	488	63	513	25	1.64
83	1063	1037	1039	622	113	683	50	1.56
84	956	937	955	542	94	609	41	1.57
85	1905	1839	1853	1081	279	1319	124	1.44
86	1778	1721	1745	937	234	1214	98	1.46
87	1409	1384	1405	883	125	874	54	1.61
88	1280	1259	1313	765	102	808	40	1.58
89	2972	2884	2804	1691	331	1950	141	1.52
90	2745	2665	2642	1501	276	1808	119	1.52
91	2885	2832	2718	1653	289	1906	122	1.51
92	2673	2614	2551	1461	244	1755	102	1.52
93	3733	3564	3484	2118	488	2477	210	1.51
94	3464	3322	3258	1897	419	2278	182	1.52
95	3299	3200	3055	1925	343	2140	148	1.54
96	3036	2959	2886	1711	293	1991	118	1.52
97	1923	1836	1761	1090	312	1290	177	1.49
98	1782	1712	1669	979	276	1206	158	1.48
99	1894	1816	1737	1087	303	1272	174	1.49
100	1765	1698	1652	968	268	1181	152	1.49

	Mean Values for each heuristic for each run							
Run	H5	H4	H3	LIFO	Random	StatusQuo	FIFO	h5/StatusQuo
101	2159	2052	1982	1235	364	1466	201	1.47
102	2012	1925	1878	1109	319	1372	175	1.47
103	2037	1951	1871	1178	317	1349	177	1.51
104	1873	1797	1751	1045	279	1253	154	1.49
105	4984	4699	4494	2886	909	3376	554	1.48
106	4645	4411	4268	2583	809	3132	488	1.48
107	4944	4687	4471	2891	900	3355	549	1.47
108	4617	4392	4235	2547	792	3123	473	1.48
109	5058	4778	4532	2951	911	3418	551	1.48
110	4681	4437	4274	2628	799	3168	478	1.48
111	5170	4861	4622	3021	947	3510	569	1.47
112	4776	4506	4340	2675	843	3236	497	1.48
113	811	801	729	502	56	474	27	1.71
114	694	691	643	421	42	409	19	1.70
115	842	829	769	522	63	506	29	1.66
116	729	716	683	437	52	443	25	1.65
117	951	940	879	592	67	581	30	1.64
118	827	817	793	506	54	501	24	1.65
119	1092	1066	1012	645	111	720	51	1.52
120	985	960	936	563	95	660	42	1.49
121	2327	2280	2043	1395	179	1442	82	1.61
122	2035	1991	1839	1186	148	1278	66	1.59
123	2337	2286	2073	1422	196	1437	89	1.63
124	2044	2003	1856	1205	159	1255	71	1.63
125	2433	2380	2154	1497	194	1531	86	1.59
126	2160	2114	1968	1281	168	1343	71	1.61
127	2582	2495	2299	1544	249	1642	119	1.57
128	2275	2212	2088	1321	205	1437	88	1.58

Table 7 Dynamic Simulation Results Summary

One notable observation is that H5 results in the highest average total realized value for 126 out of the 128 runs (note green shading indicates the highest value for each run). As before, in the cases where H5 is not the highest performing heuristic, it is virtually indistinguishable from the one that is (H3 in both cases). H5 also significantly outperforms Status Quo on every run—by as small a factor as 1.38 (run 73) and as large a factor as 2.24 (run 51). Additionally, we note that H5 performs well regardless of arrival rate, as long as the aggregate arrival rate is much larger than aggregate service rate. Finally, implementing H5 does not require knowledge of the service time distributions—all that is required is mean service times. We conclude that H5 is a viable heuristic to use in the physical system.

E. IMPLEMENTATION OF HEURISTIC STRATEGY H5

Previously in this chapter, we have demonstrated quite conclusively that H5 is a well-performing heuristic strategy to employ when making processor assignment decisions within the context of the physical system. In this section, we describe a lower bound based on H5 and depict an $O\left(n^2\right)$ algorithm for implementing H5.

1. Using H5 to Construct a Lower Bound on V^*

In Chapter III, we alluded to a lower bound for optimal realized value V^* constructed using the function $T_i\left(t\right)$. Here, we combine these functions with strategy H5 to construct a lower bound.

Let s_i be the deterministic service time for information package *i*. Define $T_i\left(t\right)=\dfrac{Val_i\left(t\right)}{s_i}$ as before, where $Val_i\left(t\right)$ is of arbitrary (or "mixed") form, and define the permutation mapping $\varphi^{H5}\left(j\right)$ as the information package to be processed j^{th}, where processing sequence is determined by heuristic strategy H5. Let $t_i^{H5}=s_{\varphi^{H5}(1)}+s_{\varphi^{H5}(2)}+\cdots+s_{\varphi^{H5}(Q)}$. Define V^{H5}—the total realized value obtained by processing the *Q* information packages according to heuristic strategy H5—as follows:

$$V^{H5}=\sum_{j=1}^{Q}s_{\varphi^{H5}(j)}T_{\varphi^{H5}(j)}\left(t_j^{H5}\right) \tag{4.4}$$

We employ the same geometric reasoning in constructing (4.4) as we did in (3.45). Finally, it is clear that no heuristic strategy can outperform an optimal strategy, which leads us to the following proposition regarding a lower bound on V^*.

Proposition 4.1: $V^{H5} \le V^*$.

Given the empirical evidence of the performance of H5 with respect to optimal sequencing, we conjecture that V^{H5} is a good lower bound for V^*.

2. An $O\left(\frac{n(n+1)}{2}\right)$ Implementation Algorithm for H5

In this section, we develop an $O\left(\frac{n(n+1)}{2}\right)$ algorithm for implementing H5 for the case where all value functions are exponential, service times are deterministic, and $m = 1$ server.

To begin, we define the term $\tilde{T}_i$ as follows:

$$\tilde{T}_i = \frac{Val_i(\chi_i + s_i)}{s_i}, \tag{4.5}$$

where $Val_i(t)$ is the value function associated with information package i, $i = 1, 2, \ldots, n$, χ_i is the time information package i is assigned to the processor, and s_i is the service time for information package i (assumed deterministic). In Section B of this chapter, we defined the heuristic H5 based on the sequencing criterion $\frac{Val_i\left(E\left[D_i \middle| \chi_i = t_d\right]\right)}{E[S_i]}$. An equivalent representation of this sequencing criterion is $\frac{Val_i\left(E[\chi_i + S_i]\right)}{E[S_i]}$, which, for deterministic service times is equivalent to $\tilde{T}_i$ in (4.5).

Let $\tilde{T}_i(t) = \frac{Val_i(t + s_i)}{s_i}$ be a continuous function of time that represents the value of the H5 sequencing criterion for information package i for any assignment time t.

$\tilde{T}_i(t)$ is related to the function $T_i(t)$ introduced earlier (Chapter III, Section D, and again in Section E1 above); specifically,

$$\tilde{T}_i(t) = T(t + s_i). \tag{4.6}$$

Expression (4.6) is used to assign information packages to processors as follows: at time t_{d_j} (the j^{th} decision time) the information package i that satisfies (4.7) (and has not *already* been sent to a processor) is sent to the available processor. This is equivalent to assigning information packages according to strategy H5.

$$\max_i \left\{ \tilde{T}_i\left(t_{d_j}\right) \right\} \tag{4.7}$$

We now construct a geometric argument which becomes the basis for our implementation algorithm. Figure 25 shows some example functions, $\tilde{T}_i(t)$, $i = 1, 2, \ldots, 4$. We begin by indexing the information packages in decreasing order of $\tilde{T}_i(0)$. Recalling that $\tilde{T}_i(t)$ is the H5 sequencing criterion, we note that, assuming $t_{d_1} = 0$, the first information package chosen for processing is information package 1 (i.e., $\varphi(1) = 1$) since $\tilde{T}_1(0)$ is greater than the remaining initial values.[26] To determine $\varphi(2)$ (i.e., the information package that will be processed second), we consider information package 2 as the first *candidate*, since it has the next highest initial value. We define $t_{i,k}$ as the time when $\tilde{T}_i(t)$ and $\tilde{T}_k(t)$ intersect; see Figure 25. We must check the intersection times of the function $\tilde{T}_i(t)$ for the candidate information package with that of all remaining information packages, and compare these times with t_{d_2}. For our

[26] In fact, the way we have chosen to sequence information packages will *always* lead to package number one being processed first.

example, we compute $t_{2,3}$ and $t_{2,4}$. If $t_{d_2} \le \min\{t_{2,3}, t_{2,4}\}$, then $\varphi(2) = 2$; otherwise, we re-designate information package 3 as the candidate for $\varphi(2)$, and repeat this process of comparing intersection times with t_{d_2} until $\varphi(2)$ is determined. We formalize this notion—that of using intersection points of the functions $\tilde{T}_i(t)$ to determine processing sequence—in the algorithm below.

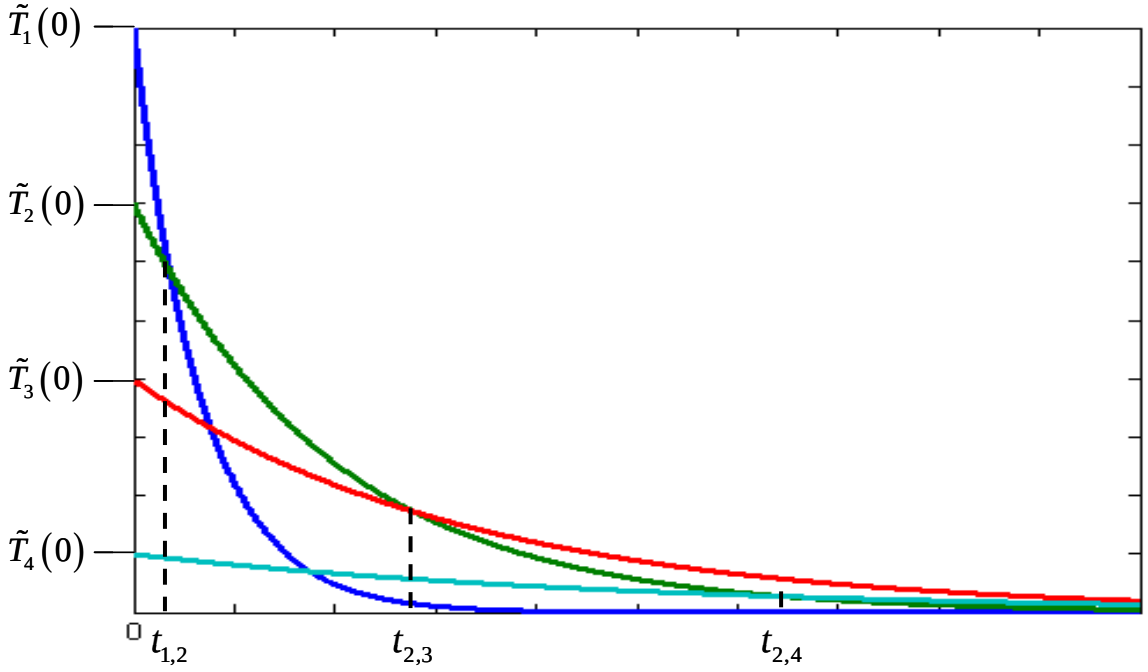


Figure 25 H5 Implementation Algorithm—a Geometric Perspective

First, we define some terms. Let $Val_i(t) = \beta_i e^{-\alpha_i t}$ be the value function for information package i, $i = 1, 2, \ldots, n$, let $m = 1$ processor, and let s_i be the deterministic service time for information package i. Define $\tilde{T}_i(t) = \dfrac{Val_i(t + s_i)}{s_i}$, and let $\varphi(j)$ be the information package to be processed j^{th}.

We now define the H5 implementation algorithm as follows.

- Step 1: Initialize and determine first information package. This step requires *n* function evaluations.
  - ◊ for $i = 1$ to n, evaluate $\tilde{T}_i(0)$, end.
  - ◊ $\varphi(1) = i : \tilde{T}_i(0) \geq \tilde{T}_k(0),\ \forall k \neq i$ (the first information package assigned).

- Step 2: for $j = 2$ to n, do:
  - ◊ "Discard" information package $\varphi(j-1)$. (note: now $n-(j-1)$ information packages remain)
  - ◊ Let $c(j,1) = i : \tilde{T}_i(0) \geq \tilde{T}_k(0),\ \forall k \neq i,\ \forall i, k \notin \{\varphi(1), \ldots, \varphi(j-1)\}$. This is the first candidate for the j^{th} assignment; it has the highest initial value of those information packages not yet assigned.
  - ◊ Let $t_{d_j} = \sum_{k=1}^{j-1} s_{\varphi(k)}$ (the j^{th} *decision time*)
  - ◊ Let $t_{i,k} = \dfrac{\ln\left(\dfrac{\beta_k s_i}{\beta_i s_k}\right) + \alpha_i s_i - \alpha_k s_k}{\alpha_k - \alpha_i}$ (the time of intersection of the functions $T_i(t)$ and $T_k(t)$)
  - ◊ Compare t_{d_j} to $t_{c(j,1),k}$ for all $k \neq c(j,1)$ (each comparison requires one function evaluation)
    - if $t_{d_j} < t_{c(j,1),k}\ \forall k \neq c(j,1)$, then $\varphi(j) = c(j,1)$
    - else, $c(j,2) = k : t_{c(j,1),k} < t_{d_j}$
    - repeat (up to $n-j+1$ times) until $\varphi(j)$ is determined.

Step 1 requires *n* function evaluations (to find the *n* initial values) and results in the determination of $\varphi(1)$. The first loop through Step 2 (that is, the loop during which $\varphi(2)$ is determined) requires the examination of at most $n-1$ intersections, with each intersection examination requiring one function evaluation. The second loop through Step 2 requires the examination of at most $n-2$ intersections (and the execution of $n-2$ function evaluations), and so on. So, completely sequencing all *n* information packages

requires at most $n+(n-1)+(n-2)+\cdots+2+1=\frac{n(n+1)}{2}$ function evaluations. Hence, we conclude that this algorithm is $O\left(\frac{n(n+1)}{2}\right)$.

V. SUMMARY AND CONCLUSION

A. RESULTS SUMMARY

The results and contributions of this research can be categorized into three main areas: 1) development of mathematical concepts and tools that can be used to model the dynamics of battlefield information flow; 2) analysis of these models to derive optimal control strategies and complexity results; and 3) development of robust heuristic control strategies and the construction of simulation platforms for strategy comparison. We now present a detailed summary of these contributions and results.

1. Modeling

- Some fundamental concepts are introduced, namely the incorporation of information value and information volume (workload) in a dynamic information flow context.
- A controlled queue model is developed for maximizing cumulative expected realized value. This involved:
  - ◊ Applying *G*/*G*/*m* multi-class queuing system as the framework to describe the system.
  - ◊ Defining various types of value functions for information packages.
  - ◊ Developing the optimal assignment problem for the queuing system.
- A discrete time information flow model is developed to represent the dynamics of total volume (or workload) in the system.

2. Analysis and Complexity Results

- The following results are proved regarding the *complexity* of special cases of the controlled queue model.

◊ Step Value Functions ("single step"): if $Val_q(t) = \begin{cases} \beta_q & a_q \le t \le a_q + \tau_q \\ 0 & \text{otherwise} \end{cases}$ then the optimal control problem $\max_{\{x_q\} \in \varphi_Q} E\left[V^r\right] = \max_{\{x_q\} \in \varphi_Q} \sum_{q=1}^{Q} Val_{x_q}\left(t_d + \sum_{h=1}^{q} s_{x_h}\right)$ is NP-hard.

◊ The same is true regarding "multi-step" value functions.

- The following analytical solutions for special cases of the controlled queue model are proved.

◊ Linear value functions ($Val_q(t) = \beta_q - \alpha_q t$)

– Proved that the optimal sequencing problem

$$\max_{\{x_q\} \in \sigma_Q} E\left[V^r\right] = \max_{\{x_q\} \in \sigma_Q} E\left[\sum_{q=1}^{Q}\left(\beta_{x_q} - \alpha_{x_q}\sum_{h=1}^{q} S_{x_h}\right)\right] \tag{5.1}$$

is equivalent to the Quadratic Assignment Problem

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{Q}\sum_{j=1}^{Q}\sum_{k=1}^{Q}\sum_{l=1}^{Q} c_{ij} d_{kl} y_{ik} y_{jl} \\ \text{subject to} \quad & \sum_{i=1}^{Q} y_{ij} = 1, \ \forall j \\ & \sum_{j=1}^{Q} y_{ij} = 1, \ \forall i \\ & y_{ij} \in \{0,1\}, \ \forall i, j \end{aligned} \tag{5.2}$$

– The quadratic assignment problem in general is NP-complete, but due to the special structure of problems (5.1) and (5.2), an analytic solution is proved. The solution to both is to process information packages in decreasing order of $\alpha_q / E\left[S_q\right]$.

– This result reveals a previously-undiscovered case of the quadratic assignment problem that has an analytic optimal solution.

◊ Exponential Value Functions—the following results are proved:

- If $Val_q(t) = \beta_q e^{-\alpha t}$ and $S_q \sim \text{Exponential}(\mu_q)$ then processing information packages in decreasing order of $\beta_q \mu_q$ is optimal.
- If $Val_q(t) = \beta_q e^{-\alpha_q t}$ and $S_q \sim \text{Exponential}(\mu)$ then processing information packages in decreasing order of $\frac{\beta_q \alpha_q}{(\alpha_q + \mu)^2}$ is optimal.
- If $Val_q(t) = \beta_q e^{-\alpha_q t}$ and $S_q \sim \text{Uniform}(a,b)$ then processing information packages in decreasing order of $\beta_q \left(\frac{e^{-\alpha_q a} - e^{-\alpha_q b}}{\alpha_q} \right)^2$ is optimal..

- Upper and lower bounds for the optimal solution in the mixed value function case are derived.

3. Simulation

- Due to the complexity of the problem, a set of heuristic information package processing strategies is developed.
- Static scheduling simulation
  - ◊ Constructed a Monte Carlo simulation in Matlab to compare heuristic policies to each other (in all cases) and to exact optimal solutions (for small cases).
  - ◊ Developed a heuristic policy that is robust to variations in value function type, value decay rate, initial value, and expected service times.
- Dynamic scheduling simulation.
  - ◊ Constructed a discrete event simulation in Matlab to represent the dynamics of the physical system.
  - ◊ Developed a mathematical representation of “status quo” in battle field information flow control. Integrated this model into the simulation framework for comparison purposes.

- ◊ Developed a heuristic policy that significantly outperforms the status quo and is robust to variations in value function type, value decay rate, initial value, and arrival rates.
- ◊ Developed an $O\left(n^2\right)$ algorithm for implementing this robust heuristic policy.

These results constitute an important initial step toward greater understanding of the overall problem of battlefield information flow through mathematical modeling and analysis. More is required, however, before we have a representation of the physical system that is realistic and accurate enough for real-world application. In the following section, we describe some potential future directions for this research.

B. FUTURE WORK

1. "Ensembling" of Information Packages

One of the assumptions we have made is that information package value is linearly additive—meaning that the value gained by processing a particular information package is independent of the information packages that have already been processed. We realize that this is not the case in the physical system. In fact, the value of a particular information package *is* dependent upon what has already been processed (and even, to some extent, upon what is *planned* to be processed). It is likely the case that several *related* information packages (related in the sense that they describe the same target or set of targets, or the same geographical region) exist in Oval 2—collected by multiple sources—and that the value of processing them in aggregate is greater than the sum of their individual values. This is analogous to the advantage of multi-source over single-source information described in [5].

To increase the realism of our model, we introduce the notion of the "ensembling" of information packages; i.e., we sequence information packages for processing in ensembles—or related groups—rather than individually. The problem of interest, then, is to develop information flow models for the case where information packages can be

ensembled, together with optimal sequencing results and/or heuristic processing strategies. We introduce two new concepts here that represent a small step toward that end.

The first of these is known as the conditional value function. As we observe above, the value of processing a particular information package is dependent upon what has already been processed. We define a conditional value function, denoted $Val_{i|j}(t)$, as the value of information package *i* at time *t* given that information package *j* has already been processed. It would be safe to assume, as we have with the original value function, that these conditional value functions are piecewise constant, non-increasing functions. Defining these functions would rely heavily on existing sensor data fusion work (see [5], for example).

The second, related, concept is that Oval 2 can be thought of as containing subsets of information packages rather than one large set. Information packages within subsets may be associated by the targets or target sets they describe, the geographical region they describe, the time they arrived, or other factors. As such, it is likely that subsets within Oval 2 will have nonempty intersections. We may wish to consider an entire subset as an ensemble to be processed together, or only certain portions of a subset.

Developing an information flow model that treats information package as ensembles rather than individually would represent an important achievement in that it would relate two previously disjoint modeling areas: dynamic information flow and sensor data fusion.

2. Additional Expected Realized Value Results

Let $Val_i(t) = \beta_i e^{-\alpha_i t}$ be the value function for information package i, $\mathrm{i} = 1, 2, \ldots, n$. Let information package service times be exponentially distributed with rate μ_i and define the permutation mapping $\varphi(j)$ to be the information package processed i^{th}. From (3.40), then, we know that

$$E\left[V^r\right] = \sum_{j=1}^{n} \beta_{\varphi(j)} \prod_{i=1}^{j} \frac{\mu_{\varphi(i)}}{\mu_{\varphi(i)} + \alpha_{\varphi(j)}}. \tag{5.3}$$

This expression represents the expected cumulative realized value obtained by processing the *n* information packages in any sequence φ. The following questions remain open:

- Suppose we process the *n* information packages in the sequence determined by heuristic H5. Does a closed form expression for $E\left[V^r\right]$ exist and can it be derived?
- If not, can we define reasonable *bounds* on $E\left[V^r\right]$ for *n* information packages processed in accordance with H5?
- Can we derive exact expressions (or bounds) for $E\left[V^r\right]$ for other cases of the problem (e.g., different service time distributions and different value function types)?

Answers to these questions would provide a theoretical basis for choosing a heuristic processing strategy to employ in addition to the empirical evidence shown in Chapter IV.

3. Additional Theoretical Results

In Chapter III, we derived results for simplified cases of the optimal control problem (2.4). It would be desirable to extend these results to at least the following cases:

- Mixed value function type case (suspected to be NP-hard)
- Multi-processor case ($m \geq 2$)

4. Extensions to Dynamic Information Flow Modeling

In Chapter II, we construct a dynamic information flow model that describes the flow of information between Ovals 2 and 3 of the Dynamic Model of Situated Cognition. Figure 3 depicts the notion of developing information flow models for other portions of the DMSC toward the ultimate goal of tying these "sub-models" together to form a cohesive representation of dynamic information flow between Ovals 1 and 6. For example, the analysis in Chapter III is all based on the assumption that the number, *Q*, of

information packages to be sequenced is pre-determined, or fixed. We model the static scheduling problem as if the commander has no control over the *contents* of Oval 2; he only controls how the contents of Oval 2 are *scheduled* for processing. This is not the case in actuality. The commander controls his organic sensor assets; he determines how they are allocated, which directly affects the number of information packages present in Oval 2 at any time, and even the *type* of IPs present. So, a model is required that relates the contents of Oval 2 to a commander's sensor allocation strategy (the control). This model could be thought of as "residing" between Ovals 1 and 2 of the dynamic model of situated cognition, describing how information flows between "ground truth" (Oval 1) and the set of information collected (Oval 2). Tying such a model together with the dynamic information flow model presented in Chapters II and III would represent a significant step toward the ultimate goal of mathematically modeling the flow of information between Ovals 1 and 6.

5. Implementing a "Warm" Start in the Dynamic Scheduling Simulation

The dynamic scheduling problem we have investigated assumes a "cold" start; i.e., zero information packages are present in the system at time zero. Another realistic starting condition would be a "warm" start where the scheduling process does not begin until some time after information packages have begun arriving. Preliminary findings seem to indicate that there is no significant difference between the two approaches, but a more comprehensive simulation investigation is required before drawing any firm conclusions.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. SIMULATION CODE (MATLAB) AND ADDITIONAL EXPERIMENTAL RESULTS

A. STATIC SIMULATION CODE

```
% StaticPolicyTest.m
% compares svc policies for sets of IPs
% assumes a fixed # of IPs to be sequenced with no new arrivals
% Created 10 Feb 2008.


% Assumptions:
% - exponential, linear, or step value functions (arbitrary decay rate
and initial value)
%
% No new arrivals
% 3 "classes" of IPs - determines service time distributions

clear all;

% define parameter matrix - each row represents a run.
% cols are:  run #, reps, Q, ValFctMix, k_alpha, theta_alpha, k_beta,
theta_beta,
% k_tau, theta_tau, k_step, theta_step, E_S1, E_S2, E_S3.  Tau is the
ending time for step
% functions.  k_step and theta_step are the parameters of the gamma
% distribution from which the initial value for step value functions is
% drawn.

% ValFctMix is 1 if all exp val fcts; 2 if all linear val fcts;
% 3 if all step fcts; 4 if "mix" - equal mix for now.
param=importdata('runmatrix.csv');

overall=zeros(size(param,1),6);
tic

for run=1:size(param,1)

    reps=param(run,2); %# of iterations
    Q=param(run,3); % number of IPs;
    ValFctMix = param(run,4);
    results=zeros(reps,6); % tracks output for each run

    count=[0 0 0];
    for z=1:reps
        count=[count(1)+1 run size(param,1)]

        %rand('twister',5489); % returns the same values each time
        rand('twister', sum(100*clock)); % random seed driven by clock
time; returns different values each time
```

```
         % User defined inputs:

        numclass = 3;
        E_S=[param(run,13); param(run,14); param(run,15)]; % mu_i are
mean service times for IPs of class i (service time distributions are
assumed to be exponential)

        alpha = gamrnd(param(run,5),param(run,6),Q,1); % Q-long vector
of decay rates (lin and exp)
        beta = gamrnd(param(run,7),param(run,8),Q,1); % Q-long vector
of initial values (lin, exp, and step)
        %step_val = gamrnd(param(run,11),param(run,12),Q,1); % vector
of constant vals for step val fcts
        tau = gamrnd(param(run,9),param(run,10),Q,1); % vector of
ending times (step val fcts only)

        class = ceil(numclass*rand(Q,1)); % Q-long vector of IP classes

        expectedsvctime = E_S(class);
        ValFctForm=zeros(Q,1);
        if ValFctMix == 1
            ValFctForm = ones(Q,1); %if all exp val fcts
        elseif ValFctMix == 2
            ValFctForm = 2*ones(Q,1); % if all linear val fcts
        elseif ValFctMix == 3
            ValFctForm = 3*ones(Q,1); % if all step val fcts
        else
            %ValFctForm = ceil(3*rand(Q,1)); % creates a uniform mix of
all three val fct types.
            vr=rand(Q,1);
            ValFctForm = (vr<.75)+2*(vr>=.75 & vr<.9)+3*(vr>=.9);
        end



        %initialize the IP characteristic data array
        IP=zeros(Q,12);
        IP(:,1) = 1:Q;
        IP(:,2) = alpha;
        IP(:,3) = beta;
        IP(:,4) = class;
        IP(:,5) = expectedsvctime; % this is E[S_sub_q]
        %IP(:,6) = step_val;
        IP(:,7) = tau;
        IP(:,8) = ValFctForm;


        % col's 9-12 of IP are the vector "param" for input to the
        % function ValueFunction.

        for jj=1:Q
            if IP(jj,8)==1 % exp val fct
                IP(jj,9)=IP(jj,2); % alpha
```

```
                IP(jj,10)=IP(jj,3); % beta
            elseif IP(jj,8)==2 % lin val fct
                IP(jj,9)=IP(jj,2); % alpha
                IP(jj,10)=IP(jj,3); % beta
            elseif IP(jj,8)==3 % step val fct
                IP(jj,9)=IP(jj,3);
                IP(jj,10)=IP(jj,7);
            end
        end

        param_val=IP(:,9:12);
        tic
        if Q<=10  %only find exact solution for small Q
            seq = single(perms(1:Q)); %a Q! x Q matrix of all possible
permutations of 1:Q; each row represents a processing sequence.
            seq = single([seq zeros(factorial(Q),1)]); % appends a
column of zeros at the end of seq (overwrites the original to save
memory)
            % this last column will hold realized
            % values for each row (sequence of IPs)
            max_val = 0; % used to track optimal value - updated while
simulation runs

            for j=1:factorial(Q)

                t1=IP(seq(j,1:Q),5);
                t = cumsum(t1); % vector of expected server availble
times
                form=IP(seq(j,1:Q),8); % val fct forms
                a=zeros(Q,1); %arrival time is zero
                param_shuffle=param_val(seq(j,1:Q),:);
                val_k = diag(ValueFunction(t, form, a, param_shuffle));
                realized_val = sum(val_k);
               % for k=1:Q
                %    t = t + IP(seq(j,k),5); % updates running expected
time
                %    val_k = ValueFunction(t, IP(seq(j,k),8), 0,
param_val(seq(j,k),:));
                %    realized_val = realized_val + val_k;
                %end

                seq(j,Q+1) = realized_val;

                if realized_val > max_val
                    max_val = realized_val; % keeps track of current
optimal value
                end


            end

           OptimalValue = max_val;
            results(z,1) = OptimalValue;
```

```
        end
        toc

        %HEURISTIC #1 DOES NOT APPLY TO STEP FUNCTIONS - SO WE OMIT.

        % Heuristic #2:  Really Greedy (sorts on value at t = 0,
without resorting)
        % so, really, this just sorts on Beta.

        IPh2 = sortrows(IP,-3); % sorts by Beta in descending order
        param_val=IPh2(:,9:12);

        t=0;
        rvh2=0;
        for k=1:Q
            t = t + IPh2(k,5); % updates running expected time
            val_k = IPh2(k,3) * exp(-IPh2(k,2)*t);
            val_k = ValueFunction(t, IPh2(k,8), 0, param_val(k,:));
            rvh2 = rvh2 + val_k;
        end

        results(z,3) = rvh2;

        % Heuristic #3:  Pretty Greedy (same as #2, only this time we
re-sort after
        % every IP selection, evaluating the value function at current
time).
        % Expect performance to be no worse than h2.

        [r c] = size(IP);
        IPh3=zeros(r,c+1);
        IPtemp = [IP zeros(r,1)];

        t=0;
        for k=1:Q
            param_val=IPtemp(:,9:12);
            a=zeros(size(IPtemp,1),1);
            IPtemp(:,c+1)=(ValueFunction(t, IPtemp(:,8), a,
param_val))'; %current value
            IPtemp = sortrows(IPtemp,-(c+1));
            IPh3(k,:) = IPtemp(1,:);
            t = t + IPtemp(1,5);  % updates the clock
            IPtemp(1,:) = []; %deletes first row of IPtemp (the row
that was just copied to IPh3)
            % note - IPtemp loses a row each iteration -
            % it eventually disappears!

        end

        t=0;
        rvh3=0;
        param_val=IPh3(:,9:12);
        for k=1:Q
```

```
            t = t + IPh3(k,5); % updates running expected time
            val_k = ValueFunction(t, IPh3(k,8), 0, param_val(k,:));
            rvh3 = rvh3 + val_k;
        end

        results(z,4) = rvh3;

        % Heuristic #4:  Somewhat Greedy - look ahead to predict value
AFTER
        % service time; resort each time

        IPh4=zeros(r,c+1);
        IPtemp = [IP zeros(r,1)];
        t=0;
        for k=1:Q
            param_val=IPtemp(:,9:12);
            a=zeros(size(IPtemp,1),1);
            IPtemp(:,c+1)=diag(ValueFunction(t+IPtemp(:,5),
IPtemp(:,8), a, param_val)); % val at end of svc
            IPtemp = sortrows(IPtemp,-(c+1));
            IPh4(k,:) = IPtemp(1,:);
            t = t + IPtemp(1,5);  % updates the clock
            IPtemp(1,:) = []; %deletes first row of IPtemp (the row
that was just copied to IPh3)
            % note - IPtemp loses a row each iteration -
            % it eventually disappears!
        end

        t=0;
        rvh4=0;
        param_val=IPh4(:,9:12);
        for k=1:Q
            t = t + IPh4(k,5); % updates running expected time
            val_k = ValueFunction(t, IPh4(k,8), 0, param_val(k,:));

            rvh4 = rvh4 + val_k;
        end

        results(z,5) = rvh4;

        %Heuristic #5:  "bang for buck" - descending sort on
        %Val_q(E[d_q])/E[S_q] - resort each time

        IPh5=zeros(r,c+1);
        IPtemp = [IP zeros(r,1)];

        t=0;
        for k=1:Q
            param_val=IPtemp(:,9:12);
            a=zeros(size(IPtemp,1),1);
            IPtemp(:,c+1)=(diag(ValueFunction(t+IPtemp(:,5),
IPtemp(:,8), a, param_val))./IPtemp(:,5); % val at end of svc
            IPtemp = sortrows(IPtemp,-(c+1));
            IPh5(k,:) = IPtemp(1,:);
```

```
            t = t + IPtemp(1,5);  % updates the clock
            IPtemp(1,:) = []; %deletes first row of IPtemp (the row
that was just copied to IPh3)
            % note - IPtemp loses a row each iteration -
            % it eventually disappears!
        end

        t=0;
        rvh5=0;
        param_val=IPh5(:,9:12);
        for k=1:Q
            t = t + IPh5(k,5); % updates running expected time
            val_k = ValueFunction(t, IPh5(k,8), 0, param_val(k,:));
            rvh5 = rvh5 + val_k;
        end

        results(z,6) = rvh5;


    end

    %Normalize results matrix - either to exact optimal (if Q<=10) or
to rvh3
    %(the most reasonable greedy heuristic), then construct numerical
    %histogram.

    [row,col] = size(results);
    results_norm=zeros(row,col);
    if Q<=10 % we normalize on exact optimal results in this case
        for i=1:col
            results_norm(:,i)=results(:,i)./results(:,1); % first
column of results is optimal
        end
        bin=(0:.05:1)'; %bin sizes for histogram
        h = hist(results_norm,bin);
        overall(run,1)=param(run,1);
        for bb=2:6
            overall(run,bb)=sum(results(:,bb))/sum(results(:,1));
        end
    end

    if Q>10 % we normalize on rvh3 in this case
        for i=1:col
            results_norm(:,i)=results(:,i)./results(:,4); % fourth
column of results is rvh3
        end
        bin=(0:.5:100)';
        h = hist(results_norm,bin);

    end

    h=[bin h];
    csvwrite(['hist_run_',num2str(run),'.csv'],h);
```

```
csvwrite(['norm_results_run_',num2str(param(run,1)),'.csv'],results_nor
m);

    %bar3(bin, h)

end
csvwrite('overall.csv',overall);
toc
```

B. VALUE FUNCTION CODE

This function is incorporated as a sub-routine in both the static and dynamic simulation codes.

```
%% y = ValueFunction(t, ValFctForm,a,param)
%% This m-file outputs Val_j(t) for independent variable t (scalar or
%% vector).  ValFctForm indicates the type of value function (1 =
%% exponential; 2 = linear; 3 = step).  a is the arrival time for
package
%% "j" - the given package.  param is a row vector containing function
%% parameters.
%% assume t is k x 1; ValFctForm and a are n x 1; param is n x 4

%% Version 2, 27 June 2007
%% Donovan Phillips

function y = ValueFunction(t, ValFctForm,a,param)

% convert all input to column vectors if not already

if (size(t,1)==1) && (size(t,2)~=1)
    t=t';
end

if (size(ValFctForm,1)==1) && (size(ValFctForm,2)~=1)
    ValFctForm=ValFctForm';
end

if (size(a,1)==1) && (size(a,2)~=1)
    a=a';
end

if size(param,1) ~= max(size(a))
    param=param';
end
```

```
v1 = ValFctForm ~=1; v2 = ValFctForm ~= 2; v3 = ValFctForm ~= 3;

v = v1 .* v2 .* v3; % v will contain non-zero entries if ValFctForm
contains entries other than 1, 2, or 3.

if sum(v) > 0
    error([num2str(ValFctForm),' is an invalid functional form.  Must
use 1, 2, or 3.'])
end

if (size(ValFctForm,1) ~= size(a,1)) || (size(ValFctForm,1) ~=
size(param,1))
    error('the vectors ValFctForm, a, and param must have the same
number of rows')
end


size_param = size(param,2);
if size_param ~= 4
    error('param must have 4 elements for every package')
end
size_t = max(size(t)); size_a = size(a,1);

% organize input data

j = cumsum(ones(size_a,1));

input = [j ValFctForm a param];
input_sort = sortrows(input,2); % sorts on ValFctForm in ascending
order

i1 = input_sort(:,2) == 1; i2 = input_sort(:,2) == 2; i3 =
input_sort(:,2) == 3;

[c1 h1] = min(i1); % c1 is the min value, h1 is the index of the first
instance of this min value
                   % so, row h1 of input_sort is the first row having
                   % ValFctForm other than 1 (note:  could be the
first
                   % row - ie, there ARE no 1's)
[c2 h2] = max(i2); % if c2=1, then there is at least one ValFctForm=2;
the first of these has index h2
[c3 h3] = max(i3); % if c3 = 1 (usually will), h3 is the index of the
first
                   % "3" in ValFctForm.
sumi1=sum(i1); sumi2=sum(i2); sumi3=sum(i3); % indicates the number of
1's in each vector

% Note:  if h1 = h3, then there are no "2's" in ValFctForm; if h1=1,
there
% are no 1's; - need to cover all these bases...

% initialize output matrix, y
```

```
y = zeros(size_t, size_a); % output will have a column for every pkg
input

%************************************************
% Exponential Value Function Form (ValFctForm = 1)
%************************************************

if sumi1 > 0 % ie, if there is at least one row of input_sort with
ValFctForm=1
    input_sort1 = input_sort(1:sumi1,:); % only the rows with
ValFctForm=1
    for k=1:sumi1
        pkg = input_sort1(k,1);
        a = input_sort1(k,3); %arrival time of the kth pkg
        p1 = input_sort1(k,4); p2 = input_sort1(k,5); % function
parameters
        arriv = t >= a; % arriv is a vector - same length as t -
                                        % that is 0 if t<arrival time;
1
                                        % otherwise
        out = p2*exp(-p1*(t-a)); % p1 = alpha, p2 = beta (both
nonnegative)
        y(:,pkg) = arriv.*out; % only considers values of t greater
than arrival time
    end

end

%************************************************
% Linear Value Function Form (ValFctForm = 2)
%************************************************


if sumi2 > 0 % ie, there is at least one ValFctForm = 2
    input_sort2 = input_sort(h2:(h2+sumi2-1),:); % only the rows of
input_sort having ValFctForm=2
    for k = 1:sumi2
        pkg = input_sort2(k,1);
        a = input_sort2(k,3); %arrival time of the kth pkg
        p1 = input_sort2(k,4); p2 = input_sort2(k,5); % function
parameters
        arriv = t >= a; % arriv is a vector - same length as t -
                                        % that is 0 if t<arrival time;
1
                                        % otherwise
        out = max(0, p2 - p1*(t-a)); % p1=alpha, p2=beta, both
nonnegative
        y(:,pkg) = arriv.*out; % only considers values of t greater
than arrival time
    end
end

%************************************************
% Step Value Function Form (ValFctForm = 3)
```

```
%*********************************************

if sumi3 > 0 % ie, there is at least one ValFctForm = 3
    input_sort3 = input_sort(h3:(h3+sumi3-1),:); % only the rows of
input_sort having ValFctForm=3
    for k = 1:sumi3
        pkg = input_sort3(k,1);
        a = input_sort3(k,3); %arrival time of the kth pkg
        p1 = input_sort3(k,4);
        p2 = input_sort3(k,5);
        p3 = input_sort3(k,6);
        p4 = input_sort3(k,7); % function parameters
        %arriv = t >= a; % arriv is a vector - same length as t -
                                        % that is 0 if t<arrival time;
1
                                        % otherwise
        a_to_p2 = (t>=a) & (t<p2); % true if a<=t<p2
        p2_to_p4 = (t>=p2) & (t<p4);

        y(:,pkg) = p1*a_to_p2 + p3*p2_to_p4;
    end
end
```

C. DYNAMIC SIMULATION CODE

```
% simulation to test various service (processing) policies

% first, must create information packages, with associated arrival
times
% (initially assumed exponential), service times (again, exponential -
% different rates for each class of info pkg), and value function.

clear all;
tic
parameters=importdata('DynamicInput.csv');
NumRuns = size(parameters,1);

overall_mean=zeros(NumRuns,8); % first col is run #; then mean/stdev
for h5, h4, h3, LIFO, Random, StatusQuo, & FIFO
overall_std=zeros(NumRuns,8);

for run = 1:NumRuns
    NumReps = parameters(run,2);
    RunResults=zeros(NumReps+2,7); % last two rows hold mean, stdev;
cols are h5 - FIFO (same order as above)
    for rep = 1:NumReps

        run
        rep %visual status output

        %rand('twister',5489); % returns the same values each time
```

```
        rand('twister', sum(100*clock)); % random seed driven by clock
time; returns different values each time

        % USER INPUT SECTION *****************

        NumClass = parameters(run,4);  % k = 1...3 classes (1 -
"EO/IR"; 2 - "Still Imagery"; 3 - "Voice")
        NumValFctForm = parameters(run,27);  % j = 1...3 ValFctForm (1
= exp; 2 = linear; 3 = step)
        ValFctMix = parameters(run,6:8); % percent of val fcts that are
exp, lin, and step, respectively.
        % Number of elements must equal NumValFctForm.
        % elements must sum to 1.
        NumPkg = parameters(run,3); % j = 1...1000 pkgs
        NumSvr = parameters(run,5); % number of information package
processors
        lambda = parameters(run,21:23)'; % arrival rates for each class
        mu = parameters(run,24:26)'; % service RATES by class
        gamparam =
[parameters(run,9:10);parameters(run,11:12);parameters(run,13:14);param
eters(run,15:16);parameters(run,17:18);parameters(run,19:20)];
        % gamparam is a 2-column matrix of parameters for gamma
distributions from
        % which the parameters for each value function type are drawn.
The two
        % cols represent "k" and "theta" - the two parameters of the
gamma distn;
        % each row represents a parameter as follows:  1st and 2nd rows
represent
        % alpha and beta, respectively, for exponential value
functions; 3rd and
        % 4th rows - alpha and beta for linear val fcts; 5th row is
beta (initial
        % value) for step fcts; 6th row is "tau" - ending time for step
val fcts.

        % val fcts are defined as follows.  Exp:  val


        % END USER INPUT SECTION *****************

        % generate class and arrival times (each of length NumPkg):

        mean = 1./lambda;
        inter_arr1 = exprnd(mean(1),NumPkg,1); arr1 =
cumsum(inter_arr1);
        inter_arr2 = exprnd(mean(2),NumPkg,1); arr2 =
cumsum(inter_arr2);
        inter_arr3 = exprnd(mean(3),NumPkg,1); arr3 =
cumsum(inter_arr3);

        arr1c = [arr1 ones(NumPkg,1)];
        arr2c = [arr2 2*ones(NumPkg,1)];
        arr3c = [arr3 3*ones(NumPkg,1)];
```

```
         arrc = [arr1c; arr2c; arr3c];
        arr_sort = sortrows(arrc,1);

        class = arr_sort(1:NumPkg,2); % information pkg class (1, 2 or
3)
        a = arr_sort(1:NumPkg,1);  % arrival times - sorted first to
last
        ExpSvcTime = 1./mu; % expected svc times for each class
        es = ExpSvcTime(class); % expected svc times for each IP
        endtime = a(NumPkg)*1.01;

        % generate ValFctForm vector according to ValFctMix:

        vr=rand(NumPkg,1);
        ValFctForm=zeros(NumPkg,1);
        vt = [0 cumsum(ValFctMix)];
        for j=1:NumValFctForm
            ValFctForm=ValFctForm + j*(vr > vt(j) & vr <= vt(j+1));
        end

        % generate service times

        s = exprnd(1./mu(class));

        % initialize and construct the parameter array:

        param = zeros(NumPkg,4); %keep the 4 cols:  ValueFunction.m
requires it (cols 3 & 4 will be zeros)
        for i=1:NumPkg
            if ValFctForm(i)==1 % exponential
                param(i,1) = gamrnd(gamparam(1,1),gamparam(1,2)); %
alpha
                param(i,2) = gamrnd(gamparam(2,1),gamparam(2,2)); %
beta
            elseif ValFctForm(i) == 2 % linear
                param(i,1) = gamrnd(gamparam(3,1),gamparam(3,2)); %
alpha
                param(i,2) = gamrnd(gamparam(4,1),gamparam(4,2)); %
beta
            elseif ValFctForm(i) == 3 % step
                param(i,1) = gamrnd(gamparam(5,1),gamparam(5,2)); %
beta
                param(i,2) = a(i) +
gamrnd(gamparam(6,1),gamparam(6,2)); % tau
            end
        end


        % determine "realized value" over time for various
        % service policies:

        %*************************************************
```

```
        % SERVICE POLICY #5:  (value at expected svc
completion)/(expected svc
        % time) H5
        %**************************************************

        IPs5 = [(1:NumPkg)' a zeros(NumPkg,1)]; % first col is IP
permanent index
        % sequenced by arrival time; second col is arrival time; third
col is an
        % indicator which is 1 if the IP has already been sent to
server or 0 if
        % not.

        SvrWkld5 = zeros(NumSvr,1); % tracks total service time
assigned to each server.
        SvrAsgn5 = zeros(NumPkg,3); %each row represents a package
(sorted by arrival
        % time, just like IPs); first column
        %is the server assigned; second column is
        %processing start time; third column is
        %processing completion time.
        ct5 = zeros(NumPkg,1);
        ct5(1) = a(1); % ct="clock time" - starts upon first arrival.
        % note - ct is the time an assignment decision is made - at
        % least one server must be available.

        for i = 1:NumPkg
            % extract from IPs those pkgs that have 1) arrived (i.e.,
a<=ct) and 2)
            % those that have not yet been sent to svr (3rd col of IPs
is zero)

            IPtemp = sortrows(IPs5,2); %sorts in ascending order on
arrival time
            arr = IPtemp(:,2) <= ct5(i); % arr = 1 if IP has arrived
(a<=ct5); arr = 0 otherwise
            % note:  arr is always size (NumPkg x 1).
            % Also, sum(arr) = # of arrivals as of
            % ct(i).
            [cc dd] = min(arr); % dd is the index of the first zero
term, if there is a zero
            if cc ~=1 % ie, if there is at least one IP that hasn't
"arrived"
                rr=size(IPtemp,1); % number of rows remaining in IPtemp
                IPtemp(dd:rr,:) = []; % deletes all rows w/arrival
times greater than current time
            end

            IPtemp = sortrows(IPtemp, -3); % sorts in descending order
on col 3 (sent to svr)
            [aa bb] = min(IPtemp(:,3)); % bb is the row containing the
first zero
```

```
            IPtemp(1:(bb-1),:) = []; % deletes rows having svr assg
indicator of 1 (sent to svr)

            r = size(IPtemp,1); % number of rows remaining in IPtemp

            % here's where we implement h5:

            IPtemp = [IPtemp zeros(r,1)]; % add a column to hold the
current value
            rows = IPtemp(:,1); % these are the IP indices remaining in
IPtemp
            ExpDep = ct5(i) + es(rows); % expected departure time for
each IP if it is sent to svc at time=ct(i)

            IPtemp(:,4) =
(diag(ValueFunction(ExpDep,ValFctForm(rows),a(rows),param(rows,:)))./e
s(rows);
            % computes h5 for each IP left in IPtemp

            IPtemp = sortrows(IPtemp,-4); % sorts on h5, descending
            IPpick = IPtemp(1,1); %choose IP w/highest h5 value
            IPs5(IPpick,3) = 1; %IP w/highest h5 value goes to server

            [v index] = min(SvrWkld5); % index is the next avail svr; v
is the time it's avail.
            SvrAsgn5(IPpick,1) = index; %assigns IPpick to the next
available server
            if a(IPpick) <= v
                SvrAsgn5(IPpick,2) = v;
            else
                SvrAsgn5(IPpick,2) = a(IPpick); % svc commencement is
the later of arrival time and svr avail time.
            end
            SvrAsgn5(IPpick,3) = SvrAsgn5(IPpick,2) + s(IPpick); % svc
completion time
            SvrWkld5(index) = SvrAsgn5(IPpick,3); % time when this
server is next available.

            % update clock time (except for the last iteration):
            if i < NumPkg

                if (sum(arr) >= i + 1) && (min(SvrWkld5) < a(i+1))
                    ct5(i+1) = min(SvrWkld5);
                else
                    ct5(i+1) = max(a(i+1), min(SvrWkld5));
                end

            end

        end

        % save this for output and analysis as necessary.
```

```
        IPdata5 = [IPs5(:,1:2) SvrAsgn5 zeros(NumPkg,1)]; % col 1 is IP
index (sequenced by arrival time);
        % col 2 is arrival time; col 3 is server
        % assigned; col 4 svc start time; col 5 is svc
        % completion time; col 6 is realized value.

        IPdata5(:,6) = diag(ValueFunction(IPdata5(:,5), ValFctForm, a,
param));

        IPdata5sort = sortrows(IPdata5,5); % sort on svc completion
time
        IPdata5sort = [IPdata5sort cumsum(IPdata5sort(:,6))]; % add
additional col for cumulative realized value

        cutoff = IPdata5sort(:,5) <= endtime;
        [g2 h2]=min(cutoff);

        RunResults(rep,1) = IPdata5sort(h2,7); % this is total realized
value for h5 for this rep

        %*************************************************
        % SERVICE POLICY #4:  Value at expected svc completion h4
        %*************************************************

        IPs4 = [(1:NumPkg)' a zeros(NumPkg,1)]; % first col is IP
permanent index
        % sequenced by arrival time; second col is arrival time; third
col is an
        % indicator which is 1 if the IP has already been sent to
server or 0 if
        % not.

        SvrWkld4 = zeros(NumSvr,1); % tracks total service time
assigned to each server.
        SvrAsgn4 = zeros(NumPkg,3); %each row represents a package
(sorted by arrival
        % time, just like IPs); first column
        %is the server assigned; second column is
        %processing start time; third column is
        %processing completion time.
        ct4 = zeros(NumPkg,1);
        ct4(1) = a(1); % ct="clock time" - starts upon first arrival.
        % note - ct is the time an assignment decision is made - at
        % least one server must be available.

        for i = 1:NumPkg
            % extract from IPs those pkgs that have 1) arrived (i.e.,
a<=ct) and 2)
            % those that have not yet been sent to svr (3rd col of IPs
is zero)

            IPtemp = sortrows(IPs4,2); %sorts in ascending order on
arrival time
```

```
            arr = IPtemp(:,2) <= ct4(i); % arr = 1 if IP has arrived
(a<=ct4); arr = 0 otherwise
            % note:  arr is always size (NumPkg x 1).
            % Also, sum(arr) = # of arrivals as of
            % ct(i).
            [cc dd] = min(arr); % dd is the index of the first zero
term, if there is a zero
            if cc ~=1 % ie, if there is at least one IP that hasn't
"arrived"
                rr=size(IPtemp,1); % number of rows remaining in IPtemp
                IPtemp(dd:rr,:) = []; % deletes all rows w/arrival
times greater than current time
            end

            IPtemp = sortrows(IPtemp, -3); % sorts in descending order
on col 3 (sent to svr)
            [aa bb] = min(IPtemp(:,3)); % bb is the row containing the
first zero

            IPtemp(1:(bb-1),:) = []; % deletes rows having svr assg
indicator of 1 (sent to svr)

            r = size(IPtemp,1); % number of rows remaining in IPtemp

            % here's where we implement h4:

            IPtemp = [IPtemp zeros(r,1)]; % add a column to hold the
current value
            rows = IPtemp(:,1); % these are the IP indices remaining in
IPtemp
            ExpDep = ct4(i) + es(rows); % expected departure time for
each IP if it is sent to svc at time=ct(i)

            IPtemp(:,4) =
(diag(ValueFunction(ExpDep,ValFctForm(rows),a(rows),param(rows,:))));
            % computes h4 for each IP left in IPtemp

            IPtemp = sortrows(IPtemp,-4); % sorts on h4, descending
            IPpick = IPtemp(1,1); %choose IP w/highest h4 value
            IPs4(IPpick,3) = 1; %IP w/highest h4 value goes to server

            [v index] = min(SvrWkld4); % index is the next avail svr; v
is the time it's avail.
            SvrAsgn4(IPpick,1) = index; %assigns IPpick to the next
available server
            if a(IPpick) <= v
                SvrAsgn4(IPpick,2) = v;
            else
                SvrAsgn4(IPpick,2) = a(IPpick); % svc commencement is
the later of arrival time and svr avail time.
            end
            SvrAsgn4(IPpick,3) = SvrAsgn4(IPpick,2) + s(IPpick); % svc
completion time
```

```
            SvrWkld4(index) = SvrAsgn4(IPpick,3); % time when this
server is next available.

            % update clock time (except for the last iteration):
            if i < NumPkg

                if (sum(arr) >= i + 1) && (min(SvrWkld4) < a(i+1))
                    ct4(i+1) = min(SvrWkld4);
                else
                    ct4(i+1) = max(a(i+1), min(SvrWkld4));
                end

            end

        end

        % save this for output and analysis as necessary.
        IPdata4 = [IPs4(:,1:2) SvrAsgn4 zeros(NumPkg,1)]; % col 1 is IP
index (sequenced by arrival time);
        % col 2 is arrival time; col 3 is server
        % assigned; col 4 svc start time; col 5 is svc
        % completion time; col 6 is realized value.

        IPdata4(:,6) = diag(ValueFunction(IPdata4(:,5), ValFctForm, a,
param));

        IPdata4sort = sortrows(IPdata4,5); % sort on svc completion
time
        IPdata4sort = [IPdata4sort cumsum(IPdata4sort(:,6))]; % add
additional col for cumulative realized value

        cutoff = IPdata4sort(:,5) <= endtime;
        [g2 h2]=min(cutoff);

        RunResults(rep,2) = IPdata4sort(h2,7); % this is total realized
value for h4 for this rep

        %*************************************************
        % SERVICE POLICY #3:  Highest Current Value (h3)
        %*************************************************

        IPs3 = [(1:NumPkg)' a zeros(NumPkg,1)]; % first col is IP
permanent index
        % sequenced by arrival time; second col is arrival time; third
col is an
        % indicator which is 1 if the IP has already been sent to
server or 0 if
        % not.

        SvrWkld3 = zeros(NumSvr,1); % tracks total service time
assigned to each server.
        SvrAsgn3 = zeros(NumPkg,3); %each row represents a package
(sorted by arrival
```

```
        % time, just like IPs3); first column
        %is the server assigned; second column is
        %processing start time; third column is
        %processing completion time.
        ct = zeros(NumPkg,1);
        ct(1) = a(1); % ct="clock time" - starts upon first arrival.
        % note - ct is the time an assignment decision is made - at
        % least one server must be available.

        for i = 1:NumPkg
            % extract from IPs3 those pkgs that have 1) arrived (i.e.,
a<=ct) and 2)
            % those that have not yet been sent to svr (3rd col of IPs3
is zero)

            IPtemp = sortrows(IPs3,2); %sorts in ascending order on
arrival time
            arr = IPtemp(:,2) <= ct(i); % arr = 1 if IP has arrived
(a<=ct); arr = 0 otherwise
            % note:  arr is always size (NumPkg x 1).
            % Also, sum(arr) = # of arrivals as of
            % ct(i).
            [cc dd] = min(arr); % dd is the index of the first zero
term, if there is a zero
            if cc ~=1
                rr=size(IPtemp,1); % number of rows remaining in IPtemp
                IPtemp(dd:rr,:) = []; % deletes all rows w/arrival
times greater than current time
            end

            IPtemp = sortrows(IPtemp, -3); % sorts in descending order
on col 3
            [aa bb] = min(IPtemp(:,3)); % bb is the row containing the
first zero

            IPtemp(1:(bb-1),:) = []; % deletes rows having svr assg
indicator of 1 (sent to svr)

            r = size(IPtemp,1); % number of rows remaining in IPtemp
            IPtemp = [IPtemp zeros(r,1)]; % add a column to hold the
current value
            rows = IPtemp(:,1); % these are the IP indices remaining in
IPtemp

            IPtemp(:,4) =
ValueFunction(ct(i),ValFctForm(rows),a(rows),param(rows,:));
            % computes the current value of each IP left in IPtemp

            IPtemp = sortrows(IPtemp,-4); % sorts on current value,
descending
            IPpick = IPtemp(1,1); %choose IP w/highest current value
            IPs3(IPpick,3) = 1; %IP w/highest current value goes to
server
```

```
            [v index] = min(SvrWkld3); % index is the next avail svr; v
is the time it's avail.
            SvrAsgn3(IPpick,1) = index; %assigns IPpick to the next
available server
            if a(IPpick) <= v
                SvrAsgn3(IPpick,2) = v;
            else
                SvrAsgn3(IPpick,2) = a(IPpick); % svc commencement is
the later of arrival time and svr avail time.
            end
            SvrAsgn3(IPpick,3) = SvrAsgn3(IPpick,2) + s(IPpick); % svc
completion time
            SvrWkld3(index) = SvrAsgn3(IPpick,3); % time when this
server is next available.

            % update clock time (except for the last iteration):
            if i < NumPkg

                if (sum(arr) >= i + 1) && (min(SvrWkld3) < a(i+1))
                    ct(i+1) = min(SvrWkld3);
                else
                    ct(i+1) = max(a(i+1), min(SvrWkld3));
                end

            end

        end

        % save this for output and analysis as necessary.
        IPdata3 = [IPs3(:,1:2) SvrAsgn3 zeros(NumPkg,1)]; % col 1 is IP
index (sequenced by arrival time);
        % col 2 is arrival time; col 3 is server
        % assigned; col 4 svc start time; col 5 is svc
        % completion time; col 6 is realized value.
        % completion time.

        IPdata3(:,6) = diag(ValueFunction(IPdata3(:,5), ValFctForm, a,
param));

        IPdata3sort = sortrows(IPdata3,5); % sort on svc completion
time
        IPdata3sort = [IPdata3sort cumsum(IPdata3sort(:,6))]; % add
additional col for cumulative realized value

        cutoff = IPdata3sort(:,5) <= endtime;
        [g2 h2]=min(cutoff);

        RunResults(rep,3) = IPdata3sort(h2,7); % this is total realized
value for h3 for this rep

        %*************************************************
        % SERVICE POLICY #7:  LIFO - h7
        %*************************************************
```

```
        IPs7 = [(1:NumPkg)' a zeros(NumPkg,1)]; % first col is IP
permanent index
        % sequenced by arrival time; second col is arrival time; third
col is an
        % indicator which is 1 if the IP has already been sent to
server or 0 if
        % not.

        SvrWkld7 = zeros(NumSvr,1); % tracks total service time
assigned to each server.
        SvrAsgn7 = zeros(NumPkg,3); %each row represents a package
(sorted by arrival
        % time, just like IPs); first column
        %is the server assigned; second column is
        %processing start time; third column is
        %processing completion time.
        ct7 = zeros(NumPkg,1);
        ct7(1) = a(1); % ct="clock time" - starts upon first arrival.
        % note - ct is the time an assignment decision is made - at
        % least one server must be available.

        for i = 1:NumPkg
            % extract from IPs those pkgs that have 1) arrived (i.e.,
a<=ct) and 2)
            % those that have not yet been sent to svr (3rd col of IPs
is zero)

            IPtemp = sortrows(IPs7,2); %sorts in ascending order on
arrival time
            arr = IPtemp(:,2) <= ct7(i); % arr = 1 if IP has arrived
(a<=ct7); arr = 0 otherwise
            % note:  arr is always size (NumPkg x 1).
            % Also, sum(arr) = # of arrivals as of
            % ct(i).
            [cc dd] = min(arr); % dd is the index of the first zero
term, if there is a zero
            if cc ~=1 % ie, if there is at least one IP that hasn't
"arrived"
                rr=size(IPtemp,1); % number of rows remaining in IPtemp
                IPtemp(dd:rr,:) = []; % deletes all rows w/arrival
times greater than current time
            end

            IPtemp = sortrows(IPtemp, -3); % sorts in descending order
on col 3 (sent to svr)
            [aa bb] = min(IPtemp(:,3)); % bb is the row containing the
first zero

            IPtemp(1:(bb-1),:) = []; % deletes rows having svr assg
indicator of 1 (sent to svr)

            r = size(IPtemp,1); % number of rows remaining in IPtemp

            % here's where we implement h7:
```

```
            [ar lastin] = max(IPtemp(:,2));
            IPpick = IPtemp(lastin,1);

            IPs7(IPpick,3) = 1; %IP w/highest h7 value goes to server

            [v index] = min(SvrWkld7); % index is the next avail svr; v
is the time it's avail.
            SvrAsgn7(IPpick,1) = index; %assigns IPpick to the next
available server
            if a(IPpick) <= v
                SvrAsgn7(IPpick,2) = v;
            else
                SvrAsgn7(IPpick,2) = a(IPpick); % svc commencement is
the later of arrival time and svr avail time.
            end
            SvrAsgn7(IPpick,3) = SvrAsgn7(IPpick,2) + s(IPpick); % svc
completion time
            SvrWkld7(index) = SvrAsgn7(IPpick,3); % time when this
server is next available.

            % update clock time (except for the last iteration):
            if i < NumPkg

                if (sum(arr) >= i + 1) && (min(SvrWkld7) < a(i+1))
                    ct7(i+1) = min(SvrWkld7);
                else
                    ct7(i+1) = max(a(i+1), min(SvrWkld7));
                end

            end

        end

        % save this for output and analysis as necessary.
        IPdata7 = [IPs7(:,1:2) SvrAsgn7 zeros(NumPkg,1)]; % col 1 is IP
index (sequenced by arrival time);
        % col 2 is arrival time; col 3 is server
        % assigned; col 4 svc start time; col 5 is svc
        % completion time; col 6 is realized value.

        IPdata7(:,6) = diag(ValueFunction(IPdata7(:,5), ValFctForm, a,
param));

        IPdata7sort = sortrows(IPdata7,5); % sort on svc completion
time
        IPdata7sort = [IPdata7sort cumsum(IPdata7sort(:,6))]; % add
additional col for cumulative realized value

        cutoff = IPdata7sort(:,5) <= endtime;
        [g2 h2]=min(cutoff);

        RunResults(rep,4) = IPdata7sort(h2,7); % this is total realized
value for LIFO for this rep
```

```
        %*************************************************
        % SERVICE POLICY #8:  Random - h8
        %*************************************************

        IPs8 = [(1:NumPkg)' a zeros(NumPkg,1)]; % first col is IP
permanent index
        % sequenced by arrival time; second col is arrival time; third
col is an
        % indicator which is 1 if the IP has already been sent to
server or 0 if
        % not.

        SvrWkld8 = zeros(NumSvr,1); % tracks total service time
assigned to each server.
        SvrAsgn8 = zeros(NumPkg,3); %each row represents a package
(sorted by arrival
        % time, just like IPs); first column
        %is the server assigned; second column is
        %processing start time; third column is
        %processing completion time.
        ct8 = zeros(NumPkg,1);
        ct8(1) = a(1); % ct="clock time" - starts upon first arrival.
        % note - ct is the time an assignment decision is made - at
        % least one server must be available.

        for i = 1:NumPkg
            % extract from IPs those pkgs that have 1) arrived (i.e.,
a<=ct) and 2)
            % those that have not yet been sent to svr (3rd col of IPs
is zero)

            IPtemp = sortrows(IPs8,2); %sorts in ascending order on
arrival time
            arr = IPtemp(:,2) <= ct8(i); % arr = 1 if IP has arrived
(a<=ct8); arr = 0 otherwise
            % note:  arr is always size (NumPkg x 1).
            % Also, sum(arr) = # of arrivals as of
            % ct(i).
            [cc dd] = min(arr); % dd is the index of the first zero
term, if there is a zero
            if cc ~=1 % ie, if there is at least one IP that hasn't
"arrived"
                rr=size(IPtemp,1); % number of rows remaining in IPtemp
                IPtemp(dd:rr,:) = []; % deletes all rows w/arrival
times greater than current time
            end

            IPtemp = sortrows(IPtemp, -3); % sorts in descending order
on col 3 (sent to svr)
            [aa bb] = min(IPtemp(:,3)); % bb is the row containing the
first zero

            IPtemp(1:(bb-1),:) = []; % deletes rows having svr assg
indicator of 1 (sent to svr)
```

```
            r = size(IPtemp,1); % number of rows remaining in IPtemp

            % here's where we implement h8 - pick an IP from IPtemp at
random to send to svr:
            pick = ceil(r*rand);
            IPpick = IPtemp(pick,1);

            IPs8(IPpick,3) = 1; %IP w/highest h8 value goes to server

            [v index] = min(SvrWkld8); % index is the next avail svr; v
is the time it's avail.
            SvrAsgn8(IPpick,1) = index; %assigns IPpick to the next
available server
            if a(IPpick) <= v
                SvrAsgn8(IPpick,2) = v;
            else
                SvrAsgn8(IPpick,2) = a(IPpick); % svc commencement is
the later of arrival time and svr avail time.
            end
            SvrAsgn8(IPpick,3) = SvrAsgn8(IPpick,2) + s(IPpick); % svc
completion time
            SvrWkld8(index) = SvrAsgn8(IPpick,3); % time when this
server is next available.

            % update clock time (except for the last iteration):
            if i < NumPkg

                if (sum(arr) >= i + 1) && (min(SvrWkld8) < a(i+1))
                    ct8(i+1) = min(SvrWkld8);
                else
                    ct8(i+1) = max(a(i+1), min(SvrWkld8));
                end

            end

        end

        % save this for output and analysis as necessary.
        IPdata8 = [IPs8(:,1:2) SvrAsgn8 zeros(NumPkg,1)]; % col 1 is IP
index (sequenced by arrival time);
        % col 2 is arrival time; col 3 is server
        % assigned; col 4 svc start time; col 5 is svc
        % completion time; col 6 is realized value.

        IPdata8(:,6) = diag(ValueFunction(IPdata8(:,5), ValFctForm, a,
param));

        IPdata8sort = sortrows(IPdata8,5); % sort on svc completion
time
        IPdata8sort = [IPdata8sort cumsum(IPdata8sort(:,6))]; % add
additional col for cumulative realized value

        cutoff = IPdata8sort(:,5) <= endtime;
```

```
        [g2 h2]=min(cutoff);

        RunResults(rep,5) = IPdata8sort(h2,7); % this is total realized
value for RANDOM for this rep

        %*************************************************
        % SERVICE POLICY #9:  "Status Quo"
        %*************************************************

        % this heuristic attempts to replicate the current strategy for
choosing
        % IPs to process by combining three of the above heuristics:
h3, LIFO,
        % and Random.  These can be "evenly" combined, or weighted.
        weight = [.3 .6 .1]; % weights for h3, LIFO, and Random,
respectively.
        %Must sum to one.
        st = rand(NumPkg,1);
        strategy = zeros(NumPkg,1);
        str = [0 cumsum(weight)];
        for j=1:3
            strategy = strategy + j*(st > str(j) & st <= str(j+1));
        end
        % strategy is a vector of length NumPkg that indicates a 1 if
h3 is to be
        % used, 2 if LIFO is to be used, and 3 if Random is to be used.
Its
        % elements are randomly generated according to the vector
"weight."

        IPs9 = [(1:NumPkg)' a zeros(NumPkg,1)]; % first col is IP
permanent index
        % sequenced by arrival time; second col is arrival time; third
col is an
        % indicator which is 1 if the IP has already been sent to
server or 0 if
        % not.

        SvrWkld9 = zeros(NumSvr,1); % tracks total service time
assigned to each server.
        SvrAsgn9 = zeros(NumPkg,3); %each row represents a package
(sorted by arrival
        % time, just like IPs); first column
        %is the server assigned; second column is
        %processing start time; third column is
        %processing completion time.
        ct9 = zeros(NumPkg,1);
        ct9(1) = a(1); % ct="clock time" - starts upon first arrival.
        % note - ct is the time an assignment decision is made - at
        % least one server must be available.

        for i = 1:NumPkg
            % extract from IPs those pkgs that have 1) arrived (i.e.,
a<=ct) and 2)
```

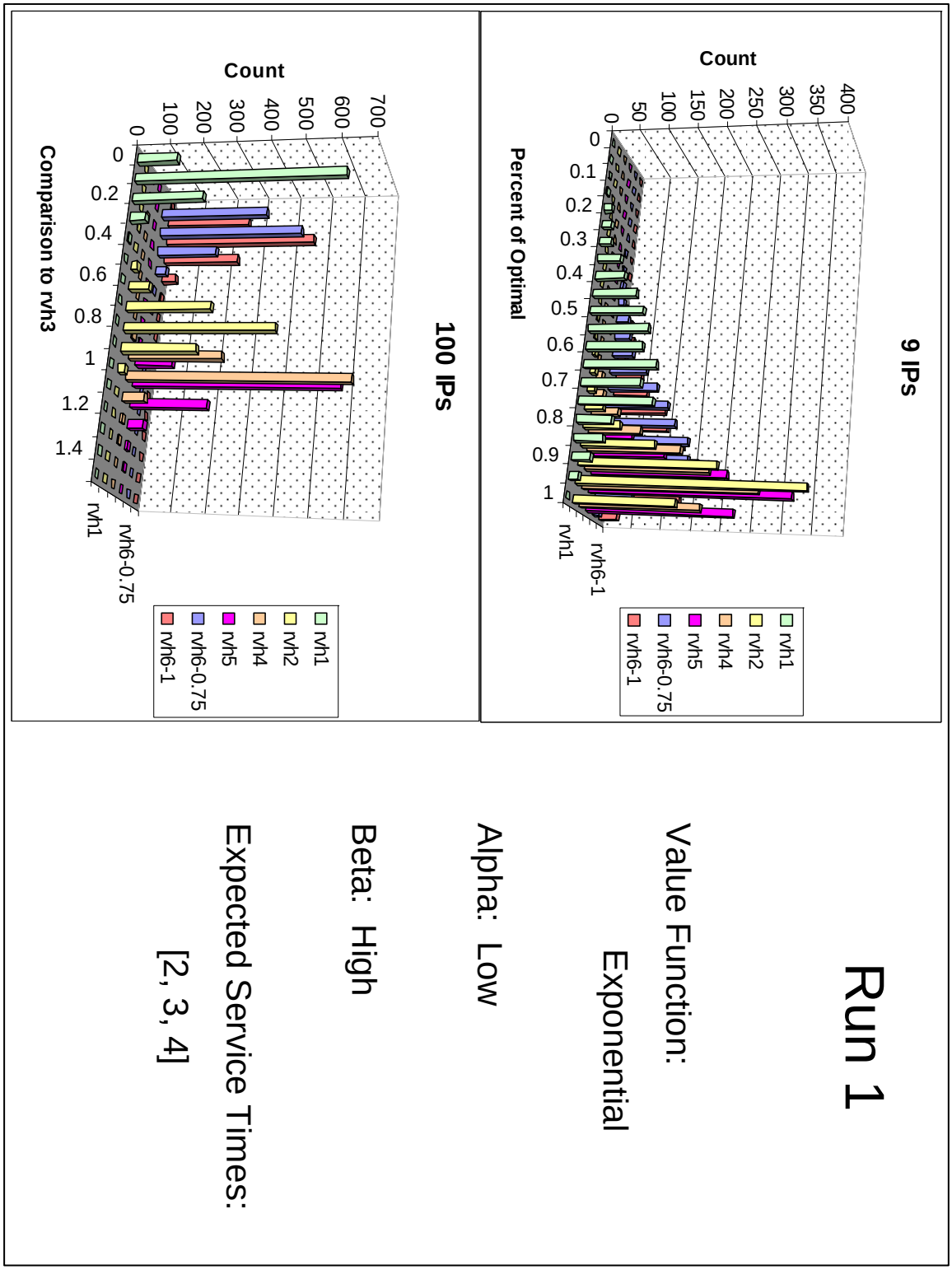
```
            % those that have not yet been sent to svr (3rd col of IPs
is zero)

            IPtemp = sortrows(IPs9,2); %sorts in ascending order on
arrival time
            arr = IPtemp(:,2) <= ct9(i); % arr = 1 if IP has arrived
(a<=ct9); arr = 0 otherwise
            % note:  arr is always size (NumPkg x 1).
            % Also, sum(arr) = # of arrivals as of
            % ct(i).
            [cc dd] = min(arr); % dd is the index of the first zero
term, if there is a zero
            if cc ~=1 % ie, if there is at least one IP that hasn't
"arrived"
                rr=size(IPtemp,1); % number of rows remaining in IPtemp
                IPtemp(dd:rr,:) = []; % deletes all rows w/arrival
times greater than current time
            end

            IPtemp = sortrows(IPtemp, -3); % sorts in descending order
on col 3 (sent to svr)
            [aa bb] = min(IPtemp(:,3)); % bb is the row containing the
first zero

            IPtemp(1:(bb-1),:) = []; % deletes rows having svr assg
indicator of 1 (sent to svr)

            r = size(IPtemp,1); % number of rows remaining in IPtemp

            % here's where we implement h9:  based on value of
strategy(i), choose
            % h3, LIFO, or Random.

            if strategy(i) == 3 % random
                pick = ceil(r*rand);
                IPpick = IPtemp(pick,1);
            elseif strategy(i) == 2 % LIFO
                [ar lastin] = max(IPtemp(:,2));
                IPpick = IPtemp(lastin,1);
            else                        % h3
                IPtemp = [IPtemp zeros(r,1)]; % add a column to hold
the current value
                rows = IPtemp(:,1); % these are the IP indices
remaining in IPtemp

                IPtemp(:,4) =
ValueFunction(ct(i),ValFctForm(rows),a(rows),param(rows,:));
                % computes the current value of each IP left in IPtemp

                IPtemp = sortrows(IPtemp,-4); % sorts on current value,
descending
                IPpick = IPtemp(1,1); %choose IP w/highest current
value
            end
```

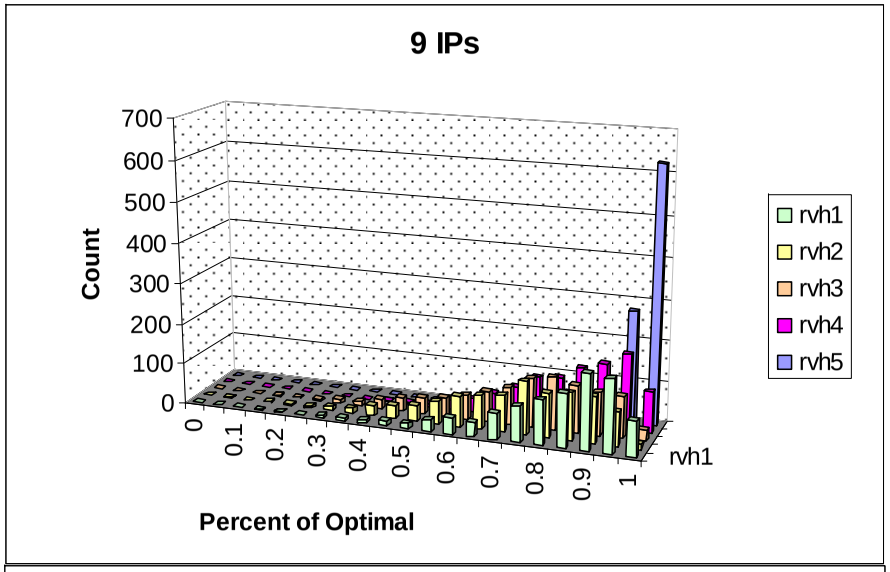
```
            IPs9(IPpick,3) = 1; %IP w/highest h9 value goes to server

            [v index] = min(SvrWkld9); % index is the next avail svr; v
is the time it's avail.
            SvrAsgn9(IPpick,1) = index; %assigns IPpick to the next
available server
            if a(IPpick) <= v
                SvrAsgn9(IPpick,2) = v;
            else
                SvrAsgn9(IPpick,2) = a(IPpick); % svc commencement is
the later of arrival time and svr avail time.
            end
            SvrAsgn9(IPpick,3) = SvrAsgn9(IPpick,2) + s(IPpick); % svc
completion time
            SvrWkld9(index) = SvrAsgn9(IPpick,3); % time when this
server is next available.

            % update clock time (except for the last iteration):
            if i < NumPkg

                if (sum(arr) >= i + 1) && (min(SvrWkld9) < a(i+1))
                    ct9(i+1) = min(SvrWkld9);
                else
                    ct9(i+1) = max(a(i+1), min(SvrWkld9));
                end

            end

        end

        % save this for output and analysis as necessary.
        IPdata9 = [IPs9(:,1:2) SvrAsgn9 zeros(NumPkg,1)]; % col 1 is IP
index (sequenced by arrival time);
        % col 2 is arrival time; col 3 is server
        % assigned; col 4 svc start time; col 5 is svc
        % completion time; col 6 is realized value.

        IPdata9(:,6) = diag(ValueFunction(IPdata9(:,5), ValFctForm, a,
param));

        IPdata9sort = sortrows(IPdata9,5); % sort on svc completion
time
        IPdata9sort = [IPdata9sort cumsum(IPdata9sort(:,6))]; % add
additional col for cumulative realized value

        cutoff = IPdata9sort(:,5) <= endtime;
        [g2 h2]=min(cutoff);

        RunResults(rep,6) = IPdata9sort(h2,7); % this is total realized
value for StatusQuo for this rep

        %****************************************
        % SERVICE POLICY #1:  FIFO:
```

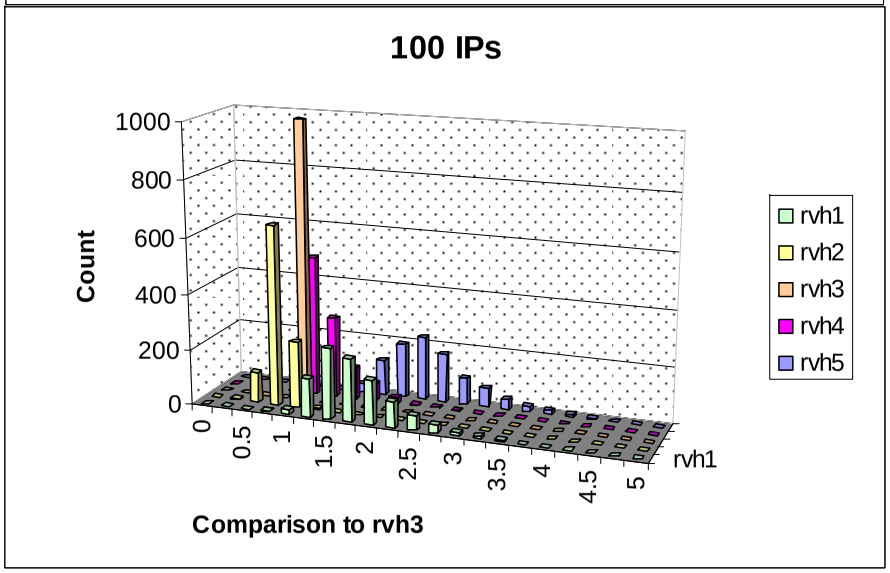
```
        %****************************************

        SvrWkld = zeros(NumSvr,1); % tracks total service time assigned
to each server.
        SvrAsgn = zeros(NumPkg,3); %each row represents a package;
first column
        %is the server assigned; second column is
        %processing start time; third column is
        %processing completion time.

        %NOTE:  the following construction assumes that pkgs are listed
in order of
        %arrival time!!
        for j=1:NumPkg
            [v index] = min(SvrWkld); % index gives the next server
available
            SvrAsgn(j,1) = index; %assign the next available server to
pkg j.
            if a(j) <= v
                SvrAsgn(j,2) = v;
            else
                SvrAsgn(j,2) = a(j); % service commencement is the
smallest of v and a(j)
            end
            SvrAsgn(j,3) = SvrAsgn(j,2) + s(j); % computes service
completion time for pkg j.
            SvrWkld(index) = SvrAsgn(j,3); % time when this server is
next available

        end

        val_r = diag(ValueFunction(SvrAsgn(:,3),ValFctForm,a,param));
%realized value for each package

        % the sorting below accounts for the fact that even though pkgs
are
        % ENTERING service in order of arrival, they do not necessarily
exit the
        % server in the same order (some later arriving pkgs are
finished
        % processing before some earlier arriving ones).
        valmat = [SvrAsgn(:,3) val_r]; % svc completion times paired
w/realized values
        valmatsort = sortrows(valmat,1); % sort on svc completion times

        val_r_t = cumsum(valmatsort(:,2)); % cumulatively sum the
realized values for each pkg

        SvrCompleteTimes = valmatsort(:,1); %NOTE:  SvrCompleteTimes is
NOT sorted in pkg number order!!

        %plot(SvrCompleteTimes,val_r_t) % plots realized value vs. time
for all pkgs
```

```
        cutoff=SvrCompleteTimes<=endtime;
        [g h]=min(cutoff);  % h is the index of the first zero, which
equates to the first svc complete time later than endtime.

        RunResults(rep,7) = val_r_t(h); % this is total realized value
for FIFO for this rep

    end
    meanrow=NumReps+1;
    stdrow=NumReps+2;
    TempResults = RunResults(1:NumReps,:);
    RunResults(meanrow,:) = Mean(TempResults); %mean for each policy

    RunResults(stdrow,:) = std(TempResults); % standard deviation

    overall_mean(run,1) = parameters(run,1); % run number
    overall_std(run,1) = parameters(run,1);
    overall_mean(run,2:8) = RunResults(meanrow,:);
    overall_std(run,2:8) = RunResults(stdrow,:);


csvwrite(['run_',num2str(parameters(run,1)),'_results.csv'],RunResults)
;

end
toc

csvwrite([num2str(parameters(1,1)),'_overall_mean.csv'],overall_mean);
csvwrite([num2str(parameters(1,1)),'_overall_std.csv'],overall_std);
```

D. VALIDATION OF COMPARISON STUDY—DETAILED RESULTS







Run 2

Value Function:

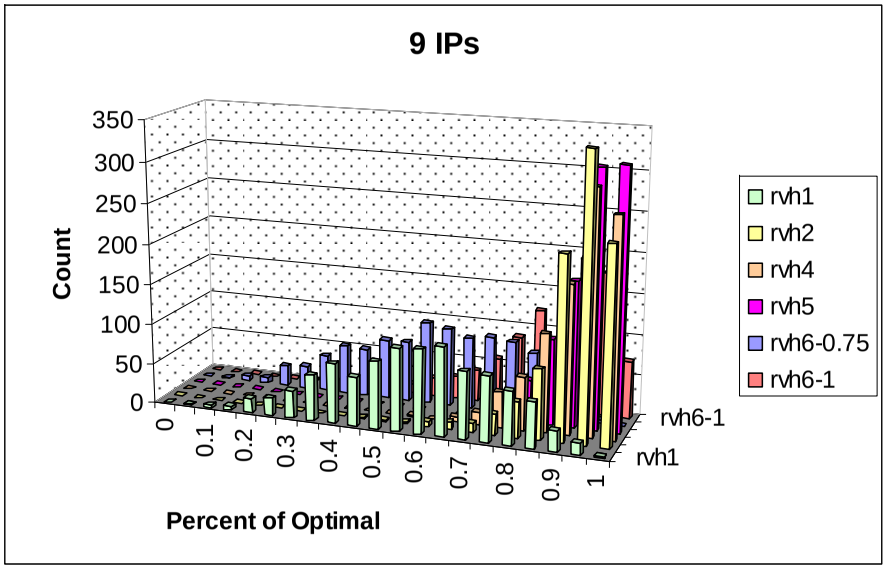
Exponential

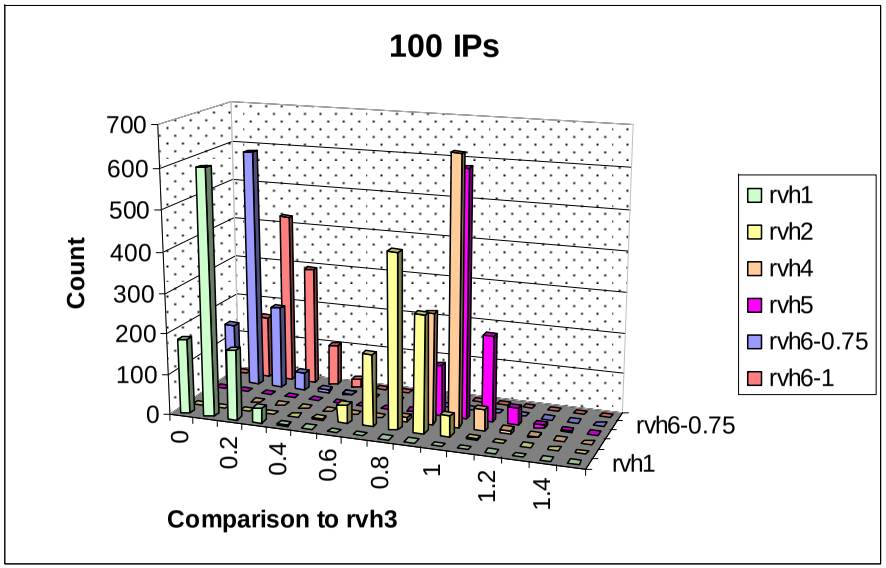
Alpha: Low

Beta: High

Expected Service Times:

[0.1, 3, 7]





Run 3

Value Function:

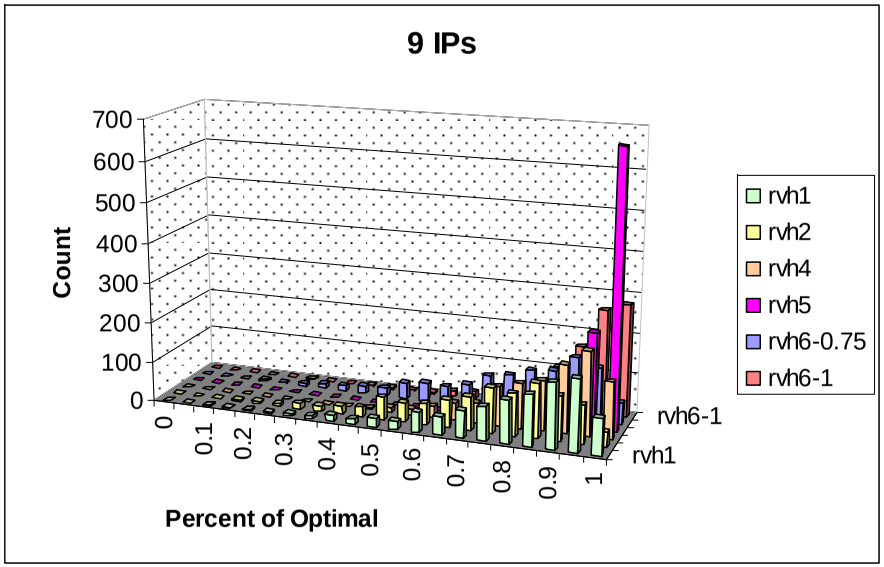
Exponential

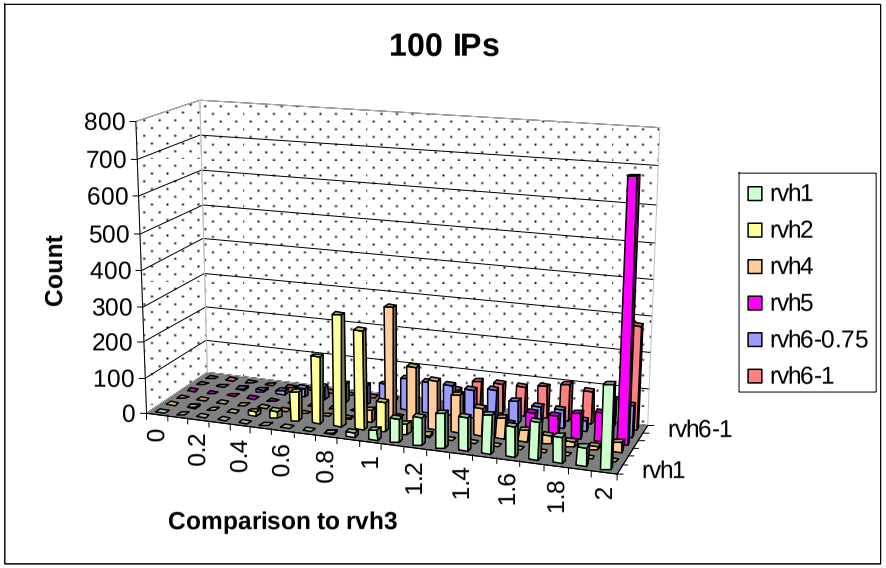
Alpha: Low

Beta: Low

Expected Service Times:

[2, 3, 4]





Run 4

Value Function:

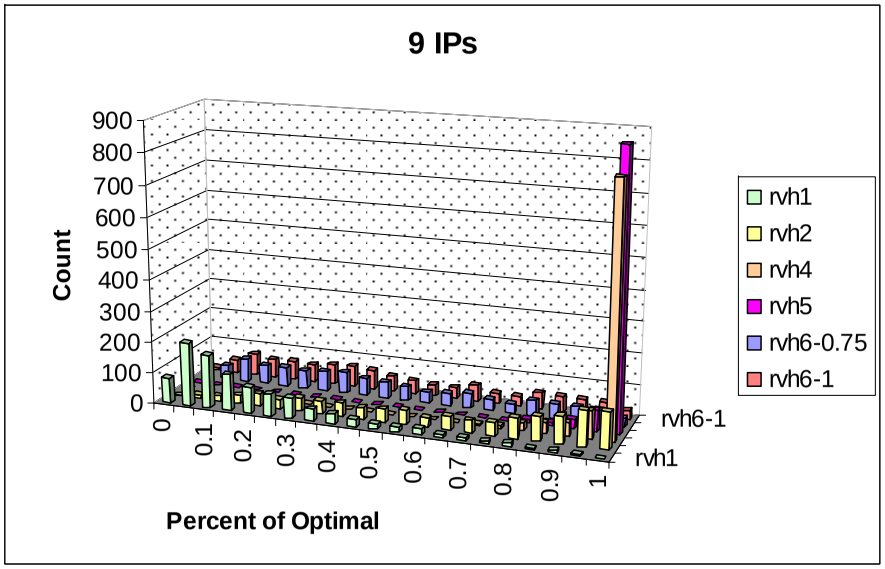
Exponential

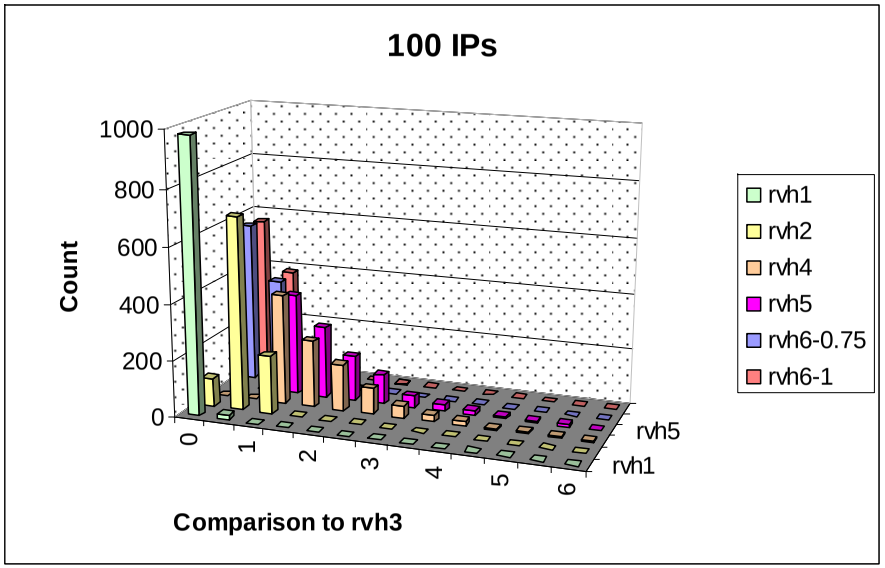
Alpha: Low

Beta: Low

Expected Service Times:

[0.1, 3, 7]





Run 5

Value Function:

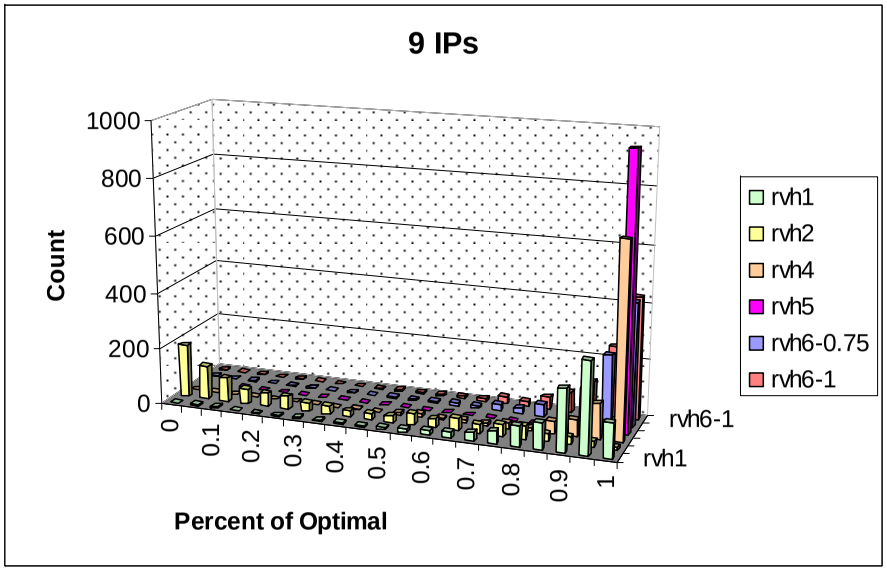
Exponential

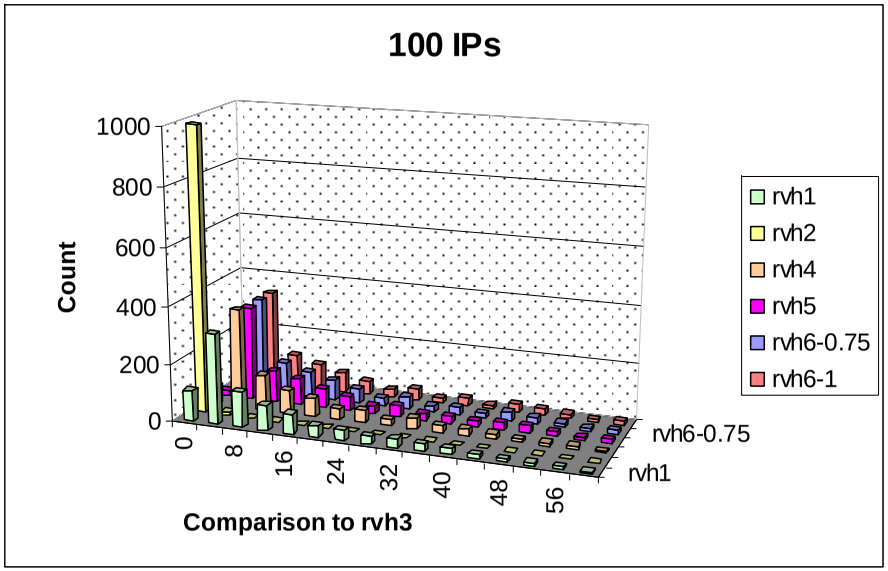
Alpha: High

Beta: High

Expected Service Times:

[2, 3, 4]





Run 6

Value Function:

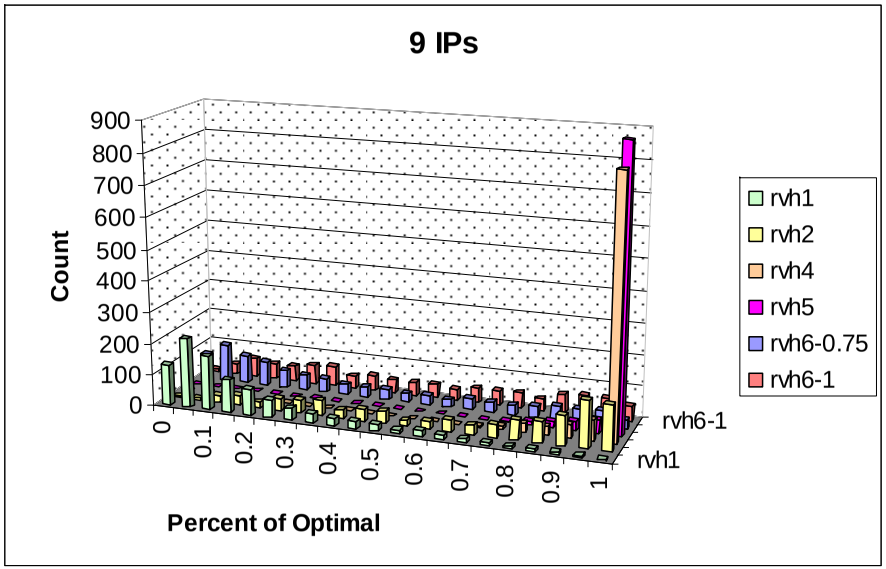
Exponential

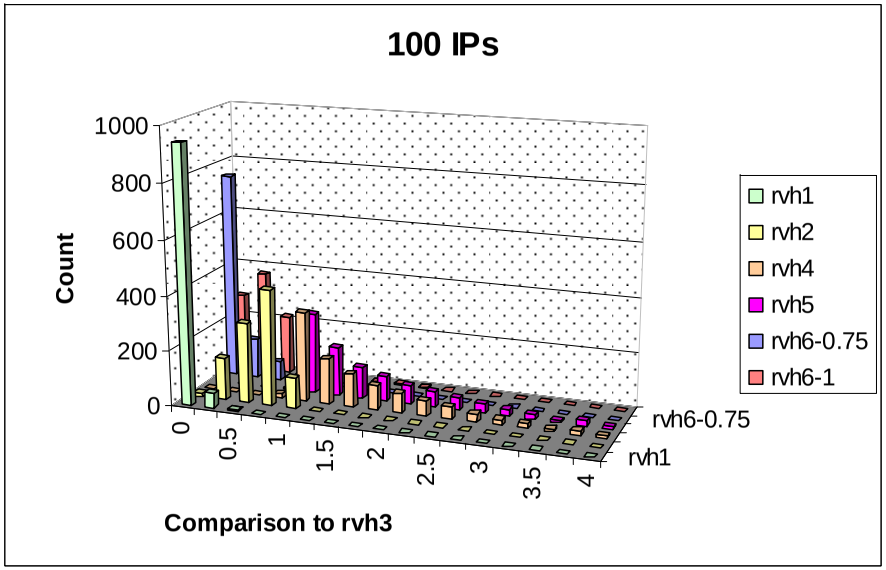
Alpha: High

Beta: High

Expected Service Times:

[0.1, 3, 7]





Run 7

Value Function:

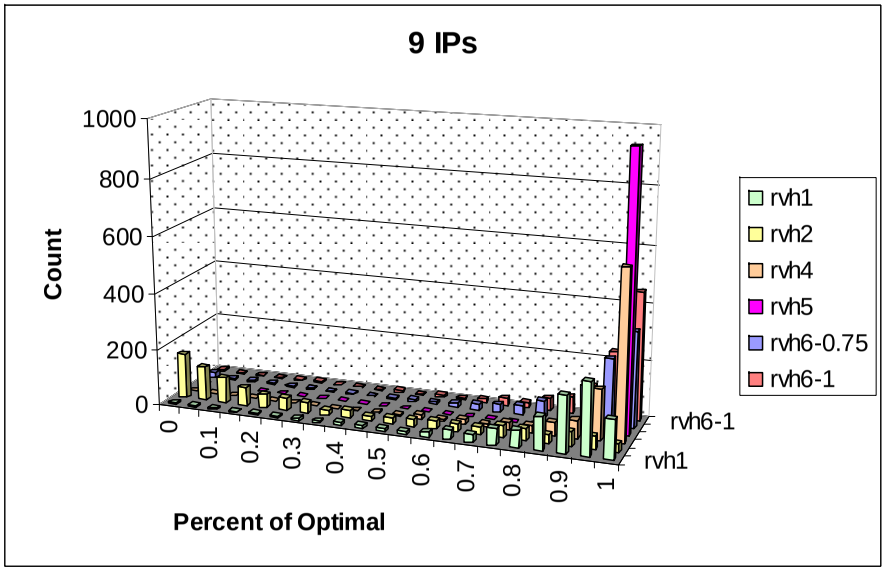
Exponential

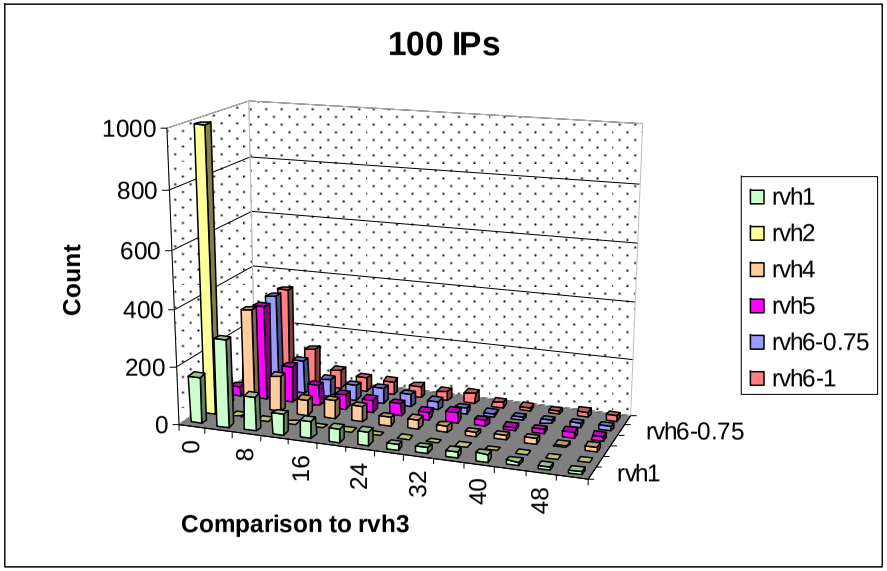
Alpha: High

Beta: Low

Expected Service Times:

[2, 3, 4]





Run 8

Value Function:

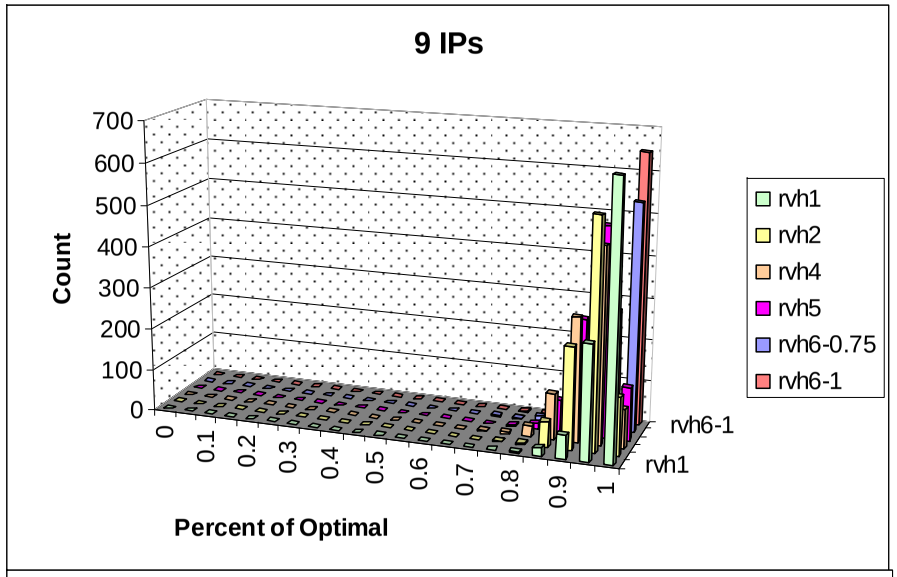
Exponential

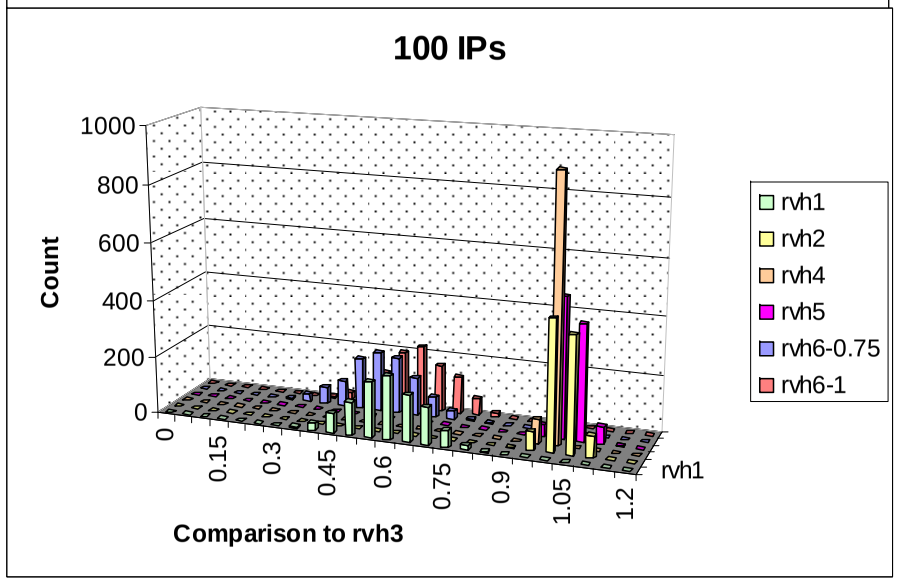
Alpha: High

Beta: Low

Expected Service Times:

[0.1, 3, 7]





Run 9

Value Function:

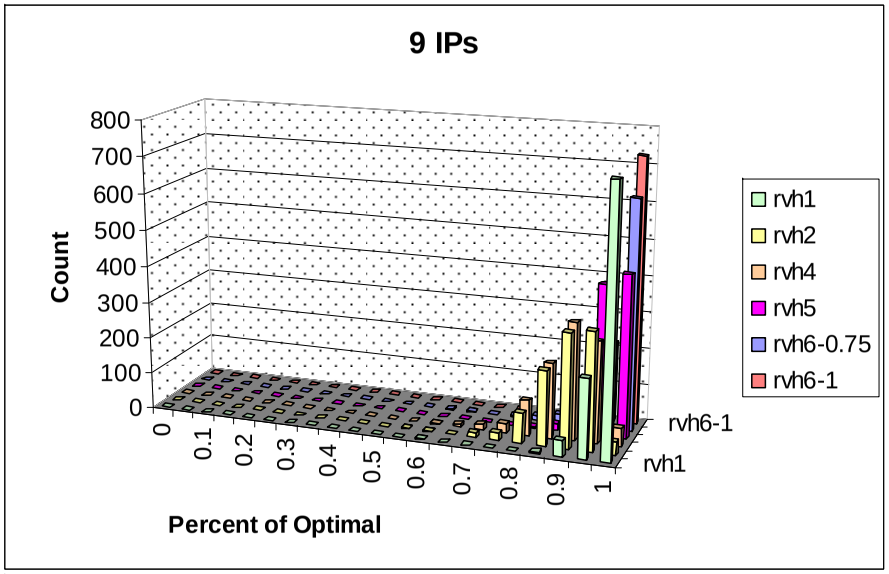
Linear

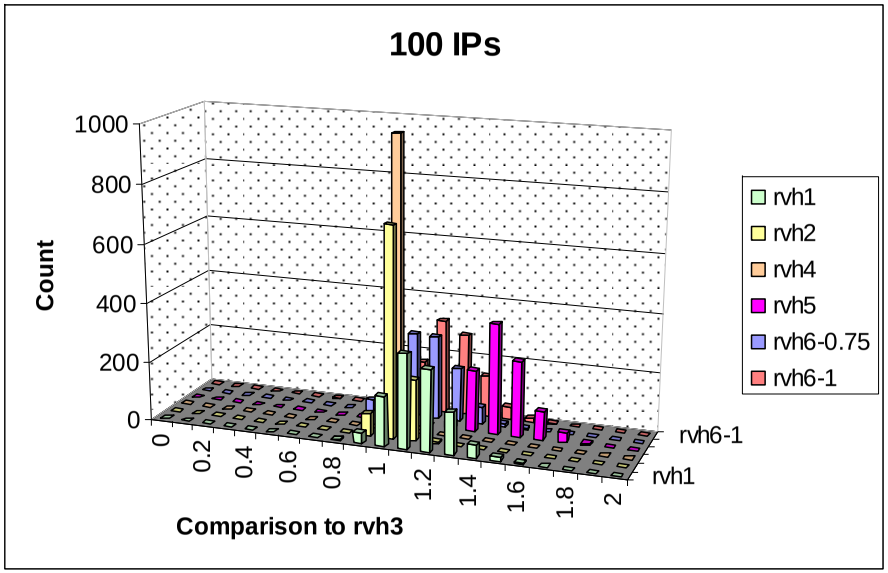
Alpha: Low

Beta: High

Expected Service Times:

[2, 3, 4]





Run 10

Value Function:

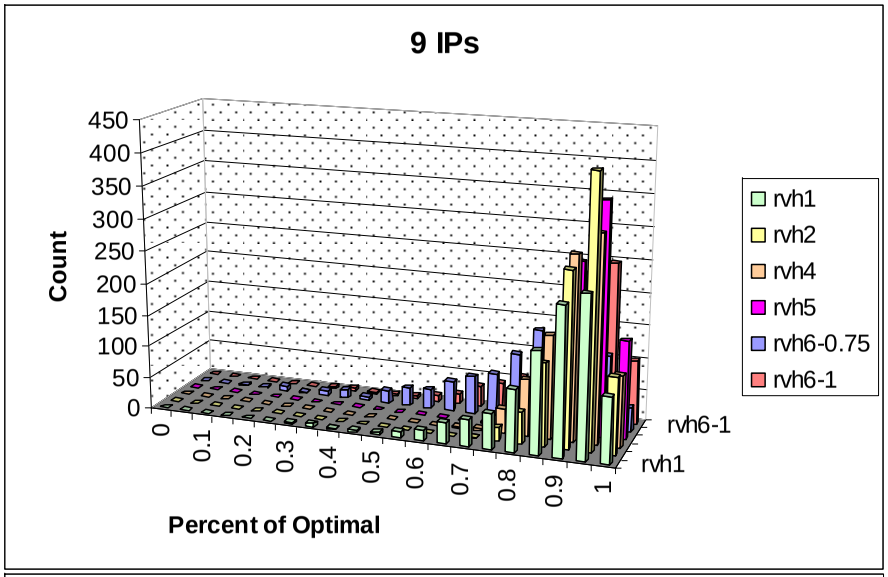
Linear

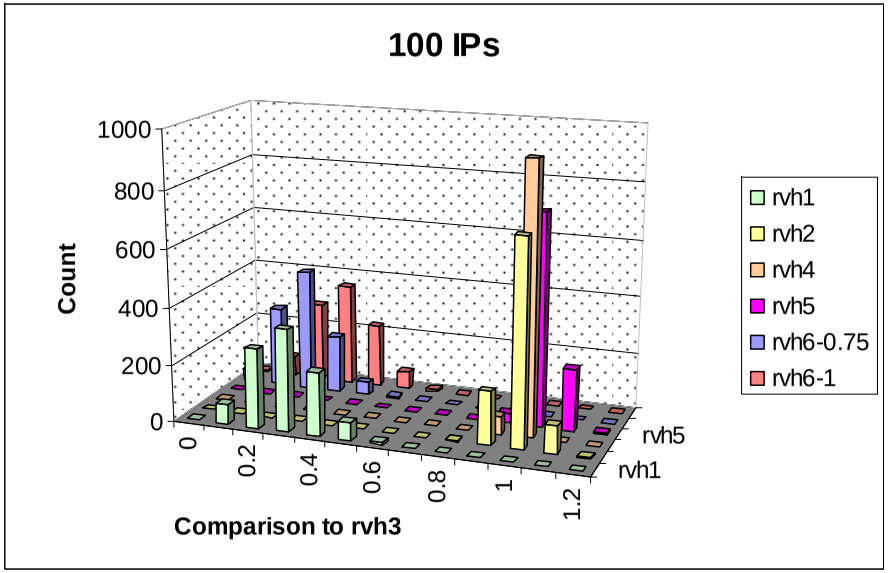
Alpha: Low

Beta: High

Expected Service Times:

[0.1, 3, 7]





Run 11

Value Function:

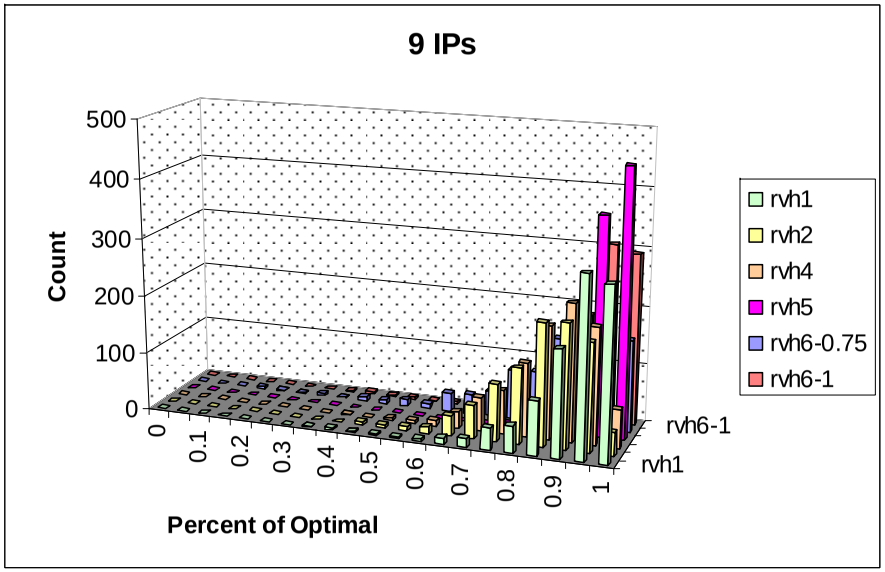
Linear

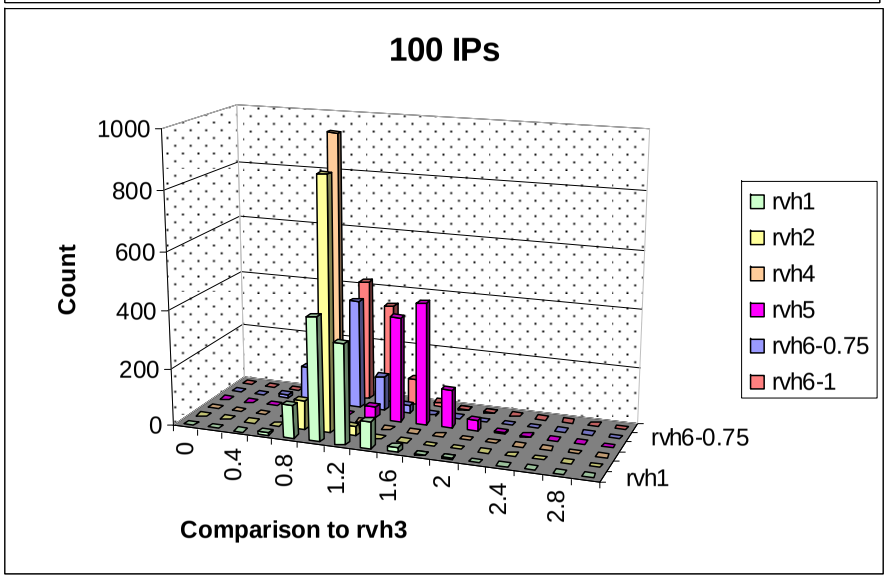
Alpha: Low

Beta: Low

Expected Service Times:

[2, 3, 4]





Run 12

Value Function:

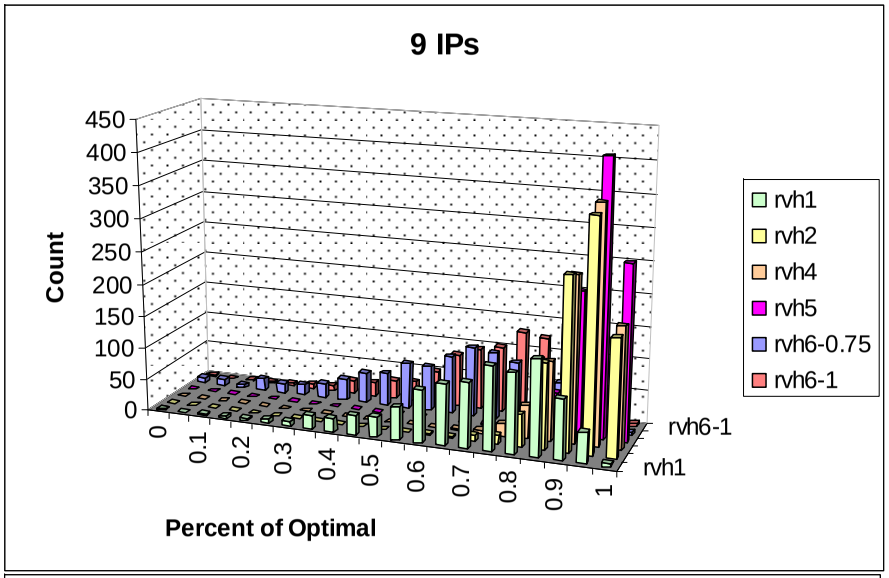
Linear

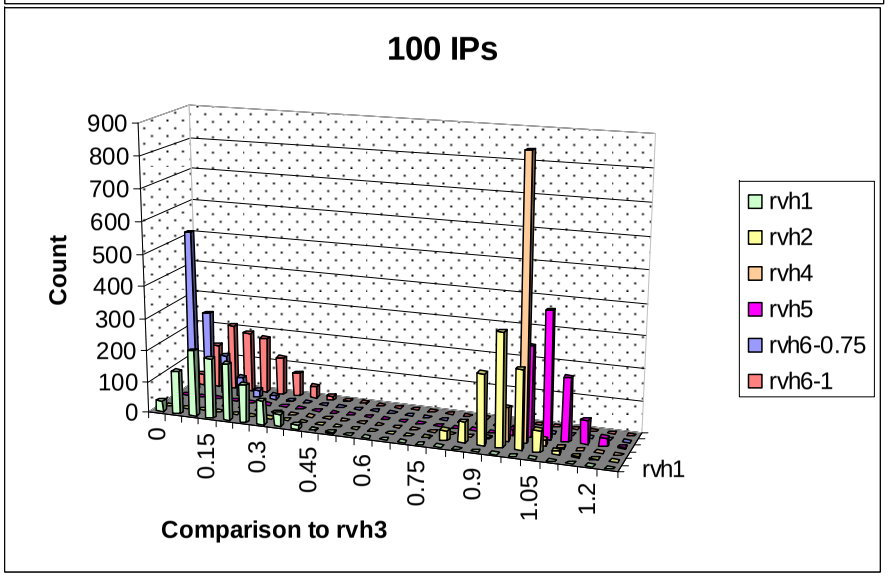
Alpha: Low

Beta: Low

Expected Service Times:

[0.1, 3, 7]





Run 13

Value Function:

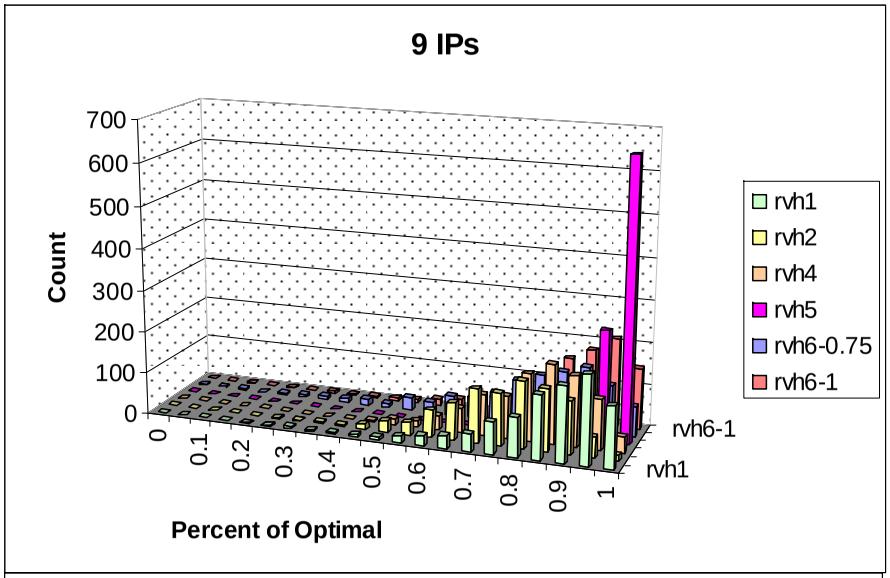
Linear

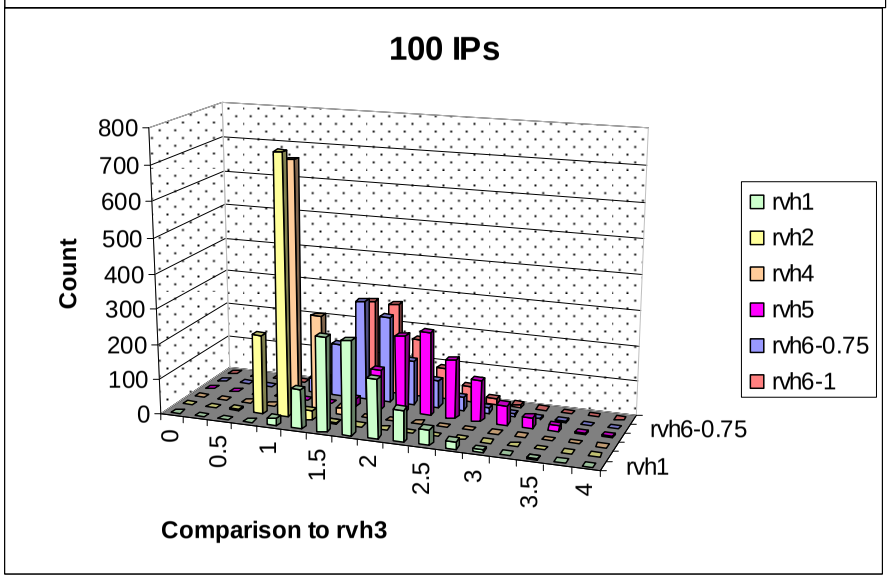
Alpha: High

Beta: High

Expected Service Times:

[2, 3, 4]





Run 14

Value Function:

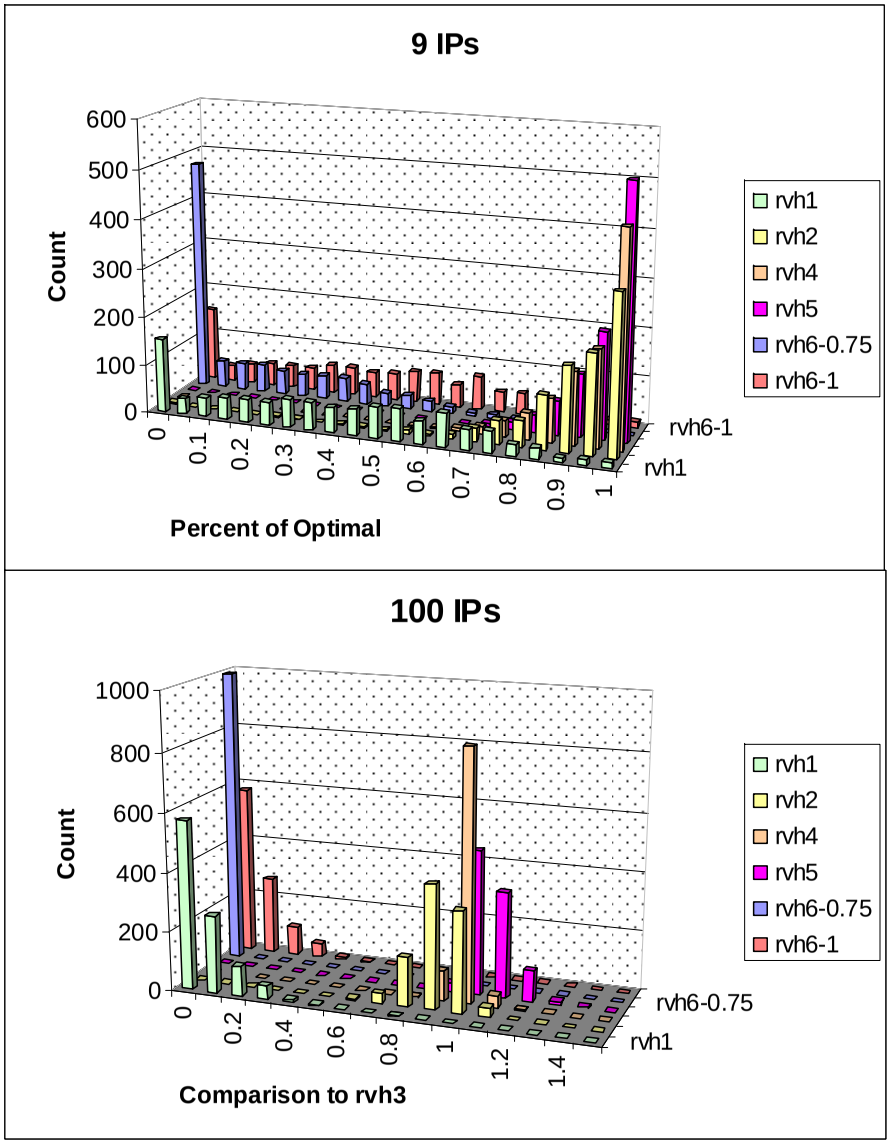
Linear

Alpha: High

Beta: High

Expected Service Times:

[0.1, 3, 7]



Run 15

Value Function:

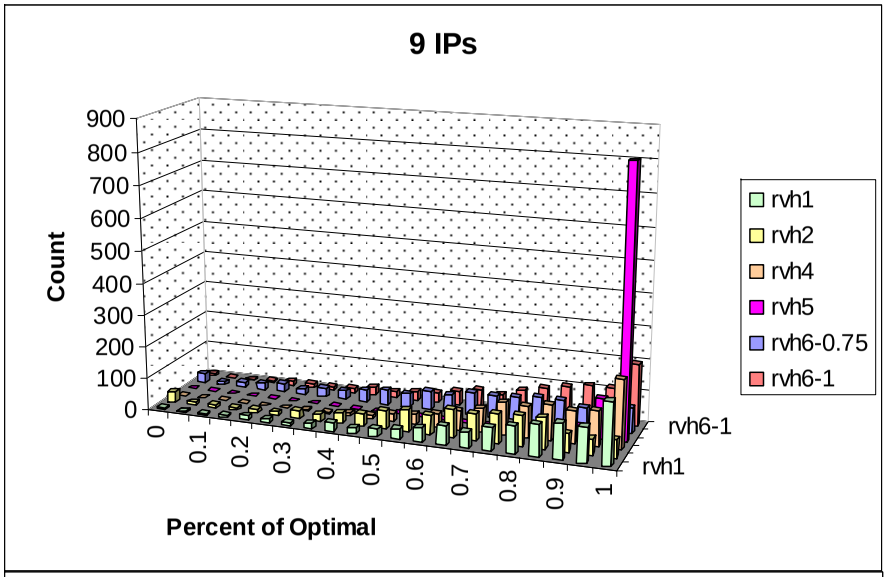
Linear

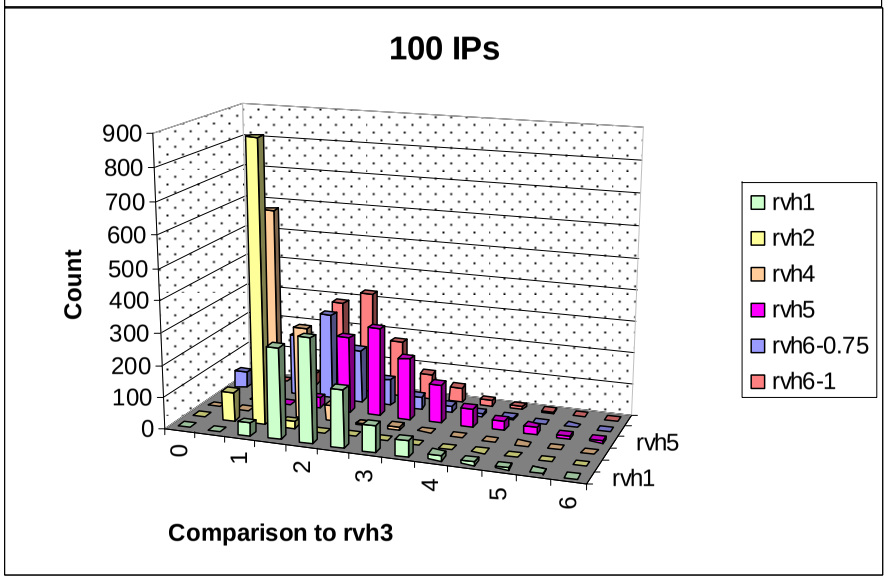
Alpha: High

Beta: Low

Expected Service Times:

[2, 3, 4]





Run 16

Value Function:

Linear

Alpha: High

Beta: Low

Expected Service Times:

[0.1, 3, 7]

E. DYNAMIC SIMULATION RUN MATRIX

Run	Reps	# IPs	Num-Class	Num-Svrs	ValFctMix (E/L/S)	Alpha (lin,exp)	Beta (lin,exp)	Beta (step)	tau (step)	Lambda	Mu
1	250	1000	3	3	33/33/33	L	L	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
2	250	1000	3	3	33/33/33	L	L	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
3	250	1000	3	3	33/33/33	L	L	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
4	250	1000	3	3	33/33/33	L	L	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
5	250	1000	3	3	33/33/33	L	L	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
6	250	1000	3	3	33/33/33	L	L	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
7	250	1000	3	3	33/33/33	L	L	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
8	250	1000	3	3	33/33/33	L	L	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
9	250	1000	3	3	33/33/33	L	H	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
10	250	1000	3	3	33/33/33	L	H	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
11	250	1000	3	3	33/33/33	L	H	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
12	250	1000	3	3	33/33/33	L	H	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
13	250	1000	3	3	33/33/33	L	H	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
14	250	1000	3	3	33/33/33	L	H	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
15	250	1000	3	3	33/33/33	L	H	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
16	250	1000	3	3	33/33/33	L	H	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
17	250	1000	3	3	33/33/33	H	L	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
18	250	1000	3	3	33/33/33	H	L	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
19	250	1000	3	3	33/33/33	H	L	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
20	250	1000	3	3	33/33/33	H	L	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
21	250	1000	3	3	33/33/33	H	L	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
22	250	1000	3	3	33/33/33	H	L	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
23	250	1000	3	3	33/33/33	H	L	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
24	250	1000	3	3	33/33/33	H	L	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
25	250	1000	3	3	33/33/33	H	H	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
26	250	1000	3	3	33/33/33	H	H	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
27	250	1000	3	3	33/33/33	H	H	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
28	250	1000	3	3	33/33/33	H	H	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
29	250	1000	3	3	33/33/33	H	H	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
30	250	1000	3	3	33/33/33	H	H	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
31	250	1000	3	3	33/33/33	H	H	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
32	250	1000	3	3	33/33/33	H	H	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
33	250	1000	3	3	75/15/10	L	L	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
34	250	1000	3	3	75/15/10	L	L	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
35	250	1000	3	3	75/15/10	L	L	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
36	250	1000	3	3	75/15/10	L	L	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
37	250	1000	3	3	75/15/10	L	L	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
38	250	1000	3	3	75/15/10	L	L	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
39	250	1000	3	3	75/15/10	L	L	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
40	250	1000	3	3	75/15/10	L	L	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
41	250	1000	3	3	75/15/10	L	H	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
42	250	1000	3	3	75/15/10	L	H	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
43	250	1000	3	3	75/15/10	L	H	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
44	250	1000	3	3	75/15/10	L	H	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
45	250	1000	3	3	75/15/10	L	H	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
46	250	1000	3	3	75/15/10	L	H	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
47	250	1000	3	3	75/15/10	L	H	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
48	250	1000	3	3	75/15/10	L	H	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
49	250	1000	3	3	75/15/10	H	L	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
50	250	1000	3	3	75/15/10	H	L	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]

Run	Reps	# IPs	Num-Class	Num-Svrs	ValFctMix (E/L/S)	Alpha (lin,exp)	Beta (lin,exp)	Beta (step)	tau (step)	Lambda	Mu
51	250	1000	3	3	75/15/10	H	L	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
52	250	1000	3	3	75/15/10	H	L	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
53	250	1000	3	3	75/15/10	H	L	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
54	250	1000	3	3	75/15/10	H	L	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
55	250	1000	3	3	75/15/10	H	L	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
56	250	1000	3	3	75/15/10	H	L	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
57	250	1000	3	3	75/15/10	H	H	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
58	250	1000	3	3	75/15/10	H	H	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
59	250	1000	3	3	75/15/10	H	H	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
60	250	1000	3	3	75/15/10	H	H	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
61	250	1000	3	3	75/15/10	H	H	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
62	250	1000	3	3	75/15/10	H	H	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
63	250	1000	3	3	75/15/10	H	H	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
64	250	1000	3	3	75/15/10	H	H	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
65	250	1000	3	5	33/33/33	L	L	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
66	250	1000	3	5	33/33/33	L	L	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
67	250	1000	3	5	33/33/33	L	L	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
68	250	1000	3	5	33/33/33	L	L	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
69	250	1000	3	5	33/33/33	L	L	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
70	250	1000	3	5	33/33/33	L	L	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
71	250	1000	3	5	33/33/33	L	L	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
72	250	1000	3	5	33/33/33	L	L	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
73	250	1000	3	5	33/33/33	L	H	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
74	250	1000	3	5	33/33/33	L	H	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
75	250	1000	3	5	33/33/33	L	H	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
76	250	1000	3	5	33/33/33	L	H	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
77	250	1000	3	5	33/33/33	L	H	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
78	250	1000	3	5	33/33/33	L	H	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
79	250	1000	3	5	33/33/33	L	H	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
80	250	1000	3	5	33/33/33	L	H	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
81	250	1000	3	5	33/33/33	H	L	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
82	250	1000	3	5	33/33/33	H	L	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
83	250	1000	3	5	33/33/33	H	L	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
84	250	1000	3	5	33/33/33	H	L	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
85	250	1000	3	5	33/33/33	H	L	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
86	250	1000	3	5	33/33/33	H	L	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
87	250	1000	3	5	33/33/33	H	L	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
88	250	1000	3	5	33/33/33	H	L	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
89	250	1000	3	5	33/33/33	H	H	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
90	250	1000	3	5	33/33/33	H	H	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
91	250	1000	3	5	33/33/33	H	H	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
92	250	1000	3	5	33/33/33	H	H	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
93	250	1000	3	5	33/33/33	H	H	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
94	250	1000	3	5	33/33/33	H	H	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
95	250	1000	3	5	33/33/33	H	H	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
96	250	1000	3	5	33/33/33	H	H	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
97	250	1000	3	5	75/15/10	L	L	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
98	250	1000	3	5	75/15/10	L	L	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
99	250	1000	3	5	75/15/10	L	L	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
100	250	1000	3	5	75/15/10	L	L	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]

Run	Reps	# IPs	Num-Class	Num-Svrs	ValFctMix (E/L/S)	Alpha (lin,exp)	Beta (lin,exp)	Beta (step)	tau (step)	Lambda	Mu
101	250	1000	3	5	75/15/10	L	L	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
102	250	1000	3	5	75/15/10	L	L	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
103	250	1000	3	5	75/15/10	L	L	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
104	250	1000	3	5	75/15/10	L	L	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
105	250	1000	3	5	75/15/10	L	H	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
106	250	1000	3	5	75/15/10	L	H	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
107	250	1000	3	5	75/15/10	L	H	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
108	250	1000	3	5	75/15/10	L	H	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
109	250	1000	3	5	75/15/10	L	H	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
110	250	1000	3	5	75/15/10	L	H	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
111	250	1000	3	5	75/15/10	L	H	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
112	250	1000	3	5	75/15/10	L	H	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
113	250	1000	3	5	75/15/10	H	L	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
114	250	1000	3	5	75/15/10	H	L	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
115	250	1000	3	5	75/15/10	H	L	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
116	250	1000	3	5	75/15/10	H	L	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
117	250	1000	3	5	75/15/10	H	L	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
118	250	1000	3	5	75/15/10	H	L	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
119	250	1000	3	5	75/15/10	H	L	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
120	250	1000	3	5	75/15/10	H	L	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
121	250	1000	3	5	75/15/10	H	H	L	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
122	250	1000	3	5	75/15/10	H	H	L	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
123	250	1000	3	5	75/15/10	H	H	L	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
124	250	1000	3	5	75/15/10	H	H	L	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
125	250	1000	3	5	75/15/10	H	H	H	L	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
126	250	1000	3	5	75/15/10	H	H	H	L	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]
127	250	1000	3	5	75/15/10	H	H	H	H	[0.631, 0.5, 0.8]	[0.134, 0.1, 1]
128	250	1000	3	5	75/15/10	H	H	H	H	[1.2, 0.2, 0.6]	[0.134, 0.1, 1]

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. DISCRETE STOCHASTIC INFORMATION FLOW MODEL MATLAB CODE

```
%% This m-file simulates the system x(k+1) = Ax(k) + Bu(k), where u(k)
is
%% a vector of random variables depicting 1) the number of info pkgs
%% arriving in Oval 2 during time period k, and 2) which processing
time
%% "bin" these pkgs are assigned to.

%% Version 1.0, 24 Jan 2007
%% Donovan Phillips

clear all

% First, define some parameters:

n = 1000;  % number of runs in the simulation
p = 51;  % number of data pkgs to generate each run - same number as
orig. data
di = 2;  % delta_i:  size (in min) of each processing time "bin"
dt = 2;  % time step size (min)
mu_arrtime = 0.843;  % average pkg inter-arrival time
mu_svctime = 7.03529;  % average pkg processing time
max_endtime = 300; % simulation end time, min
alpha_svc = 0.410298;
beta_svc = 17.1468; % parameters for beta fit of svc time data
alpha_arr = 0.7351;
beta_arr = 1.614;  % parameters for beta fit of arr time data

%% initialize the output matrix:
vol = zeros(n,max_endtime/dt);
t = dt:dt:max_endtime;

% Now, perform the simulation:

for i=1:n
    rand('state',sum(100*clock)); %randomly sets the random number
generator state (or "seed")
    %inter_arr_times = exprnd(mu_arrtime,p,1);
    inter_arr_times = gamrnd(alpha_arr, beta_arr, p,1);
    rand('state',sum(100*clock));
    %svc_times = exprnd(mu_svctime,p,1);
    svc_times = gamrnd(alpha_svc, beta_svc, p, 1);
    arr_times = zeros(p,1);
    arr_times(1) = inter_arr_times(1);
    for j=1:(p-1)
        arr_times(j+1) = arr_times(j) + inter_arr_times(j+1);
    end

    P=zeros(p,3);
```

```
    P(:,1)=arr_times;
    P(:,3)=svc_times;
    P(:,2)=arr_times + svc_times;

    d = CoefMatrixMod(di,dt,P); % rows of d are processing time bins;
cols are time steps
    s = size(d);
    s1 = s(1); % number of rows in d
    s2 = s(2); % number of cols in d; note:  size of d will vary with
each run.
    bins = (di:di:(s1*2))';

    for k=1:s2
        vol(i,k) = sum(bins.*d(:,k)); %computes the "volume" of info in
oval 2 at time period k
    end
end

sumvol = sum(vol);

close all

plot(t,vol)

hold on

ad = importdata('processtimeNoIntelDump22.csv'); %actual data, for
comparison
matrix = CoefMatrixMod(di,dt,ad);
bins = (di:di:(2*size(matrix,1)))';

vol_actual = zeros(1,max_endtime/dt);

for i=1:(size(matrix,2))
    vol_actual(i)=sum(bins.*matrix(:,i));
end

plot(t,vol_actual,'kx:')
xlabel('Time (min)');
ylabel('Total Volume in Oval 2 (min)');
xlim([0 1.05*dt*max(max(find(sumvol)), max(find(vol_actual)))]);

hold off
figure
hold on

avg_vol = mean(vol);
stdev_vol = std(vol);
ci = 1.0*stdev_vol;
upper = avg_vol + ci;
lower = max(avg_vol - ci,0);
plot(t,avg_vol)
plot(t,upper,'b--')
```

```
plot(t,lower,'b--')
plot(t,vol_actual,'kx:')
xlabel('Time (min)');
ylabel('Total Volume in Oval 2 (min)');
xlim([0 1.05*dt*max(max(find(sumvol)), max(find(vol_actual)))]);
hold off
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Department of the Army, "Concepts for the Objective Force," U.S. Army White Paper, undated.

[2] Department of the Army, "Future Combat System (Brigade Combat Team) (FCS (BCT)) 18 + 1 + 1 Systems Overview," Program Manager FCS (BCT), April 2006.

[3] J. Becker, "A buzz in the air," C4ISR Journal, vol. 2, pp. 22-24, March-April 2003.

[4] B. Koerner, "Intel's tiny hope for the future," Wired, vol. 11, pp. 236, Dec 2003.

[5] D.L. Hall, "An introduction to multisensor data fusion," Proceedings of the IEEE, vol. 85, pp. 6-23, 1997.

[6] N.L. Miller and L.G. Shattuck, "A process model of situated cognition in military command and control," Proceedings of the Command and Control Research and Technology Symposium, June 2004.

[7] L.G. Shattuck and N. Miller, "Extending Naturalistic Decision Making to Complex Organizations: A Dynamic Model of Situated Cognition," Organization Studies, vol. 27, pp. 989, 2006.

[8] N.L. Miller and L.G. Shattuck, "A Dynamic Process Model for the Design and Assessment of Network Centric Systems," Proceedings of the Command and Control Research and Technology Symposium, June 2006.

[9] Department of the Army, Field Manual 2-0, Intelligenc*e*, 2004.

[10] D.K. Ahner, Planning and Control of Unmanned Aerial Vehicles in a Stochastic Environment, Doctor of Philosophy dissertation, Boston University, 2005.

[11] G. Klein, "A recognition-primed decision (RPD) model of rapid decision making," in Decision Making in Action: Models and Methods, G. Klein, J. Orasanu, R. Calderwood and C. Zsambok, editors, Norweed, NJ: Ablex, 1993, pp. 138-147.

[12] M.R. Endsley, "Situation awareness in dynamic human decision making: theory," in Proceedings of the Center for Applied Human Factors in Aviation (CAHFA) Conference on Situational Awareness in Complex Systems, pp. 27-58, 1993.

[13] M.R. Endsley, "Situation awareness i ndynamic human decision making: measurement," in Proceedings of the Center for Applied Human Factors in Aviation (CAHFA) Conference on Situational Awareness in Complex Systems, pp. 79-97, 1993.

[14] T.C. Koopmans and M. Beckman, "Assignment Problems and the Location of Economic Activities," Econometrica, vol. 25, pp. 53-76, Jan. 1957.

[15] S. Sahni and T. Gonzalez, "P-Complete Approximation Problems," J.ACM, vol. 23, pp. 555-565, 1976.

[16] M.R. Garey and D.S. Johnson, Computers and intractability : a guide to the theory of NP-completeness, San Francisco: W. H. Freeman, 1979.

[17] R.E. Burkard, E. Çela, P.M. Pardalos and L.S. Pitsoulis, "The Quadratic Assignment Problem," in Handbook of Combinatorial Optimization, pp. 241-338, 1998.

[18] R.E. Burkard, "Selected topics on assignment problems," Discrete Applied Mathematics, vol. 123, pp. 257-302, 2002.

[19] R.E. Burkard, "The quadratic assignment problem with an Anti-Monge and a Toeplitz matrix: easy and hard cases," Mathematical Programming, vol. 82, pp. 125, 1998.

[20] R.E. Burkard, E. Çela, V.M. Demidenko, N.N. Metelski and G.J. Woeginger, "A unified approach to simple special cases of external permutation," Optimization, vol. 44, pp. 123-138, 1998.

[21] E. Çela, The quadratic assignment problem: Special cases and relatives, Austria: Karl-Franzens-Universitaet Graz (Austria), 1996.

[22] E. Çela, The quadratic assignment problem: theory and algorithms, Boston: Kluwer Academic Publishers, 1998.

[23] V. Demidenko, "Well solvable cases of the quadratic assignment problem with monotone and bimonotone matrices," Journal of Mathematical Modeling and Algorithms, vol. 5, pp. 167-187, 2006.

[24] K.R. Baker, Introduction to sequencing and scheduling, New York: Wiley, 1974.

[25] S.C. Graves, "A Review of Production Scheduling," Operations Research, vol. 29, pp. 646-675, 1981.

[26] C. Derman, G. Lieberman and S. Ross, "A sequential stochastic assignment problem," Management Science, vol. 18, pp. 349-355, 1972.

[27] S.C. Albright, "Optimal sequential assignment with random arrival times," Management Science, vol. 21, pp. 60-67, 1974.

[28] A.K. Agrawala, E.G. Coffman, M.R. Garey and S.K. Tripathi, "A stochastic optimal algorithm minimizing exponential flow times on uniform processors," IEEE Transactions on Computers, vol. 33, pp. 351-356, 1984.

[29] E.G. Coffman, Jr., L. Flatto, M.R. Garey and R.R. Weber, "Minimizing expected makespans on uniform processor systems," Advances in Applied Probability, vol. 19, pp. 177-201, 1987.

[30] R. Righter, "Multiprocessor Scheduling and the Sequential Assignment Problem," in Strategies for sequential search and selection in real time: proceedings of the AMS-IMS-SIAM joint summer research conference, pp. 105-115, 1992.

[31] S.M. Ross, Introduction to stochastic dynamic programming, New York: Academic Press, 1983.

[32] T.G. Voutsinas and C.P. Pappis, "Scheduling jobs with values exponentially deteriorating over time," International Journal of Production Economics, vol. 79, pp. 163-169, 2002.

[33] L. Shattuck and N. Miller, "A Process Tracing Approach to the Investigation of Situated Cognition," in Proceedings of the Human Factors and Ergonomics Society, vol. 48, pp. 658, 2004.

[34] D. Phillips, W. Kang, D. Ahner and B. Mansager, "A dynamical system model of information flow on the battlefield," in Proceedings of IEEE International Conference on Systems, Man and Cybernetics, pp. 685-690, 2007.

[35] S.M. Ross, Introduction to probability models, San Diego, CA: Academic Press, 1997.

[36] M. Dell'Amico and P. Toth, "Algorithms and codes for dense assignment problems: the state of the art," Discrete Applied Mathematics, vol. 100, pp. 17-48, 2000.

[37] R. Burkard, B. Klinz and R. Rudolf, "Perspectives of Monge properties in optimization," Discrete Applied Mathematics, vol. 70, pp. 95-161, 1996.

[38] A.J. Hoffman, "On simple linear programming problems," Convexity, Proceedings of Symposia in Pure Mathematics, vol. 7, pp. 317-327, 1963.

[39] United States Joint Chiefs of Staff, Joint Publication 3-09, Joint Fire Support, 2006.

[40] L. Shattuck and N. Miller, personal conversation, March 2008.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Wei Kang
   Naval Postgraduate School
   Monterey, California

4. Kyle Lin
   Naval Postgraduate School
   Monterey, California

5. Hong Zhou
   Naval Postgraduate School
   Monterey, California

6. Carlos Borges
   Naval Postgraduate School
   Monterey, California

7. Darryl Ahner
   Army Training and Doctrine Command Analysis Center
   Monterey, California

8. Chris Arney
   Army Research Office
   Durham, North Carolina