GENERALIZED SEQUENTIAL ASSIGNMENT PROBLEM

BY

ARASH KHATIBI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Industrial Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

ABSTRACT

The Sequential Stochastic Assignment Problem (SSAP) deals with assigning sequentially arriving tasks with stochastic parameters to workers with fixed success rates. The reward of each assignment is the product of the worker's success rate and the task value assigned to the worker. The objective is to maximize the total expected reward. There has been a surge of interest in studying sequential assignment problems due to their applications in online matching markets, asset selling, and organ transplant.

This dissertation studies several variations of SSAP by relaxing the main assumptions. The first part assumes that the workers' success rates are random values coming from a known distribution. This generalization modifies the SSAP from a problem with a single random value (i.e., the task value) at each stage to an online matching problem with several random parameters (i.e., the task value and the workers' success rates). The optimal assignment policy uses backward induction to first solve smaller subproblems, and then use them to optimally assign tasks to workers from the first stage. An approximation algorithm is proposed that achieves a fraction of the optimal reward in a polynomial time.

Assuming that the value of sequentially arriving elements are independently drawn from a known distribution is unrealistic in many applications. The second part of thesis relaxes this assumption and uses the well-known Secretary Problem to derive constant-competitive algorithms for SSAP with tasks having a random arrival order. Several deterministic and randomized algorithms are proposed and their performance are compared with the maximum offline reward. These algorithms use the first stages of the problem as a training phase to compute thresholds for the task values. These thresholds are used to assign tasks to workers after the training phase.

The third part uses the linear programming technique to derive bounds on the performance of optimal policy for several variations of SSAP. Formulating an online matching problem as a linear program is a useful tool. In addition to deriving bounds on performance of optimal policies, the linear programming technique can be used to formulate extensions of the problem as linear programs by simple changes in the objective function and constraints of the basic formulation. The linear programming formulation of the incentive compatible problem and the sequential assignment problem with unknown number of elements are also proposed. The edge-weighted online bipartite matching problem is used to design assignment policies for each of the formulated problems.

The last part relaxes the assumption that at most one task must be assigned to each worker in SSAP. It is assumed that a worker is available for possible future assignments after performing the previously assigned task. The number of stages that the worker is not available due to a prior task assignment is referred to as the task duration. This problem is studied under various models for the task duration. First, it is assumed that the task duration is fixed. Then, assignment policies are proposed for the problem with a memoryless model for the task duration. The proposed algorithms are extensions of the optimal algorithm for the sequential assignment problem. They divide the n-stage assignment process to periods whose lengths are equal to the expected task duration. Then, they assign tasks to workers in each period by applying the optimal algorithm of the sequential assignment problem.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would like to thank my adviser, Professor Sheldon H. Jacobson for his continual guidance and support. In addition to involving me in interesting research topics, his help in the time of need and respect for his group members is invaluable.

I am grateful for the time and insightful comments of my PhD final examination committee members, Professor Xin Chen, Professor David A. Forsyth, and Professor Richard Sowers.

I am thankful to the faculty and staff of the Department of Industrial and Enterprise Systems Engineering and the Department of Computer Science at University of Illinois. I am very thankful to Professor Ramavarapu S. Sreenivas, Holly Kizer, Mary Beth A. Kelley, Elaine Wilson, and Viveka Perera.

I am grateful to the several sources who provided funding throughout my PhD studies. The research presented in this thesis has been supported in part by the Air Force Office of Scientific Research (FA9550-15-1-0100). Any opinion, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views, official policy or position of the Air Force Office of Scientific Research. I would also like to thank the Department of Industrial and Enterprise Systems Engineering and the Department of Computer Science at University of Illinois who provided funding to me as a teaching assistant.

I would like to extend my sincere gratitude to my labmates, Ge Yu, Hee Youn Kwon, Wenda Zhang, Rahul Swamy, Shouvik Dutta, Douglas M. King, Estelle Kone, and David Morrison, for their help during the past six years. Special thanks to Golshid Baharian and Banafsheh Behzad who helped me in the first days of joining Professor Jacobson's group and to my dear friend, Jason Sauppe, for his friendship and company during these years.

I would also like to thank my old friends, Omid Zobeiri, Houshmand Shirani-mehr, Bahram Ehsandoust, Mahdi Mirhosseini, Alborz Rezazadeh, Amirreza Safaripour, and my brother, Amirreza Khatibi. Special thanks to Aliakbar Daemi for being both a friend and a role model. Lastly, thanks to my wife, Maryam Kazerooni, for her unconditional support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ABBREVIATIONS

CDF	Cumulative Distribution Function
DSSAP	Doubly Stochastic Sequential Assignment Problem
GSSAP	Generalized Sequential Stochastic Assignment Problem
IID	Independent and Identically Distributed
LP	Linear Programming
PDF	Probability Density Function
SSAP	Sequential Stochastic Assignment Problem

CHAPTER 1

INTRODUCTION

The Sequential Stochastic Assignment Problem (SSAP) assigns sequentially arriving tasks to a set of workers so as to maximize the expected value of a reward function [19]: There are n sequentially arriving tasks with random values that must be optimally assigned to n workers with fixed success rates. Associated with the j^{th} arriving task is a random variable X_j and a value x_j, which is revealed upon its arrival. The tasks' associated random variables are assumed to be *independent and identically distributed* (IID). The reward of each assignment is the product of the worker's success rate and the task value assigned to the worker. The objective is to maximize the expected sum of the assignment rewards. The number of tasks and the distribution of the task's random variable are known. The main challenge is that the decision-maker irrevocably assigns each arriving task to one of the workers, without knowing the future task values. This formulation of SSAP is referred to as the SSAP basic formulation throughout this thesis. The optimal policy for the SSAP is threshold-based. If there are n tasks to be assigned, the task distribution is used to compute $n + 1$ interval thresholds. Each task is then assigned to a worker based on the interval that the task value is placed [19].

This chapter describes the optimal policy of the basic formulation of SSAP (Section 1.1). Section 1.2 reviews several applications of SSAP. Chapter 1.3 presents the objective of this dissertation and briefly describes the contributions. The outline of the remaining chapters is presented in Section 1.4.

1.1 The Optimal Policy of SSAP

Let x_j denote the random value associated with the j^{th} arriving task and p_i denote the fixed success rate of the i^{th} worker. The reward of assigning the j^{th} arriving task to the i^{th} worker is

$p_i x_j$, where x_j is the realized value of the j^{th} task. The total expected reward is given by

$$E[\sum_{j=1}^{n} p_{i_j} x_j], \tag{1.1}$$

where i_j denotes the index of the worker assigned to the j^{th} task. The objective is to assign the n tasks to the n workers such that the total expected reward is maximized. Theorem 1 describes the optimal assignment policy for the SSAP.

Theorem 1. *[19] For each $n \geq 1$, there exists numbers*

$$-\infty = a_{0,n} \leq a_{1,n} \leq ... \leq a_{n,n} = +\infty,$$

such that whenever there are n workers (with $p_1 \leq p_2 \leq ... \leq p_n$) to be assigned to n tasks, then the optimal choice in the initial stage is to use p_i if the random variable X_1 is contained in the interval $(a_{i-1,n}, a_{i,n}]$, $i = 1, 2, ..., n$. The $a_{i,n}$ depend on F_X (the CDF of tasks associated random variable), but do not depend on the $\{p_i\}$. Furthermore, $a_{i,n}$ is the expected value, in an $(n-1)$ stage problem, of the quantity assigned to the i^{th} smallest p (assuming an optimal policy is being followed), and the total expected reward is given by

$$\sum_{i=1}^{n} a_{i,n+1} p_i. \tag{1.2}$$

The constants $\{a_{i,n}\}$ are computed recursively, as described in Corollary 1. The interval thresholds depend on the number of tasks and the tasks distribution, but are independent of the workers' success rates.

Corollary 1. *[19] Define $a_{0,n} = -\infty$ and $a_{n,n} = +\infty$. Then*

$$a_{i,n+1} = \int_{a_{i-1,n}}^{a_{i,n}} x \, dF_X(x) + a_{i-1,n} F_X(a_{i-1,n}) + a_{i,n}(1 - F_X(a_{i,n})), \tag{1.3}$$

for i=1,2,...,n, where $-\infty.0$ and $+\infty.0$ are both defined to be 0.

The proof of the optimality is based on induction and uses Hardy's Lemma.

Lemma 1. *(Hardy's Lemma). If $x_1 \leq x_2 \leq ... \leq x_n$ and $y_1 \leq y_2 \leq ... \leq y_n$ are sequences of numbers, then*

$$\max_{(i_1,i_2,...,i_n)\in P} \sum_{j=1}^{n} x_{i_j} y_j = \sum_{j=1}^{n} x_j y_j \tag{1.4}$$

where P is the set of all permutations of the integers $(1, 2, ..., n)$.

Hardy's Lemma indicates that the maximum sum is achieved when the largest of the x's and y's are paired, the second largest are paired, and so on until the smallest of them are paired.

The optimal policy of SSAP is threshold-based. This means that a set of thresholds are computed and tasks are assigned to workers by comparing the task values to thresholds. Since the task values are drawn from a known distribution, the thresholds are computed prior to the assignment process. When there is no prior information on task values, there must be a training phase to compute thresholds, after which the assignment process starts.

1.2 Applications

SSAP is an online matching problem and has applications in several areas such as online matching markets, organ transplant, and house hunting. Online matching problems have been extensively studied in recent years due to their application in online advertising.

In an online matching market, a set of sequentially arriving items must be matched to buyers. Workers in SSAP correspond to buyers (bidders) while the workers' success rates correspond to bids. For example, the Adwords market is a large auction where companies bid for keywords. The objective it to determine what advertisements to display with each query to maximize the profit [45].

The organ transplant problem seeks to match a set of sequentially arriving donors to patients such that a reward function, defined based on the probability of successful transplants, is maximized [59]. Donors in the organ transplant correspond to tasks in the SSAP and patients correspond to workers. The reward function is defined based on the probability of successful transplant and the life expectancy following the transplantation.

In the house hunting problem, a sequence of offers are made for a set of houses [19]. The decision maker must either accept or reject each offer to maximize the total profit. Matching requesters and workers in a crowdsourcing market, such as the Amazon Mechanical Turk, and assigning passengers to screening devices in an airport are other applications of SSAP [39, 44, 43].

1.3 Motivation and Contribution

The basic formulation of SSAP is a simple model for most of the described applications. For example, SSAP assumes that the arriving donors in the organ transplant problem can be modeled by a single value coming from a known distribution. In an online matching market, SSAP assumes that at most one item is sold to each bidder and the number of bidders is known prior to the assignment process.

The objective of this dissertation is to relax the main assumptions in the basic formulation of SSAP. Fixed workers' success rates, task values coming from a known distribution, and the one-time assignment of tasks to any of the workers (i.e., the assumption that at most one task is assigned to each worker) are three main assumptions of SSAP. Relaxing these three assumptions is the main contribution of this dissertation. Moreover, several variations of SSAP are formulated as linear programs and the linear programming technique is used to derive upper bounds on the performance of any assignment policy.

The first part of this thesis relaxes the assumptions that the workers' success rates are fixed. Due to randomness in both the task values and the workers' success rates, this problem is referred to as the Doubly Stochastic Sequential Assignment Problem (DSSAP). DSSAP can be considered as an online matching market where each of the n buyers has a valuation for each of sequentially arriving elements. The objective is to assign each of the online arriving elements to one buyer such that the total profit is maximized. First, several special cases of DSSAP are presented and the optimal policy of each problem is proposed. The optimal policy for DSSAP with one worker (and n tasks) transforms the randomness from workers to tasks and uses the optimal policy of SSAP. Then, the Greedy Algorithm is proven to be optimal for DSSAP with IID success rates.

The optimal assignment policy for the general case with no assumptions on the success rate distributions is derived. The optimal algorithm uses backward induction to find the optimal assignment policy for each subset of workers. Then, the optimal solution of smaller problems is used to determine the optimal assignment from the first stage. Due to computational complexity of the optimal policy, an efficient approximation algorithms is proposed. The approximation algorithm ranks the workers based on their expected success rates and assumes higher assignment priority for workers with larger expected success rates.

The second part of this dissertation relaxes the assumption that task values are independently drawn from a given distribution. No prior information on task values is assumed (which is equivalent to assuming that task values are selected by an adversary). However, it is assumed that tasks have a random arrival order.

Definition 1. *A set of n sequentially arriving values have a random arrival order if the i^{th} element is the j^{th} largest value with probability $\frac{1}{n}$ for $i, j = 1, 2, ..., n$.*

SSAP with no prior information on task values is referred to as the Generalized Sequential Stochastic Assignment Problem (GSSAP). GSSAP is a generalization of the well-known Secretary Problem. The Secretary Problem seeks to hire a secretary among a known number of sequentially arriving candidates in an optimal manner. GSSAP can be considered as a Secretary Problem where all arriving secretaries are hired and assigned to various positions, with the reward of each assignment defined as the product of the quality of the secretary and the weight of the position. The Weighted Secretary Problem is used to design assignment policies for GSSAP. The reward of an algorithm for an online matching problem without any prior information on the values of the online elements depends on the input sequence. Therefore, the proposed algorithms are evaluated by their competitive ratios.

Definition 2. *An algorithm is α-competitive if its expected reward is at least $\frac{1}{\alpha}$ times the optimal offline reward, where the optimal offline reward is the maximum reward achieved when all elements are known in advance.*

Several constant competitive algorithms for GSSAP are proposed. Similar to the optimal policy of SSAP, the proposed algorithms for GSSAP are threshold-based. However, while the interval thresholds of SSAP are computed prior to the assignment process using the task's distribution, the GSSAP algorithms use the first stages of the problem as a training phase to learn task values and compute thresholds. This defines a new question for GSSAP: How to assign the tasks arriving in the training phase? This dissertation suggests several answers to this question. The first idea is to assign the first arriving tasks to workers with smallest success rates. The intuition behind this idea is that you do not lose any of the workers with larger success rates while learning the task values. The second policy deterministically divides the workers to two sets with approximately the same quality (i.e., one of the best two workers is assigned to each group, one of the next two best is assigned to each group, and so on). Then, the tasks in the training phase are assigned to the first group. This provides the possibility of assigning the tasks with large values arriving in the training phase to workers with large success rates selected for the first group. Finally, a randomized algorithm divides the workers to two sets to improve fairness. However, the cost of providing the possibility of assigning each of the arriving tasks to any of the n workers is a slight decrease in the achieved reward.

The linear programming technique is a very useful tool in studying online matching problems. Formulating the sequential assignment problem as a linear program provides the possibility of deriving bounds on the performance of optimal policies. Moreover, various extensions of the problem can be modeled by simple changes in the objective function and constraints. The third part of this dissertation formulates several extensions of the sequential assignment problem as linear programs. Instead of defining the reward of each assignment as the product of the task value and the worker's success rate, the linear programming formulation assumes a binary reward function: Assigning the i^{th} largest task value to the worker with the i^{th} largest success rate (for $i = 1, 2, ..., n$) is referred to as a correct matching. The reward of each correct matching is one and any other matching yields a zero reward. For example, assigning the third largest task (among all n tasks) to the worker with the third largest success rate values yields a reward of one while assigning the largest task to the second largest success rate yields a reward of zero.

The linear programming formulation defines an objective function that provides an upper bound on the reward achieved by any assignment policy. The constraints of the linear program ensure that each task is assigned to at most one worker and each worker is assigned to at most one task. Duality is used to prove that no algorithm can achieve a competitive ratio better than e for the sequential assignment problem. With the binary reward function, the optimal offline reward is n since the best offline matching assigns each task to the correct worker and yields a reward of one. Therefore, the competitive ratio of e is equivalent to an optimal objective value of n/e for the formulated linear program. The Edge-Weighted Online Bipartite Matching Problem is used to propose the e-competitive algorithm for the sequential assignment problem. The algorithm is very similar to the assignment policy proposed for GSSAP. However, the length of the training phase is smaller. Moreover, the algorithm assigns each task to a worker by finding the optimal offline matching on the set of tasks observed so far. Note that this algorithm is a generalization of the optimal algorithm of the Secretary Problem, which yields an e-competitive ratio for hiring the best secretary among a set of n sequentially arriving candidates.

The linear programming formulation of the sequential assignment problem is extended to formulate two other online matching problems as linear programs. The first one is the incentive compatible problem that provides fairness by assuming that the probability of assigning each task to each worker is equal. The linear programming technique is used to prove that no algorithm can achieve a reward with a lower bound better than 29% of the maximum offline reward. Then, the optimal algorithm for the sequential assignment problem is extended to present an incentive compatible algorithm. It is proven that the incentive compatible algorithm guarantees a lower bound of 27% compared to the optimal offline reward. The sequential assignment problem with an unknown number of elements is the last online matching problem formulated as a linear program. It is assumed that the number of elements is selected by an adversary from a given set. The linear programming formulation shows that no algorithm can achieve a constant competitive ratio.

Assuming that at most one task is assigned to each worker does not accurately model many online matching problems. The last part of this thesis assumes that each worker becomes available for possible future assignments after performing the previously assigned task. The number of stages that a worker is not available due to the previous assignment is referred to as the task duration. The problem is studied under several models for task duration and task values. Assignment policies are proposed for fixed task duration. Then, a random memoryless model that assumes the task duration follows a geometric distribution is presented. The proposed algorithms are extensions of the optimal policy of the basic formulation. These algorithms find the optimal offline matching on the observed tasks so far (after the training phase). Then, assign tasks to workers based on the optimal matching if the worker is available.

1.4 Outline

This dissertation consists of five more chapters. Chapter 2 reviews the existing literature on SSAP and provides a brief summary of the results on various extensions of the basic formulation. Chapter 3 relaxes the assumption that the workers' success rates are fixed and proposes policies for SSAP with random success rates. Chapter 4 studies the relation between the Secretary Problem and a variation of SSAP, where there is not prior information on task values. The Weighted Secretary Problem is used to propose several assignment policies for SSAP with tasks having a random arrival order. The linear programming formulations of several sequential assignment problems are presented in Chapter 5 and used to derive bounds on the performance of optimal policies. Chapter 6 proposes assignment policies for the Dynamic Sequential Assignment Problem, where more than one task might be assigned to each worker. A summary of the results and several concluding remarks are presented in Chapter 7.

CHAPTER 2

LITERATURE REVIEW

This chapter reviews the SSAP literature. SSAP was first introduced by Derman et al. in 1972 [19]. They formulate the problem as a job assignment problem that can also be applied in house hunting, where an owner of k identical houses must select among a set of n sequentially arriving offers. Derman et al. propose the optimal assignment policy and discuss an extension of the problem with a more general reward function.

Various extensions of the basic formulation SSAP have been studied in the literature. The objective of these papers is to find the optimal policy for the problem with relaxed assumptions. Many of these papers focus on the task distribution and either study a more general version of the problem or assume partial information about the task values is available. For example, Kennedy [33] removes the independence assumption between the task values. He proposes the optimal assignment policy for finite and infinite number of tasks. The optimal policy has the same threshold structure of the optimal policy of the basic formulation. However, the optimal interval thresholds are random values, depending on the tasks' associated distribution.

Lee and Jacobson [41] generalize the SSAP by introducing uncertainty into the task value distribution. They present three estimators for various uncertainty levels and show that closed-loop policies based on observed task values improve the overall reward.

Albright [3] assumes that several parameters of the task distribution are unknown and the information on these unknown parameters is updated in a Bayesian manner upon arrival of each task. He uses the optimal policy of SSAP to derive the optimal policy and presents the explicit form for several distribution functions.

Derman et al. [20] describe an extension of SSAP as an investment problem and derive the optimal solution under some assumptions on the reward function. They assume that an investment opportunity occurs with a fixed probability in each of the N time periods. The expected reward of investing y in each time period is given by $P(y)$, a non-decreasing continuous function of the investment amount. The objective is to compute the optimal investment at each opportunity to maximize the total expected profit. Derman et al. derive the structure of the optimal policy for concave reward functions. Bounds on the optimal value function, asymptotic results, and the closed-form expression optimal for the optimal value for several special cases are also presented. They extend the results to a continuous-time model with opportunities arriving as a Poisson process.

David and Yechiali [18] assume that the sequentially arriving offers must be matched to a set of candidates. Assigning an offer to a candidate yields a reward of R if they match. Otherwise, a smaller reward r is achieved. They also assume a fixed discount rate for later stages of the problem. They derive optimal policies for several variations of the problem with different assumptions on the parameters and the assignment regime.

Nakai [48] considers the problem as a process observed at some time points, which are the states of a stationary Markov chain with the states not known explicitly. He uses the Bayes' theorem to specify a learning process and show that the optimal policy is not always a threshold-based algorithm.

SSAP formulates the online matching problem as a discrete-time model. The number of stages is a discrete known value and the reward function is independent of the stage number. Several papers consider a continuous-time model with a time-dependent reward. For example, Albright [3] studies the SSAP by assuming that tasks arrive according to a known continuous distribution and the reward of each assignment (i.e., the product of the task value and the worker's success rate) is multiplied by a time-dependent discount function. Albright proposes time-dependent threshold-based policies for two task arrival models: A non-homogeneous Poisson process with a continuous intensity function and a renewal process. He computes a threshold $y_n(t)$ as a function of the number of remaining workers n and task's arrival time t. If a task with value x arrives at time t and the number of remaining workers is n, then the task is assigned to one of the workers if $x > y_n(t)$.

Albright [4] considers a continuous-time model with workers becoming available after performing the previous job. He also assumes a cost per unit time for each worker being idle. Albright uses a queueing model with a limited waiting size to optimize the expected reward based on Markov decision analysis techniques.

Righter [54] considers the problem of allocating resources to activities to maximize the total expected reward. She assumes that either there is a single random deadline for all activities or activities have independent random deadlines. Righter shows that similar to the optimal policy of SSAP, the optimal policy is independent of the activity values if the deadlines are independent.

Using the same framework, Righter [55] studies the effects of allowing the model's parameters to be determined by independent Markov processes on the structure of optimal policy. The model's parameters include the resource arrival rate and deadline rates, the activity values, and the variability of the resource distribution. Righter shows that the total reward is increasing and convex in the activity values, but decreasing and convex in the deadline rates. She proves that if the variability of the distribution of resource values is increasing, the total reward is increasing. Righter also derives conditions for monotonicity of the optimal total expected reward in the states of the Markov processes.

Several papers relax the assumption on the number of tasks. Nikolaev and Jacobson [49] study an extension of the problem with random number of elements. They assume that the number of tasks is unknown until after the final arrival. However, they assume that the number of tasks follows a discrete distribution. Nikolaev and Jacobson define an auxiliary problem with fixed number of jobs, where the job values are dependent, to propose the optimal policy as an extension of the optimal policy of SSAP basic formulation. They extend this result to derive the optimal policy for the case that the number of tasks has an infinite support.

Albright and Derman [2] investigate the limiting behavior of the optimal intervals as the number of assignments approaches infinity. They derive a simple relation between asymptotic values of interval thresholds and the tasks' distribution.

Some researchers have extended the basic SSAP formulation by considering new objectives. Baharian and Jacobson [10] perform the task assignment under a threshold criterion, which minimizes the probability that the total reward fails to achieve a given value. Their result is extended by modeling the problem as a Markov Decision Process in an uncountable state space. They obtain sufficient conditions for existence of a deterministic Markov optimal policy.

Chun and Sumichrast [16] study a rank-based version of SSAP with the objective to minimize the sum of weighted ranks of jobs and machines. They propose a rank-based assignment policy that can be used in several areas including machine scheduling, job interview, kidney transplant problem, and emergency evacuation of patients in a mass-casualty situation.

Baharian and Jacobson [9] study stationary policies for SSAP, which achieve the optimal expected reward per task as the number of tasks approaches infinity. They first derive the optimal policy and the convergence rate by assuming IID task values with a known observable distribution. Then, they address the problem with unobservable task distribution with a hidden Markov chain.

Baharian and Jacobson [11] combine the results of [9] and [10] to study the limiting behavior of target-dependent stochastic sequential assignment problem. They address the problem under both assumptions of observable and unobservable task distribution.

Khatibi et al. [35] study the SSAP with random success rates, which are assumed to take on the same value during the entire assignment process. They propose several assignment policies for different uncertainty levels in workers' success rates.

Feng and Hartman [31] allow the decision maker to hold a number of jobs that may be rejected or accepted later to study the value of postponing decisions. While this queueing assumption significantly increases the complexity, they show the existence of threshold-based optimal policies under mild assumptions.

Nakai [47] assumes that after one period from assigning a task to a worker, the worker is fired with a fixed probability or remains permanently employed. He also assumes a search cost and a discount factor for the reward of each assignment.

Bloch and Houy [13] consider the problem of assigning durable objects to agents who live for two periods. They show that the optimal policy is stationary, favors old agents and is determined by a selectivity function. They further show that the selectivity function satisfies an iterative functional differential equation.

Several papers focus on the applications of SSAP. For example, Su and Zenios [59] use a sequential stochastic assignment model for the Kidney Allocation problem with the reward defined as the quality-adjusted life expectancy following the transplantation. They divide the kidney types into different domains based on patient types to propose a partition policy and prove asymptotic optimality when patients must accept all assigned kidneys. Su and Zenios use an incentive compatible condition to guarantee that the assignments determined by the policy are not rejected by patients. They use this condition to propose a second policy that reflects the patient choice in the kidney transplant problem.

David and Levi [17] study a sequential assignment problem in the framework of an online matching market and propose policies to maximize the total expected discounted revenue. They show that the optimal assignment is a function of a set of thresholds, which are sorted by the values of a discrete approximation of the bid-distribution.

CHAPTER 3

DOUBLY STOCHASTIC SEQUENTIAL ASSIGNMENT PROBLEM

3.1 Introduction

This chapter relaxes the assumption that the workers success rates are fixed in SSAP. This generalization transforms the SSAP from an assignment problem with one random value (the task value) observed at each stage to a problem with several random values (the task value and the workers' success rates) observed at each stage.

The Doubly Stochastic Sequential Assignment Problem (DSSAP) is a generalization of SSAP with workers' success rates assumed to be random. Upon a task arrival, the workers' success rate values, which are possibly different from those of the previous stages, are observed. Then, the task is irrevocably assigned to one of the available workers. This procedure is repeated until all tasks are assigned to workers. The reward of each assignment is the product of the worker's success rate and the task value assigned to the worker. The objective is to find a policy that maximizes the total expected reward. SSAP is a special case of DSSAP, where the workers' success rates are deterministic.

DSSAP extends the applications of SSAP to many online matching problems, where a set of (online) sequentially arriving elements must be matched to a set of bidders. The Online Bipartite Matching is one of the first online matching problems, which deals with matching the vertices in a bipartite graph [32]: Given a bipartite graph $G(U, V, E)$, the V vertices arrive sequentially, and the edges incident to a vertex are exposed upon the vertex arrival. The goal is to find an algorithm that matches each arrived vertex $v \in V$ to a vertex $u \in U$ such that the size of the matching is maximized.

In the Edge-Weighted Online Bipartite Matching problem, the vertex set U and the number of sequentially arriving vertices $v \in V$ are initially given [40]. Upon arrival of a vertex $v \in V$, its incident edges and their weights are observed. Then, the arrived vertex v must be irrevocably matched to one of the vertices of the set U or remain unmatched. The goal is to maximize the weight of the matched set. The sequentially arriving tasks in the DSSAP correspond to the sequentially arriving vertex set V in the Edge-Weighted Online Bipartite Matching problem and the workers in the DSSAP problem correspond to the vertex set U in the Online Bipartite Matching. The reward of each assignment in the DSSAP, which is defined as the product of the i^{th} task value and j^{th} worker's success rate, is the weight of the the edge between i^{th} arriving vertex of the set V and the j^{th} vertex of the set U. Therefore, the DSSAP is a special case of the Edge-Weighted Online Bipartite Matching, where each arriving vertex is adjacent to all vertices that are not previously matched and the edge weights are independently drawn from known distribution functions: The weight of an edge between the left vertex i and the right vertex j is defined as the product of the i^{th} task value and the j^{th} worker. While online algorithms for the Bipartite Matching problem, which assume no prior information on the weights of edges, achieve a minimum fraction of the optimal offline matching, the optimal algorithm for DSSAP, which assumes that the task and success rate values are drawn from known distributions, maximizes the total expected reward.

DSSAP models an online matching market: There are n sequentially arriving items and n bidders. Upon an item's arrival, the bid values are observed. Individual bidders specify their bid as a percentage of the average market value of the objects and these percentages, centralized at a mean value of 1, are IID realization from bidder specific distributions. Each bidder buys only one item (and hence, leaves the market once an item is sold to him). The objective is to sell each item to a bidder such that the total money spent by the bidders is maximized. This is equivalent to an Adwords problem, where each bidder remains in the market until it gets one item and bids of each bidder come independently from a known distribution [45]. The sequentially arriving items (or the sequence of keywords in Adwords) correspond to tasks and the bidders correspond to workers in DSSAP.

The remainder of this chapter is organized as follows. Section 3.2 analyzes two special cases of the DSSAP. Section 3.2.1 assumes there is only one worker, which must be optimally assigned to one of the n sequentially arriving tasks. Section 3.2.2 assumes that all workers' success rates are independently drawn from the same distribution (DSSAP with IID success rates). Section 3.3

Table 3.1: Summary of the problems discussed in this chapter

Section	Assumptions
3.2.1	DSSAP with n tasks and one worker
3.2.2	DSSAP with IID random success rates
3.3	The DSSAP algorithms

Table 3.2: Summary of the results

Theorem number	Assumptions and contribution
1	Optimal policy for SSAP (fixed success rates)
2	Optimal policy for DSSAP with n tasks and one worker
3	Optimal policy for SSAP with uncertainty in task distribution
4	Optimal policy for SSAP with multiple tasks arriving at each stage
5	Optimal policy for DSSAP with IID success rates
6	Optimality and running time of DSSAP Algorithm for general case
7	Performance of DSSAP Ranking Algorithm
8	Polynomial running time of DSSAP Ranking Algorithm

proposes an optimal assignment algorithm for DSSAP (hereafter referred to as the DSSAP Algorithm). Due to the exponential running time of the DSSAP Algorithm, the DSSAP Ranking Algorithm is proposed. We derive a lower bound on the expected reward achieved by the DSSAP Ranking Algorithm compared to the optimal expected reward while proving a polynomial running time. Section 3.4 reports several numerical experiment results to illustrate the average performance of the proposed policies for several distributions. Section 3.5 provides concluding remarks and future research directions.

Table 3.1 summarizes the sequential stochastic assignment problems discussed in this chapter while Table 3.2 provides a brief summary of the theorems. Table 3.3 lists the most common notations used in the chapter.

3.2 Optimal Policies For DSSAP

This section discusses the DSSAP with the random success rate values observed upon each task arrival. Section 3.2.1 investigates the DSSAP with one worker. Section 3.2.2 studies the DSSAP with IID random success rates. In the following sections, the order of operation is that the task value and the realized values of the random success rates are observed, and the task is assigned to

Table 3.3: List of notations used in Chapter 3

Symbol	Meaning
X	Random variable associated with tasks
x	Realized value of a task
F_X	*cdf* of random variable X
p_i	Fixed success rate of worker i
$a_{i,n}$	Optimal interval thresholds for the n-stage problem by Derman's policy
Q	Random variable associated with a worker's success rate
q	Realized value of a worker's success rate
$c_{i,n}$	Optimal interval thresholds for the n-stage problem for $X \times Q$
$Q_{(j)}$	Random variable associated with the worker with j^{th} largest expected success rate
$c_{i,n}^{(j)}$	Optimal interval thresholds for the n-stage problem for $X \times Q_{(j)}$
$S_{t,i}$	Subset number t with i of the workers success rates
$R(S_{t,i})$	Reward of assigning tasks to workers with success rates in $S_{t,i}$

one of the workers. When the next task arrives, its value and the (possibly) new success rate values of the remaining workers are observed, and the task is assigned to one of the workers. This process continues until all tasks are assigned to remaining workers. It is assumed that the *cdf* of the task's and the success rates' associated random variables are known.

3.2.1 DSSAP with One Worker

Consider the DSSAP with n sequentially arriving tasks and one worker with a random success rate. The objective is to maximize the expected reward of assigning one of the tasks to the single worker. Example 1 illustrates DSSAP with one worker.

Example 1. *Assume that there are three sequentially arriving tasks with IID random values and one worker. The worker has a random success rate (or task completion rate) which upon each task arrival, is generated independent of the previous stages (from a known distribution). The objective is to assign one of the tasks to the worker such that the expected product value of the task and the worker's success rate is maximized. Assume that the task values are* 0.4, 0.3, *and* 0.7, *and the success rate values of the three stages are* 0.5, 0.9, *and* 0.3, *respectively. Then, the reward of assigning the second task to the worker is given by* $0.3 \times 0.9 = 0.27$. *Note that the task and success rate values of each stage are only observed if no task is assigned to the worker in a prior stage. Therefore, upon assignment of the second task to the worker, the decision maker does not* *have access to the task and success rate values of the next stage.*

Theorem 2 describes the optimal policy for DSSAP with one worker.

Theorem 2. *Assume there are n sequentially arriving tasks (with values independently drawn from a given distribution) and one worker with a random success rate to be assigned one of the tasks. The worker's success rate takes new values (independently drawn from a given distribution) upon each task arrival. Let X and Q denote the random variables associated with the tasks and the success rate, respectively. Then, there exists numbers*

$$-\infty = c_{0,n} \leq c_{1,n} \leq \dots \leq c_{n-1,n} \leq c_{n,n} = +\infty$$

such that the optimal policy in the initial stage is to assign the arriving task to the worker if the product of the task and the success rate realized values is contained in the interval $(c_{n-1,n}, c_{n,n}]$. The maximum expected reward is given by $c_{n,n+1}$. Moreover, the $\{c_{i,n}\}$ satisfy

$$c_{i,n+1} = \int_{c_{i-1,n}}^{c_{i,n}} y\, dF_Y(y) + c_{i-1,n} F_Y(c_{i-1,n}) + c_{i,n}(1 - F_Y(c_{i,n})) \tag{3.1}$$

where $c_{0,n} = -\infty$, $c_{n,n} = +\infty$, and $F_Y(y)$ is the cdf of $Y = X \times Q$ (the distribution of the product of tasks and the random success rates).

Proof: Let P_I denote the DSSAP with one worker. To find the optimal policy for P_I, an equivalent SSAP (i.e., a sequential stochastic assignment problem with fixed success rates) is defined: Let P_{II} denote the SSAP with n sequentially arriving tasks with an associated random variable $Y = X \times Q$ and one worker with a fixed success rate $p_n = 1$ (and $n-1$ workers of success rate zero). Problem P_{II} is a SSAP with fixed success rates with an optimal policy described in Theorem 1: Assign the first arriving task to the worker with success rate $p_n = 1$ if its realized value (i.e., the realized value of $y = x \times q$) is contained in $(c_{n-1,n}, c_{n,n}]$, where the constants $\{c_{i,n}\}$ are computed using Corollary 1 for the random variable Y. Moreover, the maximum expected reward is given by $c_{n,n+1}$.

Induction is used to prove that the optimal policy for P_{II} achieves the maximum total expected reward for P_I: The optimal policy for P_I assigns the arriving task in the first stage to the worker if

$x \times q$ is contained in $(c_{n-1,n}, c_{n,n}]$. For the case of $n = 1$, the single arriving task is assigned to the single worker. To complete the proof, suppose that the optimal policy in the $(n-1)$-stage problem is to assign the first arriving task to the worker if $x \times q \in (c_{n-2,n-1}, c_{n-1,n-1}]$, which achieves a total expected reward $c_{n-1,n}$ (the induction assumption). Two mutually exclusive and collectively exhaustive cases are considered to prove the optimality of the policy for n tasks.

Case 1: Suppose that the product of the first arriving task and the success rate $x \times q$ takes a value in interval $(c_{n-1,n}, c_{n,n}]$, but the task is not assigned to the worker. By the induction assumption, the maximum total expected reward in the $(n-1)$-stage problem is $c_{n-1,n} (< x \times q)$. Therefore, if $x \times q \in (c_{n-1,n}, c_{n,n}]$, assigning the task to the worker achieves a larger expected reward than discarding the task.

Case2: Suppose that the first arriving task is assigned to the worker with $x \times q < c_{n-1,n}$. However, if the task is not assigned to the worker, the maximum total expected reward achieved in the $(n-1)$-stage problem is $c_{n-1,n}$. Therefore, the optimal policy does not assign the first arriving task to the worker if the product of the task and the success rate values is not in the interval $(c_{n-1,n}, c_{n,n}]$.

Therefore, the optimal policy for P_I assigns the first arriving task in the n-stage problem to worker if the task's realized value is contained in the interval $(c_{n-1,n}, c_{n,n}]$. The maximum total expected reward is $c_{n,n+1}$ by Theorem 1. □

The DSSAP with one worker is a full information Secretary Problem with each secretary's quality value defined as the product of two random numbers revealed upon the secretary arrival. While the objective in the Secretary Problem is to maximize the probability of hiring the best secretary based on an ordinal criterion, the optimal policy of the DSSAP with one worker maximizes the expected reward.

Theorem 2 provides the optimal policy to hire one applicant (assign one of the sequentially arriving tasks to a single worker). Extensions of Theorem 2 solve similar problems with different objectives. For example, if the objective is to maximize the total expected reward of hiring k secretaries when there are n applicants, the optimal policy is to hire the interviewed applicant if $x \times q \in (c_{n-k,n}, c_{n,n}]$ (i.e., the product of the realized values of the success rate and the arriving task is contained in one of the k largest intervals). Moreover, the total expected reward is given by $\sum_{i=n-k+1}^{n} c_{i,n+1}$. This problem is referred to as the K-choice Secretary Problem [8].

3.2.2 DSSAP with IID Random Success Rates

This section studies the DSSAP with IID success rates. First, an extension of SSAP with multiple tasks arriving simultaneously at each stage is discussed. Then, the relation between the SSAP with multiple tasks and the DSSAP is described. The optimal policy of SSAP with multiple tasks, which can be derived by small modifications of Derman's policy, is then used to derive the optimal policy of the DSSAP with IID success rates.

Suppose that multiple tasks with IID associated random values arrive simultaneously at each stage in the SSAP and there are workers with fixed success rates to be assigned the tasks. At the i^{th} stage, $n-i+1$ tasks arrive, with one task selected and assigned to one of the workers. This process continues until all workers are assigned to tasks. The goal is to maximize the total expected reward of the assignment. To find the optimal policy, consider the SSAP with uncertainty in tasks' distribution. For the sake of self-completeness, the optimal policy of this problem (proposed by Albright [5]) is briefly discussed.

Let n denote the total number of tasks in the SSAP and assume that there are one or more unknown parameters in the task's distribution. These parameters have a prior distribution, which is updated via Bayes' rule as successive task values are observed. Let q and q_x denote the prior and posterior distributions, respectively. Then, Theorem 1 can be modified to find the optimal policy for SSAP with uncertainty in task's distribution.

Theorem 3. *[5] Suppose n X's remain to be seen and the current prior is q. Then there exist numbers*

$$-\infty = a_{0,n} \leq a_{1,n}(q_x) \leq ... \leq a_{n-1,n}(q_x) \leq a_{n,n} = +\infty$$

such that if the next X has value x, it is best to assign the worker with the i^{th} smallest success rate value to this x if and only if $a_{i-1,n}(q_x) < x \leq a_{i,n}(q_x)$.These critical numbers do not depend on the workers' success rates. Furthermore, before this first X is observed, $a_{i,n+1}(q)$ is the expected value of the X which, under the optimal policy, will eventually be paired with p_i. Finally, the critical numbers satisfy the recursion

$$a_{i,n+1}(q) = \int_A x dH(x) + \int_{\underline{A}} a_{i-1,n}(q_x) dH(x) + \int_{\overline{A}} a_{i,n}(q_x) dH(x) \tag{3.2}$$

where H is the current marginal distribution of the X's, $\underline{A} = \{x : x \leq a_{i-1,n}(q_x)\}$, $A = \{x : a_{i-1,n}(q_x) \leq x \leq a_{i,n}(q_x)\}$, and $\overline{A} = \{x : x > a_{i,n}(q_x)\}$.

Theorem 3 assumes uncertainty in tasks distribution (while there is only one task arriving at each stage). Theorem 4 describes the optimal policy for SSAP with multiple IID arriving tasks at each stage by transforming the problem to SSAP with uncertainty in task distribution and using the result of Theorem 3. It is assumed that the number of arriving tasks (with IID values) at each stage is equal to the number of remaining workers. One of the arriving tasks at each stage is selected and assigned to one of the workers.

Theorem 4. *For each $n \geq 1$, there exists numbers*

$$-\infty = m_{0,n} \leq m_{1,n} \leq ... \leq m_{n-1,n} \leq m_{n,n} = +\infty$$

such that whenever there are n simultaneously arriving tasks with IID associated random variables $X_1, X_2, ..., X_n$ and n workers with fixed success rates $p_1 \leq p_2 \leq ... \leq p_n$ to be assigned one of the tasks at each stage, then the optimal choice in the initial stage is to assign the task with the maximum value to the worker with success rate p_i if the maximum task value is contained in $(m_{i-1,n}, m_{i,n}]$. The maximum total expected reward is given by

$$\sum_{i=1}^{n} m_{i,n+1} p_i, \tag{3.3}$$

where the $\{m_{i,n}\}$ depend on the cdf of the task's associated random variable, but are independent of the $\{p_i\}$. Moreover, the $\{m_{i,n}\}$ satisfy the following recursion.

$$m_{i,n+1} = \int_{m_{i-1,n}}^{m_{i,n}} y d(F_X^n(y)) + \int_{-\infty}^{m_{i-1,n}} m_{i-1,n} d(F_X^n(y)) + \int_{m_{i,n}}^{+\infty} m_{i,n} d(F_X^n(y)) \tag{3.4}$$

where F_X denotes the cdf of the random variable X_i, $i = 1, 2, ..., n$.

Proof: Since the tasks are IID, the task with the maximum realized value at each stage must be assigned to one of the workers. Therefore, the task assigned at each stage has a value coming from a known distribution, which is the maximum of a set of IID random variables. If F_X

denotes the cdf of each task's associated random variable, the cdf of the maximum of n IID tasks is given by F_X^n, which is the current marginal distribution of the tasks' values. However, there is an uncertainty in the task's associated random variable; it is updated from the maximum of $(n-i+1)$ IID random variables at stage i (F_X^{n-i+1}) to the maximum of $(n-i)$ random variables at stage $i+1$ (F_X^{n-i}). Hence, the problem can be considered as the SSAP with a single task arriving at each stage with uncertainty in task's associated random variable. This is an special case of Theorem 3 with the task's marginal distribution at stage i (F_X^{n-i+1}) corresponding to H and the optimal interval thresholds $\{m_{i,n}\}$ corresponding to $\{a_{i,n}(q_x)\}$ in (3.4). $\square$

Let X and Q_i denote the random variables associated with the task and the i^{th} worker's success rate, respectively. The DSSAP with n IID random success rates is similar to the SSAP with multiple arriving tasks, discussed in Theorem 4, with tasks' associated random variable $X \times Q$ and fixed success rates $p_i = 1$, $(i = 1, 2, ..., n)$. Assigning the task X to one of the workers Q with IID random success rates in the DSSAP is similar to assigning one of the simultaneously arriving tasks $X \times Q$ to a worker with fixed success rate 1 in the SSAP with multiple arriving tasks at each stage. The Greedy algorithm, which assigns the arriving task to the worker with the maximum realized value, achieves the maximum total expected reward in the DSSAP with IID random success rates.

Theorem 5. *The Greedy algorithm, which assigns the arriving task to the worker with the maximum success rate value at each stage, achieves the maximum total expected reward in the DSSAP with IID random success rates. Moreover, the maximum total expected reward is given by*

$$E[X] \times (\sum_{i=1}^{n} \int_{-\infty}^{\infty} y d(F_Q^{n-i+1}(y))), \tag{3.5}$$

where $E[X]$ denotes the expected value of the random variable X and F_Q denotes the cdf of the workers' random success rate.

Proof: The proof has two parts. The first part uses induction to prove that the Greedy algorithm is optimal. The second part computes the maximum total expected reward achieved by the Greedy algorithm. For the first part, in the base case of induction $(n = 1)$, the single arriving task is assigned to the single worker. Assume that the Greedy algorithm is optimal for the $(n-1)$-stage problem (the induction assumption). Based on the induction assumption, the optimal policy in stage $i = (2, 3, ..., n)$ is to assign the task to the worker with the maximum success rate value. Consider the problem with n tasks. Suppose that $Q^M(i)$ denotes the maximum of $(n - i + 1)$ IID random success rates (i.e., the maximum of the workers success rates at stage i):

$$Q^M(i) = \max_{j \in \{1,2,...,n-i+1\}} \{Q_j\}$$

The total expected reward of the Greedy algorithm for the n-stage problem is given by

$$E[X] \times (q^M(1) + E[Q^M(2)] + ... + E[Q^M(n)]) \tag{3.6}$$

where $q^M(1)$ is the maximum success rate value at stage one. Contrary to the claim, assume that the optimal policy assigns the first task to a worker with success rate $q(1)$ $(< q^M(1))$, which achieves a total expected reward

$$E[X] \times (q(1) + E[Q^M(2)] + ... + E[Q^M(n)]) \tag{3.7}$$

Since $Q^M(i)$ $(i = 2, 3, ..., n)$ is the maximum value of $(n - i + 1)$ IID random variables, it is independent of the first stage's choice. Therefore, $E[X] \times (q^M(1) - q(1)) > 0$ and the expected reward achieved by the Greedy algorithm (3.6) is larger than (3.7). This contradicts the claim and hence, the Greedy algorithm, which assigns the task to the worker with the maximum success rate value, achieves the maximum total expected reward.

For the second part, the maximum total expected reward achieved by the Greedy algorithm is computed. In the first part, it is proven that the success rate assigned at the i^{th} stage is the maximum of the remaining $(n - i + 1)$ success rates, which are assumed to be IID. The cdf of the maximum of $(n - i + 1)$ IID success rates is given by $F_Q^{n-i+1}(y)$. Therefore, the expected reward of the i^{th} assignment is given by

$$E[X] \times E[Q^M(i)] = E[X] \times \int_{-\infty}^{\infty} y d(F_Q^{n-i+1}(y)) \tag{3.8}$$

The total expected reward is the sum of the assignment rewards, as given by (3.5). $\square$

3.3 The DSSAP Algorithms

This section relaxes the assumption of IID random success rates and describes assignment algorithms for the DSSAP with distinct success rates distributions. There are n sequentially arriving tasks that must be assigned to n workers with random success rates (which are not necessarily identically distributed). Upon a task arrival, its value and the value of the workers success rates are observed. Then, the task is irrevocably assigned to one of the workers. The workers success rates take new values upon each task arrival. The distributions of the task values and the workers success rates are known. Example 2 illustrates the general case of DSSAP.

Example 2. *Assume that there are two sequentially arriving tasks with IID random values and two workers. The workers have random success rates (that are not necessarily identically distributed) which upon each task arrival, are generated independent of the previous stages. The objective is to assign each task to one worker such that the total expected reward, defined as the sum of the reward of individual assignments, is maximized. Assume that the task values are given by 0.4 and 0.5 and worker 1's success rates at the first and second stages are 0.7 and 0.5, respectively. Moreover, worker 2's success rates at the two stages are given by 0.3 and 0.8. Then, the reward of assigning the first task to worker 1 and the second task to worker 2 is given by $0.4 \times 0.7 + 0.5 \times 0.8 = 0.68$ while the reward of assigning the first task to worker 2 and the second task to worker 1 is given by $0.4 \times 0.3 + 0.5 \times 0.5 = 0.37$.*

The optimal assignment policy, which maximizes the total expected reward, is referred to as the DSSAP Algorithm. Since the DSSAP Algorithm has exponential running time, the DSSAP Ranking Algorithm that achieves a fraction of the maximum expected reward in a polynomial running time, is proposed in Section 3.3.2.

3.3.1 The DSSAP Algorithm

The DSSAP Algorithm is a computational approach that uses backward induction to determine the optimal assignment at each stage. The maximum total expected reward for n tasks is computed as the sum of the reward of assigning the first arriving task and the maximum total expected reward of assigning $n-1$ tasks to the remaining $n-1$ workers. To find the optimal assignment at each stage, the expected reward of assigning the arriving task to each of the remaining workers is computed and compared to determine the optimal range of task and success rate values for each assignment. Then, the probability of each of these realizations is computed and the maximum total expected reward is derived as the weighted sum of the expected rewards.

Algorithm 1 The DSSAP Algorithm

```
for i=1 to n
    $S_{t,i}$ =subsets of $i$ workers (with $t = 1, 2, ..., \binom{n}{i}$)
    for all $Q_j \in S_{t,i}$
        $R(S_{t,i}|\textit{first task assigned to } Q_j) = x \times q_j + E[R_D(S_{t,i} - \{Q_j\})]$
        $P_j = Pr[\{x, q_1, q_2, ..., q_n\} \in \phi_{S_{t,i}}(Q_j)]$
        $E[R_D(S_{t,i})] = \sum_{Q_j \in S_{t,i}} P_j \times E[R_D(S_{t,i}|\{x, q_1, ..., q_n\} \in \phi_{S_{t,i}}(Q_j)]$
    end for
end for
```

The DSSAP Algorithm computes the maximum total expected reward for any subset of i $(= 1, 2, ..., n)$ workers. Let $S_{t,i}$ denote the subset number t with i workers. For example, there are $\binom{n}{2}$ subsets of two workers: $S_{1,2} = \{Q_1, Q_2\}$, $S_{2,2} = \{Q_1, Q_3\}$,..., $S_{n-1,2} = \{Q_1, Q_n\}$, $S_{n,2} = \{Q_2, Q_3\}$,..., $S_{\binom{n}{2},2} = \{Q_{n-1}, Q_n\}$. Let $R(S_{t,i}|\textit{first task assigned to } Q_j)$ denote the reward of assigning tasks to workers $S_{t,i}$ if the first arriving task is assigned to worker with success rate Q_j. The DSSAP Algorithm computes the reward of assigning the first arriving task to each worker $Q_j \in S_{t,i}$ as the sum of the first assignment's reward and the maximum expected reward of assigning the $i - 1$ tasks to the remaining workers in the set

$$R(S_{t,i}|\textit{first task assigned to } Q_j) = x \times q_j + E[R_D(S_{t,i} - \{Q_j\})], \tag{3.9}$$

where $E[R_D(S_{t,i} - \{Q_j\})]$ is the expected reward achieved by the DSSAP Algorithm when the set of workers is given by $S_{t,i} - \{Q_j\}$. Then, it compares $R(S_{t,i}|\textit{first task assigned to } Q_j)$ for all $\{Q_j\} \in S_{t,i}$ to find $\phi_{S_{t,i}}(Q_j)$, the range of task and random success rate values for which assigning the first task to worker with success rate Q_j is optimal (i.e., assigning the first task to worker with success rates Q_j achieves the largest values in (3.9)). The DSSAP Algorithm assigns the first task to worker with success rate Q_j if the task and success rate values are in the optimal range $(\phi_{S_{t,i}}(Q_j))$. The next step computes the probability (P_j) that the task and random success rates take values in the optimal range $(\phi_{S_{t,i}}(Q_j))$ to find the maximum total expected reward.

Theorem 6. *The DSSAP Algorithm achieves the maximum total expected reward for the DSSAP in an exponential running time.*

Proof: The proof uses induction on n. For the case of $n = 1$, the single arriving task is assigned to the single worker. Assume that the DSSAP Algorithm maximizes the total expected reward for $(n-1)$ workers (the induction assumption). Therefore, $E[R_D(S_{1,n} - \{Q_j\})]$ $(j = 1, 2, ..., n)$ is the maximum total expected reward for the DSSAP with $n-1$ workers, where $S_{1,n} = \{Q_1, Q_2, ..., Q_n\}$ denotes the set of all workers success rates.

Let $\phi_{S_{t,i}}(Q_j)$ denote the range of task and success rate values for which assigning the first arriving task in the n-stage problem to worker j achieves the maximum total expected reward (i.e., the maximum value of $(x \times q_l + E[R_D(S_{1,n} - \{Q_l\})]$, with $l = 1, 2, ..., n)$. Hence, if $(x, q_1, q_2, ..., q_n) \in \phi_{S_{t,i}}(Q_j)$, then

$$x \times q_j + E[R_D(S_{1,n} - \{Q_j\})] = \max_{Q_l \in S_{1,n}} (x \times q_l + E[R_D(S_{1,n} - \{Q_l\})]) \tag{3.10}$$

If (3.10) holds, the DSSAP Algorithm assigns the first arriving task to the worker with success rate Q_j. For all task and success rate values in the optimal range $(\phi_{S_{t,i}}(Q_j))$, the expected reward of assigning the first task to worker Q_j is maximum. Therefore,

$$\begin{aligned} E[x \times q_j + E[R_D(S_{1,n} - \{Q_j\})] | (x, q_1, q_2, ..., q_n) \in \phi_{S_{t,i}}(Q_j)] = \max_{Q_l \in S_{1,n}} (E[x \times q_l \\ + E[R_D(S_{1,n} - \{Q_l\})] | (x, q_1, q_2, ..., q_n) \in \phi_{S_{t,i}}(Q_j)]) \end{aligned} \tag{3.11}$$

Taking expectation of both sides yields:

$$E[R_D(S_{1,n})] = \max_{Q_l \in S_{1,n}} (E[x \times q_l + E[R_D(S_{1,n} - \{Q_l\})]]) \tag{3.12}$$

which proves the optimality of the DSSAP Algorithm for n workers.

The complexity of the DSSAP Algorithm depends on the number of random success rates with distinct distributions. When there are n random success rates with distinct distributions, the number of cases that are recursively considered is $\sum_{j=1}^{n}\binom{n}{j} = 2^n - 1$. Therefore, the computational time is exponential. $\square$

The DSSAP Algorithm determines the optimal assignment once the task and success rate values are revealed at each stage, i.e., the optimal intervals are computed before the assignment process. The optimal policies for the SSAP and the DSSAP with IID success rates are special cases of the DSSAP Algorithm. When all success rates are fixed, the DSSAP Algorithm gives the optimal policy for the SSAP (Theorem 1). It also reduces to the optimal policy described in Theorem 5 when all workers have IID random success rates.

The optimal policy of DSSAP (i.e., the DSSAP Algorithm) uses dynamic programming to solve the smaller sub-problems (since the optimal assignment at each stage depends on the assignment of the prior stages). This is similar to the optimal policy of SSAP, whose interval thresholds are computed recursively using smaller sub-problems. The workers success rates are fixed in the SSAP. Therefore, using the Hardy's Lemma, the optimal interval thresholds are independent of the workers success rates. This is not the case in DSSAP, where by Theorem 2, the optimal intervals depend on the distribution of the workers success rates. For example, the optimal policy of a 2-stage DSSAP computes the range of optimal assignment for each of the workers. If we change one of the workers, the optimal intervals must be computed again by comparing the expected reward of the possible two assignments. Hence, the factor that determines the order of computational complexity for the optimal policy is the total number of subsets of workers. This cannot be reduced without any further assumptions on the distribution of the random success rates.

In fact, adding a new worker with a distinct success rate requires computing the optimal thresholds and the maximum expected reward for all new subsets of workers, which doubles the number of possible subsets (since the new worker's success rate must be added to all existing subsets). Therefore, while adding a new worker to SSAP does not change the order of computational complexity (the computational complexity is only changed from $(n-1)^2$ to n^2), adding one worker to DSSAP doubles the computational complexity of the optimal policy.

3.3.2 The DSSAP Ranking Algorithm

The DSSAP Algorithm achieves the maximum total expected reward, but requires an exponential running time. This section proposes a polynomial-time algorithm that achieves a fraction of the maximum expected reward. The *DSSAP Ranking Algorithm* gives a priority level in task assignment to each worker by comparing the expected values of the workers' random success rates. Assuming that the task and success rate values come from a known distribution, the DSSAP Ranking Algorithm can be used to achieve an expected reward close to the optimal reward.

The intuition behind the DSSAP Ranking Algorithm is that the computational time can be decreased by assuming that the higher ranked workers (i.e., the workers with assignment priority) have a larger effect on the total expected reward. Therefore, if the assignment priority is given to the workers with larger expected success rates, a reasonable fraction of the maximum total expected reward is achieved while keeping the computational time polynomial. Note that the DSSAP Ranking Algorithm achieves a reasonable fraction of the maximum expected reward in a random setting since it assumes that in expectation, a worker with a larger expected success rate achieves a larger proportion of the total reward.

The DSSAP Ranking Algorithm ranks the workers by comparing the expected values of their random success rates and computes the optimal interval thresholds for the product of the task and each worker's success rate. Let $Q_{(j)}$ $(j = 1, 2, ..., n)$ denote the success rate of the worker with the j^{th} largest expected value and $c_{i,n}^{(j)}$ $(i = 0, 1, 2, ..., n)$ denote the interval thresholds in the n-stage problem for the random variable $X \times Q_{(j)}$, defined as in Corollary 1. Starting from the worker with the highest priority (i.e., the worker with the largest expected reward value), the arriving task is assigned to the worker with success rate $Q_{(j)}$ if the realized value of $x \times q_{(j)}$ is in one of the i largest intervals defined by $c_{i,n}^{(j)}$. Therefore, in the first stage of the n-stage problem, the arriving task is assigned to the worker with success rate $Q_{(1)}$ if $x \times q_{(1)} > c_{n-1,n}^{(1)}$. Otherwise, the task is assigned to the worker with success rate $Q_{(2)}$ if $x \times q_{(2)} > c_{n-2,n}^{(2)}$, and so on. The DSSAP Ranking Algorithm is formalized as Algorithm 2.

Consider the problem discussed in Theorem 2 as an extension of the secretary problem with the secretaries' quality values as the product of two random numbers revealed upon the secretary arrival. If the goal is to hire k secretaries with the maximum sum of the quality values, the optimal choice in the initial stage is to hire the secretary if the quality value is in one of the k largest optimal intervals. The DSSAP Ranking Algorithm uses the same method as Theorem 2 to assign a task to the worker with the j^{th} largest expected value when the assignment's reward is in one of the j largest optimal intervals. Theorem 7 proves a lower bound on the total expected reward achieved by the DSSAP Ranking Algorithm. Note that a constant competitive ratio can not be derived since based on the problem's assumptions, the ratio between the expected reward of the DSSAP Ranking Algorithm and the maximum expected reward depends on the parameters of the assumed distribution functions.

Algorithm 2 The DSSAP Ranking Algorithm

```
Rank the workers based on their success rates' expected value
Compute the optimal interval thresholds for $X \times Q_{(j)}$ $(j = 1, 2, ..., n)$
Upon arrival of a task with $m$ $(= 1, 2, ..., n)$ remaining workers
for j=1 to m
    Let $q_{(j)}$ denote the realized value of the success rate with rank $j$
    if $x \times q_{(j)} > c^{(j)}_{m-j,m}$
        assign the task to worker with success rate $Q_{(j)}$
        break
    end if
end for
```

Theorem 7. *Let $S_{1,n}$ denote the set of n random success rates $\{Q_1, Q_2, ..., Q_n\}$ and $E[R_{DR}(S_{1,n})]$ and $E[R_D(S_{1,n})]$ denote the expected reward achieved by the DSSAP Ranking Algorithm and the DSSAP Algorithm (in the n-stage problem with workers success rates $S_{1,n}$), respectively. Then,*

$$\frac{E[R_{DR}(S_{1,n})]}{E[R_D(S_{1,n})]} \geq \frac{\sum_{j=1}^{n} c^{(j)}_{n-j+1,n+1}}{\sum_{j=1}^{n} c^{(j)}_{n,n+1}} \tag{3.13}$$

where the $\{c^{(j)}_{i,k}\}$ are the optimal interval thresholds computed by Corollary 1 for random variable $X \times Q_{(j)}$ and $Q_{(j)}$ is the success rate of the worker with the j^{th} largest expected value.

Proof: The proof has two parts. The first part proves that $E[R_{DR}(S_{1,n})] \geq \sum_{j=1}^{n} c^{(j)}_{n-j+1,n+1}$ and the second part proves that $E[R_D(S_{1,n})] \leq \sum_{j=1}^{n} c^{(j)}_{n,n+1}$, which together prove (3.13).

For the first part, a surrogate SSAP is defined for each of the n workers. Then, induction is used to prove that using the DSSAP Ranking Algorithm, each worker achieves an expected reward at least as large as the surrogate SSAP. This is used to prove that $E[R_{DR}(S_{1,n})] \geq \sum_{j=1}^{n} c_{n-j+1,n+1}^{(j)}$.

Consider the following SSAP defined for the worker with success rate $Q_{(j)}$: There are n tasks with values coming from $X \times Q_{(j)}$ that must be assigned to n workers. Assume that the success rate of worker j is 1 (while the success rates of workers $1, 2, ..., j-1$ is larger than 1 and the success rates of workers $j+1, j+2, ..., n$ is smaller than 1). Define Policy A as follows: The first task in the n-stage problem is assigned to worker j if and only if $x \times q_j \in (c_{n-j,n}^{(j)}, c_{n-j+1,n}^{(j)}]$. If $x \times q_j > c_{n-j+1,n}^{(j)}$, then the task is assigned to one of the workers with a success rate larger than 1, and if $x \times q_j \leq c_{n-j,n}^{(j)}$, then the task is assigned to a worker with a success rate smaller than 1, and the process continues in the next stage. By Theorem 1, $c_{n-j+1,n+1}^{(j)}$ is the expected value, in an n-stage problem, of the task assigned to the worker with the $(n-j+1)^{th}$ smallest success rate. Therefore, the expected reward achieved by worker j (under Policy A) in the SSAP is given by $c_{n-j+1,n+1}^{(j)}$. Induction is used to prove that the expected reward achieved by the worker with success rate $Q_{(j)}$ in the DSSAP Ranking Algorithm is at least as large as the expected reward achieved by worker j with success rate 1 in the SSAP (using Policy A). This proves that the total expected reward achieved by the DSSAP Ranking Algorithm is at least $\sum_{j=1}^{n} c_{n-j+1,n+1}^{(j)}$.

The base case is trivial: In the 1-stage problem, the expected reward achieved by the single worker using the DSSAP Ranking Algorithm is the same as Policy A. As the induction assumption, assume that the expected reward achieved by the worker with success rate $Q_{(j)}$ (with $j = 1, 2, ..., n-1$) in the DSSAP Ranking Algorithm for the $(n-1)$-stage problem is at least as large as the expected reward achieved by worker j with success rate 1 under Policy A in the surrogate SSAP. The expected reward achieved by worker j using the DSSAP Ranking Algorithm in the n-stage problem is given by

$$\begin{aligned} E_n[R_{DR}(Q_{(j)})] &= E_n[R_{DR}(Q_{(j)}|x \times q_j > c_{n-j+1,n}^{(j)})] \times Pr(x \times q_j > c_{n-j+1,n}^{(j)}) \\ &+E_n[R_{DR}(Q_{(j)}|c_{n-j,n}^{(j)} < x \times q_j \leq c_{n-j+1,n}^{(j)})] \times Pr(c_{n-j,n}^{(j)} < x \times q_j \leq c_{n-j+1,n}^{(j)}) \\ &+E_n[R_{DR}(Q_{(j)}|x \times q_j \leq c_{n-j,n}^{(j)})] \times Pr(x \times q_j \leq c_{n-j,n}^{(j)}), \end{aligned} \tag{3.14}$$

where $R_{DR}(Q_{(j)})$ denotes the reward achieved by the worker with success rate $Q_{(j)}$ using the

DSSAP Ranking Algorithm and $E_n[R_{DR}(Q_{(j)})]$ denotes the expected reward achieved by this worker in the n-stage problem. Note that (3.14) is computed by conditioning on the reward of the first stage. The expected reward achieved by worker j using Policy A in the n-stage SSAP is given by

$$\begin{aligned} E_n[R_A(Q_{(j)})] &= E_n[R_A(Q_{(j)}|x \times q_j > c^{(j)}_{n-j+1,n})] \times Pr(x \times q_j > c^{(j)}_{n-j+1,n}) \\ +E_n[R_A(Q_{(j)}|c^{(j)}_{n-j,n} &< x \times q_j \le c^{(j)}_{n-j+1,n})] \times Pr(c^{(j)}_{n-j,n} < x \times q_j \le c^{(j)}_{n-j+1,n}) \\ &+E_n[R_A(Q_{(j)}|x \times q_j \le c^{(j)}_{n-j,n})] \times Pr(x \times q_j \le c^{(j)}_{n-j,n}) \end{aligned} \tag{3.15}$$

It is sufficient to prove the following:

$$E_n[R_{DR}(Q_{(j)}|x \times q_j > c^{(j)}_{n-j+1,n})] \ge E_n[R_A(Q_{(j)}|x \times q_j > c^{(j)}_{n-j+1,n})],$$

$$E_n[R_{DR}(Q_{(j)}|c^{(j)}_{n-j,n} < x \times q_j \le c^{(j)}_{n-j+1,n})] \ge E_n[R_A(Q_{(j)}|c^{(j)}_{n-j,n} < x \times q_j \le c^{(j)}_{n-j+1,n})],$$

$$E_n[R_{DR}(Q_{(j)}|x \times q_j \le c^{(j)}_{n-j,n})] \ge E_n[R_A(Q_{(j)}|x \times q_j \le c^{(j)}_{n-j,n})].$$

1. If $x \times q_j > c^{(j)}_{n-j+1,n}$ and the DSSAP Ranking Algorithm does not assign the task to the worker with success rate $Q_{(j)}$ (since the task is assigned to a higher rank worker), then by the induction assumption, it achieves an expected reward at least as large as Policy A (since in both problems, the rank of worker j becomes $j-1$ in the $(n-1)$-stage problem). If the DSSAP Ranking Algorithm assigns the task to the worker with success rate $Q_{(j)}$, then worker j achieves a reward larger than $c^{(j)}_{n-j+1,n}$ while the expected reward achieved by worker j using Policy A in the $(n-1)$-stage SSAP is given by $c^{(j)}_{n-j+1,n}$ (This is true since the rank of worker j with success rate 1 in the $(n-1)$-stage problem becomes $j-1$, and by Theorem 1, the expected reward achieved by this worker in the $(n-1)$-stage SSAP is given by $c^{(j)}_{n-j+1,n}$).

2. If $c^{(j)}_{n-j,n} < x \times q_j \le c^{(j)}_{n-j+1,n}$ and the DSSAP Ranking Algorithm assigns the task to the worker with success rate $Q_{(j)}$, then both algorithms achieve the same (since the task is also assigned to the worker in the surrogate SSAP). Otherwise, the DSSAP Ranking Algorithm assigns the task to a worker of higher rank. Therefore, the rank of the worker with success rate $Q_{(j)}$ in the $(n-1)$-stage DSSAP is reduced to $j-1$. By the induction assumption, the expected reward

achieved by this worker in the $(n-1)$-stage problem is at least as large as the expected reward achieved by worker $j-1$ (with success rate 1) in the $(n-1)$-stage SSAP $(E_{n-1}[R_A(Q_{(j-1)})])$, and by Theorem 1, $E_{n-1}[R_A(Q_{(j-1)})] = c^{(j)}_{n-j+1,n}$. This proves the claim since the expected reward achieved by worker j in the n-stage SSAP in this case is at most $c^{(j)}_{n-j+1,n}$.

3. If $x \times q_j \leq c^{(j)}_{n-j,n}$, both algorithms assign the task to a worker other than j. While the task is assigned to a worker with a success rate smaller than worker j in the SSAP, the task is assigned either to a worker of higher rank or a worker of a lower rank in the DSSAP. Therefore, while the rank of worker j in the $(n-1)$-stage SSAP remains j, the rank of worker with success rate $Q_{(j)}$ in the $(n-1)$-stage DSSAP would be j or better. Therefore, the claim holds by the induction assumption.

For the second part, consider the DSSAP with n tasks and one worker with success rate $Q_{(j)}$ (Theorem 2). The maximum expected reward of the DSSAP with one worker is achieved when the product of the task and the worker's success rate is in the largest interval, and is given by $c^{(j)}_{n,n+1}$. Therefore, the maximum expected reward that the worker with success rate $Q_{(j)}$ can achieve in an n-stage DSSAP is bounded above by $c^{(j)}_{n,n+1}$ and the maximum total expected reward is bounded above by $\sum_{j=1}^{n} c^{(j)}_{n,n+1}$. Hence, $E[R_D(S_{1,n})] \leq \sum_{j=1}^{n} c^{(j)}_{n,n+1}$, which together with $E[R_{DR}(S_{1,n})] \geq \sum_{j=1}^{n} c^{(j)}_{n-j+1,n+1}$, prove (3.13). $\square$

To see how large is the ratio (3.13), consider the case where $c^{(j)}_{i,n+1}$ is equal for all values of j. Using Corollary 1 yields $E[X \times Q_{(j)}]/c^{(j)}_{n,n+1}$ for (3.13). This ratio is equal to 0.59 for a $U(0,1)$ distribution in a 10-stage problem and approaches 0.5 as $n \to +\infty$.

Note that (3.13) does not provide a tight lower bound for the expected reward achieved by the DSSAP Ranking Algorithm. The Numerical Experiment Section compares the performance of the DSSAP Ranking Algorithm with the optimal policy and shows the ratio for several cases.

Computing the optimal interval thresholds is the main factor in determining the running time of the DSSAP Ranking Algorithm.

Theorem 8. *The running time of the DSSAP Ranking Algorithm is $O(nlog(n) \times n^3)$.*

Proof: Computing the optimal interval thresholds requires $O(n^2)$ running time, which must be done for n distinct distributions. Assuming that the expected success rate of each worker is known, a running time of $O(nlog(n))$ is required for ranking the success rates expected values.

Therefore, the running time of the DSSAP Ranking Algorithm is $O(nlog(n) \times n^3)$. $\square$

Note that if the workers are ranked randomly in the DSSAP Ranking Algorithm, the probability that worker j $(= 1, 2, ..., n)$ has rank i $(= 1, 2, ..., n)$ in the random ordered set is $\frac{1}{n}$. The expected reward of assigning a task to worker with the i^{th} highest rank is at least $\frac{1}{n}\sum_{j=1}^{n} c^{j}_{n-i+1,n+1}$ (since the i^{th} highest rank is worker j $(= 1, 2, ..., n)$ with probability $\frac{1}{n}$ and the i^{th} highest rank is assigned a task with a reward value in one of the i largest interval that by Theorem 7, guarantees a reward of at least $c^{j}_{n-i+1,n+1}$). Therefore, the total expected reward of the *randomized* DSSAP Ranking Algorithm is at least

$$\frac{\sum_{j=1}^{n}\sum_{i=1}^{n} c^{j}_{n-i+1,n+1}}{n}$$

3.4 Numerical Experiments

This section describes several numerical experiments. Each of the experiments reports the results of 10000 replications that compares the performance of the optimal policy with the expectation policy.

Definition 3. *The expectation policy ranks the workers by comparing their expected success rates and follows the assignment policy described in Theorem 1 (i.e., instead of fixed success rates p_i, the workers are ranked by the expected success rate values).*

3.4.1 DSSAP with one worker

The first experiment illustrates the performance of the optimal policy in Theorem 2. Let n denote the total number of sequentially arriving tasks (i.e., secretaries) and assume that there is a single worker with a random success rate, taking new values with each task arrival. The objective is to find the optimal task to assign to the worker (i.e., hiring the best secretary). Assume that tasks are distributed IID $U(0, 1)$. The expectation policy computes the optimal interval thresholds $\{a_{i,n}\}$ for the uniform distribution and assigns the arriving task to the single worker if the realized value of the task is contained in $(a_{n-1,n}, a_{n,n}]$ (since it assumes that there are $n - 1$ workers with fixed

Table 3.4: The performance of the expectation and the optimal policies for DSSAP with one worker with a uniform success rate (Theorem 2)

Number of tasks (n)	$(E[R_{exp}(S_{1,n})], \sigma_{exp})$	$(E[R_{opt}(S_{1,n})], \sigma_{opt})$	$\frac{E[R_{opt}(S_{1,n})]}{E[R_{exp}(S_{1,n})]}$
5	(0.39, 0.25)	(0.48, 0.21)	1.24
10	(0.42, 0.26)	(0.57, 0.20)	1.36
50	(0.50, 0.28)	(0.79, 0.12)	1.61
100	(0.48, 0.28)	(0.84, 0.10)	1.75
200	(0.48, 0.29)	(0.88, 0.07)	1.84

Table 3.5: The performance of the expectation and the optimal policies for DSSAP with one worker with a Gamma success rate (Theorem 2)

Number of tasks (n)	$(E[R_{exp}(S_{1,n})], \sigma_{exp})$	$(E[R_{opt}(S_{1,n})], \sigma_{opt})$	$\frac{E[R_{opt}(S_{1,n})]}{E[R_{exp}(S_{1,n})]}$
5	(1.51, 1.58)	(1.96, 2.05)	1.30
10	(1.64, 1.66)	(2.44, 2.42)	1.49
50	(1.90, 1.96)	(3.89, 3.40)	2.05
100	(1.93, 2.04)	(4.82, 3.82)	2.50
200	(2.04, 2.19)	(5.22, 4.19)	2.56

success rates of zero). However, the optimal policy proposed in Theorem 2 computes the product distribution of the tasks and the random success rate, computes the corresponding interval thresholds ($\{c_{i,n}\}$), and assigns the arriving task to the worker if the task and the success rate's product is contained in $(c_{n-1,n}, c_{n,n}]$. The simulations for both policies are independent. Table 3.4 displays the sample average and sample standard deviation (σ) of the assigned reward for the expectation policy (denoted by subscript exp) and the optimal policy (denoted by subscript opt), when the random success rate follows a $U(0, 1)$ distribution. Table 3.5 provides results when the success rate follows a Gamma ($\theta = 2, k = 1$) distribution (i.e., the shape parameter k and the scale parameter θ, corresponding to an exponential random variable with mean $\theta = 2$).

As shown in Tables 3.4 and 3.5, the proposed optimal policy results in a larger average reward than the expectation policy in all experiments. In contrast to the expectation policy, which assigns a task to the worker based on the task's realized value, the optimal policy takes the product of the tasks and success rates values into account. Hence, the total average reward is increased. Moreover, larger improvements are achieved as the number of tasks increases.

Table 3.6: The performance of ranking the tasks by the expected value and the optimal policy for the SSAP with multiple arriving tasks (Theorem 4)

Number of tasks (n)	$(E[R_C(S_{1,n})], \sigma_C)$	$(E[R_{opt}(S_{1,n})], \sigma_{opt})$	$\frac{E[R_{opt}(S_{1,n})]}{E[R_C(S_{1,n})]}$
5	(1.58, 0.64)	(1.96, 0.54)	1.24
10	(3.19, 1.00)	(4.36, 0.78)	1.37
50	(16.21, 2.56)	(24.38, 1.93)	1.50
100	(32.54, 3.74)	(49.09, 2.88)	1.51
200	(65.70, 5.21)	(99.16, 4.08)	1.51

3.4.2 SSAP with multiple arriving tasks

The next experiment investigates the optimal policy proposed in Theorem 4 (SSAP with multiple arriving tasks). Assume that n IID $U(0, 1)$ tasks arrive simultaneously and suppose that there are n workers with fixed success rates to be assigned the tasks. The (fixed) success rate values are distributed uniformly $U(0, 1)$ and generated once for the experiment (i.e., they are not resampled for each replication). One of the n arriving tasks is selected and assigned to a worker, and the procedure with $n - 1$ tasks and workers continues. The optimal policy computes optimal interval thresholds ($\{m_{i,n}\}$) and assigns the arriving task to the worker with the i^{th} smallest success rate if the maximum realized value of the tasks is contained in the i^{th} smallest interval (Theorem 4). The optimal policy is compared with a policy that ranks the arriving tasks based on their expected values (which is a random selection in this example since all tasks have the same expected value). The sample average and sample standard deviation of the assigned rewards for this policy are denoted by $(E[R_C(S_{1,n})], \sigma_C)$. The simulations for the different policies are independent. The results are displayed in Table 3.6. The increase in the sample average reward is between 23% and 51% for various number of workers.

3.4.3 DSSAP with IID success rates

The next experiment illustrates the Greedy policy for the DSSAP with IID success rates (Theorem 5) and compares its performance with the expectation policy. The tasks are assumed to follow a $U(0, 1)$ distribution. The optimal policy assigns the arriving task to the worker with the maximum success rate at each stage. Table 3.7 provides the sample average reward and the

Table 3.7: The performance of the expectation and Greedy policies for the DSSAP with IID uniform random success rates (Theorem 5)

Number of tasks (n)	$(E[R_{exp}(S_{1,n})], \sigma_{exp})$	$(E[R_{Greedy}(S_{1,n})], \sigma_{Greedy})$	$\frac{E[R_{Greedy}(S_{1,n})]}{E[R_{exp}(S_{1,n})]}$
5	(1.26, 0.48)	(1.77, 0.52)	1.41
10	(2.48, 0.70)	(3.98, 0.80)	1.60
50	(12.47, 1.60)	(23.19, 1.96)	1.86
100	(24.90, 2.16)	(47.74, 2.82)	1.92
200	(50.04, 3.05)	(97.58, 3.82)	1.95

Table 3.8: The performance of the expectation and the Greedy policies for the DSSAP with IID exponential random success rates (Theorem 5)

Number of tasks (n)	$(E[R_{exp}(S_{1,n})], \sigma_{exp})$	$(E[R_{Greedy}(S_{1,n})], \sigma_{Greedy})$	$\frac{E[R_{Greedy}(S_{1,n})]}{E[R_{exp}(S_{1,n})]}$
5	(5.04, 2.94)	(6.97, 3.51)	1.38
10	(10.08, 4.06)	(16.10, 5.42)	1.60
50	(49.64, 9.32)	(92.29, 13.83)	1.86
100	(100.39, 12.90)	(192.00, 19.84)	1.91
200	(199.80, 18.52)	(390.25, 28.14)	1.95

sample standard deviation values for $U(0, 1)$ success rates and Table 3.8 provides results for exponentially distributed random success rates with mean two. The optimal policy results in a significant improvement in the sample average reward, with larger increase as n increases.

3.4.4 DSSAP algorithms

The last experiment illustrates the DSSAP Algorithm and compares the expected reward of the DSSAP Algorithm with the DSSAP Ranking and the randomized DSSAP Ranking Algorithms (which follows the same step as the DSSAP Ranking Algorithm, but ranks the workers randomly). Note that $E[R_{Rand}(S_{1,n})]$ denotes the sample average reward achieved by the randomized DSSAP Ranking Algorithm. The tasks are assumed to be IID $U(0, 1)$, and the i^{th} worker's success rate is assumed to have a Gamma $(\theta = i, k = 1)$ distribution. The simulations for the different policies are independent. The DSSAP Ranking Algorithm achieves a larger average reward than the randomized DSSAP Ranking Algorithm, and achieves more than 90% of the average reward achieved by the DSSAP Algorithm for $n = 5$ and $n = 10$ workers (Table 3.9). Due to the running time of the DSSAP Algorithm, its average reward for larger number of

Table 3.9: The performance of the DSSAP, the DSSAP Ranking, and the randomized DSSAP Ranking Algorithms for Uniform task and Gamma random success rates

n	$E[R_D(S_{1,n})]$	$E[R_{DR}(S_{1,n})]$	$E[R_{Rand}(S_{1,n})]$	$\frac{E[R_{DR}(S_{1,n})]}{E[R_D(S_{1,n})]}$	$\frac{E[R_{Rand}(S_{1,n})]}{E[R_D(S_{1,n})]}$
5	22.76	22.32	18.50	0.98	0.81
10	99.47	94.50	74.92	0.95	0.75
20	-	405.99	301.51	-	-
50	-	2808.6	1972.5	-	-
100	-	12382	8257.8	-	-

workers is not reported. However, the DSSAP Ranking Algorithm can be used for larger number of workers with the sample average reward values shown in Table 3.9. Note that the value of $\sum_{j=1}^{n} c_{n,n+1}^{(j)}$ which is an upper bound on the maximum total expected reward (as discussed in Theorem 7) for $n = 5, 10, 20, 50, 100$ is $27.30, 134.31, 647.43, 5044.3, 23386$, respectively.

3.5 Summary

This chapter introduces the Doubly Stochastic Sequential Assignment Problem (DSSAP), an extension of the SSAP where the workers' success rates are random. As a generalization of SSAP, DSSAP has applications in several areas including online matching markets, revenue management, asset selling, and organ transplant. The main difference between SSAP and DSSAP is the number of random parameters observed at each stage of the problem. While the task value is the only random parameter in SSAP, both task and the workers' success rates are stochastic in DSSAP. This makes the problem of finding an optimal assignment policy for DSSAP very challenging.

Optimal assignment policies for several special cases are proposed. The optimal algorithm for DSSAP with one worker transforms the randomness in workers' success rates to task values by defining a surrogate problem and applies the optimal policy of SSAP to the surrogate problem. The Greedy Algorithm is proven to be optimal for DSSAP with IID success rates.

The optimal policy for DSSAP with possibly different success rate distributions (referred to as the DSSAP Algorithm) is a computational approach that uses backward induction to find the optimal assignment at each stage by first solving smaller subproblems. The DSSAP Algorithm computes the expected reward for each possible assignment in the n-stage problem and hence, suffers from an exponential running time. The DSSAP Ranking Algorithm is an efficient approximation algorithm that achieves a fraction of the maximum total expected reward in a polynomial time. The DSSAP Ranking Algorithm assumes assignment priority for workers with larger expected success rates. The algorithms assigns the arriving task to the worker with the i^{th} largest expected success rate if the task is not assigned to a worker with a higher priority, and the reward achieved by this assignment is in one of the best i intervals computed for the worker. The proposed policies are illustrated with several numerical experiments.

CHAPTER 4

GENERALIZED SEQUENTIAL ASSIGNMENT PROBLEM

4.1 Introduction

This chapter relaxes the assumption that task values are independently drawn from a known distribution. SSAP without any prior information on task values is referred to as the Generalized SSAP (GSSAP). First, the relation between the Secretary Problem and GSSAP is described.

The Secretary Problem finds an optimal policy for selecting the best element among a known number of sequentially arriving objects [23, 42]. The main challenge in the Secretary Problem is that the decision-maker must make an irrevocable decision by comparing the relative rank of the candidates interviewed so far, without any information on the quality of the next (future) arriving candidates. Similar to the SSAP, the optimal policy of the Secretary Problem has a threshold structure: A number of candidates are interviewed and the quality of the best interviewed candidate is set as a threshold. Then, the first candidate with a quality greater than the threshold is hired. While the optimal thresholds of the SSAP are computed prior to the assignment process using the distribution of the task values, there is a training phase in the Secretary Problem to set the threshold since there is no prior information on the secretary quality values.

There are two differences between the SSAP and the Secretary Problem. First, while the selected elements in the Secretary Problem are equivalent and in terms of an assignment problem, are assigned to similar positions, each arriving task in the SSAP is assigned to a worker with a distinct success rate value. SSAP can be considered as a Secretary Problem with n secretaries and n employers, where the secretaries must be assigned (matched) to the employers such that the total reward is maximized. Second, while one element or a subset of elements (smaller than the total number of arriving elements) are selected in the Secretary Problem, the number of sequentially arriving tasks and available workers are assumed to be equal for the SSAP.

Therefore, the assignment process for the SSAP begins with the arrival of the first task and there is no training phase during which tasks can be discarded. Note that the equal number of tasks (which are the *online* elements that arrive sequentially and must be assigned upon arrival) and workers (which are the *offline* elements that are known prior to the assignment process) in the SSAP is the most general case in an online matching problem. If the number of tasks n is larger than the number of workers m, then $n - m$ auxiliary workers with zero success rate can be created. If the number of tasks is less than the number of workers, the $m - n$ workers with smallest success rates can be discarded.

The similarities between the SSAP and the Secretary Problem can be used to propose policies for one problem using the other problem. In one direction, the SSAP is used to derive optimal policies for the *full information* Secretary Problem, where the quality values of the sequentially arriving elements are independently drawn from a known distribution. In the other direction, the Secretary Problem is used to generalize the SSAP by relaxing the assumption that task values come from a known distribution. The Weighted Secretary problem [7] is used to design two deterministic and one randomized assignment policies for GSSAP, and an assignment policy for GSSAP with a time-dependent reward function. While the deterministic algorithms provide slightly better lower bounds on the expected reward, the randomized algorithm is *incentive compatible*: It assigns each task to any of the workers with equal probability. This provides fairness since the sequentially arriving elements do not have an incentive to appear in specific stages to be assigned to better workers.

As mentioned previously, SSAP has applications in several areas, including the organ transplant, revenue management, and asset selling problem [59, 35]. While assuming that task values are independently drawn from a known distribution is useful for applications with a large historical data set, it is not a realistic assumption in some other applications. This chapter relaxes this assumption and proposes assignment policies with a guaranteed lower bound on the expected reward, which can be used in real applications with no prior information on the sequentially arriving elements. Babaioff et al. [8] present the Secretary Problem as a framework for online auctions. Considering the GSSAP as a generalization of the Secretary Problem, the GSSAP studies the problem of matching a set of bidders with a number of available items in an online auction, where bids correspond to the tasks and items correspond to the workers.

The remainder of this chapter is organized as follows. Section 4.2 describes several extensions of the Secretary Problem and SSAP and discusses the relation between the two problems. Section 4.3 proposes the assignment policies for GSSAP using the Weighted Secretary Problem. Section 4.4 uses the Weighted Discounted Secretary Problem to obtain an algorithm for the GSSAP with a time-dependent reward function. Section 4.5 briefly discusses the GSSAP with general reward functions. Section 4.6 provides a summary of the results and concluding comments.

4.2 The Secretary Problem and The Sequential Stochastic Assignment

There are n sequentially arriving secretaries interviewed one by one in the Secretary Problem. Once a secretary is interviewed, its relative quality (i.e., the quality compared to secretaries interviewed in the previous stages) is observed. An irrevocable decision, either to hire or to reject, is made at each stage. Once rejected, a secretary can not be recalled. Based on the objective and the assumptions, four different types of the Secretary Problem can be defined: The objective is maximizing either the probability of hiring the best secretary or the expected quality of the hired secretary. The quality values are assumed either to come independently from a known distribution (the *full information* Secretary Problem) or to have a random arrival order (without any assumption on the values).

The optimal policy of the Secretary Problem with no assumptions on the quality values and the objective to maximize the probability of hiring the best secretary divides the arriving elements into two sets. First, a set of secretaries are interviewed without anyone being hired. These secretaries are referred to as the *training set*, with a threshold defined as the quality of the best secretary in the training set. In the second phase, the first secretary that has a quality above the threshold is hired. It can be shown that as $n \to +\infty$, the optimal number of secretaries in the training phase approaches $\frac{n}{e}$ and the probability of hiring the best secretary approaches $\frac{1}{e}$ [25].

The optimal policy of the Secretary Problem with no assumptions on secretaries quality values and the objective to maximize the expected quality of the hired secretary (or equivalently, minimize the expected rank of the hired secretary) is of the following form: Hire the r^{th} secretary if their relative rank s (i.e., their rank among the secretaries interviewed so far) satisfies $s \leq s^*(r)$, where $s^*(r)$ is a function of the stage number that computes the optimal relative rank. The intuition is that after a few interviews of not observing a secretary of relative rank 1, the best secretary has been rejected in the training phase with a large probability. Therefore, since the objective is to minimize the expected rank of the hired secretary (and not maximizing the probability of hiring the best secretary), a secretary of relative rank 2 is hired, and so on. [15] prove that as $n \to +\infty$, the absolute rank of the secretary hired by the optimal policy tends to

$$\prod_{j=1}^{\infty}(\frac{j+2}{j})^{1/j+1} \cong 3.8695$$

Enns [24] derives the optimal policy for the full information Secretary Problem with the objective to maximize the probability of hiring the best secretary and proves that the probability of obtaining the maximum value is independent of the distribution.

Consider the full information Secretary Problem with the objective to maximize the expected quality of the hired secretary $E[x_1]$, where x_1 denotes the quality value of the hired secretary. This problem is a special case of the SSAP [19]: Assume that n tasks with associated random values coming independently from a known distribution arrive sequentially, and there is one worker with a fixed success rate $p_1 = 1$ (and $n-1$ workers with success rate of $p_2 = p_3 = ... = p_n = 0$) to be assigned one of the tasks [36]. The goal is to maximize the expected reward of the assignment, which is given by

$$E[\sum_{j=1}^{n} x_{i_j} p_j] = E[x_{i_1} p_1] = E[x_{i_1}], \tag{4.1}$$

where x_{i_1} denotes the task value assigned to the single available worker. This problem is equivalent to the full information Secretary Problem since both problems maximize the expected quality of the selected elements. The optimal policy of SSAP determines the optimal policy for the Secretary Problem: n optimal intervals are defined using Corollary 1 and the task is assigned to the worker with success rate $p_1 = 1$ (or equivalently the secretary is hired) if the task (secretary) value is in the highest interval $x \in (a_{n-1,n}, a_{n,n}]$ (as in Theorem 1).

Several extensions of the Secretary Problem and the SSAP are discussed in Sections 4.2.1 to

4.2.5. The optimal policies of the SSAP extensions can be used to find the optimal policy for the corresponding full information Secretary Problem when the objective is to maximize the expected quality of the hired secretaries.

4.2.1 The Multiple Choice Problem

The Multiple Choice Secretary Problem selects a subset of k secretaries so as to maximize either the expected sum of their qualities [38], or the probability of selecting the k best secretaries [27]. Another version of the Multiple Choice Secretary Problem seeks to maximize the probability of hiring the best secretary among the k choices [57].

The full information Multiple Choice Secretary Problem maximizes $E[\sum_{i=1}^{k} x_i]$, where k is the number of secretaries to be hired and x_i denotes the quality value of the i^{th} hired secretary. The optimal policy of the SSAP can be used to maximize the expected sum of the qualities in the full information Multiple Choice Secretary Problem [36]. Assume that the workers success rates are $p_i = 1$ for $i = 1, 2, ..., k$ and $p_i = 0$ for $i = k + 1, k + 2, ..., n$. Then, Theorem 1 determines the optimal policy for the SSAP that maximizes

$$E[\sum_{j=1}^{n} x_{i_j} p_j] = E[\sum_{j=1}^{k} x_{i_j} p_j] = E[\sum_{j=1}^{k} x_{i_j}], \tag{4.2}$$

which is the same as the objective function of the Multiple Choice Secretary Problem. The optimal policy hires the arriving secretary if their quality value is in one of the k highest intervals defined by Corollary 1, $x \in (a_{n-k,n}, a_{n,n}]$, with x denoting the quality value, and n denoting the total number of candidates.

4.2.2 Uncertain Employment

Smith [58] studies the Secretary Problem with Uncertain Employment, where each secretary refuses an employment offer with a fixed probability. The objective is to maximize the probability of hiring the best secretary. Tamaki [60] studies the same problem under two assumptions; one model assumes that the availability of a secretary is revealed when the employment offer is made and the other model assumes that the availability is revealed only upon secretary arrival. The optimal policy of the Secretary Problem with Uncertain Employment has a training phase to set a threshold and a selection phase to hire the first secretary with a value above the threshold [58].

SSAP with random success rates (DSSAP), which is described in the previous chapter, can be used to solve a problem similar to the full information Secretary Problem with Uncertain Employment. Assume that the secretary quality values (x) are independently drawn from a known distribution. Moreover, assume that there is an availability probability (q) revealed upon each secretary interview, which are also independently drawn from a known distribution. The objective is to maximize the expected quality of the hired secretary, which is defined as $E[x \times q]$ (since if a secretary with quality x is selected, it has an availability probability of q). This problem is a special case of the DSSAP with n tasks and one worker with a random success rate q. The optimal policy computes a threshold using the product distribution of task values and the random success rate and assigns the first arriving task to the single worker (i.e., hires the first secretary) in the n-stage problem if $x \times q \in (c_{n-1,n}, c_{n,n}]$ [35].

4.2.3 Time Discounting

Rasmussen and Pliska [53] study the Secretary Problem with a discount factor α. The reward of hiring the best secretary if she appears at stage i is given by α^i. If the hired secretary is not the best one, then the achieved reward is zero. Rasmussen and Pliska show that the optimal policy has a threshold structure: The first $r^* - 1$ secretaries are interviewed without hiring anyone. Then, starting from stage r^*, the first secretary that is better than any previously interviewed ones is hired. They derive an expression for the optimal value of r^*.

Albright [3] studies the SSAP by assuming that tasks arrive according to a non-homogeneous Poisson process with a continuous intensity function $\lambda(t)$, and the reward of assigning the task with value x at time t to a worker with success rate p_i is defined as $x \times r(t) \times p_i$. The discount function $r(t)$ is assumed to be a piecewise continuous, non-negative, and non-increasing function with $r(0) = 1$. Albright first discusses the problem with $p_1 = p_2 = ... = p_n = 1$. The optimal policy when there are n remaining workers defines a set of time-dependent *critical curves* $y_i(t)$, $i = 1, 2, ..., n$ such that the arriving task is assigned to a worker if $x > y_n(t)$. The critical curves satisfy the following integral equation:

$$r(t)s_n(t) = \int_t^\infty [\lambda(\tau)r(\tau)H(y_n(\tau)) + \lambda(\tau)\bar{F}(y_n(\tau))r(\tau)s_{n-1}(\tau)]exp(-\int_t^\tau \lambda(\beta)\bar{F}(y_n(\beta))d\beta)d\tau,$$

where $s_n(t) = \sum_{i=1}^n y_i(t)$, $H(y) = \int_y^\infty xF(dx)$, and $\bar{F}(y) = 1 - F(y)$. The optimal policy can be used to maximize the expected sum of the secretary qualities in a full information Discounted Secretary Problem where the time discounting is a non-increasing function of time. This is a more general assumption than the Discounted Secretary Problem studied by [53], since the discount function is a continuous function of time. Assuming that the workers success rates are $p_1 = 1$ and $p_i = 0$ for $i = 2, 3, ..., n$, the optimal policy defines a critical curve $y_1(t)$ such that the current secretary is hired if their quality value is larger than $y_1(t)$.

4.2.4 Infinite Problem

Gianini and Samuels [29] study the Infinite Secretary Problem. They assume that an infinite number of secretaries arrive sequentially at $IID\ Uniform\ (0,1)$ times. The relative rank of the secretaries is observed, with rank 1 denoting the best secretary. They seek a policy to select one secretary such that the objective, which is defined as a positive increasing function of the actual rank, is minimized. The optimal policy computes a set of time break points $0 < t_1 \leq t_2 \leq ... \leq 1$ such that the first secretary arriving in the interval $[t_s, t_{s+1})$ is hired if their relative rank is s or better.

Albright and Derman [2] investigate the limiting behavior of the optimal intervals as the number of assignments approaches infinity. They show that if the CDF of the task values F is absolutely continuous and the task's values have finite means, the optimal interval thresholds of the infinite SSAP can be computed as

$$\lim_{n\to\infty} a_{[n\pi],n+1} = F^{-1}(\pi), \tag{4.3}$$

where $0 < \pi < 1$. Therefore, the optimal policy for the full information Secretary Problem with infinite number of candidates is to hire the arriving secretary if their quality value falls in the highest interval computed as (4.3).

4.2.5 Random Number of Elements

The Secretary Problem with a random number of candidates assumes that the number of sequentially arriving candidates is random. Presman and Sonin [52] study the problem with the objective to maximize the probability of selecting the best element. The optimal policy computes a set of (possibly) discontinuous stopping points, referred to as *islands* (i.e., while the optimal policy of the Secretary Problem hires the first relatively best candidate in stages $k, k+1, ..., n$, the optimal policy of the Secretary Problem with a random number of candidates is of the form "hire the first relatively best candidate in stages $\{k_1, k_1+1, ..., k_2\} \cup \{k_2+z, k_2+z+1, ..., k_3\} \cup ...$", with $z \geq 0$ a constant that depends on the distribution of the number of tasks).

Gianini-Pettitt [30] studies the same problem with the objective to minimize the expected rank of the selected individual. The optimal policy hires the r^{th} arriving secretary if their relative rank is less than $s(r)$, where s is a function that depends on the probability distribution of the number of elements.

Oveis Gharan and Vondrak [51] study the Secretary Problem with an Unknown Number of Candidates. They show that if the number of elements is selected by an adversary from $\{1, 2, ..., N\}$, then there exists a randomized algorithm that finds the best secretary with probability no less than $1/(H_{N-1}+1)$, where $H_K = \sum_{i=1}^{K} \frac{1}{i}$ is the K^{th} harmonic number. They further show that there is no algorithm that hires the best secretary with probability more than $1/H_N$.

Nikolaev and Jacobson [49] consider SSAP with a random number of tasks. They assume that the distribution of the number of tasks is known. To find the optimal policy, they define a surrogate problem as the SSAP with a fixed number of tasks N_{max}, where N_{max} is the maximum possible number of tasks. The task values in the surrogate problem (x') are computed using the distribution of the number of tasks: If $x'_{j-1} = 0$, then $x'_j = 0$. Otherwise, $x'_j = x_j$ with probability $\sum_{i=j}^{N_{max}} P_i / \sum_{i=j-1}^{N_{max}} P_i$ and $x'_j = 0$ with probability $P_{j-1} / \sum_{i=j-1}^{N_{max}} P_i$. Then, they use the SSAP with task values coming from not necessarily independent random variables [33] to find the optimal policy of the surrogate problem, which is used to derive the optimal assignment of the SSAP with a random number of tasks. The optimal policy has the same structure as the optimal policy of the SSAP: A set of thresholds are defined recursively and the n^{th} arriving task is assigned to the worker with the m^{th} largest success rate value if the task value (x'_n) is contained in the m^{th} highest interval. Assuming that the distribution and the maximum number of tasks are known, the optimal policy of the SSAP with $p_1 = 1$ and $p_i = 0$ for $i = 2, 3, ..., n$, can be used to find the optimal policy of the full information Secretary Problem with a random number of candidates.

4.3 Algorithms for Generalized SSAP Using the Weighted Secretary Problem

This section uses the Weighted Secretary Problem to obtain assignment policies for the Generalized SSAP (GSSAP) without any prior information on the task values. Babaioff et al. [7] study the Weighted Secretary Problem, where up to K secretaries are selected and assigned irrevocably to K positions with weights $w_1 \geq w_2 \geq ... \geq w_K$ such that $\sum_{i=1}^{K} x_i w_i$ is maximized, where x_i is the value of the secretary assigned to position i, and $x_i = 0$ if position i is not filled. They propose the *Interval Reservation Algorithm*, which observes the first $l = \lfloor \frac{n}{2} \rfloor$ elements (referred to as the *training set*) and computes a set of thresholds to assign secretaries to the positions. The intervals are defined as $I_i = (\hat{x}_i, \hat{x}_{i-1})$ for $i > 1$ and let $I_1 = (\hat{x}_1, \infty)$, where $\hat{x}_i$ is the i^{th} largest quality value in the training set. Upon arrival of a secretary with value x_e from the remaining $n - l$ secretaries (hereafter called the *selection set*), the secretary is assigned to position m with smallest index such that $m \geq m_e$, where m_e is such that $x_e \in I(m_e)$.

To analyze the Interval Reservation Algorithm, Babaioff et al. [7] propose an algorithm, referred to as Algorithm B, which follows the same steps as the Interval Reservation Algorithm, but assigns secretary e to position m_e if the position is not yet filled. Otherwise, secretary e is not assigned to a position. Note that the expected value achieved by the Interval Reservation Algorithm is greater than or equal to the one achieved by Algorithm B since each position is filled in the Interval Reservation Algorithm by a secretary with a value at least as large as the secretary filling the same position in Algorithm B [7]. Therefore, the competitive ratio achieved by the Interval Reservation Algorithm is at least as good as the competitive ratio achieved by Algorithm B. This result is formalized as Lemma 9.

Lemma 2. *[7] The expected weighted value achieved by the Interval Reservation Algorithm is at least that achieved by Algorithm B.*

Babaioff et al. [7] prove a $4-$competitive ratio for Algorithm B. Then, as formalized in Theorem 9, they conclude that the Interval Reservation Algorithm is 4-competitive.

Theorem 9. *[7] Algorithm B is 4-competitive. Therefore, by Lemma 9, the Interval Reservation Algorithm is 4-competitive.*

The Weighted Secretary Problem is equivalent to the SSAP with no prior information on task values (i.e., the GSSAP), where secretaries correspond to the sequentially arriving tasks and positions (weights) correspond to workers (success rates). In the GSSAP, the decision-maker assigns each arriving task to one of the available workers starting at the first stage (i.e., the number of workers is equal to the number of arriving tasks). Therefore, while the Interval Reservation Algorithm does not make any assignments in the training phase, an assignment policy for the GSSAP must start the assignment process from the first stage. Note that discarding a set of tasks in the GSSAP is not possible for some applications, such as organ transplant [59]. Moreover, maximizing the number of assignments in the GSSAP is equivalent to maximizing the number of matchings in an online matching problem, which is necessary for customer satisfaction. Sections 4.3.1, 4.3.2, and 4.3.3 propose three different methods to divide workers into groups recursively and assign tasks to the workers of each group using the Interval Reservation Algorithm. While the first two algorithms achieve a 4-competitive ratio, the third algorithm is a 6-competitive randomized algorithm that assigns each task to any of the workers with equal probability.

4.3.1 The Recursive Interval Reservation Algorithm (RIRA)

The *Recursive Interval Reservation Algorithm* (RIRA) recursively applies the Interval Reservation Algorithm to groups of workers: The first group of workers includes the $l = \lfloor \frac{n}{2} \rfloor$ workers with the smallest success rates, which are assigned the tasks of the training set (i.e., the first l tasks), and the second group includes workers with the $n - l$ largest success rates, which are assigned tasks in the selection set (i.e., the last $n - l$ tasks). The tasks in the training set are assigned to the workers by recursively applying the Interval Reservation Algorithm to the training set (i.e., workers of the first group are again divided into two groups, with the first $\lfloor \frac{l}{2} \rfloor$ tasks assigned to the group with the smallest success rates and the last $l - \lfloor \frac{l}{2} \rfloor$ tasks assigned to the group with the largest success rates, and so on). Similar to the Interval Reservation Algorithm, RIRA assigns the task with a value in the k^{th} highest interval to the worker $p_{(l)}$ if l is the smallest index such that $l \geq k$. If there is no such worker, then the task is assigned to the worker with success rate $p_{(l)}$ such that l is the largest index with $l < k$. This guarantees that each task is assigned to a worker. The intuition behind the Recursive Interval Reservation Algorithm is that less skilled workers (i.e., workers with smaller success rates) are assigned tasks without a training data set (or more precisely, with a smaller training set), and more skilled workers, which have a larger effect on the total reward, are assigned tasks after a (larger) training phase.

The Recursive Interval Reservation Algorithm is formalized as Algorithm 3, with x_j denoting the j^{th} arriving task's value, and $x_{(i)}$ $(p_{(i)})$ denoting the i^{th} largest task (success rate) value. There are $s = \lfloor log(n) \rfloor + 1$ rounds of dividing the workers into two groups and assigning the tasks to the second group of workers using the thresholds defined by the tasks assigned to the first group. The first part of the algorithm recursively divides the workers into groups, saved in the rows of matrix B. The first group of workers A is divided into two groups in the next iteration. The second group $B(j)$, which includes the workers with the largest success rates, are assigned tasks in the $s - j + 1^{th}$ round. For example, in the GSSAP with four workers with success rates $p_{(4)} \leq p_{(3)} \leq p_{(2)} \leq p_{(1)}$, there are $s = \lfloor log(4) \rfloor + 1 = 3$ rounds of dividing the workers. The first round divides the workers into two groups, with the first group $A_{new} = \{p_{(3)}, p_{(4)}\}$ (which is divided in the next iteration), and the second group $B(1) = \{p_{(1)}, p_{(2)}\}$ (which are assigned the last two tasks). The next round divides A_{old} (which is A_{new} of the previous iteration) into two group, with $A_{new} = \{p_{(4)}\}$ and $B(2) = \{p_{(3)}\}$. The third round includes the worker with the smallest success rate $B(3) = \{p_{(4)}\}$.

Algorithm 3 The Recursive Interval Reservation Algorithm (RIRA)

```
$RIRA\ (n, P = \{p_i | i \in \{1, 2, ..., n\}\})$

$A_{old} = P$
$s = \lfloor log(n) \rfloor + 1$
for j=1 to s
    $m = |A_{old}|$
    if $m > 1$
        $A_{new} = \{p_{(i)} \in A_{old} | i \in \{m - \lfloor m/2 \rfloor + 1, m - \lfloor m/2 \rfloor + 2, ..., m\}\}$
        $B(j) = A_{old} - A_{new}$
        $A_{old} = A_{new}$
    else
        $B(j) = A_{old}$
    end if
end for
$I_1 = (-\infty, +\infty)$
for j=1 to s
    $r = |B(s - j + 1)|$
    Assign the next $r$ sequentially arriving tasks to workers with success rates $p_{(l)} \in B(s - j + 1)$:
    if $x_i \in I_k$
        Assign task $i$ to worker with success rate $p_{(l)}$ with $l$ the smallest index satisfying $l \geq k$
        if there is no such worker
            Assign task $i$ to worker with success rate $p_{(l)}$ with $l$ the largest index satisfying $l < k$
        end if
    end if
    Update the intervals using the $r$ tasks assigned:
    $I_k = (x_{(k)}, x_{(k-1)})$ for $k = 1, 2, ..., n + 1$, with $x_{(n+1)} = -\infty$ and $x_{(0)} = +\infty$
end for
```

The second part of the algorithm assigns the sequentially arriving tasks to the groups of workers obtained in the first part. It updates the optimal intervals using the tasks assigned at each round. In the above example, $B(s) = B(3) = \{p_{(4)}\}$ includes one worker, which is assigned the first arriving task. Then, the interval thresholds are updated using the value of the first arriving task. However, since $B(2) = \{p_{(3)}\}$ includes only one worker, the second task is assigned to this worker. The third round assigns the next two tasks to the workers in $B(1)$ using the thresholds defined by the first two task values.

Lemma 3. *The Recursive Interval Reservation Algorithm achieves a reward strictly larger than the Interval Reservation Algorithm for $K > n - \lfloor \frac{n}{2} \rfloor$ and achieves the same reward for $K \leq n - \lfloor \frac{n}{2} \rfloor$, where K is the number of available workers.*

Proof: Follows directly from the definition of the Recursive Interval Reservation Algorithm.

Theorem 10. *The Recursive Interval Reservation Algorithm is 4-competitive.*

Proof: Follows directly from Theorem 9 and Lemma 3.

Lemma 4. *The reward achieved by the Recursive Interval Reservation Algorithm for the problem with n tasks and workers is at least $\frac{1}{4}\sum_{i=n-l+1}^{n} x_{(i)}p_{(i)}$ larger than the reward achieved by the Interval Reservation Algorithm.*

Proof: Let $\hat{x}_i$ denote the task assigned to position i and R_{IRA} denote the total reward achieved by the Interval Reservation Algorithm. Since the Interval Reservation Algorithm observes the first l secretaries without assigning any of them, the total reward it achieves is given by

$$R_{IRA} = \sum_{i=1}^{n} \hat{x}_i p_{(i)} = \sum_{i=1}^{n-l} \hat{x}_i p_{(i)}. \tag{4.4}$$

Let x_i denote the task value assigned to position i and R_{RIRA} denote the total reward achieved by the Recursive Interval Reservation Algorithm. Then,

$$R_{RIRA} = \sum_{i=1}^{n} x_i p_{(i)} = \sum_{i=1}^{n-l} x_i p_{(i)} + \sum_{i=n-l+1}^{n} x_i p_{(i)} = \sum_{i=1}^{n-l} \hat{x}_i p_{(i)} + \sum_{i=n-l+1}^{n} x_i p_{(i)} = R_{IRA} + \sum_{i=n-l+1}^{n} x_i p_{(i)}. \tag{4.5}$$

Therefore,

$$R_{RIRA} - R_{IRA} \geq \sum_{i=n-l+1}^{n} x_i p_{(i)}. \tag{4.6}$$

Let $x^*_{(j)}$ denote the j^{th} largest task value among the first l tasks (i.e., the training set). By Theorem 9,

$$\sum_{i=n-l+1}^{n} x_i p_{(i)} \geq \frac{1}{4} \sum_{i=1}^{l} x^*_{(i)} p_{(n-l+i)}. \tag{4.7}$$

Since $x^*_{(j)} \geq x_{(n-l+j)}$ for $j = 1, 2, ..., l$ (i.e., the best task among the first l tasks is at least as good as the $(n-l+1)^{th}$ task among all n tasks and so on),

$$\sum_{i=n-l+1}^{n} x_i p_{(i)} \geq \frac{1}{4} \sum_{i=n-l+1}^{n} x_{(i)} p_{(i)}, \tag{4.8}$$

which together with (4.6) completes the proof. □

4.3.2 The Determinstic Dividing Algorithm (DDA)

The i^{th} largest task value has the same probability of appearing in the first and second half of the sequentially arriving tasks in a random arrival model. Therefore, assigning all workers with larger success rates in the second half of the assignment process decreases the expected number of assignments, where the i^{th} largest task value is assigned to the i^{th} largest success rate (as in the optimal offline assignment). Similar to the Recursive Interval Reservation Algorithm, the *Deterministic Dividing Algorithm* (DDA) recursively divides the workers into two groups and applies the Interval Reservation Algorithm to each group. However, while the Recursive Interval Reservation Algorithm divides the workers into a group with the smallest success rates and a group with the largest success rates, the Deterministic Dividing Algorithm divides the workers into two groups of approximately the same success rate values: The tasks in the training set, in the $n-$stage problem, are assigned to workers with success rates $\{p_{(2k)} | 2k \in \{1, 2, ..., n\}\}$, and the selection set's tasks are assigned to workers with success rates $\{p_{(2k+1)} | 2k+1 \in \{1, 2, ..., n\}\}$. This policy increases the expected number of assignments, where the i^{th} largest task and success rate values are matched. Theorem 11 proves that the Deterministic Dividing Algorithm, which is formally presented as Algorithm 4, is 4-competitive.

Theorem 11. *The Deterministic Dividing Algorithm is* 4*-competitive.*

Proof: Let $p_{(n)} \leq p_{(n-1)} \leq ... \leq p_{(1)}$ denote the workers success rates and $x_{(n)} \leq x_{(n-1)} \leq ... \leq x_{(1)}$ denote the task values. The Deterministic Dividing Algorithm assigns a task to the worker with the largest success rate $p_{(1)}$ exactly in the same manner as the Interval Reservation Algorithm. Therefore, in expectation, the reward of assigning a task to this worker by the Deterministic Dividing Algorithm is at least $\frac{1}{4}x_{(1)}p_{(1)}$.

The same procedure as in the proof of $4-$competitiveness for the Interval Reservation Algorithm is used to prove that the expected reward that the Deterministic Dividing Algorithm achieves by assigning a task to each of the other workers is at least $1/4$ times the maximum offline reward. Define Algorithm B in the same way as discussed in Lemma 9: If $x_j \in I_k$, Algorithm B assigns the task to worker with success rate $p_{(k)}$ if and only if the worker has not already been assigned, and discards the task otherwise [7]. It will be proven that Algorithm B achieves at least $1/4$ of the optimal offline reward. Together with Lemma 9, this result proves that the expected reward achieved by the Deterministic Dividing Algorithm is at least $1/4$ of the optimal offline reward. The Deterministic Dividing Algorithm assigns a task to the worker with success rate $p_{(2)}$ in one of stages $z = \lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor + 1, \lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor + 2, ..., \lfloor\frac{n}{2}\rfloor$. Given that the task $x_{(2)}$ or a task with a larger value (i.e., $x_{(1)}$) appears in a random order at position z, then with probability $\frac{\lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor}{z-1}$ the next most valuable task (i.e., the task with the next largest value) that appeared before $x_{(2)}$ (or $x_{(1)}$), appears in the training set (i.e., in one of the stages $1, 2, ..., \lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor$). Conditioning on this event, with probability $\frac{\lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor-1}{z-2}$ the next less valuable task (i.e., the task with the next smallest value) that appeared before $x_{(2)}$ (or $x_{(1)}$), appears in the training set. These two events are sufficient conditions that a task with a value at least as large as $x_{(2)}$ is assigned to the worker with success rate $p_{(2)}$ by Algorithm B [7]. The probability that the task in position z has a value at least as large as $x_{(2)}$, is $\frac{2}{n}$ (since this is equal to the probability that the task has the value $x_{(1)}$ or $x_{(2)}$). Therefore, the probability that the worker with success rate $p_{(2)}$ is assigned to a task with a value at least as large as $x_{(2)}$ by Algorithm B is given by

$$\sum_{z=\lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor+1}^{\lfloor\frac{n}{2}\rfloor} \frac{2}{n} \times \frac{\lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor(\lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor - 1)}{(z-1)(z-2)} = \frac{2}{n} \times \lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor(\lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor - 1) \times \sum_{z=\lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor+1}^{\lfloor\frac{n}{2}\rfloor} (\frac{1}{z-2} - \frac{1}{z-1})$$

$$= \frac{2}{n} \times \lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor(\lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor - 1) \times (\frac{1}{\lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor - 1} - \frac{1}{\lfloor\frac{n}{2}\rfloor - 1}) = \frac{2}{n} \times \lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor \times \frac{\lfloor\frac{n}{2}\rfloor - \lfloor\frac{1}{2}\lfloor\frac{n}{2}\rfloor\rfloor}{\lfloor\frac{n}{2}\rfloor - 1},$$

which is at least $\frac{1}{4}$, for $n \to \infty$. Therefore, in expectation, the reward achieved by the worker with success rate $p_{(2)}$ using the Deterministic Dividing Algorithm is at least $\frac{1}{4}$ times the reward achieved by the same worker in the optimal assignment.

Algorithm 4 The Deterministic Dividing Algorithm

```
$DDA\ (n, P = \{p_i | i \in \{1, 2, ..., n\}\})$

$A_{old} = P$
$s = \lfloor log(n) \rfloor + 1$
for j=1 to s
   $m = |A_{old}|$
   if $m > 1$
      $A_{new} = \{p_{(i)} \in A_{old} | i = 2k\}$
      $B(j) = A_{old} - A_{new}$
      $A_{old} = A_{new}$
   else
      $B(j) = A_{old}$
   end if
end for
Follow the assignment process as RIRA
```

To prove that the same competitive ratio holds for the reward achieved by the other workers, note that $x_{(i)}p_{(i)} \geq x_{(i+1)}p_{(i+1)}$ for $i = 3, 5, 7, ...$. Therefore, the reward achieved by the workers with success rates $p_{(3)}, p_{(5)}, p_{(7)}, ...,$ is at least half of the remaining reward and hence, proving a $2-$competitive ratio for the reward achieved by these workers is equivalent to proving a $4-$competitive ratio for the total reward achieved by the workers with success rates $p_{(3)}, p_{(4)}, p_{(5)}, ..., p_{(n)}$.

Consider the worker with success rate $p_{(k)}$ with $\{k = 2l + 1 | 2l + 1 \in \{3, ..., n\}\}$. The probability that the task in position z has a value at least as large as $x_{(k)}$, is at least $\frac{2}{n}$ (since this is equal to the probability that the task has a value $x_{(k)}$ or $x_{(k+1)}$). Therefore, following the same steps as (4.9), for stages $z = \lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, ..., n$, the probability that the worker with success rate $p_{(k)}$ $(k = 3, 5, 7, ...)$ is assigned to a task with a value at least as large as $x_{(k)}$ is given by

$$\frac{2}{n} \times \lfloor \frac{n}{2} \rfloor \times \frac{n - \lfloor \frac{n}{2} \rfloor}{n-1},$$

which is at least $\frac{1}{2}$ as $n \to +\infty$. Therefore, in expectation, the reward achieved by the workers with success rate $p_{(3)}, p_{(4)}, p_{(5)}, ..., p_{(n)}$ using the Deterministic Dividing Algorithm is at least $\frac{1}{4}$ times the reward achieved by the same workers in the optimal assignment. □

Lemma 5 compares the expected reward achieved by the Deterministic Dividing Algorithm with the minimum expected reward achieved by the Interval Reservation Algorithm. Together with Lemma 4, this result can be used to compare the lower bound of the reward achieved by the Deterministic Dividing Algorithm and the Recursive Interval Reservation Algorithm.

Lemma 5. *Let $E[R_{DDA}]$ denote the expected reward achieved by the Deterministic Dividing Algorithm. Then,*

$$E[R_{DDA}] - \frac{1}{4}\sum_{j=1}^{n} x_{(j)}p_{(j)} \geq \frac{1}{4} \sum_{\{i=2k+1|k\geq 1, 2k+1\in\{1,2,...,n\}\}} (x_{(\frac{i+1}{2})} - x_{(i)})p_{(i)} + \frac{1}{4}\sum_{i=n-l+1}^{n} x_{(i)}p_{(n)}, \tag{4.9}$$

where $\frac{1}{4}\sum_{j=1}^{n} x_{(j)}p_{(j)}$ is the lower bound of the expected reward achieved by the Interval Reservation Algorithm.

Proof: Let x_{i_j} denote the task assigned to the worker with success rate $p_{(j)}$, $j = 1, 2, ..., n$, by the Deterministic Dividing Algorithm. Then, the expected reward achieved by the Deterministic Dividing Algorithm is given by

$$E[R_{DDA}] = \sum_{j} x_{i_j}p_{(j)} = \sum_{\{j=2k+1|2k+1\in\{1,2,...,n\}\}} x_{i_j}p_{(j)} + \sum_{\{j=2k|2k\in\{1,2,...,n\}\}} x_{i_j}p_{(j)}. \tag{4.10}$$

A lower bound for each of the two terms in the right hand-side of (4.10) is proven, which are used to prove (4.9). By Theorem 11,

$$\sum_{\{j=2k|2k\in\{1,2,...,n\}\}} x_{i_j}p_{(j)} \geq \frac{1}{4} \sum_{\{j=2k|2k\in\{1,2,...,n\}\}} x_{(j)}p_{(j)}. \tag{4.11}$$

The Deterministic Dividing Algorithm uses the Interval Reservation Algorithm to assign tasks to workers with success rates $p_{(1)}, p_{(3)}, ..., p_{(n-1)}$. In the SSAP with n tasks and workers, the Interval Reservation Algorithm does not assign any tasks to the workers with the smallest $\lfloor \frac{n}{2} \rfloor$ success rates. To prove a tighter lower bound on the reward that the Deterministic Dividing Algorithm achieves by assigning tasks to the workers with success rates $p_{(1)}, p_{(3)}, ..., p_{(n-1)}$, (i.e., the first term in the right hand-side of (4.10)), define $n - (n - \lfloor \frac{n}{2} \rfloor)$ auxiliary workers with success rates $p_{(n)}$, which increases the set of available workers to n workers with success rates

$$p_{(1)}, p_{(3)}, ..., p_{(n-1)}, p_{(n)}, p_{(n)}, ..., p_{(n)},$$

where the smallest $\lfloor \frac{n}{2} \rfloor$ success rates are all equal to $p_{(n)}$. By Thoerem 9,

$$\sum_{\{j=2k+1|2k+1\in\{1,2,...,n\}\}} x_{i_j} p_{(j)} \geq \frac{1}{4}\sum_{j=1}^{n-l} x_{(j)} p_{(2j-1)} + \frac{1}{4}\sum_{i=n-l+1}^{n} x_{(i)} p_{(n)}. \tag{4.12}$$

Inserting (4.11) and (4.12) into (4.10),

$$\begin{aligned} E[R_{DDA}] = \sum_j x_{i_j} p_{(j)} \geq \frac{1}{4} \sum_{\{j=2k|2k\in\{1,2,...,n\}\}} x_{(j)} p_{(j)} + \frac{1}{4}\sum_{j=1}^{n-l} x_{(j)} p_{(2j-1)} \\ + \frac{1}{4}\sum_{i=n-l+1}^{n} x_{(i)} p_{(n)} = \frac{1}{4}\sum_{j=1}^{n} x_{(j)} p_{(j)} + \frac{1}{4} \sum_{\{i=2k+1|2k+1\in\{1,2,...,n\}\}} (x_{(\frac{i+1}{2})} - x_{(i)}) p_{(i)} \\ + \frac{1}{4}\sum_{i=n-l+1}^{n} x_{(i)} p_{(n)}, \end{aligned} \tag{4.13}$$

which completes the proof. □

4.3.3 The Random Dividing Algorithm

This section describes an incentive compatible algorithm for GSSAP. The *Random Dividing Algorithm* (RDA) randomly orders the workers such that the i^{th} $(i = 1, 2, ..., n)$ largest success rate has $\frac{1}{n}$ probability of being in the j^{th} $(j = 1, 2, ..., n)$ position. It then recursively applies Algorithm B to groups of workers. Algorithm 7 formalizes this procedure. While the Recursive Interval Reservation Algorithm and the Deterministic Dividing Algorithm divide the workers into groups deterministically, the Random Dividing Algorithm generates groups of workers randomly. First, it is proven that the Random Dividing Algorithm is incentive compatible: It assigns each task to each of the workers with equal probability. Then, it is proven that the Random Dividing Algorithm is 6-competitive.

Theorem 12. *The Random Dividing Algorithm is incentive compatible.*

Proof: Let $P^* = \{p_1^*, p_2^*, ..., p_n^*\}$ denote the success rates of randomly ordered workers. Induction is used to prove that each of the tasks arriving in stages $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, ..., n$ are assigned to any of the workers with success rates $P_1^* = \{p_{\lfloor n/2 \rfloor+1}^*, p_{\lfloor n/2 \rfloor+2}^*, ..., p_n^*\}$ with equal probability. Since this process is recursively applied and each worker has equal probability of being in the set P_1^*, this proves that each task is assigned to each worker with equal probability. The task arriving at stage $\lfloor n/2 \rfloor + 1$ is assigned to the j^{th} worker if the task value is in the j^{th} interval defined by the first $\lfloor n/2 \rfloor$ tasks. Since the tasks have a random arrival order, the task value is equally likely to be in any of these intervals. Therefore, the task arriving at stage $\lfloor n/2 \rfloor + 1$ is assigned to any of the workers with success rates $\{p_{\lfloor n/2 \rfloor+1}^*, p_{\lfloor n/2 \rfloor+2}^*, ..., p_n^*\}$ with equal probability. As the induction assumption, assume that tasks arriving at stages $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, ..., t-1$ are assigned to any of the workers with success rates $\{p_{\lfloor n/2 \rfloor+1}^*, p_{\lfloor n/2 \rfloor+2}^*, ..., p_n^*\}$ with equal probability. The probability that task t is assigned to worker j is given by the probability that task t is assigned to worker j given that the worker is available at stage t multiplied by the probability that worker j is available at stage t. By the induction assumption, since each worker has equal probability of being in the set P_1^* and each task in stages $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, ..., t-1$ is equally likely to be assigned to any of the workers in the set P_1^*, $Pr($*worker j is available at stage t*$)$ is equal for all workers. Using the random arrival order of the tasks, task t is assigned to each worker with equal probability. $\square$

Algorithm 5 The Random Dividing Algorithm (RDA)

```
$RDA\ (n, P = \{p_i | i \in \{1, 2, ..., n\}\})$

Randomly order the workers $P^* = \{p_1^*, p_2^*, ..., p_n^*\}$
$A_{old} = P^*$
$s = \lfloor log(n) \rfloor + 1$
for j=1 to s
    $m = |A_{old}|$
    if $m > 1$
        $A_{new} = \{p_{(i)} \in A_{old} | i \in \{m - \lfloor m/2 \rfloor + 1, m - \lfloor m/2 \rfloor + 2, ..., m\}\}$
        $B(j) = A_{old} - A_{new}$
        $A_{old} = A_{new}$
    else
        $B(j) = A_{old}$
    end if
end for
$I_1 = (-\infty, +\infty)$
for j=1 to s
    $r = |B(s - j + 1)|$
    Assign the next $r$ sequentially arriving tasks to workers with success rates $p_{(l)} \in B(s - j + 1)$:
    if $x_i \in I_k$
        Assign task $i$ to worker with success rate $p_{(k)}$ if it is not assigned before
    end if
    Update the intervals using the $r$ assigned tasks:
    $I_k = (x_{(k)}, x_{(k-1)})$ for $k = 1, 2, ..., n + 1$, with $x_{(n+1)} = -\infty$ and $x_{(0)} = +\infty$
end for
```

Lemma 6 proves the lower bound of the reward achieved by the Random Dividing Algorithm by assigning the last $n - \lfloor n/2 \rfloor$ arriving tasks to workers. Lemma 7 proves that the maximum reward of the first $\lfloor n/2 \rfloor$ stages is $1/4$ of the maximum reward of the n-stage problem. These two results will be used to prove a 6-competitive ratio for the Random Dividing Algorithm in Theorem 13.

Lemma 6. *The expected reward that the Random Dividing Algorithm achieves by assigning the last $n - \lfloor n/2 \rfloor$ arriving tasks to the workers is at least $\frac{1}{8}$ times the maximum total reward.*

Proof: Assume that n is an even number. The proof for the case of n odd follows in a similar manner. The last $n/2$ tasks are assigned to the randomly selected $n/2$ workers in the selection set. Let $\{p_1^*, p_2^*, ..., p_{n/2}^*\}$ denote the randomly selected set of workers to be assigned the last $n/2$ tasks, and let $\{x_1^*, x_2^*, ..., x_{n/2}^*\}$ denote the last $n/2$ tasks, with x_i^* the task assigned to the worker with success rate p_i^* by the Random Dividing Algorithm. Then, by Lemma 9,

$$\sum_{i=1}^{n/2} x_i^* p_i^* \geq \frac{1}{4} \sum_{i=1}^{n/2} x_{(i)} p_{(i)}^*, \tag{4.14}$$

where $x_{(i)}$ is the i^{th} largest task value (among all n tasks) and $p_{(i)}^*$ is the i^{th} largest success rate value in the set $\{p_1^*, p_2^*, ..., p_{n/2}^*\}$. It must be proven that

$$E[\sum_{i=1}^{n/2} x_i^* p_i^*] \geq \frac{1}{8} R_{max}, \tag{4.15}$$

where $R_{max} = \sum_{i=1}^{n} x_{(i)} p_{(i)}$ is the maximum offline reward. Using (4.14), proving (4.15) is equivalent to proving that

$$E[\sum_{i=1}^{n/2} x_{(i)} p_{(i)}^*] \geq \frac{1}{2} R_{max} = \frac{1}{2} \sum_{i=1}^{n} x_{(i)} p_{(i)}. \tag{4.16}$$

Since the workers are ordered randomly,

$$E[p_{(i)}^*] = \sum_{j=i}^{n/2+1} \frac{\binom{j-1}{i-1}\binom{n-j}{n/2-i}}{\binom{n}{n/2}} p_{(j)}, \tag{4.17}$$

where $\frac{\binom{j-1}{i-1}\binom{n-j}{n/2-i}}{\binom{n}{n/2}}$ is the probability that the i^{th} largest success rate in the randomly selected set of workers is the j^{th} largest success rate among all n workers. Inserting (4.17) into (4.16),

$$E[\sum_{i=1}^{n/2} x_{(i)} p^*_{(i)}] = \sum_{i=1}^{n/2} x_{(i)} E[p^*_{(i)}] = \sum_{i=1}^{n/2} x_{(i)} \sum_{j=i}^{n/2+1} \frac{\binom{j-1}{i-1}\binom{n-j}{n/2-i}}{\binom{n}{n/2}} p_{(j)}. \tag{4.18}$$

Note that $x_{(i)}$ denotes the i^{th} largest task value and the randomness is in the success rate of the worker assigned to the i^{th} largest task. The constant factor multiplied by $p_{(j)}$ in (4.18) is given by

$$\sum_{i=1}^{j} \frac{\binom{j-1}{i-1}\binom{n-j}{n/2-i}}{\binom{n}{n/2}} = \frac{\binom{n-1}{n/2-1}}{\binom{n}{n/2}} = 1/2, \tag{4.19}$$

which completes the proof. □

Lemma 7. *The maximum expected reward of the first $\lfloor n/2 \rfloor$ stages of the problem (i.e., with the first $\lfloor n/2 \rfloor$ tasks and the first $\lfloor n/2 \rfloor$ workers in the randomly ordered set) is $\frac{1}{4}$ times the maximum total reward of the n-stage problem.*

Proof: Assume that n is an even number. The proof for the case of n odd follows in a similar manner. Let $\{\hat{p}_1, \hat{p}_2, ..., \hat{p}_{n/2}\}$ denote the success rates of the randomly selected set of workers to be assigned the first $n/2$ tasks, and let $\{\hat{x}_1, \hat{x}_2, ..., \hat{x}_{n/2}\}$ denote the first $n/2$ tasks. Let $\hat{x}_{(i)}$ denote the i^{th} largest task value among the first $n/2$ tasks and $\hat{p}_{(i)}$ is the i^{th} largest success rate in the set $\{\hat{p}_1, \hat{p}_2, ..., \hat{p}_{n/2}\}$. It must be proven that

$$E[\sum_{i=1}^{n/2} \hat{x}_{(i)} \hat{p}_{(i)}] = \frac{1}{4} R_{max} = \frac{1}{4} \sum_{i=1}^{n} x_{(i)} p_{(i)}, \tag{4.20}$$

where $R_{max} = \sum_{i=1}^{n} x_{(i)} p_{(i)}$ is the maximum offline reward. Since the workers and the task are (independently) ordered randomly,

$$E[\hat{x}_{(i)} \hat{p}_{(i)}] = E[\hat{x}_{(i)}] E[\hat{p}_{(i)}] = \sum_{j=i}^{n/2+1} \frac{\binom{j-1}{i-1}\binom{n-j}{n/2-i}}{\binom{n}{n/2}} x_{(j)} \sum_{j=i}^{n/2+1} \frac{\binom{j-1}{i-1}\binom{n-j}{n/2-i}}{\binom{n}{n/2}} p_{(j)}, \tag{4.21}$$

where $\frac{\binom{j-1}{i-1}\binom{n-j}{n/2-i}}{\binom{n}{n/2}}$ is the probability that the i^{th} largest success rate (task value) in the randomly selected set of workers (among the first $n/2$ tasks) is the j^{th} largest success rate (task value) among all n workers (tasks). Inserting (4.21) into (4.20),

$$E[\sum_{i=1}^{n/2} \hat{x}_{(i)}\hat{p}_{(i)}] = \sum_{i=1}^{n/2} E[\hat{x}_{(i)}]E[\hat{p}_{(i)}] = \sum_{i=1}^{n/2}[\sum_{j=i}^{n/2+1} \frac{\binom{j-1}{i-1}\binom{n-j}{n/2-i}}{\binom{n}{n/2}} x_{(j)} \sum_{j=i}^{n/2+1} \frac{\binom{j-1}{i-1}\binom{n-j}{n/2-i}}{\binom{n}{n/2}} p_{(j)}]. \quad (4.22)$$

Using (4.19), the constant factor multiplied by $p_{(j)}$ $(x_{(j)})$ in (4.22) is $1/2$. Therefore, the constant factor multiplied by $x_{(j)}p_{(j)}$ is $1/4$, which completes the proof. $\square$

Note that the result of Lemma 7 is intuitive. The expected reward of the first $\lfloor n/2 \rfloor$ tasks is $1/2$ of the expected reward of n tasks since they arrive in a random order. When the workers are also ordered randomly, the maximum expected reward of the first $\lfloor n/2 \rfloor$ stages become $1/4$ of the maximum reward of the n-stage problem.

Theorem 13. *The Random Dividing Algorithm is* 6*-competitive.*

Proof: Let $\{p_1^*, p_2^*, ..., p_{l^k}^*\}$ denote the success rates of the workers assigned at the k^{th} round with $l^{(k)} = \lfloor \frac{1}{2} ... \lfloor \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rfloor ... \rfloor$ (with k multipliers of $\frac{1}{2}$). Let x_i^* and x_i^{opt} denote the values of the tasks assigned to the worker with success rate p_i^* by the Random Dividing Algorithm and the optimal policy, respectively. Then, by Theorem 9,

$$\sum_{i=1}^{l^k} x_i^* p_i^* \geq \frac{1}{4} \sum_{i=1}^{l^k} x_i^{opt} p_i. \quad (4.23)$$

Using Lemmas 6, each assignment round achieves $1/8$ of the maximum reward of that round, and using Lemma 7, the maximum reward of each round is $1/4$ of the maximum reward of the previous round. Therefore,

$$E[R_{RDA}] = E[\sum_{l^k} \sum_{i=1}^{l^k} x_i^* p_i^*] \geq \frac{1}{4} E[\sum_{l^k} \sum_{i=1}^{l^k} x_i^{opt} p_i] \geq \frac{1}{4}(\frac{1}{2} + \frac{1}{8} + ...)R_{max}, \quad (4.24)$$

where $E[R_{RDA}]$ denotes the expected reward achieved by the Random Dividing Algorithm, and $R_{max} = \sum_{i=1}^{n} x_{(i)}p_{(i)}$ is the maximum offline reward. Therefore, as $n \to +\infty$, the Random Dividing Algorithm is 6-competitive. $\square$

4.4 GSSAP with a time-dependent reward

Babaioff et al. [7] study the Discounted Secretary Problem with a general discount function. They propose an $O(logn)$-competitive algorithm for the Discounted Secretary Problem and combine this algorithm with the Interval Reservation Algorithm to propose an $O(log(n))$-competitive algorithm for the Weighted Discounted Secretary Problem. GSSAP with a time-dependent reward is a weighted discounted secretary problem where the number of sequentially arriving items (i.e., sequentially arriving tasks in GSSAP) is equal to the number of available positions (i.e., workers in GSSAP). Therefore, the same idea can be used to propose an $O(log(n))$-competitive algorithm for the GSSAP with time-dependent reward values.

Let d denote the discount function, and let $d(i)$ and d_{max} denote the discount value at stage i and the maximum discount, respectively. Suppose that $[n]$ denotes the set of numbers $\{1, 2, ..., n\}$. The $O(log(n)) - competitive$ algorithm for the Weighted Discounted Secretary Problem selects $c \in [\Theta(log(nK))]$ uniformly at random (i.e., c is equally likely to be any of the numbers from 1 to a number belonging to $\Theta(log(nK))$), where $K(\leq n)$ is the number of available positions. It then runs the Interval Reservation Algorithm in the c^{th} discount class. Note that $f(n) \in \Theta(g(n))$ means that f is bounded both above and below by g asymptotically.

Algorithm 6 The Time-dependent GSSAP Algorithm

```
$Time - dependent\ GSSAP\ (n, d(i), P = \{p_i | i \in \{1, 2, ..., n\}\})$

$m = 0$
while $n \neq m$
    Select $c \in [\Theta(log((n - m)^2))]$ uniformly at random
    $I_c = (2^{-c} d_{max}, 2^{-(c-1)} d_{max}]$
    $T = \{j | d(j) \in I_c\}$
    $m = m + |T|$
    $L$=$\{p_{(i)} \in P | i \in \{1, 2, ..., m\}\}$
    Assign the tasks at stages $j \in T$ to workers with success rates $p_i \in L$ using RIRA
    $P = P - L$
```

Theorem 14. *[7] There is an $O(log(n))$-competitive algorithm for the Weighted Discounted Secretary Problem.*

The $O(log(n))$-competitive algorithm for the Weighted Discounted Secretary Problem selects a discount class randomly and runs the Interval Reservation Algorithm on the tasks arriving in the selected discount class. All the tasks in other discount classes and the tasks in the training phase of the selected discount class are discarded. However, starting from the first stage in GSSAP, each task must be assigned to one of the workers. The *Time-dependent GSSAP Algorithm*, which is formalized as Algorithm 6, achieves an $O(log(n))$-competitive algorithm while guaranteeing that each task is assigned to a worker. The Time-dependent GSSAP Algorithm generates a random number $c \in [\Theta(log(n^2))]$. Then, it assigns the workers with the m largest success rates to the tasks in the c^{th} discount class using the Recursive Interval Reservation Algorithm, where m is the number of tasks in the c^{th} discount class. The remaining $n - m$ tasks are assigned to the remaining workers by recursively generating random numbers $c \in \Theta[(log((n-m)^2))]$ and running the Recursive Interval Reservation Algorithm on the c^{th} discount class.

Lemma 8. *The Time-dependent GSSAP Algorithm is $O(log(n))$-competitive.*

Proof: Let c_i denote the i^{th} generated random number by the Time-dependent GSSAP Algorithm. The total expected reward achieved by the Time-dependent GSSAP Algorithm is the sum of the expected rewards achieved at different discount classes and given by

$$E[R_{TGSSAP}] = \sum_i E[R_{c_i}], \tag{4.25}$$

where $E[R_{TGSSAP}]$ denotes the expected reward achieved by the Time-dependent GSSAP Algorithm and $E[R_{c_i}]$ is the expected reward achieved by the c_i^{th} discount class. The tasks arriving at stages $\{j | d(j) \in I_{c_1}\}$ are assigned to the workers with the largest c_1 success rates. By Theorem 14, the reward achieved by the discount class c_1 (i.e., $E[R_{c_1}]$) is $O(log(n))$-competitive. Since $E[R_{TGSSAP}] \geq E[R_{c_1}]$, the Time-dependent GSSAP Algorithm is $O(log(n))$-competitive. $\square$

4.5 SSAP With General Reward Functions

The reward of each assignment in the SSAP is defined as the product of the worker's success rate and the value of the task assigned to the worker. There are a few papers that assume objectives other than maximizing the total expected reward for the SSAP. For example, Baharian and Jacobson [10] perform the task assignment under a threshold criterion, which minimizes the probability that the total reward fails to achieve a given value.

This section studies the GSSAP with general reward functions. It is assumed that the reward of each assignment is a function of the worker's success rate and the relative quality of the task (compared to the quality of other tasks). Assuming a random arrival order, backward induction is used to derive the optimal policy that maximizes the total expected reward.

Assume that the reward of each assignment is a known function of the worker's success rate and the relative quality of the task: Let $R(\sigma_i, p_j)$ denote the reward achieved by assigning the i^{th} arriving task to the worker with success rate p_j given that the rank of the i^{th} task among all n tasks is σ_i. For example, if the second arriving task has the third largest value among all tasks, then $\sigma_2 = 3$. Let S_i denote the set of remaining $n-i+1$ success rates at stage i (i.e, the success rates of the workers that are not assigned to a task in stages $1, 2, ..., i-1$). Let $ER_i^*(S_i)$ denote the maximum expected reward achieved at stage i with S_i as the set of remaining workers. The optimal policy assigns the i^{th} task to the worker with success rate p_{j_i} such that

$$E[\sum_{i=1}^{n} R(\sigma_i, p_{j_i})] = ER_1^*(S_1). \tag{4.26}$$

Let $E[R(\sigma_i, p_j)]$ denote the expected reward achieved by assigning the current task to the worker with success rates p_j, which is given by

$$E[R(\sigma_i, p_j)] = \sum_{\sigma_i=k}^{n-(i-k)} R(\sigma_i, p_j) \frac{\binom{\sigma_i-1}{k-1} \times \binom{n-\sigma_i}{i-k}}{\binom{n}{i}}, \tag{4.27}$$

where k denotes the rank of task i among the first i tasks, which is computed by comparing the relative quality of the i^{th} task with the tasks arrived in stages $1, 2, ..., i-1$. Moreover, $\frac{\binom{\sigma_i-1}{k-1} \times \binom{n-\sigma_i}{i-k}}{\binom{n}{i}}$ is the probability that the i^{th} task value is the σ_i^{th} largest among all n tasks given that it is the k^{th} largest among the first i tasks. Using backward induction, the optimal policy is to assign the arriving task to the worker that maximizes the expected sum of the assignment reward and the reward of the following stages. The maximum expected reward at stage i of the problem is given by

$$ER_i^*(S_i) = \frac{1}{n-i+1} \sum_{\sigma_i} \max_{p_j \in S_i} (E[R(\sigma_i, p_j)] + ER_{i+1}^*(S_i - \{p_j\})). \tag{4.28}$$

Assuming that tasks have a random arrival order (look at Definition **??**), the expected reward of the last assignment (as the base case of the backward induction) is given by

$$ER_n^*(S_n) = \frac{1}{n} \sum_{i=1}^{n} R(i, S_n), \tag{4.29}$$

where S_n denotes the single worker remaining in the last step. Solving (4.28) recursively by using (4.29) and (4.27) determines the maximum total expected reward and the optimal assignment at each stage of the SSAP.

Note that the Secretary Problem with a general utility function, which assumes that the reward of selecting the i^{th} best item is U_i [25], also uses backward induction to find the optimal decision at each stage [46]. However, the computational time of the optimal policy for the SSAP with a general reward function is larger than the Secretary Problem: At stage at i in the SSAP, $n-i+1$ reward values must be compared (to determine which of the remaining $n-i+1$ workers must be assigned to the task) while in the Secretary Problem, the optimal policy compares only two values at each stage, which are the expected utility of stopping at the current stage or proceeding to the next stage without hiring the current candidate.

4.6 Summary

This chapter proposes assignment policies for SSAP with no prior information on task values. This problem is referred to as the Generalized Sequential Stochastic Assignment Problem (GSSAP). GSSAP is described as a generalization of the Secretary Problem, where each of the selected elements is assigned to a distinct position. This relation can be used to derive assignment policies for GSSAP based on the Secretary Problem. Moreover, SSAP can be used to find the optimal policy for variations of the Secretary Problem when the values of sequentially arriving elements are independently drawn from a known distribution.

The Weighted Secretary Problem is used to propose assignment policies for GSSAP. The proposed assignment policies consist of a training phase to compute threshold values. These thresholds are used in the second phase to assign tasks to workers based on the interval that the task value is placed. Tasks arriving in the training phase are assigned to workers by recursively applying the same procedure of defining a training phase to compute thresholds. The relation between the Secretary Problem and GSSAP is further used to derive an assignment policy for GSSAP with time-dependent reward function.

CHAPTER 5

THE LINEAR PROGRAMMING TECHNIQUE

5.1 Introduction

This chapter presents the linear programming formulation of the *sequential assignment problem*, an extension of SSAP, where the reward of assigning the i^{th} $(i = 1, 2, ..., n)$ best task to the i^{th} best worker is equal to 1 while the reward of any other assignment is zero. The sequential assignment problem is an online version of the stable marriage problem, where each person's top preference is distinct and each person is the top preference of her best choice [26]. In this chapter, the assignment policies for the sequential assignment problem are evaluated by their competitive ratios. Moreover, it is assumed that the task values are selected by an adversary while having a random arrival order.

Formulating a matching problem as a linear program has been used as an effective tool to derive bounds on the performance of optimal algorithms and model variations of a problem by changing the objective function and constraints of the basic formulation. The linear programming technique also reduces the task of finding optimal algorithms to solving a linear program. The linear programming technique for studying online matching problems can be applied in one of the following two ways: First, an online matching problem can be formulated as a linear program with a known objective function and constraints. The objective function bounds the reward achieved by any assignment policy and the constraints are computed based on the problem's assumptions. This method is most appropriate for bounding the performance of optimal assignment policies. For example, Buchbinder et al. [14] formulate several variants of the Secretary Problem as linear programs and derives upper bounds on the performance of optimal algorithms.

Second, an online matching problem can be formulated as an *online linear program*, where the objective function and the constraint matrix in the linear program are revealed column by column. Each instance of the online linear program is equivalent to one stage of the online matching problem. This formulation provides the possibility of directly applying the solution of the linear program to derive algorithms for the online matching problem (since similar to the matching problem, the linear program must be solved stage by stage using the observed values in the prior stages). For example, Devanur and Hayes derive the online linear programming formulation of matching a set of keywords to available bidders [22]. Agrawal et al. proposes a near-optimal algorithm for a general class of online problems by defining an online linear program [1].

The main contribution of this chapter is to derive bounds on the performance of optimal policies for several extensions of the sequential assignment problem using the linear programming technique. First, duality is used to prove that the optimal policy of the sequential assignment problem can not achieve an approximation ratio better than e. Then, the linear programming formulation of several variations of the sequential assignment problem, including the problem with an unknown number of tasks, and the incentive compatible problem, which assumes equal assignment probability for each pair, are presented. This paper also describes the sequential assignment problem as a variation of the Adwords problem, and the online linear program of Adwords is presented as a framework to study extensions of SSAP [22].

The remainder of this chapter is organized as follows. Section 5.2 proposes the linear programming formulation of the sequential assignment problem and uses duality to prove that no algorithm can achieve a competitive ratio better than e. The optimal e-competitive algorithm is presented in Section 5.3. Section 5.4 derives the linear programming formulation of the incentive compatible problem, which assumes an equal probability for assigning each task to any of the workers. Section 5.5 extends the optimal algorithm for the sequential assignment problem to propose an incentive compatible mechanism. Section 5.6 studies the sequential assignment problem with an unknown number of tasks using the linear program. Section 5.7 casts the online linear program of Adwords problem into a new format, which can be used to study a generalization of the sequential assignment problem. Summary of this chapter and several concluding remarks are discussed in Section 5.8.

5.2 Linear Programming Formulation of the Sequential Assignment Problem

This section presents the linear programming formulation of the sequential assignment problem. The linear programming formulation is used to prove an e-competitive ratio for the optimal policy of the sequential assignment problem. First, the sequential assignment problem is formally defined in Section 5.2.1. Then, Section 5.2.2 presents the linear programming formulation. Table 5.1 summarizes the mathematical notation used in this chapter.

5.2.1 The Sequential Assignment Problem

There are n sequentially arriving tasks with a random arrival order that must be assigned to n workers. Assume that the workers are sorted such that worker j has the j^{th} largest success rate. Let R_{ij} denote the reward of assigning the i^{th} arriving task to worker j. The reward of assigning the j^{th} $(j = 1, 2, ..., n)$ best task to worker j is assumed to be 1, and the reward of any other assignment is assumed to be zero. The reward function is formally defined as:

$$R_{ij} = \begin{cases} 1, & \text{if } x_i = x_{(j)}^{(n)} \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

where $i, j = 1, 2, ..., n$. Note that $x_i = x_{(j)}^{(n)}$ means that the i^{th} arriving task is the j^{th} best task overall (i.e., the i^{th} arriving task has the j^{th} largest value among all n tasks). The main challenge is that the decision maker must assign a task to one of the workers upon its arrival, with the relative rank of the task (i.e., the rank of the arriving task among tasks observed so far) as the only available information.

The Secretary Problem is a special case of the sequential assignment problem with $R_{i1} = 1$ if $x_i = x_{(1)}^{(n)}$ for $i = 1, 2, ..., n$ (i.e., the reward of selecting the best secretary is equal to 1). SSAP is a variation of the sequential assignment problem with $R_{ij} = x_i \times p_j$ (i.e., the reward of assigning task i to worker j is the product of the task value and the worker's success rate).

Table 5.1: List of notations used in Chapter 5

Symbol	Meaning
$pr(E)$	Probability of event E
Y_{ij}	Binary (decision) variable of assigning the i^{th} arriving task to worker j
$pr(Y_{ij} = 1) = y_{ij}$	Probability of assigning the i^{th} arriving task to worker j
x_i	Value of the i^{th} arriving task
$x_{(j)}^{(i)}$	j^{th} largest task value among the first i arriving tasks
$x_{(j)}^{(n)}$	j^{th} largest task value among all n arriving tasks
R_{ij}	Reward of assigning the i^{th} arriving task to worker j

5.2.2 The Linear Programming Formulation

This section proposes a linear programming formulation which provides an upper bound on the reward achieved by any assignment policy for the sequential assignment problem. Before presenting the linear programming formulation, the set of admissible policies are introduced as Definition 4.

Definition 4. *An assignment policy for the sequential assignment problem is admissible if the probability of assigning task i to worker j (denoted by y_{ij}) satisfies the following constraints:*

$$\sum_{j=1}^{n} y_{ij} \leq 1; \quad i = 1, 2, \ldots, n$$

$$y_{ij} \leq 1 - \sum_{l=1}^{i-1} y_{lj}; \quad i, j = 1, 2, \ldots, n$$

$$y_{ij} \geq 0; \quad i, j = 1, 2, \ldots, n$$

The first constraint guarantees that each task is assigned to at most one worker while the second constraint guarantees that at most one task is assigned to each worker.

Lemma 9 generalizes the linear programming formulation of the Secretary Problem [14] to formulate the sequential assignment problem as a linear program. While the linear program of the Secretary Problem has n decision variables, corresponding to the probability of hiring the secretary at stage i $(= 1, 2, ..., n)$, the linear program of the sequential assignment problem has n^2 decision variables, corresponding to the probability of assigning task i to worker j, with $i, j = 1, 2, ..., n$.

Lemma 9. *Let y_{ij} denote the probability of assigning the i^{th} arriving task to worker j. Then, the optimal solution to the following linear program provides an upper bound for the reward achieved by any assignment policy for the sequential assignment problem.*

$$\begin{aligned} \textit{maximize} \quad & \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij} \\ \textit{subject to} \quad & \sum_{j=1}^{n} y_{ij} \leq 1; \quad i = 1, 2, \ldots, n. \\ & y_{ij} \leq 1 - \sum_{l=1}^{i-1} y_{lj}; \quad i, j = 1, 2, \ldots, n. \\ & y_{ij} \geq 0; \quad i, j = 1, 2, \ldots, n. \end{aligned}$$

Proof: The achieved reward of any assignment policy for the sequential assignment problem is given by

$$\sum_{i=1}^{n}\sum_{j=1}^{n} pr(Y_{ij} = 1 | x_i = x_{(j)}^{(n)}) \times pr(x_i = x_{(j)}^{(n)}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} pr(Y_{ij} = 1 | x_i = x_{(j)}^{(n)})$$
$$\leq \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} pr(Y_{ij} = 1) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij}$$

Therefore, the objective function of the linear program provides an upper bound on the performance of any assignment policy. The constraints are simply the same as the requirements of admissible policies, described in Definition 4. □

Lemma 9 shows that the formulated linear program provides an upper bound for the optimal reward achieved by any policy for the sequential assignment problem. The total expected reward of a policy, which assigns task i to worker j with probability y_{ij}, is given by $\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} pr(Y_{ij} = 1 | x_i = x_{(j)}^{(n)})$. Therefore, any assignment policy for which equation $y_{ij} = pr(Y_{ij} = 1 | x_i = x_{(j)}^{(n)})$, achieves a total reward of $\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} i y_{ij}$.

Any assignment policy for the sequential assignment problem satisfies the constraints of the linear program defined in Lemma 9. Therefore, the set of feasible assignment policies for the sequential assignment problem is a subset of the solutions of the linear programming formulation.

Theorem 15 uses this result to provide an upper bound on the performance of any mechanism for the sequential assignment problem.

Theorem 15. *No algorithm for the sequential assignment problem can achieve a competitive ratio better than e.*

Proof: The dual of the linear program presented in Lemma 9 is given by:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} + \sum_{k=1}^{n} \hat{d}_k \\
\text{subject to} \quad & d_{ij} + \sum_{k=i+1}^{n} d_{kj} + \hat{d}_i \geq \frac{1}{n}; \quad i,j = 1,2,\dots,n. \\
& d_{ij} \geq 0; \quad i,j = 1,2,\dots,n. \\
& \hat{d}_k \geq 0; \quad k = 1,2,\dots,n.
\end{aligned}$$

Any feasible solution to dual provides an upper bound for the optimal solution of the primal. Consider the feasible dual solution $d_{ij} = 0$ for $1 \leq i \leq n/e$, $d_{ij} = \frac{1}{n}(1 - \sum_{k=i}^{n-1} \frac{1}{k})$ for $n/e < i \leq n$, and $\hat{d}_k = 0$ for $k = 1, 2, ..., n$. The dual objective value of this solution is n/e, which proves the claim. □

While the Secretary Problem deals with finding only the best element, the sequential assignment problem seeks to find the optimal matching of all sequentially arriving elements. However, the upper bound on the approximation ratio of the sequential assignment problem is the same as the Secretary Problem. Notice the similarity between the linear programming formulation, the dual problem, and the dual feasible solution of the Secretary Problem [14] and the sequential assignment problem.

5.3 The Optimal Algorithm

This section uses the Online Bipartite Matching problem to propose the optimal (e-competitive) algorithm for a generalization of the sequential assignment problem, where the reward of assigning task i with value x_i to a worker with success rate p_j is given by $x_i \times p_j$. The Online Bipartite Matching problem matches a set of sequentially arriving elements to a set of available entities: Assume a bipartite graph consists of n left vertices and n right vertices. The left vertices arrive sequentially. Upon each left vertex arrival, the set of adjacent right vertices are revealed, and the left vertex must be irrevocably assigned to one of the adjacent right vertices. The objective is to maximize the size of the matching set.

The Edge-Weighted Online Bipartite Matching assumes that each edge between the arriving left vertex and its adjacent right vertices has a weight, observed upon the left vertex arrival. The sequentially arriving tasks in the sequential assignment problem correspond to the left vertices in the Edge-Weighted Online Bipartite Matching while the workers correspond to the right vertices.

Algorithm 7 The Optimal Algorithm

```
Input: $(n, P = \{p_i | i \in \{1, 2, ..., n\}\})$

Order the workers $\{p_{(1)} \geq p_{(2)} \geq ... \geq p_{(n)}\}$
Divide the workers to two groups $P_S = \{p_{(1)}, p_{(2)}, ..., p_{(n-\lfloor n/e \rfloor)}\}$ and $P_T = \{p_{(n-\lfloor n/e \rfloor+1)}, p_{(n-\lfloor n/e \rfloor+2)}, ..., p_{(n)}\}$
Observe the first $\lfloor n/e \rfloor$ tasks
From stage $l = \lfloor n/e \rfloor + 1$, find the optimal matching on $\{x_1, x_2, ..., x_l\} \cup P_S$
If the optimal matching assigns $x_l$ to $p_j$, assign the task to the worker if the worker is available
Assign the first $\lfloor n/e \rfloor$ tasks to workers $P_T$ by recursively applying the same procedure
```

A simple variation of the Online Bipartite Matching Algorithm proposed by [34] is used as the optimal policy for the generalization of the sequential assignment problem. The algorithm observes the first $\lfloor \frac{n}{e} \rfloor$ tasks. Starting from stage $l = \lfloor \frac{n}{e} \rfloor + 1$, the algorithm finds the best matching on the set $\{x_1, x_2, ..., x_l\} \cup P_S$, where $\{x_1, x_2, ..., x_l\}$ denotes the set of first l tasks and P_S represents the set of all $n - \lfloor n/e \rfloor$ workers with the largest success rates. If worker j is assigned to task l in the optimal matching, the algorithm assigns task l to worker j if the worker is available (i.e., it is not previously matched to one of the prior tasks). The first $\lfloor n/e \rfloor$ tasks are assigned to the $\lfloor n/e \rfloor$ workers with the smallest success rates by recursively applying the same procedure.

Notice the similarity between the optimal algorithm of the sequential assignment problem and the Secretary Problem. Both algorithms use the first $\lfloor \frac{n}{e} \rfloor$ elements as a training phase. Then, starting from stage $\lfloor \frac{n}{e} \rfloor + 1$, the assignment policies assign tasks to workers/hire secretaries based on the optimal matching of the elements observed so far.

Lemma 10. *Algorithm 7 is e-competitive for the sequential assignment problem.*

The proof is very similar to the proof of the optimality of the algorithm for the Edge-Weighted Online Bipartite Matching.

5.4 Incentive Compatibility

The linear programming technique reduces the problem of finding assignment policies for variants of the sequential assignment problem to simply changing the objective function and constraints of the basic formulation. This section and Section 5.6 provide two examples of challenging problems that are formulated by a few changes in the linear program presented in Lemma 9.

This section proposes the linear programming formulation of the *incentive compatible* problem, which assumes that the probability of assigning task i to worker j is equal for all $i, j \in \{1, 2, ..., n\}$. While many algorithms for the online matching problems have a training phase and assign the arriving elements to (better) positions after the training phase, the incentive compatible problem provides fairness: There is no incentive for the sequence of arriving elements to appear in later stages. The incentive compatible formulation may be applied to online matching markets, where a bidder might be motivated to bid in later stages to increase its chance of being matched to a better item.

Lemma 11. *Let p denote the probability that task i $(= 1, 2, ..., n)$ is assigned to worker j $(= 1, 2, ..., n)$ by any incentive compatible mechanism. Let g_{ij} denote the probability of assigning task i to worker j given that task i is j^{th} best overall. Then, p and g_{ij} is a feasible solution to the following linear program. Moreover, the optimal value of the objective function, which provides an upper bound on the expected reward of any assignment policy for the incentive compatible problem, is achieved by the assignment probabilities $g_{ij} = ip$ for $1 \leq i \leq \frac{1}{2p}$ and $g_{ij} = 1 - (i-1)p$ for $i > \frac{1}{2p}$.*

$$\begin{aligned}
&\textit{maximize} \quad \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} g_{ij} \\
&\textit{subject to} \quad g_{ij} \leq i \times p; \quad i,j = 1,2,\ldots,n. \\
&\qquad\qquad g_{ij} + (i-1) \times p \leq 1; \quad i,j = 1,2,\ldots,n. \\
&\qquad\qquad p \leq \frac{1}{n} \\
&\qquad\qquad p, g_{ij} \geq 0; \quad i,j = 1,2,\ldots,n.
\end{aligned}$$

Proof: The objective function of the linear program is the same as the basic formulation. For the first constraint, notice that

$$p = \frac{1}{i}\sum_{k=1}^{i} pr(Y_{ij} = 1 | x_i = x_{(k)}^{(i)}) \tag{5.2}$$

Moreover, by conditioning on the task's relative rank,

$$g_{ij} = pr(Y_{ij} = 1 | x_i = x_{(j)}^{(n)}) = \sum_{k=1}^{i} pr(Y_{ij} = 1 | x_i = x_{(j)}^{(n)} \cap x_i = x_{(k)}^{(i)}) \times pr(x_i = x_{(k)}^{(i)}) \tag{5.3}$$

However, knowing that the task's overall rank is j means that the relative rank of the task is at most $min(i,j)$ (Given the overall rank of the task does not mean that the overall rank is known to the assignment policy; this only limits the possible values of the task's relative rank). Therefore,

$$g_{ij} = \sum_{k=1}^{min(i,j)} pr(Y_{ij} = 1 | x_i = x_{(k)}^{(i)}) \times pr(x_i = x_{(k)}^{(i)}) \leq \sum_{k=1}^{min(i,j)} pr(Y_{ij} = 1 | x_i = x_{(k)}^{(i)})$$
$$\leq \sum_{k=1}^{i} pr(Y_{ij} = 1 | x_i = x_{(k)}^{(i)}),$$

where we have used $pr(x_i = x_{(k)}^{(i)}) \leq 1$ and $min(i,j) \leq i$. Together with (5.2), this proves the first constraint. The second and third constraints guarantee that each task is assigned to at most one worker and the probability of assigning a task to a worker is not greater than $1/n$. The optimal assignment probabilities $g_{ij} = ip$ for $1 \leq i \leq \frac{1}{2p}$ and $g_{ij} = 1 - (i-1)p$ for $i > \frac{1}{2p}$ are simply the optimal solutions of the linear program. □

Note that if the goal is to guarantee that each task is assigned to one of the workers (similar to the *must hire Secretary Problem*), only the fourth constraint must be changed to $p = \frac{1}{n}$ [14]. The optimal values of the incentive compatible Secretary Problem are given by $g_i = ip$ for $1 \leq i \leq \frac{1}{2p}$ and $g_i = 1 - (i-1)p$ for $i > \frac{1}{2p}$, where g_i denotes the probability of hiring the Secretary arriving at stage i [14]. Here, instead of selecting one secretary, the objective is to maximize the number of correct matchings (out of n possible pairs). Since all correct matchings are assumed to have the same reward, the optimal policy is the same for all n pairs: There is a training phase of the same length to find the optimal task for each worker. As the stage number increases, the probability that the i^{th} relatively best task (i.e., the i^{th} largest task value so far) is the i^{th} best overall increases. Therefore, each pair must have the same training phase and the same selection phase to find the optimal matching.

While Lemma 11 shows that any assignment policy is a feasible solution to the linear program, the optimal solution of the linear program does not necessarily correspond to an assignment policy for the incentive compatible sequential assignment problem. Contradiction is used to prove this result. The probability of assigning task i to worker j is given by

$$p = \sum_{k=1}^{i} pr(Y_{ij} = 1 | x_i = x_{(k)}^{(i)}) \times \frac{1}{i} \tag{5.4}$$

The optimal policy must satisfy

$$g_{ij} = ip = pr(Y_{ij} = 1 | x_i = x_{(j)}^{(n)}), \tag{5.5}$$

for $1 \leq i \leq \frac{1}{2p}$. Given that task i is the j^{th} best task overall, its relative rank (i.e., its rank among the first i tasks) is between $|n-(i+j)|$ and $min(i,j)$. Therefore,

$$pr(Y_{ij} = 1 | x_i = x_{(j)}^{(n)}) = \sum_{k=|n-(i+j)|}^{min(i,j)} pr(Y_{ij} = 1 | x_i = x_{(k)}^{(i)}) \times \frac{1}{r}, \tag{5.6}$$

where $r = min(i,j) - |n-(i+j)| + 1$. Using (5.5) and (5.6),

$$ip = \sum_{k=|n-(i+j)|}^{min(i,j)} pr(Y_{ij} = 1 | x_i = x_{(k)}^{(i)}) \times \frac{1}{r}, \tag{5.7}$$

and hence,

$$rip = \sum_{k=|n-(i+j)|}^{min(i,j)} pr(Y_{ij} = 1 | x_i = x_{(k)}^{(i)}), \tag{5.8}$$

Note that $rip \geq ip$ and in general, there exists (n, i, j) such that $r > 1$, and hence,

$$\sum_{k=|n-(i+j)|}^{min(i,j)} pr(Y_{ij} = 1 | x_i = x_{(k)}^{(i)}) > \sum_{k=1}^{i} pr(Y_{ij} = 1 | x_i = x_{(k)}^{(i)}), \tag{5.9}$$

which is a contradiction since the right hand-side includes all the terms of the summation in the left hand-side. Therefore, while any feasible assignment policy for the incentive compatible mechanism is a subset of the solutions of the linear programming formulation, every solution of the linear program does not necessarily correspond to an assignment policy for the incentive compatible sequential assignment problem.

Lemma 12 uses the linear programming formulation to derive upper bounds on the performance of any assignment policy for the incentive compatible sequential assignment problem and the problem with n assignments (which assigns each task to one of the workers).

Lemma 12. *There is no incentive compatible policy for the sequential assignment problem which achieves a competitive ratio better than* $1 - \frac{1}{\sqrt{2}}$. *Moreover, if the objective is to assign each task to one of the workers, no assignment policy can achieve a competitive ratio better than* 0.25.

The proof follows by finding the optimal objective value of the linear programming formulations. Note that the same competitive ratios hold for the incentive compatible Secretary Problem, and the incentive compatible and must-hire Secretary Problem [14].

5.5 The Incentive Compatible Algorithm

This section extends the optimal algorithm proposed in Section 5.3 to propose an incentive compatible algorithm. The idea is similar to the incentive compatible algorithm proposed in the previous chapter. The set of workers are randomly divided into two groups. The first $\lfloor \frac{n}{e} \rfloor$ tasks are used as a training phase to assign task to the workers of the second group after the training phase. The tasks arriving in the training phase are assigned to the first group of workers by recursively applying the same procedure. Note that the proof of incentive compatibility is very similar to the proof of Theorem 12.

Algorithm 8 The Incentive Compatible Algorithm

```
Input: $(n, P = \{p_i | i \in \{1, 2, ..., n\}\})$

Randomly order the workers $\{p_1^*, p_2^*, ..., p_n^*\}$
Divide the workers to two groups $P_S^* = \{p_1^*, p_2^*, ..., p_{n-\lfloor n/e \rfloor}^*\}$ and $P_T^* = \{p_{n-\lfloor n/e \rfloor+1}^*, p_{n-\lfloor n/e \rfloor+2}^*, ..., p_n^*\}$
Observe the first $\lfloor n/e \rfloor$ tasks
From stage $l = \lfloor n/e \rfloor + 1$, find the optimal matching on $\{x_1, x_2, ..., x_l\} \cup P_S$
If the optimal matching assigns $x_l$ to $p_j^*$, assign the task to the worker if the worker is available
Assign the first $\lfloor n/e \rfloor$ tasks to workers $P_T^*$ by recursively applying the same procedure
```

Theorem 16. *The Incentive Compatible Algorithm is* $(e + 1)$*-competitive for the sequential assignment problem.*

The proof is very similar to the proof of Theorem 13.

5.6 Unknown Number of Elements

One of the main assumptions of the sequential assignment problem, SSAP, and many other online matching problems, is that the number of arriving elements is known. While designing algorithms with reasonable competitive ratios requires some prior information on the number of tasks, assuming a fixed value is unrealistic for many applications. For example, the number of bidders in an online matching market might change during the assignment process.

The linear programming technique simplifies the challenging problem of bounding the performance of assignment policies for the sequential assignment problem with uncertainty in the number of arriving tasks. Oveis gharan and Vondrak (2011) study the Secretary Problem with the number of elements selected by an Adversary. They use the linear programming formulation of the Secretary Problem to provide an upper bound on the probability of selecting the best element.

Theorem 17. *[51] There is no algorithm for the Secretary Problem with the number of elements selected by an adversary, that returns the best element with probability more than $\frac{1}{H_N}$, where $H_N = \sum_{i=1}^{N} \frac{1}{i}$ is th N^{th} harmonic number.*

Lemma 13 formulates the sequential assignment problem with an unknown number of tasks as a linear program using the Secretary Problem.

Lemma 13. *Assume that the number of sequentially arriving elements in the sequential assignment problem is chosen by an adversary from $\{1, 2, ..., N\}$, where N is given. Then, the optimal solution to the following Linear Programming formulation provides an upper bound on the reward achieved by any assignment policy for the online matching problem.*

$$\begin{aligned}
\textit{maximize} \quad & \alpha \\
\textit{subject to} \quad & \alpha \leq \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} i y_{ij} \qquad n = 1, 2, \ldots, N. \\
& y_{ij} \leq 1 - \sum_{l=1}^{i-1} y_{lj}; \quad i, j = 1, 2, \ldots, N. \\
& \sum_{j=1}^{n} y_{ij} \leq 1; \quad i = 1, 2, \ldots, n; n = 1, 2, \ldots, N. \\
& y_{ij} \geq 0; \quad i, j = 1, 2, \ldots, N.
\end{aligned}$$

Proof: The proof follows the same steps as the proof of Lemma 9. □

Theorem 18 provides an upper bound on the performance of any assignment policy for the sequential assignment problem, with the number of tasks selected by an adversary.

Theorem 18. *Assume that the number of arriving tasks in the sequential assignment problem is selected by an adversary from $\{1, 2, ..., N\}$, where N is given. Then, there is a randomized algorithm that finds the perfect matching (i.e., assigns each task to its optimal worker) with probability at least $\frac{1}{2(H_{N-1}+1)}$. Moreover, no algorithm find the perfect matching with probability more than $\frac{1}{H_N}$.*

Proof: Let n denote the (random) number of tasks (and hence, the size of perfect matching). Proving the first part is equivalent to showing that there is a feasible solution for the linear program (defined by Lemma 13) with objective value at least $\frac{n}{2(H_{N-1}+1)}$. Define the probability of assigning task i to worker j as $y_{ij} = \frac{1}{i(H_{N-1}+1)}$ for $j \leq i$. Consider the following algorithm: If the i^{th} arriving task is the j^{th} best so far, assign it to worker j with probability $\frac{iy_{ij}}{1-\sum_{l=1}^{i-1} y_{lj}}$. Then,

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} iy_{ij} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} i \times \frac{1}{i(H_{N-1}+1)} = \frac{n+1}{2(H_{N-1}+1)} \tag{5.10}$$

Moreover,

$$y_{ij} + \sum_{l=1}^{i-1} y_{lj} \leq iy_{ij} + \sum_{l=1}^{i-1} y_{lj} = \frac{H_{i-1}+1}{H_{N-1}+1} \leq 1, \tag{5.11}$$

proves that the second constraint is satisfied [51] while

$$\sum_{j=1}^{n} y_{ij} = \sum_{j=1}^{i} y_{ij} = \frac{1}{H_{N-1}+1} \leq 1, \tag{5.12}$$

implies the third constraint. Therefore, $(\alpha = \frac{n+1}{2(H_{N-1}+1)}, y_{ij} = \frac{1}{i(H_{N-1}+1)})$ for $i = 1, 2, ..., n$, and $j \leq i$ provide a feasible solution for the linear program with the objective value larger than $\frac{n}{2(H_{N-1}+1)}$.

For the second part, note that the Secretary Problem is a special case of the sequential assignment problem, where instead of finding the optimal match for each of the sequentially arriving elements, the objective is to find only the best element. Therefore, if there is an algorithm for the sequential assignment problem that finds the perfect matching with probability more than $\frac{1}{H_N}$, it can be used to find the best element with the same probability. This contradicts Theorem 17. □

5.7 Online Linear Programming Formulation

The previous sections proposed the linear programming formulation of several online matching problems. This section presents the *online linear programming* formulation for a generalization of the sequential assignment problem, where instead of the zero-one reward of the sequential assignment problem, the reward of assigning a task to a worker is the product of the task value and the worker's success rate.

Let x_j denote the value of the j^{th} arriving task and $q_{j,i}$ denote the success rate of worker i upon the arrival of the j^{th} task. At stage j, the task value (x_j) is observed, the workers' success rates (which are possibly different from those of the previous stages) are observed, and the task is irrevocably assigned to one of the workers. The objective is to maximize the sum of the assignments' rewards, defined as the product of the task value and the worker's success rate. Since the workers' success rates are also random, this problem is referred to as the Doubly Stochastic Sequential Assignment Problem (DSSAP). As described before, the objective function and the constraints in an online linear program are revealed column by column. The online linear programming formulation provides the possibility of applying linear programming techniques to derive assignment policies for an online matching problem.

DSSAP is presented as a special case of the Adwords problem. The Adwords problem seeks to maximize the revenue of assigning sequentially arriving keywords to competing bidders in a search engine [22]: There are n bidders with known daily budgets $B_1, B_2, ..., B_n$ and m keywords. When keyword j $(= 1, 2, ..., m)$ arrives, the bidders submit their bids, with u_{ij} denoting the bid of bidder i for query j. Then, the search engine must assign the keyword to one of the bidders such that the total revenue is maximized while the daily budget constraint of each bidder is satisfied (i.e., the total revenue collected from bidder i must be at most B_i). It is assumed that the queries have a random arrival order.

There is one difference between DDSAP and Adwords: While each task in the DSSAP is assigned to at most one worker (and each worker is assigned to at most one task), each bidder in Adwords can be assigned to multiple queries as long as its daily budget is not spent. Let x_i denote the value of task i, q_{ij} denote the success rate of worker j upon arrival of task i, and y_{ij} denote the probability of assigning task i to worker j. Then, the online linear programming formulation of

DSSAP is given by

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} x_i q_{ij} y_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{n} y_{ij} \leq 1; \quad i = 1,2,\ldots,n. \\ & \sum_{i=1}^{n} y_{ij} \leq 1; \quad j = 1,2,\ldots,n. \\ & y_{ij} \geq 0; \quad i,j = 1,2,\ldots,n. \end{aligned}$$

This is an online linear program, where the objective function is revealed column by column. The online linear program is a variation of the formulation for Adwords [22] with the bid of bidder j for query i defined as the reward of assigning task i to worker j $(x_i \times q_{ij})$ and the daily budget constraint replaced by $\sum_{i=1}^{n} y_{ij} \leq 1$, which guarantees at most one task is assigned to each worker.

Theorem 19. *[22] Let OPT denote the maximum offline reward of the n-stage problem, $R_{Max} = \max_{1\leq i,j\leq n} x_i \times q_{ij}$ denote the maximum reward value, and $\lambda = \max_{i,j,j'} \frac{q_{ij}}{q_{ij'}}$ denote the ratio of the maximum to minimum reward collected from any task. Then, there is a $\frac{1}{1-\epsilon}$-competitive algorithm for all tasks and workers success rates such that*

$$\frac{OPT}{R_{Max}} \geq \Omega(\frac{n^2 \times log(\lambda/\epsilon)}{\epsilon^3})$$

5.8 Summary

This chapter proposes the linear programming formulation of the sequential assignment problem, a variation of SSAP with a reward of one for each correct matching. A correct matching is defined as matching the i^{th} largest task value to the worker with the i^{th} largest success rate. The linear programming technique reduces the online matching problems to solving a linear program, provides a method to compute upper bounds of the reward achieved by the optimal algorithms, and models variations of the problem by changing the objective function and constraints of the basic formulation.

The sequential assignment problem with unknown number of elements and the incentive compatible problem are formulated as linear programs and the upper bounds on the performance of any assignment policy are derived.

CHAPTER 6

DYNAMIC SEQUENTIAL ASSIGNMENT

6.1 Introduction

The basic formulation of SSAP assumes that each worker is assigned to at most one task. When a task is assigned to a worker, the worker will not be available for any future assignments. This assumption is realistic for some applications of SSAP such as the organ transplant problem. However, the one-time assignment assumption is not valid for several applications such as online matching markets and aviation security. The bidders in an online matching market might intend to buy several items. The security gates in the aviation security problem screen many passengers in a single day. Therefore, relaxing the one-time assignment assumption would make the model closer to the real-life problems. This chapter studies a more general problem, where a worker would become available for future assignments upon performing the previous task. This problem is referred to as the Dynamic Sequential Assignment Problem.

This chapter studies the Dynamic Sequential Assignment Problem under two models. Both models assume that the task duration is independent of the task value and the worker performing the task. Section 6.2 describes the first model, which considers a fixed task duration. Section 6.3 assumes a Geometric model for the task duration. An extension of the problem with random arrival of tasks is studied in Section 6.4.

6.2 Fixed Task Duration

This section assumes that once a task is assigned to a worker, the worker performs the task in a fixed duration, which is a discrete value (i.e., can be computed in terms of the number of stages). The task duration is independent of the task value and the workers' success rates. For example, if the task's duration is 3 and a task is assigned to worker j on stage 2, the worker becomes available for a new assignment at stage 5. The basic formulation of SSAP is a special case of this generalized model with $T = 1$.

We first describe the optimal algorithm for the Edge-Weighted Online Bipartite Matching Problem [34]. We refer to this algorithm as the BM Algorithm. As discussed in the previous chapter, the BM Algorithm is a generalization of the optimal algorithm for the Secretary Problem, which uses the first $\lfloor n/e \rfloor$ stages as a training phase and assigns tasks to workers based on the optimal (offline) matching on the set of tasks observed so far from stage $\lfloor n/e \rfloor + 1$. Simple variations of this algorithm are used throughout this paper for variations of the problem.

Algorithm 9 The BM Algorithm [34]

```
Observe the first $\lfloor n/e \rfloor$ tasks: $\{x_1, x_2, ..., x_{\lfloor n/e \rfloor}\}$
From stage $l = \lfloor n/e \rfloor + 1$
  Find the optimal matching on $\{x_1, x_2, ..., x_l\} \cup \{p_1, p_2, ..., p_n\}$
  Let $p_j$ denote the worker's success rate assigned to $x_l$ in the optimal matching
  Assign task $l$ to worker $j$ if the worker is not previously assigned
```

Theorem 1 proves that the BM Algorithm is $\frac{e^2}{e-1}$-competitive when the duration of each task is a fixed known value. Note that the only difference between applying the BM algorithm to this problem and the Edge-Weighted Online Bipartite Matching Problem is in finding the optimal matching on the tasks observed so far. While the optimal matching of the Edge-Weighted Online Bipartite Matching (on the set of tasks observed so far) assigns the i^{th} best task among the first l tasks to the i^{th} best worker, the optimal matching for the problem with a fixed task duration is as follows: Find T workers with the largest success rate values. Assign the best l/T tasks observed so far to the best worker (where l is the number of available tasks). Assign the second best l/T tasks to the second best worker, and so on. This policy is optimal since we assume that in the offline setting, the decision maker has access to all tasks and workers, and hence, can adjust the tasks' ordering. The optimal policy puts the best l/T tasks on stages $1, T+1, 2T+1, ...$ and assigns these tasks to the best worker. The other tasks are assigned to the workers in a similar fashion.

Theorem 20. *The BM Algorithm is $\frac{e^2}{e-1}$-competitive for SSAP with random arrival order of tasks and fixed task duration.*

Proof: The proof has the same structure as the proof of e-competitiveness of the BM Algorithm for the edge-weighted bipartite matching problem. Let OPT denote the maximum offline reward in the n-stage problem. First, it is proven that if a task is assigned to a worker, the reward of this matching is at least $1/n$ of OPT. This is a direct results of the tasks' random arrival order, as shown by [34]: The total expected reward of the optimal matching on the tasks observed so far (i.e., on the set $\{x_1, x_2, ..., x_l\} \cup \{p_1, p_2, ..., p_n\}$) is at least l/n of the maximum offline reward over all tasks and workers. The expected reward of matching at stage l is at least $1/l$ of the maximum offline reward of matching on the set $\{x_1, x_2, ..., x_l\} \cup \{p_1, p_2, ..., p_n\}$. Therefore, the reward of each matching by the algorithm is at least OPT/n.

Next, we prove that the probability that task l is assigned to a worker (i.e., the worker is available on stage l) converges to $1/e$. Assume that the optimal offline matching on the set of tasks observed so far assigns task l to worker j. The probability that worker j is available on stage l is equal to the probability that no task is assigned to worker j at stages $i = l - T + 1, l - T + 2, ..., l - 1$, which is given by $(\frac{T-1}{T})^{T-1} \geq (1 - \frac{1}{T})^T$ and converges to $1/e$. Therefore, the BOM Algorithm achieves an expected reward of $\frac{OPT}{e \times n}$ on stage l. Summing over all possible assignments from stage $\lfloor n/e \rfloor + 1$ to n yields the total expected reward of $\frac{e-1}{e^2} \times OPT$. □

6.3 Geometric Model

Assume that the task duration follows a geometric distribution with parameter r: Once a task is assigned to the worker, the number of stages that the worker is busy follows a geometric distribution, with the parameter independent of the task value and the worker's success rate. This model is referred to as the Geometric Model. Let X denote the distribution of the task values and n denote the number of tasks and workers. Without loss of generality, assume that $p_i \geq p_j$ for $i \geq j$.

6.3.1 Task Distribution Known

Consider stage $i + 1$ with $n - i$ remaining tasks. Since there are n sequentially arriving tasks, the optimal algorithm assigns the remaining $n - i$ tasks to the $n - i$ remaining best workers. Therefore, if the number of remaining workers at stage $i + 1$ is more than $n - i$, the algorithm bypasses the workers that are not among the $n - i$ with the largest success rates. However, after assigning the $(i + 1)^{th}$ task, the problem can be described as SSAP with random success rates (referred to as Doubly Stochastic Sequential Assignment Problem or DSSAP) since the workers success rates in the next stages depend on the random event of previously assigned workers becoming re-available. Therefore, to find the optimal policy, we define a surrogate DSSAP, where the assignment problem consists of workers with random success rates.

Similar to the DSSAP, backward induction is used to find the optimal assignment policy. First, the expected reward of the last stage is computed. Assuming that the best available worker prior to arrival of the n^{th} task has a success rate of p_j, the expected success rate of the worker assigned to the n^{th} task is given by $E[Q_{1,n}(S_1)] = r \times p_1 + r(1 - r) \times p_2 + ... + (1 - r)^{j-1} \times p_j$, where $Q_{i,j}(S_{n-j+1})$ denotes the (random) success rate of the i^{th} $(i = 1, 2, ..., n - j + 1)$ best worker at stage $j \in \{1, 2, ..., n\}$ given the set of $n - j + 1$ best remaining workers S_{n-j+1}. The expected reward of the last stage is then given by $E[X]E[Q_{1,n}]$. Note that in this example $S_{n-j+1} = \{P_j\}$.

Assume that the two best available workers at stage $n - 1$ have success rates p_i and p_j, where $p_i \geq p_j$. If we assign the $(n - 1)^{th}$ task to the worker with success rate p_i, then with probability r the n^{th} task is assigned to the worker with success rate p_1, with probability $r(1 - r)$, the task is assigned to p_2, and if no worker with a better success rate becomes available in the last stage (which happens with probability $(1 - r)^{j-1}$), the task is assigned to the worker with success rate p_j. Similarly, we can compute the expected success rate assigned to the last task if the $(n - 1)^{th}$ task is assigned to p_j.

The optimal algorithm is computationally challenging. Therefore, an efficient approximation algorithm for the problem is now proposed. Lemma 14 provides an expression for the maximum expected reward that any policy might achieve for the SSAP with Geometric task duration.

Lemma 14. *The maximum expected reward is upper bounded by*

$$\sum_{i=n-\lceil nr \rceil}^{n} a_{i,n+1}p_1 + \sum_{i=\lceil nr \rceil-\lceil 2nr \rceil}^{n-\lceil nr \rceil} a_{i,n+1}p_2 + ... + \sum_{i=1}^{\lceil nr \rceil} a_{i,n+1}p_{\lceil \frac{1}{r} \rceil} \tag{6.1}$$

Proof: Consider the SSAP with the workers success rates p_1, p_2, ..., $p_{\lceil \frac{1}{r} \rceil}$. Assume that there are $\lceil nr \rceil$ workers with each of the mentioned success rates. The maximum expected reward of this SSAP, which is given by (6.1), provides an upper bound for the problem with Geometric task duration. □

Consider the $\lceil \frac{1}{r} \rceil$ workers with the best success rates. Divide the n stages to $\lceil nr \rceil$ intervals: One from stage 1 to stage $\lceil \frac{1}{r} \rceil$, second one from stage $\lceil \frac{1}{r} \rceil + 1$ to stage $2\lceil \frac{1}{r} \rceil$, and so on. Assign tasks of each interval to workers by defining interval thresholds based on the optimal policy of SSAP: If there are k remaining tasks to the end of the current interval, compute the thresholds $a_{i,k}$ and assign the task to worker j if the task value is in the j^{th} interval and worker j is available.

Theorem 21. *The proposed policy achieves an expected reward of at least*

$$1/e \times (1 - 1/e) \times \lceil nr \rceil \sum_{i=1}^{\lceil \frac{1}{r} \rceil} a_{i,\lceil \frac{1}{r} \rceil+1}p_{\lceil \frac{1}{r} \rceil-i+1} \tag{6.2}$$

Proof: First, notice that $\lceil nr \rceil \sum_{i=1}^{\lceil \frac{1}{r} \rceil} a_{i,\lceil \frac{1}{r} \rceil+1}p_{\lceil \frac{1}{r} \rceil-i+1}$ is the total expected reward if the workers are all available at the beginning of each interval of length $\lceil \frac{1}{r} \rceil$. Assume that the distance between any two tasks that must be assigned to the same worker is exactly $\lceil \frac{1}{r} \rceil$. Given this, the probability that the worker becomes available for future assignments (and hence, achieves the same reward as when all workers are available at the beginning of each interval) is equal to the probability that the number of stages that a worker is not available is less than its expected value. This probability is given by

$$\sum_{i=0}^{1/r-1} (1-r)^i \times r = 1 - (1-r)^{1/r}, \tag{6.3}$$

which converges to $1 - 1/e$.

The probability that the distance (i.e., the number of stages) between two tasks that must be assigned to the same worker is at least $\lceil \frac{1}{r} \rceil$ is given by

$$(\frac{\lceil \frac{1}{r} \rceil - 1}{\lceil \frac{1}{r} \rceil})^{\lceil \frac{1}{r} \rceil - 1} \geq (1 - \frac{1}{\lceil \frac{1}{r} \rceil})^{\lceil \frac{1}{r} \rceil}, \tag{6.4}$$

which converges to $1/e$. □

6.3.2 Tasks with a Random Arrival Order

Assume that tasks have a random arrival order. Again, a worker that is assigned to a task would be available in any of the next stages with probability p. We study the performance of a policy very similar to the BM Algorithm in this problem.

In contrast to the Secretary Problem, the optimal offline policy of this problem has a random parameter r. First, we derive an upper bound on the expected reward of the optimal offline policy. Policy A is not the optimal offline policy, but provides an upper bound on the reward of any offline policy for this problem.

```
Algorithm 10 Policy $A$
Input: Task values $X = \{x_1, x_2, ..., x_n\}$, workers success rates $P = \{p_1, p_2, ..., p_n\}$, parameter $p$
Output: Assign the best $\lceil nr \rceil$ tasks to $p_1$, the second best $\lceil nr \rceil$ tasks to $p_2$, and so on.
```

Lemma 15. *The expected reward of Policy A is given by*

$$E_A[X, P, r] = \sum_{i=1}^{\lceil \frac{1}{r} \rceil} \sum_{j=1}^{\lceil nr \rceil} p_i X_{((i-1)\times\lceil nr \rceil + j)} \tag{6.5}$$

Lemma 16. *Policy A provides an upper bound on the expected reward of any offline policy.*

Proof: Consider the following problem: Assume that there are n arriving tasks and only one worker with success rate $p_{(1)}$. Upon assigning a task to the worker, the worker becomes available in the next stage with probability r. The maximum (offline) expected reward of this problem is given by $p_{(1)} \times (X_{(1)} + X_{(2)} + ... + X_{(\lceil nr \rceil)})$ since in expectation, the worker is available for nr stages and it is assigned to tasks with best workers in these stages. The right hand-side of (6.6) is equal to the optimal reward when all the best $\lceil \frac{1}{r} \rceil$ workers are assigned to the optimal tasks. □

Next, we define another assignment policy, referred to as Policy B, and compare its performance to Policy A. Note that Policy B is similar to the optimal policy of the problem discussed in Theorem 1 when the task duration is fixed. The only difference is that here we consider a distance equal to the expected task duration between each two tasks that must be assigned to the same worker.

```
Algorithm 11 Policy $B$
Input: Task values $X = \{x_1, x_2, ..., x_n\}$, workers success rates $P = \{p_1, p_2, ..., p_n\}$, parameter $p$
Divide tasks to $\lceil \frac{1}{r} \rceil$ groups, with the first group containing the best $\lceil nr \rceil$ tasks, the second group containing the second best $\lceil nr \rceil$ tasks, and so on.
Sort the tasks such that each two tasks of the same group has a distance of $\lceil \frac{1}{r} \rceil$ (i.e., tasks of the first group appear at stages $1, 1 + \lceil \frac{1}{r} \rceil, 1 + 2\lceil \frac{1}{r} \rceil, ...$; tasks of the second group appear at stages $2, 2 + \lceil \frac{1}{r} \rceil, 2 + 2\lceil \frac{1}{r} \rceil, ...$, and so on.)
Starting from the first stage, assign tasks of the $i^{th}$ group to the worker with the $i^{th}$ largest success rate if it is available.
```

Lemma 17. *Let $E_A[X, P, r]$ ($E_B[X, P, r]$) denote the total expected reward achieved by Policy A (Policy B) on the set of tasks X, set of workers P, and a Geometric Model with parameters r for the task duration. Then,*

$$E_B[X, P, r] \geq (1 - \frac{1}{e})E_A[X, P, r] \tag{6.6}$$

Proof: The proof is very similar to the proof of the first part of Theorem 21 since in the worst case, each task is assigned to the worker by Policy B if the task duration is smaller than its expected value. □

Now we define our proposed algorithm for the problem with a random arrival order of tasks and Geometric task duration. Note that the algorithm is very similar to the BM Algorithm and hence, is referred to as the BM Algorithm. The only difference is that the algorithm finds a matching on tasks observed so far by applying Policy B.

Algorithm 12 The BM Algorithm for Random Arrival Order of Tasks and Geometric Task Duration

```
Observe the first $\lfloor n/e \rfloor$ tasks: $\{x_1, x_2, ..., x_{\lfloor n/e \rfloor}\}$
From stage $l = \lfloor n/e \rfloor + 1$
  Apply Policy $B$ to $\{x_1, x_2, ..., x_l\} \cup \{p_1, p_2, ..., p_n\}$
  Let $p_j$ denote the worker's success rate assigned to $x_l$ by Policy $B$
  Assign task $l$ to worker $j$ if the worker is not previously assigned
```

Theorem 22. *The total expected reward achieved by the BM Algorithm is at least* $\frac{e-1}{e^2} E_B[X, P, r]$.

Proof: The proof is similar to the proof of Theorem 1. First, it is easy to see that due to the random arrival order of tasks, each assignment by Policy B achieves an expected reward of $\frac{E_B[X,P,r]}{n}$. This means that the expected reward of each stage is at least $1/n$ of the total expected reward achieved by Policy B in the n-stage problem. Assume that Policy B assigns task l to worker j. Now, we should compute the probability that worker j is available on stage l. If the last task assigned to worker j by the BM Algorithm appeared at stage $l - \lceil \frac{1}{r} \rceil$ or earlier, then the probability that the worker is available is (at least) the same for the BM Algorithm and Policy B. This is true since Policy B orders tasks such that each two tasks that must be assigned to the same worker has a distance of $\lceil \frac{1}{r} \rceil$. The event that the last task assigned to worker j appeared at stage $l - \lceil \frac{1}{r} \rceil$ or earlier happens with probability $(\frac{\lceil \frac{1}{r} \rceil - 1}{\lceil \frac{1}{r} \rceil})^{\lceil \frac{1}{r} \rceil - 1}$, which converges to $1/e$. Again, summing over all possible assignments from stage $\lfloor n/e \rfloor + 1$ to n completes the proof. □

6.4 Random Arrival of Tasks

This section extends the results of the previous section to the case that tasks arrive with a certain probability on each stage. Assume that tasks have a random arrival order. The optimal policy for this case has a training phase of length n/e and is e-competitive. Now assume that at each time interval, a task arrives with probability u.

Consider the same assignment policy: Observe the tasks until the stage n/e. Then, use them as a training phase to assign tasks at stages $n/e + 1$ to n. Notice the difference between this policy and the previous one. Here the number of tasks observed up to stage n/e is a (binomial) random number with and expected value of $u \times n/e$.

Theorem 23. *The BM Algorithm is optimal.*

Proof: First, note that for optimality, we should prove that the algorithm is e-competitive. Since the case of $u = 1$ is a special case of this problem, which is itself a generalization of the Secretary problem. Therefore, any algorithm with a better competitive ratio, yields a better competitive ratio for the secretary problem, which is a contradiction. We follow the same steps as the proof of Lemma 1 in [34].

Let $R(x_l)$ denote the reward of assigning the l^{th} arriving task and $R_M(\{x_1, x_2, ..., x_l\})$ denote the maximum offline reward of assigning the first l tasks to workers. Then, given that a task arrives at stage l, the expected reward achieved by the task in the offline matching satisfies

$$E[R(x_l)] \geq \frac{E[R_M(\{x_1, x_2, ..., x_l\})]}{ul}$$

This is a direct result of the random arrival order of tasks. This also yields

$$E[R(\{x_1, x_2, ..., x_l\})] \geq \frac{l \times E[R_M(\{x_1, x_2, ..., x_n\})]}{n},$$

where $E[R_M(\{x_1, x_2, ..., x_n\})] = OPT$ is the maximum offline reward of the n-stage problem. Therefore, given that a task appears at stage l, the expected reward of assigning it is at least $1/un$ times the maximum offline reward:

$$E[R(x_l)] \geq \frac{OPT}{un}$$

Since a task appears at each stage with probability u, the expected reward of stage u is at least $1/n$ times the maximum offline reward.

Consider stage $k \in \{\lceil n/e \rceil, \lceil n/e \rceil + 1, ..., n\}$. The probability that a task appearing at this stage is actually assigned to a worker is equal to the probability that the worker determined by the matching policy is available; i.e., it is not assigned to a task in one of the stages $l = \lceil n/e \rceil, \lceil n/e \rceil + 1, ..., k - 1$. Due to the random arrival order, the probability that a task is assigned at stage l given that all l tasks appeared is at most $1/l$. Now conditioning on the number of tasks appeared at stage l, this probability is given by

$$\sum_{i=1}^{l} \frac{1}{i} \binom{l-1}{i-1} (1-u)^{k-i} u^i = \frac{1}{l} \sum_{i=1}^{l} \binom{l}{i} (1-u)^{k-i} u^i \leq \frac{1}{l} \tag{6.7}$$

The rest of the proof follows in the same way as [34]. $\square$

Note that Theorem 4 has another simple intuitive proof. In fact, since tasks have a random arrival order and we assume that each task arrives at each stage with probability u, independent of other stages, we could incorporate this probability into the random arrival order. Therefore, the random arrival of tasks does not change anything in optimality of the threshold policy.

6.5 Summary

This chapter proposes assignment policies for a generalization of SSAP, with a worker capable of performing more than one task. This problem is referred to as Dynamic Sequential Assignment. The number of stages that a worker is not available due to prior assignment is considered to be the task duration. SSAP is a special case of the Dynamic Sequential Assignment with a task duration larger than the number of remaining stages of the problem.

The Dynamic Sequential Assignment relaxes the one-time assignment of the basic formulation of SSAP and has applications in several areas such as the online matching markets. In an online matching market, each of the bidders might bid and buy more than one item.

Assignment policies are proposed for two models. The first model assumes a fixed task duration, independent of the task value and the worker's success rate. An assignment policy is proposed and its performance is evaluated. This result is extended to a memoryless model for task duration. It is assumed that the task duration follows a geometric distribution. Assuming randomness in task duration makes the analysis of the problem more complex. In particular, the optimal offline policy becomes computationally challenging. In order to overcome this problem, an upper bound on the optimal offline reward is computed. Then, an approximation algorithm for the optimal offline reward is proposed. This algorithm is used to design and analyze an online assignment policy.

The intuition behind the proposed algorithms is to divide the set of arriving tasks to groups of length equal to the expected task duration. The arriving tasks in each group are then assigned to workers using a training phase. While the proposed algorithms are analyzed for fixed and memoryless task duration, similar ideas can be used for more complex models. However, the competitive ratio depends on the random structure of the task duration and might be a function of the expected task duration.

CHAPTER 7

CONCLUSION AND FUTURE WORK

The Sequential Stochastic Assignment Problem (SSAP) deals with finding policies for matching a set of online arriving elements to available entities so as to maximize a reward function. The online arriving elements are referred to as tasks and the entities that are assigned to tasks are referred to workers. Each task is associated with a specific value, and each worker has a success rate. The reward of each matching is defined as the product of the task value and the worker's success rate. In the basic formulation of SSAP, it is assumed that the number of tasks and workers are known, the task values are independently drawn from a given distribution, and the workers' success rates are constant.

This thesis consists of several essays on sequential assignment problems. The first part studies the Doubly Stochastic Sequential Assignment Problem (DSSAP). DSSAP assumes that the workers' success rates are random, taking new values upon each task arrival. While SSAP can be considered as a generalization of Secretary Problem, where each hired candidate must be assigned to a distinct position, DSSAP is an Edge-Weighted Online Bipartite Matching Problem. Due to randomness in task values and the workers' success rates, DSSAP deals with several random parameters at each stage of the problem. Therefore, the optimal assignment policy is computationally challenging with arbitrary success rate distributions. This dissertation studies several special cases of DSSAP, and then proposes the optimal assignment policy using a backward induction. An approximation algorithm that achieves a fraction of the optimal reward in a polynomial time is also proposed. The approximation algorithm is a ranking algorithm that gives priority to workers with larger expected success rates. This improves the computational complexity since the expected reward of many possible matchings are not computed when the task is assigned to a higher priority worker.

The second part of this dissertation studies the Generalized Sequential Stochastic Assignment

Problem (GSSAP). GSSAP assumes no prior information on task values. Several constant competitive algorithms are proposed. With no assumption on task values, an assignment policy uses the first stages of the problem as a training phase to compute thresholds. Tasks arriving in the training phase can be assigned to workers by recursively applying the same procedure (i.e., defining a new training phase to compute thresholds and then use them to assign tasks to workers).

The linear programming technique formulates the sequential assignment problem as a linear program and uses this formulation to analyze the performance of assignment policies. The constraints of the linear program are defined such that each feasible solution of the linear program corresponds to an algorithm for the online matching problem. Moreover, the objective function yields an upper bound on the expected reward achieved by any assignment policy. Therefore, each solution of the linear program provides a bound on the reward achieved by any algorithm for the sequential assignment problem. Moreover, the linear programming formulation can be used to analyze extensions of the problem by simple changes in the objective function and constraints.

The Dynamic Sequential Assignment Problem seeks to maximize the total expected reward for a generalization of SSAP, where each worker might be assigned to several tasks. In many applications of online matching markets, one side of the market is available for more than one matching. The Dynamic Sequential Assignment finds assignment policies for such matching markets by various assumptions on the availability of workers for future assignments. First, it is assumed that upon assigning a task, the worker would not be available for a fixed time, referred to as the task duration. Then, this is generalized to a memoryless model, where the task duration follows a geometric random variable. The proposed algorithms divide the n-stage problem to several periods, with the length of each period equal to the (expected) task duration. Tasks arriving in each period of this length are assigned to workers by applying the optimal algorithm of the n-stage problem. The performance of the proposed algorithms are analyzed by computing the probability that the distance between any two tasks that must be assigned to the same worker is at least as large as the expected task duration. Moreover, due to randomness in availability of workers in the memoryless model, the reward is multiplied by the probability that a worker is available for a possible assignment. Note that similar ideas can be used for other random structures for the task duration. However, the ratio of the achieved reward to the optimal offline reward might be different.

Relaxing the assumptions of SSAP generalizes the basic formulation to model the real-world problems with higher accuracy. For example, DSSAP generalizes SSAP by assuming that the workers' success rates might change during the assignment process. GSSAP assumes a random arrival order of tasks, which is quite realistic in many applications. The Dynamic Sequential Assignment Problem provides a framework for modeling sequential assignment problems, where each element might be matched to several entities.

While various online matching problems have been studied, there are many questions to be addressed in future work. For example, one of the main challenges in SSAP is to design algorithms for the problem in a continuous-time model. The continuous-time model provides a more realistic framework for studying online matching markets, where the price of each item is a function of time. However, assuming that the reward of each assignment is a time-dependent function makes the problem significantly more complex.

The sequential assignment problem with unknown number of elements is another possible research direction. This dissertation derives bounds on the performance of assignment policies for the sequential assignment problem when the number of elements is selected by an adversary from a given set. Due to the dependence of the size of the training data set on the number of elements, designing algorithms with reasonable competitive ratio for the problem with limited information on the number of online arriving elements is very challenging.

Designing matching algorithms for the Dynamic Sequential Assignment with new models for availability of workers is a future research direction. While similar ideas to those proposed in this dissertation can be applied to new models, the performance of the proposed algorithms needs to be analyzed. The Dynamic Sequential Assignment becomes very challenging when the task duration is a function of the task value and the worker's success rate. Bounding the performance of assignment policies for the Dynamic Sequential Assignment Problem using the linear programming formulation is an interesting application of the linear programming technique. Moreover, designing algorithms for the problem with/without some assumptions on the relation between the task duration and value is another research direction.

While the Dynamic Sequential Assignment Problem defines the availability of workers as a function of the task duration, other models for reassigning workers to tasks can be studied. For example, a queueing model assumes a capacity for the maximum number of tasks performed by each worker. An algorithm for this model might assign several successive tasks to the same worker. The objective can be defined as maximizing the total expected reward while satisfying some constraints on the waiting time of each element.

REFERENCES

[1] Agarwal, S. and Zizhuo, W. and Ye, Y. 2014. A Dynamic Near-Optimal Algorithm for Online Linear Programming. *Operations Research*. 62 (4). 876–890.

[2] Albright, S. C. and Derman, C. 1972. Asymptotic Optimal Policies for Stochastic Sequential Assignment Problem. *Management Science*. 19 (1). 46–51.

[3] Albright, S. C. 1974. Optimal Sequential Assignment with Random Arrival Times. *Management Science*. 21 (1). 60–67.

[4] Albright, S. C. 1974. A Markov-Decision-Chain Approach to a Stochastic Assignment Problem. *Operations Research*. 22 (1). 61–64.

[5] Albright, S. C. 1977. A Bayesian Approach to a Generalized House Selling Problem. *Management Science*. 24 (4). 432–440.

[6] Apostol, T. M. 1967. Calculus, Vol. 1: One-Variable Calculus with an Introduction to Linear Algebra. *John Wiley and Sons, second edition*. 978-0-471-00005-1.

[7] Babaioff, M. and Dinitz, M. and Gupta, A. and Immorlica, N. and Talwar, K. 2009. Secretary Problems: Weights and Discounts. *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*. 1245–1254.

[8] Babaioff, M. and Immorlica, N. and Kempe, D. and Kleinberg, R. 2008. Online Auctions and Generalized Secretary Problems. *ACM SIGecom Exchanges*. 7 (2).

[9] Baharian, G. and Jacobson, S. H. 2013. Limiting Behavior of the Stochastic Sequential Assignment Problem. *Naval Research Logistics*. 60 (4). 321–330.

[10] Baharian, G. and Jacobson, S. H. 2013. Stochastic Sequential Assignment Problem with Threshold Criteria. *Probability in the Engineering and Informational Sciences*. 27 (3). 277–296.

[11] Baharian, G. and Jacobson, S. H. 2014. Limiting behavior of the target-dependent stochastic sequential assignment problem. *Journal of Applied Probability*. 51 (4). 943–953.

[12] Bertsimas, D. and Tsitsiklis, J. N. 1997. Introduction to Linear Optimization. *Athena Scientific*. ISBN:1886529191.

[13] Bloch, F. and Houy, N. 2012. Optimal assignment of durable objects to successive agents. *Economic Theory*. 51 (1). 13–33.

[14] Buchbinder, N. and Kamal, J. and Singh, M. 2013. Secretary Problems via Linear Programming. *Mathematics of Operations Research*. 39 (1). 190–206.

[15] Chow, Y. S. and Moriguti, S. and Robbins, H. and Samuels, S. M. 1964. Optimal selection based on relative rank. *Israel Journal of Mathematics*. 2. 81–90.

[16] Chun, Y. H. and Sumichrast, R. T. 2006. A rank-based approach to the sequential selection and assignment problem. *European Journal of Operational Research*. 174 (2). 1338–1344.

[17] David, I. and Levi, O. 2004. A new algorithm for the multi-item exponentially discounted optimal selection problem. *European Journal of Operational Research*. 153 (3). 782–789.

[18] David, I. and Yechiali, Y. 1995. One-attribute Sequential Assignment Match Processes in Discrete Time. *Operations Research*. 43 (5). 879–884.

[19] Derman, C. and Lieberman, G. J. and Ross, S. M. 1972. A Sequential Stochastic Assignment Problem. *Management Science*. 18 (7). 349–355.

[20] Derman, C. and Lieberman, G. J. and Ross, S. M. 1975. A Stochastic Sequential Allocation Model. *Operations Research*. 23 (6). 1120–1130.

[21] Derman, C. and Lieberman, G. J. and Ross, S. M. 1979. Adaptive Disposal Models. *Naval Research Logistics Quarterly*. 26 (1). 33–40.

[22] Devanur, N. R. and Hayes, T. P. 2009. The Adwords Problem: Online Keyword Matching with Budgeted Bidders under Random Permutations. *Proceedings of the 10th ACM Conference on Electronic Commerce*. 71–78.

[23] Dynkin, E. B. 1963. The Optimum Choice of the Instant for Stopping a Markov Process. *Sov. Math. Dokl*. 4.

[24] Enns, E. G. 1970. The optimum strategy for choosing the maximum of N independent random variables. *Unternehmensforschung*. 14. 89–96.

[25] Freeman, P. R. 1983. The Secretary Problem and Its Extensions: A Review. *International Statistical Review*. 51. 189–206.

[26] Gale, D. and Shapley, L. S. 1962. College Admissions and the Stability of Marriage. *The American Mathematical Monthly*. 69 (1). 9–15.

[27] Glasser, K. S. and Holzsager, R. 1983. The d Choice Secretary Problem. *Communications in Statistics*. 2 (3). 177–199.

[28] Glen, A. G. and Leemis, L. M. and Drew, J. H. 2004. Computing the Distribution of the Product of Two Continuous Random Variables. *Computational Statistics and Data Analysis*. 44 (3). 451–464.

[29] Gianini, J. and Samuels, S. M. 1976. The Infinite Secretary Problem. *Annals of Probability*. 4. 418–432.

[30] Gianini-Pettitt, J. 1979. Optimal Selection Based on Relative Ranks with a Random Number of Individuals. *Adv. Appl. Prob*. 11. 720–736.

[31] Feng,T. and Hartman, J. C. 2013. The Sequential Stochastic Assignment Problem with Postponement Options. *Probability in the Engineering and Informational Sciences*. 27 (1). 25–51.

[32] Karp, R. M. and Vazirani, U. V. and Vazirani, V. V. 1990. An Optimal Algorithm for On-line Bipartite Matching. *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*. 352–358.

[33] Kennedy, D. P. 1986. Optimal Sequential Assignment. *Mathematics of Operations Research*. 11 (4). 619–626.

[34] Kesselheim, T. and Radke, K. and Tonnis, A. and Vocking, B. 2013. An Optimal Online Algorithm for Weighted Bipartite Matching and Extensions to Combinatorial Auctions. *Algorithms-ESA*. 589–600.

[35] Khatibi, A. and Baharian, G. and Kone, E. R. and Jacobson, S.H. 2014. The Sequential Stochastic Assignment Problem with Random Success Rates. *IIE Transactions*. 46 (11). 1169–1180.

[36] Khatibi, A. and Jacobson, S. H. 2015. Doubly Stochastic Sequential Assignment Problem. *Naval Research Logistics*. 63 (2). 124–137.

[37] Khatibi, A. and Jacobson, S.H. 2015. Generalized Sequential Stochastic Assignment Problem. *Technical Report, University of Illinois.*

[38] Kleinberg, R. 2005. A Multiple-choice Secretary Algorithm with Applications to Online Auctions. *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. 630–631.

[39] Kittur, A. and Chi, E. H. and Suh, B. 2008. "Crowdsourcing user studies with Mechanical Turk." *Proceedings of the SIGCHI conference on human factors in computing systems*. 453–456.

[40] Korula, N. and Pál, M. 2009. Algorithms for Secretary Problems on Graphs and Hypergraphs. *Proceedings of the 36th Internatilonal Colloquium on Automata, Languages and Programming: Part II*. 508–520.

[41] Lee, A. J. and Jacobson, S. H. 2011. Sequential Stochastic Assignment under Uncertainty: Estimation and Convergence. *Statistical Inference for Stochastic Processes*. 14 (1). 21–46.

[42] Lindley, D. V. 1961. Dynamic programming and decision theory. *Applied Statistics*. 10 (1). 39–51.

[43] McLay, L.A. and Jacobson, S.H. and Kobza, J.E. 2006. A Multilevel Passenger Screening Problem for Aviation Security. *Naval Research Logistics*. 53 (3). 183–197.

[44] McLay, L.A. and Jacobson, S.H. and Nikolaev, A.G. 2009. A Sequential Stochastic Passenger Screening Problem for Aviation Security. *IIE Transactions*. 41 (6). 575–591.

[45] Mehta, A. and Saberi, A. and Vazirani, U. and Vazirani, V. 2007. AdWords and Generalized Online Matching. *J. ACM*. 54 (5). 1–20.

[46] Mucci, A. G. 1973. Differential Equations and Optimal Choice Problems. *Annals of Statistics*. 1 (1). 104–113.

[47] Nakai, T. 1981. Sequential Stochastic Assignment Problem With Rejection. *Journal of Information and Optimization Sciences*. 2 (2). 169–180.

[48] Nakai, T. 1986. A Sequential Assignment Problem in a Partially Observable Markov Chain. *Mathematics of Operations Research*. 11 (2). 230–240.

[49] Nikolaev, A. G. and Jacobson, S. H. 2010. Stochastic Sequential Decision-making with a Random Number of Jobs. *Operations Research*. 58 (4P1). 1023–1027.

[50] Nikolaev, A. G. and Jacobson, S. H. and McLay, L. A. 2007. A Sequential Stochastic Security System Design Problem for Aviation Security. *Transportation Science*. 41 (2). 182–194.

[51] Oveis Gharan, S. and Vondrak, J. 2011. On Variants of the Matroid Secretary Problem. *Algorithms–ESA*. 335–346.

[52] Presman, E. L. and Sonin, I. M. 1972. The Best Choice Problem For a Random Number of Objects. *Theory of Probability and Its Applications*. 17 (4). 657–668.

[53] Rasmussen, W. T. and Pliska, S. R. 1975. Choosing the maximum from a sequence with a discount function. *Applied Mathematics and Optimization*. 2. 279–289.

[54] Righter, R. L. 1987. The Stochastic Sequential Assignment Problem with Random Deadlines. *Probability in the Engineering and Informational Sciences*. 1 (2). 189–202.

[55] Righter, R. L. 1989. A Resource Allocation Problem in a Random Environment. *Operations Research*. 37 (2). 329–338.

[56] Sakaguchi, M. 1983. A Sequential Stochastic Assignment Problem with an Unknown Number of Jobs. *Mathematika Japonica*. 29 (2). 141–152.

[57] Sakaguchi, M. 1978. Dowry problems and OLA policies. *Rep. Statist. Appl. Res. JUSE*. 25. 124–128.

[58] Smith, M. H. A Secretary Problem with Uncertain Employment. *Journal of Applied Probability*. 12. 620–624.

[59] Su, X. and Zenios, S. A. 2005. Patient Choice in Kidney Allocation: A Sequential Stochastic Assignment Model. *Operations Research*. 53 (3). 443–455.

[60] Tamara, M. 1991. A secretary problem with uncertain employment and best choice of available candidates. *Operations Research*. 39. 274–284.