GROUP SCHEDULING AND ASSIGNMENT:
COMPLEXITY AND ALGORITHMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Hooyeon Lee

December 2016

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Virginia V. Williams) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Ashish Goel)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Yoav Shoham)

Approved for the Stanford University Committee on Graduate Studies

Abstract

Acknowledgments

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Chapter 2

Complexity of Group Scheduling Problem I

Scheduling an event for a group of invitees is a frustrating task; it tends to be tedious and time consuming. A typical scheduling process can be described as iterative approval voting: An event organizer selects a candidate set of date/time options, and asks invitees to respond with their availability. Given the responses, the organizer chooses an agreeable option and announces it, or she may repeat the process by proposing another set of options if no feasible option is found. Naturally the organizer and her invitees wish to reach an agreement within a small number of iterations and proposed options – the more iterations and proposed options there are, the more laborious a scheduling process becomes.

There exist several software tools that are designed to help an event organizer handle a scheduling process more efficiently – one of the most well-known tools is Doodle [1]. In Doodle, an organizer can simply list as many date/time options as she likes, and each invitee is asked to respond with her availability. Essentially, invitees participate in approval voting upon all proposed options. Hence if too many options are proposed, then invitees are given the burden of answering them all. This often leads to undesired behaviors of invitees such as herding or procrastination, instead of honest, quick responses [46]. On the other hand, if the organizer proposes only few options, there may not exist an agreeable outcome after

[1] http://www.doodle.com

all, which may result in another iteration of proposals and responses. In fact, surveys find that the most challenging part of group scheduling is due to "chasing people who do not answer" and "finding a suitable time." [2]

Doodle has many practical advantages. One of them is its simplicity – invitees simply need to approve a subset of options based on their availability. Another is a short duration of scheduling process – the duration it takes until the last invitee responds and the meeting time is settled. Each invitee needs to respond only once, which limits the degree to which the process is hijacked by invitees' delayed responses. But Doodle has many drawbacks, even in this idealized setting. In particular, Doodle forces the invitees to examine many potential date/time options. This can be inconvenient not so much due to the effort involved (though that is a factor), but mostly because invitees need to block their available slots off until a option is announced. We use the expected number of time slots floated by the organizer as a proxy for this inconvenience.

To avoid incurring much inconvenience, the organizer can select only a few options, and poll the invitees about those. If a feasible option is not found among them, then repeat with another batch of date/time options. We call this broad class of mechanisms *B-Doodle* (for "Batched Doodle"). Clearly, Doodle is a special case with one batch consisting of all options. Another extreme case is the OAAT (one-at-a-time) mechanism, in which the organizer tests a single option at each iteration. Doodle minimizes the total number of iterations, whereas OAAT minimizes (expected) inconvenience caused by the scheduling process. In-between lie many other mechanisms, with different batching schemes, that trade off time against inconvenience differently.

Figure 2.1 illustrates this via a simple example. In this scenario there are six options, four invitees, each of whom is available at each option independently with probability .8. The figure depicts a scatter plot of all 32 different B-Doodle mechanisms, including Doodle and OAAT. The vertical axis depicts the expected number of rounds to determine the option, and the horizontal axis the expected inconvenience.

We can clearly observe a time-inconvenience Pareto frontier in this plot. If in addition there is an overall cost function combining time and inconvenience, one can identify an

[2] http://en.blog.doodle.com/2012/07/26/new-findings-a-small-number-of-initiators-organize-most-of-the-meetings/

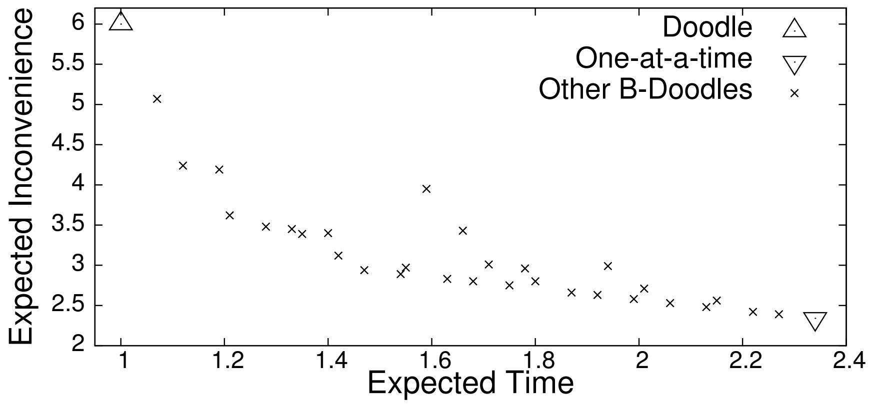

Figure 2.1: A scatter plot of B-Doodle Mechanisms.

optimal B-Doodle mechanism along this frontier as illustrated in Figure 2.2. In this example, if the overall cost is the linear function 3·Time+Inconvenience, the optimal mechanism happens to be the "Half-n-half" mechanism; this mechanism sends out 3 options in the first batch, and if no feasible time slot is found then sends out the remaining 3 options in the next batch.

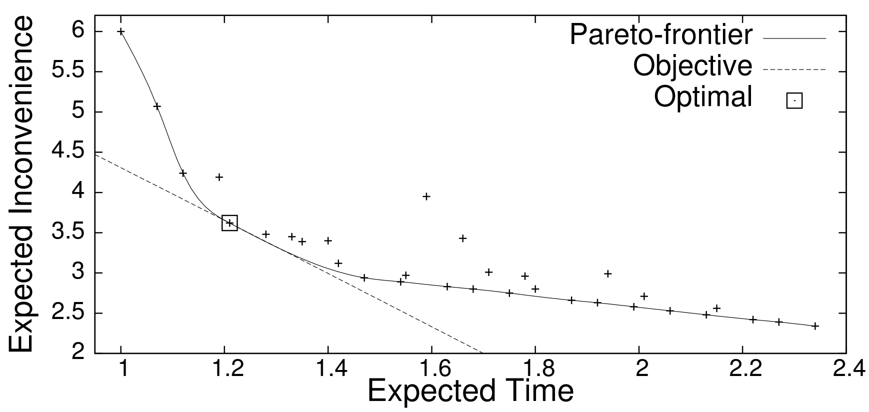

Figure 2.2: Pareto-frontier and objective function.

In this chapter we will investigate the difficulty of determining the optimal B-Doodle mechanism, and to what degree it improves on the simple Doodle in realistic scenarios. Under the assumption that the set of events in which an invitee is available for a particular option is mutually independent, we provide an efficient recursive algorithm for computing the optimal batching scheme, for a broad class of objective functions of time and inconvenience, including the linear combination as in the above example.[3]

Lastly, we assume that the event organizer is given probability estimates on availability of agents, but it is not clear how one can obtain such probabilities. Probability estimation is an interesting and challenging research question on its own, and we do not attempt to solve the question in this work. However, we provide several plausible methods for estimating the probabilities in the context of group scheduling, which can enable our model and algorithm to be deployed as a real-world application in the future. In the psychology literature, Mann et al. found that cultural differences between the Western, individualistic countries (such as the United States) and the Eastern, collectivistic countries (such as China and Japan) lead to different behaviors of respondents when it comes to a group-decision making process [35]. More recently, Reinecke et al. analyzed more than 1.5 million Doodle date/time polls from 211 countries, and confirmed similar findings regarding time perception and group's behavior [39]. Among others, they found that "in comparison to predominantly individualist societies, poll participants from collectivist countries respond earlier, agree to fewer options but find more consensus," which agrees with the findings of Mann et al. Besides the cultural differences, Doodle's own surveys on event scheduling found that people tend to respond to the scheduling surveys on Mondays, while Monday is the least popular day for having a meeting. [4] We believe that these studies and findings can be used to design a reasonable estimator for availability of agents, by utilizing the features that are known to be crucial – such as demographics of the group and purpose of the event being scheduled. Recent work by Zou et al. analyzed over 340,000 Doodle polls data to study behavioral patterns of the users, and they were able to identify response functions that match the response patterns observed in the real data [46]. We believe that a similar approach can be taken to tackle the problem of probability estimation in the context of group scheduling.

[3] As another, more minor generalization of the problem, we will allow that the organizer is content with only a fraction of the invitees attending, not necessarily requiring everyone be available. This fraction will be a part of input to the problem; of course, as a special case, 1 implies full attendance.

[4] http://en.blog.doodle.com/2012/05/23/mondays-for-planning-busy-weekends/

2.1 Notation and Definitions

We mentioned two dimensions of optimality in the scheduling process: Time and Inconvenience. Time captures the duration of the scheduling process and Inconvenience measures how much inconvenience is caused for each invitee during the scheduling process. In this section we formally define the class of B-Doodle mechanisms and the Batched Doodle Problem (BDP).

Consider a setting where there are n invitees and s date/time options from which the organizer can choose to propose. The organizer wishes to find a *feasible* date/time option that works for at least $\lceil f \cdot n \rceil$ invitees (we call $f \in [0,1]$ the *feasibility threshold*) or determines that there is no feasible option. We model the uncertainty of availability of invitees as a random matrix as follows. Consider a matrix of mutually independent Bernoulli random variables where rows represent invitees and columns represent a set of date/time options. Given probability of success for each Bernoulli random variable, the organizer can "inspect" a batch of columns at once to know of the realization of those entries. This corresponds to polling invitees' availability for a batch of date/time options. The organizer wishes to determine with certainty whether the matrix contains a feasible column or it does not. Using the random matrix model, we first investigate interesting properties of an optimal solution, and then discuss how it performs better than the classic Doodle under various settings.

Definition 1 (Feasibility)**.** Let A be a matrix whose entries are from $\{0,1\}$, and refer to the entry of A at row r and column c as $a_{r,c}$. We say that a column c of A is *feasible* if it consists only of 1's (otherwise it is *infeasible*). We say that A is *feasible* if it contains at least one feasible column (otherwise it is *infeasible*).

The organizer iteratively sends out a batch of options until a feasible option is found. In our setting Time spent by the scheduling process is measured by the number of iterations and Inconvenience caused is measured by the number of options that have been sent out by the organizer. Let us define a class of B-Doodle mechanisms that describe how the organizer sends out options in each iteration.

Definition 2 (B-Doodle Mechanism)**.** Let $S = \{1,2,\ldots,s\}$ be a set of s options. We define a B-Doodle mechanism for S as an ordered partition of S. Let $B = \langle S_1, S_2, \ldots, S_m \rangle$ be a partition of S into m subsets such that $S_j \neq \emptyset$, $S_j \cap S_k = \emptyset$ for $j \neq k$, and $\cup_{l=1}^{m} S_l = S$ where $1 \leq j,k \leq m$. We define $b_j = |S_j|$ for all $j \leq m$ and call b_j the size of the j-th batch. We write B_m to emphasize that B has m batches.

We interpret a B-Doodle mechanism B as follows: the organizer sends out options in S_1 during the first iteration. If a feasible option is found, the process ends. Otherwise she sends out the next batch, S_2, and so on. Note that there exist exponentially many B-Doodle mechanisms for any a given S (exponential in cardinality of S).

The goal is to find a B-Doodle mechanism that minimizes the expected cost of scheduling process, given an objective function of Time and Inconvenience. Earlier we considered a simple cost function that is a linear combination of the two. While this cost function fits in many realistic situations, we want to explore a larger class of cost functions. We define a class of cost functions that depend only on the number of iterations and batch sizes.

Definition 3 (Cost function)**.** A cost function c takes two integers j and b as arguments, and $c(j,b) > 0$ describes the aggregate cost of Time and Inconvenience that is incurred by sending out a batch of b options during the j-th iteration. We assume that cost is additive so that the overall cost of executing the first j iterations of $B_m = \langle S_1, S_2, \ldots, S_m \rangle$ is simply the sum, $\sum_{k=1}^{j} c(k, b_k)$, which we denote by $C(j, B_m)$; recall that $b_k = |S_k|$. A cost function c is said to be θ-*simple* if there exists some constant $\theta > 0$ such that $c(j,b) = \theta \cdot c(j-1,b)$ for all $j > 1$ and for all $b \geq 1$. We often drop θ and just state that a cost function is *simple*, for brevity.

It is reasonable to assume that cost is additive (with respect to iterations) because an additional iteration adds the time spent and inconvenience caused for the invitees.

Note that $C(j, B_m)$ is strictly increasing in j for any fixed B_m because $c(j,b) > 0$ for any j and b. While the class of simple cost functions may seem too restrictive, we present a few natural choices of cost functions that belong to the class of simple cost functions. In this work we assume that the underlying cost function is simple; further we assume that $c(j,b)$ for all $1 \leq j,b \leq s$ can be computed in polynomial time with respect to s.

Example 1 (Examples of simple cost functions)**.** The simplest choice of a cost function is a linear combination of Time and Inconvenience, parameterized by some constant $\alpha > 0$;

we call this a *linear* cost function. This cost function is 1-simple. Notice that the first term captures Time and the second captures Inconvenience in both expressions.

$$c_\alpha(j, b_j) = \alpha + b_j$$
$$C_\alpha(j, B_m) = (\alpha \cdot j) + \left(\sum_{k=1}^{j} b_k\right)$$

In some cases the organizer may want to penalize the mechanism for executing too many iterations by making later iterations to cost more than earlier iterations. The following describes such cost function, parameterized by some constant $\beta > 1$; we call this a *time-averse* cost function. Notice that the term β^j is strictly increasing as j increases (recall $\beta > 1$). This cost function is β-simple.

$$c_\beta(j, b_j) = \beta^j \cdot b_j$$
$$C_\beta(j, B_m) = \left(\sum_{k=1}^{j} \beta^k \cdot b_k\right)$$

In contrast if the organizer is interested in reducing the size of each batch because it may cause too much inconvenience, then the following cost function could be used, which is parameterized by some constant $\gamma > 1$; we call this an *inconvenience-averse* cost function which is 1-simple.

$$c_\gamma(j, b_j) = \gamma^{b_j}$$
$$C_\gamma(j, B_m) = \left(\sum_{k=1}^{j} \gamma^{b_k}\right)$$

We now formally define the Batched Doodle Problem.

Definition 4 (Batched Doodle Problem)**.** An instance of the Batched Doodle Problem (BDP) is a tuple (A, P_A, f, c) where A is a matrix of Bernoulli random variables, P_A is the probability matrix associated with it, f is the feasibility threshold with $f \in [0, 1]$, and c is a cost function.

The objective in BDP is to find an optimal B-Doodle mechanism B^* such that B^* minimizes the expected cost of the scheduling process (expectation with respect to P_A), given (A, P_A, f, c).

Since there are exponentially many B-Doodle mechanisms, it is impractical to enumerate each B-Doodle and compute its expected cost in order to find an optimal one. In what follows we make some simplifying assumptions that are assumed throughout this paper, and we present an efficient algorithm to solve BDP in next section.

2.2 Efficient Algorithm for Finding Optimal B-Doodle

2.2.1 Preliminaries

Intuitively, if the organizer knows the probability distribution of availability of invitees, then he should inspect the columns that are more likely to be feasible first. We will prove this claim using the following lemma.

Lemma 1. *Given (A, P_A, f, c), let q_t for each column t be the probability that column t is f-feasible. We can compute q_t in polynomial time.*

Proof. For any fixed t, let us define $V_t(i,z)$ to be the probability that among the column t of A, exactly z entries out of the first i entries (in their row indices) are 1's. $V_t(i,z)$ is well-defined where $1 \leq i \leq n$ and $0 \leq z \leq i$. We further define $V_t(i,z)$ for the following degenerate cases:

$$V_t(i,z) = \begin{cases} 1 & \text{if } i = z = 0 \\ 0 & \text{if } z = -1 \text{ or } (i = 0 \land z > 0) \end{cases}$$

Then we can compute $V_t(i,z)$ using a simple dynamic programming algorithm according to the following recurrence relation where $1 \leq i \leq n$ and $0 \leq z \leq i$:

$$V_t(i,z) = V_t(i-1,z-1) \cdot p_{i,t} + V_t(i-1,z) \cdot (1 - p_{i,t}) \tag{2.1}$$

It is easy to verify that the recurrence relation is correct. For the degenerate cases (or base cases) when $i = z = 0$, no entries (out of 0 entries) in column t are equal to 1, and thus $V_t(i,z) = 1$ in this case. When $(z = -1)$ or $(i = 0 \wedge z > 0)$, $V_t(i,z) = 0$ because it is impossible to have z entries out of i entries.

For the main recurrence relation, the entry $a_{r,t}$ is either 1 or 0, and we compute $V_t(i,z)$ by considering both cases. For each fixed t, there are $O(n^2)$ entries of V_t to be computed, and computing each entry takes $O(1)$; therefore it takes $O(sn^2)$ time to compute $V_t(i,z)$ for all t,i,z.

Finally we can compute q_t after we compute $V_t(i,z)$ for all i,z, as follows:

$$q_t = \sum_{z=\lceil f \cdot n \rceil}^{n} V_t(n,z).$$

□

Lemma 1 allows us to describe a B-Doodle mechanism in a simpler form, along with the following theorem because an optimal B-Doodle mechanism must inspect the columns in non-increasing order of q_t.

Theorem 1. *Consider a B-Doodle mechanism B described by $\langle S_1, S_2, \ldots, S_m \rangle$. Suppose that there is some option $t \in S_l$ and $t' \in S_{l'}$ with $q_t < q_{t'}$ and $l < l'$. Then B is not an optimal in the sense that we can find another mechanism B^* whose expected cost is strictly less than the expected cost of B.*

Proof. Given B, let us construct $B^* = \langle S_1^*, S_2^*, \ldots, S_m^* \rangle$ that is the same as B except that we swap t and t'. That is, $S_i^* = S_i$ for all $i \neq l$ and $i \neq l'$, and $S_l^* = (S_l \cup t') \setminus t$ and $S_{l'}^* = (S_{l'} \cup t) \setminus t'$. Note that the two mechanisms have the same batch sizes ($b_j = b_j^*$ for all j).

Let w_j be the probability that the scheduling process ends after the j-th iteration if we use B, and w_j^* if we use B^*. Then $w_j = w_j^*$ for all $j < l$ because $S_j = S_j^*$ for all j with $1 \leq j < l$. Clearly it holds that $w_l < w_l^*$ because of the swap of t and t'. For j with $l < j \leq l'$, it holds that $w_j > w_j^*$; this is not obvious, but the intuition is that if w_l^* increases then the subsequent w_j^*'s with $l < j \leq l'$ must decrease as they depend on $1 - w_l^*$, until this effect is canceled out in the l'-th batch. Since the probability of finding no feasible options during l' iterations is the same in both cases (because $\cup_{i=1}^{l'} S_i = \cup_{i=1}^{l'} S_i^*$) it holds that $w_j = w_j^*$ for all $j > l'$.

Let E_B be the expected cost of B and E_{B^*} of B^*. Since $b_j = b_j^*$ for all j, it holds that $C(j,B) = C(j,B^*)$ for all j, and we have:

$$\begin{aligned} E_B - E_{B^*} &= (w_l - w_l^*)C(l,B) + \sum_{j=l+1}^{l'} (w_j - w_j^*)C(j,B) \\ &= \sum_{j=l+1}^{l'} (w_j - w_j^*)(C(j,B) - C(l,B)) \end{aligned}$$

The first inequality holds by definition of the expected cost and canceling out some terms; the second inequality holds because $(w_l - w_l^*) + \sum_{j=l+1}^{l'} (w_j - w_j^*) = 0$ (as probabilities must add up to one). Since $w_j > w_j^*$ and $C(j,B) > C(l,B)$ for all j (because $C(j,B)$ is an increasing function in j for fixed B), we conclude that $E_B > E_{B^*}$. □

Theorem 1 implies that an optimal B-Doodle that minimizes the expected cost must have the following property: for any two batches S_j and S_k with $j < k$, it must hold that $q_t \geq q_{t'}$ for all $t \in S_j$ and $t' \in S_k$ (otherwise we can apply the theorem and swap the two options to obtain a mechanism with smaller expected cost). Therefore we can limit our attention to the sub-class of B-Doodle mechanisms that only describe batch sizes, but not explicitly which options in each batch. Due to Lemma 1 we can compute q_t for each t in polynomial time, and thus we can simply sort options by q_t in non-increasing order as a pre-processing step.

Therefore we can focus on the following sub-class of *simplified* B-Doodle mechanisms when we seek an optimal B-Doodle mechanism.

Definition 5 (Simplified B-Doodle Mechanism)**.** Let $S = \{1, 2, \ldots, s\}$ be a set of s options. A simplified B-Doodle mechanism B for S is a vector of integers, described as $B = \langle b_1, b_2, \ldots, b_m \rangle$ where ($b_j \geq 1$ for all $j \leq m$) and ($\sum_{k=1}^{m} b_k = s$). We write B_m to emphasize that B has m batches. It is assumed that B_m partitions S into m batches according to b_j's where options (or columns) are sorted by q_t in non-increasing order.

From now on we use the definition of a simplified B-Doodle, and simply write a B-Doodle mechanism to mean a simplified B-Doodle mechanism but this should cause no confusion.

2.2.2 Optimal Algorithm

In this section we present an algorithm that finds an optimal B-Doodle mechanism that minimizes the expected cost, given an instance of BDP. Let us first describe how one can express the expected cost of a B-Doodle mechanism.

Given an instance (A, P_A, f, c), consider some B-Doodle mechanism $B_m = \langle b_1, b_2, \ldots, b_m \rangle$ with $\sum_{k=1}^{m} b_k = |S|$. Let $\Pr(j)$ be the probability that the scheduling process ends after j-th iteration. Let us denote the expected cost of B_m by $\mathbb{E}_c[B_m]$, which can be expressed in terms of $\Pr(\cdot)$ and the cost function c:

$$\mathbb{E}_c[B_m] = \sum_{j=1}^{m} \Pr(j) \cdot C(j, B_m) = \sum_{j=1}^{m} \Pr(j) \cdot \left(\sum_{k=1}^{j} c(k, b_k) \right) \tag{2.2}$$

While one can compute $\Pr(j)$ in polynomial time, it is not necessary for our algorithm. Instead we present an important lemma that leads us to an efficient algorithm. The following lemma states that if c is simple, then we can compute $\mathbb{E}_c[B_m]$ in a recursive manner.

Lemma 2. *Consider any mechanism $B_m = \langle b_1, b_2, \ldots, b_m \rangle$. Let us denote another mechanism that is obtained after removing the first batch (b_1) from B_m, as $\hat{B}_{m-1}$ (i.e. $\hat{B}_{m-1} = \langle b_2, b_3, \ldots, b_m \rangle$). For each option $t \in S$, let q_t be the probability that t is f-feasible.*

If c is a θ-simple cost function with with $\theta > 0$, the following equality holds:

$$\mathbb{E}_c[B_m] = c(1, b_1) + \left(\prod_{t=1}^{b_1} (1 - q_t) \right) \cdot \theta \cdot \mathbb{E}_c[\hat{B}_{m-1}] \tag{2.3}$$

Proof. Let us first provide an intuitive way to understand the recurrence relation given by Equation 2.3. The first term $c(1, b_1)$ captures the cost of sending out the first batch; regardless of when the scheduling process ends, this cost incurs with probability of 1. If it turns out that the first batch does not contain a feasible option with probability of $(1 - \Pr(1))$, then the organizer must send out the remaining batches, which is precisely $\hat{B}_{m-1}$. It is easy to verify that the product term in the equation is equal to $(1 - \Pr(1))$. The expected cost of

$\hat{B}_{m-1}$ is adjusted by a factor of θ in the equation because the cost function is θ-simple.

We now formally prove the claim. Given B_m as described above, for each batch $j \in \{1,2,\ldots,m\}$, let r_j be the probability that the j-th batch has at least one f-feasible option. Let us define $v_0 = 0$ and $v_j = \sum_{k=1}^{j} b_k$ (i.e. v_j is the number of options in batches 1 through j). We can express r_j as:

$$r_j = 1 - \prod_{t=v_{j-1}+1}^{v_j} (1-q_t) \tag{2.4}$$

$\Pr(j)$ is the probability that the scheduling process ends after j-th iteration (with $\Pr(m) = 1 - \sum_{k=1}^{m-1} \Pr(k)$). For $1 \le j < m$, $\Pr(j)$ is given by:

$$\Pr(j) = r_j \prod_{k=1}^{j-1} (1-r_k) \tag{2.5}$$

It is understood that $\prod_{k=x}^{y}(\cdot)$ is equal to 1 when $x > y$.

Recall that $\hat{B}_{m-1} = \langle b_2, b_3, \ldots, b_m \rangle$. Not to be confused, let us express it as $\hat{B}_{m-1} = \langle \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{m-1} \rangle$ with $\hat{b}_j = b_{j+1}$ for all $j \ge 1$. Let $\hat{r}_j$ be the probability that a feasible option exists in j-th batch of $\hat{B}_{m-1}$, which is equal to r_{j+1} for all $j \ge 1$. Let $\hat{Pr}(j)$ be the probability that scheduling ends after the j-th iteration of $\hat{B}_{m-1}$:

$$\hat{Pr}(j) = \hat{r}_j \prod_{k=1}^{j-1} (1-\hat{r}_k) = r_{j+1} \prod_{k=2}^{j} (1-r_k) \tag{2.6}$$

We can express $\mathbb{E}_c[\hat{B}_{m-1}]$ as follows:

$$\begin{aligned}
\mathbb{E}_c[\hat{B}_{m-1}] &= \sum_{j=1}^{m-1} \hat{Pr}(j) \left(\sum_{k=1}^{j} c(k, \hat{b}_k) \right) \\
&= \sum_{j=1}^{m-1} r_{j+1} \left(\prod_{k=2}^{j} (1-r_k) \right) \left(\sum_{k=2}^{j+1} c(k-1, b_{k+1}) \right) \\
&= \sum_{j=2}^{m} r_j \left(\prod_{k=2}^{j-1} (1-r_k) \right) \left(\sum_{k=2}^{j} c(k-1, b_{k+1}) \right)
\end{aligned}$$

The first equality holds by definition. The second equality is due to Equation 2.6 and because $\hat{b}_k = b_{k+1}$. The third equality is by changing the range of j in the summation.

If we multiply $\mathbb{E}_c[\hat{B}_{m-1}]$ by $(1-r_1)\theta$, we get the following:

$$(1-r_1)\theta\mathbb{E}_c[\hat{B}_{m-1}] = \sum_{j=2}^{m} r_j \left(\prod_{k=1}^{j-1}(1-r_k)\right)\left(\sum_{k=2}^{j} c(k,b_{k+1})\right) \tag{2.7}$$

Notice that the product term now runs from $k=1$ to $j-1$ as we multiply by $(1-r_1)$ and the inner-most summation has $c(k,b_{k+1})$ as we multiply by θ (recall that c is θ-simple).

Finally we can express $\mathbb{E}_c[B_m]$ in terms of $\mathbb{E}_c[\hat{B}_{m-1}]$ as follows (where $c_1 = c(1,b_1)$ for brevity):

$$\begin{aligned}
\mathbb{E}_c[B_m] &= \sum_{j=1}^{m} \Pr(j)\left(\sum_{k=1}^{j} c(k,b_k)\right) \\
&= c_1 + \left(\sum_{j=2}^{m} r_j \left(\prod_{k=1}^{j-1}(1-r_k)\right)\left(\sum_{k=2}^{j} c(k,b_k)\right)\right) \\
&= c_1 + (1-r_1)\theta\mathbb{E}_c[\hat{B}_{m-1}] \\
&= c(1,b_1) + \left(\prod_{t=1}^{b_1}(1-q_t)\right)\cdot\theta\cdot\mathbb{E}_c[\hat{B}_{m-1}]
\end{aligned}$$

The first equality holds by definition of $\mathbb{E}_c[B_m]$. The second equality is obtained by taking $c(1,b_1)$ out from the summation (and note that $\Pr(j)$ adds up to 1) first and then applying Equation 2.5. The last two inequalities hold due to Equation 2.7 and 2.4, respectively. The last expression exactly matches Equation 2.3 in the lemma. $\square$

Using the recurrence relation in Lemma 2 and the computing method in Lemma 1, we can now design an efficient recursive algorithm finds the optimal B-Doodle mechanism. Our algorithm is presented in Algorithm 1 as a recursive method $Rec(x)$. We assume that the values of q_t have been computed as a pre-processing step prior to running our algorithm, using the method in Lemma 1. We further assume that options are sorted in decreasing order of q_t (i.e. $q_1 \geq q_2 \geq \cdots \geq q_s$); therefore we simply refer to the option by its index (i.e. $t=3$ refers to the third option in the sorted list of options).

Given $1 \leq x \leq s$, $Rec(x)$ returns optimal B-Doodle (B^*) that consists of options $\{x, x+1, \ldots, s\}$ and its expected cost (EC^*). To solve for a given instance of the problem, we simply call the method $Rec(x)$ with $x = 1$.

Algorithm 1 Recursive Algorithm: $Rec(x)$

1: $B^* \leftarrow \langle s-x+1 \rangle$, $EC^* \leftarrow c(1, s-x+1)$
2: **for** $b := 1, 2, \ldots, s-x$ **do**
3: $\quad (B, EC) \leftarrow Rec(x+b)$
4: $\quad EC_b \leftarrow c(1,b) + \left(\prod_{t=x}^{x+b-1}(1-q_t)\right) \cdot \theta \cdot EC$
5: $\quad$ **if** $EC^* > EC_b$ **then**
6: $\qquad EC^* \leftarrow EC_b$
7: $\qquad B^* \leftarrow \langle b, B \rangle$
8: $\quad$ **end if**
9: **end for**
10: **return** (B^*, EC^*)

Theorem 2. *Algorithm 1 runs in polynomial time, and returns an optimal B-Doodle mechanism that minimizes the expected cost, given an instance (N, S, f, c, P) of* BDP *when c is θ-simple.*

Proof. Our recursive method is very simple: given options $\{x, x+1, \ldots, s\}$, it iteratively considers the case of sending out b options in the first iteration, and computes the expected cost EC_b of doing so, for each b with $1 \leq b \leq s-x+1$. In line 1, our method checks for the trivial case when $b = s-x+1$ (i.e. all options are sent in a single batch); we store this mechanism in B^* and its expected cost in EC^*. Then we iterate b from 1 to $s-x$ and compute the expected cost EC_b, and compare with the optimal expected cost found so far (lines 2-9). For each b, we first recursively compute the expected cost for the remaining options (namely, $\{x+b, x+b+1, \ldots, s\}$) by calling our method $Rec(x+b)$, and store the expected cost in EC (line 3). Using Lemma 2 we can compute EC_b as in line 4 (note that the first batch contains b options from x to $x+b-1$). In lines 5-8, we simply compare EC_b with the current optimal, EC^*, and update if necessary; $\langle b, B \rangle$ should be interpreted as the concatenation of b and B into a single vector of integers. Finally in line 10, our method returns the optimal B-Doodle found, B^* and its expected cost EC^*.

We prove correctness of the algorithm by induction on x (from $x = s$ to $x = 1$). The base case is trivial: if $x = s$ the only B-Doodle mechanism is $\langle 1 \rangle$ and our method finds it in line 1 and returns it in line 10. Suppose our method correctly returns the optimal B-Doodle mechanism (and its expected cost) for all $x > k$ for some k (the inductive hypothesis), and we prove for the case of $x = k$. When $x = k$, there are precisely $(s-k+1)$ options that are to be sent, and the first batch can contain any number of options between 1 and $(s-k+1)$, inclusive. In lines 1-9 our method considers all such cases, and for each case it computes the expected cost correctly (in line 4) due to our inductive hypothesis and Lemma 2. Therefore our method finds the optimal B-Doodle for all x with $1 \leq x \leq s$; in particular when $x = 1$, it returns the desired optimal B-Doodle mechanism for the given problem instance.

Our recursive method runs in polynomial time when we use memoization on $Rec(x)$ with respect to x; that is, for each x we cache what $Rec(x)$ returns for the first time, and for any subsequence calls to $Rec(x)$ we simply return the cached values. Therefore lines 1-9 are executed at most once for each x with $1 \leq x \leq s$. Also note that $Rec(x)$ makes a call to $Rec(x+b)$ with $b \geq 1$ and $x+b \leq s$, which means there are no infinite loops. Once $Rec(x)$ is computed for all $x > k$, it takes $O(|S|^2)$ time to compute $Rec(k)$, and thus the overall running time of our algorithm is $O(|S|^3)$ when we start with $Rec(1)$. Pre-processing steps (of computing q_t) also run in polynomial time due to Lemma 1. □

2.3 Experimental Results with Synthetic Data

We showed that we can find an optimal B-Doodle mechanism that minimizes the expected cost. But a very important question remains: Is our optimal B-Doodle substantially better than Doodle in realistic settings? If the answer is no, we do not have much incentive to discard the simplest mechanism, Doodle, over using our sophisticated algorithm. Intuitively we expect Doodle to be inefficient if there are many options (i.e., s is large) because it causes much inconvenience for invitees to examine the options. Reasoning further, we can also see that inefficiency of Doodle depends on the probability of each option being feasible (i.e. q_t values), which in turn depends on $p_{i,t}$ values. If invitees are relatively free, then a small number of options would be sufficient for finding a feasible option, which makes

Doodle inefficient. Last but not least, the underlying cost function plays an important role as well. These all sound plausible, and we validate our intuition with simulation results.

In our setting there are many experimental choices one can choose from. In what follows we look at a representative example in which each invitee is available for each option with the same probability of p; that is, $p_{i,t} = p$ for some constant p. (While in realistic situations the $p_{i,t}$'s may all differ, we note that in our simulations our results seem to carry over to these more general settings as well.) We present experimental results for each of the three cost functions discussed in Example 1: The linear cost function, the time-averse cost function, and the inconvenience-averse function.

2.3.1 Case of Linear Cost Function

Recall that the linear cost function is parameterized by some constant $\alpha > 0$ and that $c_\alpha(j,b) = \alpha + b$. We used $\alpha = 2$ as an experimental choice, and we later discuss what we observed for different values of α.

We reasoned that Doodle becomes suboptimal if the number of options, s, is large. Therefore it is interesting to know for what ranges of s, Doodle is suboptimal. For some fixed (n,s,p), let $C^D(n,s,p)$ be the cost of Doodle and $C^*(n,s,p)$ be the expected cost of B^*. We want to find the smallest critical point $S^*(n,p)$ such that for any $s \geq S^*(n,p)$ we have $C^D(n,s,p) > C^*(n,s,p)$.

In Table 2.1, we show S^* for various (n,p) when $f = 1$ (i.e. the organizer requires everyone be available). For instance we find $S^*(6,.8) = 5$, which implies that Doodle is suboptimal for all $s \geq 5$ given that $n = 6$ and $p = .8$. The smaller S^* is, the less practical Doodle is for the corresponding (n,p). Across the table we can observe that S^* is small when n is small and/or when p is high – this agrees with our intuition. We highlighted 8 entries in boldface to emphasize that $S^* \leq 15$; for the corresponding (n,p) values, Doodle is suboptimal if there are 15 or more options being considered.

Not only do we want to know when Doodle is suboptimal, but we also want to know how inefficient Doodle is in realistic situations. For some fixed (n,s,p), we define the efficiency of Doodle, $e_D(n,s,p)$, as the ratio of the optimal expected cost to the cost of Doodle: $e_D(n,s,p) = C^*(n,s,p)/C^D(n,s,p)$.

S^*	$n=2$	$n=4$	$n=6$	$n=10$	$n=15$
$p=.8$	**3**	**4**	**5**	**8**	**15**
$p=.5$	**5**	**11**	22	90	>300
$p=.2$	**14**	70	>300	>300	>300

Table 2.1: Critical point S^* for various (n,p) when $f=1,\alpha=2$. Entries in boldface emphasize that $S^* \leq 15$.

In Table 2.2 we show e_D for the same settings of (n,p) we used in Table 2.1. For this experiment we used $s=15$ and $f=1$. If $e_D=1$ Doodle is optimal, and if e_D is close to zero then Doodle is very inefficient. We find that $e_D(2,.8)=.270$, which implies that Doodle is very inefficient in this case; on the other hand $e_D(10,.5)=1$, in which case Doodle is optimal. Across the table we observe the same pattern we observed before – for small n and high p, Doodle is substantially inefficient.

e_D	$n=2$	$n=4$	$n=6$	$n=10$	$n=15$
$p=.8$	**.270**	**.361**	**.486**	.777	.986
$p=.5$	**.502**	.904	1	1	1
$p=.2$	.970	1	1	1	1

Table 2.2: Efficiency of Doodle e_D for various (n,p) when $f=1,\alpha=2,s=15$. Entries in boldface emphasize that $e_D<.750$.

Inefficiency of Doodle is more pronounced when the organizer has a lower feasibility threshold such as $f=.7$; in such case, only a fraction of invitees need to be available. We can clearly observe the worsened inefficiency of Doodle in Table 2.3. Here we show e_D values for the same set of (n,p) as in Table 2.2 but with $f=.7$. Previously we highlighted four entries with $e_D<.750$ when $f=1$ in Table 2.2; when $f=.7$ the number of entires with $e_D<.750$ doubled to eight. Note that the first column is identical between the two tables; this is because $f=.7$ still requires both invitees to be available when $n=2$. Also notice that e_D is surprisingly low in the first row of Table 2.3 across all columns. This shows that regardless of the number of invitees, Doodle is significantly inefficient when the invitees are highly available ($p \geq .8$) and the organizer has a relaxed feasibility threshold.

We ran experiments with different values of (n,s,p,f,α), and observed the same trends that are presented here.

e_D	$n=2$	$n=4$	$n=6$	$n=10$	$n=15$
$p=.8$	**.270**	**.215**	**.267**	**.201**	**.211**
$p=.5$	**.502**	**.434**	.772	**.628**	.913
$p=.2$	.970	1	1	1	1

Table 2.3: Efficiency of Doodle e_D for various (n,p) when $f=.7,\alpha=2,s=15$. Entries in boldface emphasize that $e_D<.750$.

2.3.2 Case of Other Cost Functions

We present some of the experimental results we obtained with different cost functions, namely the time-averse cost function and the inconvenience-averse cost function. As the names suggest, the former places more weight on optimizing Time, while the latter on optimizing Inconvenience.

For the time-averse cost function (recall $c_\beta(j,b)=\beta^j\cdot b$), we chose $\beta=2$ as an experimental choice. Notice that the cost increases exponentially for later iterations, which forces the organizer to send out a small number of batches (as in Doodle). We ran the same set of experiments as before to measure efficient of Doodle, e_D.

The result is summarized in Table 2.4, which shows the same trends as we observed in previous experiments. Notice that in the first row ($p=.8$), as n increases we expect e_D to decrease, but e_D fluctuates while decreasing in general. The fluctuation is due to the rounding of attendance requirement ($\lceil a\cdot n\rceil$). For instance when $n=4$, $\lceil a\cdot n\rceil$ = 3 which effectively requires 75% of attendees be available.

e_D	$n=2$	$n=4$	$n=6$	$n=10$	$n=15$
$p=.8$	**.180**	**.104**	**.175**	**.088**	**.099**
$p=.5$	**.561**	**.457**	.876	**.726**	.987
$p=.2$	1	1	1	1	1

Table 2.4: Efficiency of Doodle e_D for various (n,p) when $f=.7,\beta=2,s=15$. Entries in boldface emphasize that $e_D<.750$.

For the inconvenience-averse cost function (recall $c_\gamma(j,b)=\gamma^b$), we chose $\gamma=1.1$ as an experimental choice. While γ is fairly small, because the cost function is an exponential function of b, an optimal B-Doodle must send out small-size batches. This cost function optimizes Inconvenience primarily. We ran the same set of experiments as before to measure efficiency of Doodle, e_D.

The result is summarized in Table 2.5. Notice that e_D is not equal to 1 even when $p = .2$ in this setting, which we did not observe with the other cost functions previously. Due to integer-rounding, we again observe some fluctuations in e_D across columns within the same row, but the general trend is that e_D decreases as n increases.

e_D	$n = 2$	$n = 4$	$n = 6$	$n = 10$	$n = 15$
$p = .8$	**.333**	**.299**	**.329**	**.294**	**.298**
$p = .5$	**.499**	**.450**	**.695**	**.592**	.799
$p = .2$	.850	.887	.974	.976	.980

Table 2.5: Efficiency of Doodle e_D for various (n, p) when $f = .7, \gamma = 1.1, s = 15$. Entries in boldface emphasize that $e_D < .750$.

2.3.3 Summary of Experiments

While we only presented experimental results with specific values of (n, s, p, f) and parameters (α, β, γ), we observed that Doodle is in general substantially inefficient, including (but not limited to) when one or more of the following conditions hold:

- There is a relatively small number of invitees ($n \leq 10$).
- There is a large number of options ($s \geq 15$).
- invitees are relatively free ($p > .5$).
- Feasibility threshold is relaxed ($f < .8$).
- The cost function places more weight on Inconvenience than Time ($\alpha < 20$, $\beta < 5$, or $\gamma > 1.05$).

Intuitively the first four conditions affect q_t (the probability that option t is feasible) in the same way, and if q_t's are high then Doodle is more likely to be inefficient because a few options may be sufficient for finding a feasible one. While the last condition is independent of q_t, the cost function determines what is being optimized, and Doodle becomes more inefficient if c favors reducing Inconvenience over reducing Time.

2.4 Discussion

In this chapter we identified two important dimensions of optimality in group scheduling: Time and Inconvenience. We generalized the popular Doodle mechanism to a class of B-Doodle mechanisms that partition date/time options into batches. We showed an example of the Pareto-frontier of B-Doodle mechanisms on the Time-Inconvenience dimensions. We then described an efficient algorithm for finding an optimal B-Doodle mechanisms, given a simple cost function that aggregates Time and Inconvenience, assuming probabilistic independence among availability of invitees. We showed in simulations that optimal B-Doodle mechanism is superior to Doodle in realistic situations, sometimes greatly so.

Chapter 3

Complexity of Group Scheduling Problem II

In Chapter 2, we assumed that an event organizer is only allowed to inspect a group of columns (as a batch) by polling all invitees at once. In this chapter, we relax this restriction, and assume that the organizer can inspect each entry of a random matrix at unit cost (by querying a certain invitee about a certain date/time option). Consequently, Timeand Inconvenienceare always equal in this setting, and the goal is to find an optimal inspection sequence which is a permutation of entries of a given random matrix. As in BDP, we would like to optimize the expected cost, which is the expected number of inspections until a feasible column is found or all columns are deemed infeasible – we call this problem the Probabilistic Matrix Inspection Problem (PMIP).

The main difference between the Batched Doodle Problem (BDP) and the Probabilistic Matrix Inspection Problem (PMIP) lies in the solution domain (a unit operation on a group of columns versus entries) and the cost model (a function of Time and Inconvenience versus the number of inspections). The Batched Doodle generalizes Doodle by allowing columns-by-columns inspections, and we further generalize it by allowing entry-by-entry inspections.

3.1 Notation and Definitions

We use the same definition of feasibility as in previous chapter, which is copied below.

Definition 6 (Feasibility)**.** Let A be a matrix whose entries are from $\{0,1\}$, and refer to the entry of A at row r and column c as $a_{r,c}$. We say that a column c of A is *feasible* if it consists only of 1's (otherwise it is *infeasible*). We say that A is *feasible* if it contains at least one feasible column (otherwise it is *infeasible*).

In the Probabilistic Matrix Inspection Problem, we do not know of the values of the entries of A, as they are Bernoulli random variables, but we know of probability distribution of each entry. An input to the problem is this probability distribution.

Definition 7 (Input instance)**.** An input instance of the Probabilistic Matrix Inspection problem is a pair of matrices (A, P_A) of size n by m where A is a matrix of Bernoulli random variables and P_A is the probability matrix associated with it. We denote an entry of A as $a_{r,c}$ and of P_A as $p_{r,c}$. Each entry $a_{r,c}$ of A is a Bernoulli random variable, and $p_{r,c}$ is the probability of success for $a_{r,c}$ (i.e., $p_{r,c} = \mathbf{P}[a_{r,c} = 1]$). We define $s_c = \prod_{r=1}^{n} p_{r,c}$ to denote the probability that column c is feasible (probability of success for column c).

Throughout this work we will assume that the set of random variables $\{a_{r,c}\}$ are mutually independent. This assumption is crucial to our technical results because it allows to compute (in polynomial time) the probability of a specific realization of A conditioning on the event in which some entries of A have already been realized. Without this assumption, it is unclear how one can compute such probabilities without having an access to the joint probability distribution over all realizations of A (whose size is exponential in the size of A).

We define an "inspection" as an operation that can be performed on A. One can inspect an arbitrary entry $a_{r,c}$ of A at unit cost, so as to know of the realization of the random variable. In group scheduling, an inspection corresponds to querying an agent about her availability for a certain outcome. The objective of the problem is to determine whether A is feasible or not, with minimum (expected) number of inspections possible. The expectation is with respect to the probability distribution specified by P_A.

Because we are interested in determining feasibility of A with minimum number of inspections, there are certain "unnecessary inspections" that an optimal strategy must avoid. For instance, if a certain entry $a_{r,c}$ is found to be unsuccessful (i.e., $a_{r,c} = 0$ is realized), then there is no need to inspect any other entry from the same column because the column is already known to be infeasible. Similarly, if a certain column is found to be feasible (which implies A is feasible) or if all columns are found to be infeasible (which implies A is infeasible), then there is no need to inspect any other entries of the matrix. Lastly, if $p_{r,c} = 0$ or $p_{r,c} = 1$, then there is no need to inspect the entry $a_{r,c}$ because we already know its realization with probability 1. Therefore, without loss of generality, we will assume that $p_{r,c} \in (0,1)$ (i.e., $p_{r,c} \neq 0,1$) in this work.

Let us define what constitutes a solution to the problem.

Definition 8 (Inspection policy)**.** Given an input instance (A, P_A), a solution is any permutation of the entries of A, and we call it an "inspection policy" (or simply, a "policy").

The interpretation of a permutation is as follows. The entries of A will be inspected in order specified by the permutation. After each inspection, if A is found to be feasible or infeasible, the inspection process ends. Otherwise, it continues inspecting the entries as specified, but it will not make any unnecessary inspections as mentioned earlier.

If π is a permutation of the entries of A, we write $C(\pi)$ to denote the number of inspections performed by π. $C(\pi)$ is a random variable whose probability distribution is determined by P_A. We are interested in finding an optimal policy which minimizes the expected number of inspections, $\mathbf{E}[C(\pi)]$. Note that there are $(nm)!$ permutations of the entries of A, and therefore exhaustive search for an optimal permutation will not produce an efficient algorithm.

3.1.1 Example

Consider a 2-by-2 matrix A of Bernoulli random variables whose probability of success is given by P_A as follows. In group scheduling, this corresponds to two agents and a set of two date/time options being considered.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}, P_A = \begin{bmatrix} 0.6 & 0.7 \\ 0.9 & 0.8 \end{bmatrix}$$

Let us consider an inspection policy π which inspects the entries of A column-by-column while inspecting them from top to bottom within a column:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ a_{1,1} & a_{2,1} & a_{1,2} & a_{2,2} \end{pmatrix}.$$

Suppose that the realization of A happens to be the identity matrix of size 2 (i.e., $a_{1,1} = a_{2,2} = 1$ and $a_{2,1} = a_{1,2} = 0$). If we use π, it will first inspect $a_{1,1}$ and learn its realization. Since $a_{1,1} = 1$, it will inspect $a_{2,1}$ next only to find that column 1 is infeasible after all. It will then inspect $a_{1,2}$ and learn that column 2 is also infeasible, which implies that A is infeasible. At this point, the inspection process terminates without inspecting $a_{2,2}$. This specific realization of A happens with probability $p_{1,1}(1-p_{2,1})(1-p_{1,2})p_{2,2}$, and yields $C(\pi) = 3$ because 3 inspections would occur. If the realization of A happens to be the null matrix (i.e., all entries are 0's), then π would only inspect $a_{1,1}$ and $a_{1,2}$ but skip $a_{2,1}$ and $a_{2,2}$. In this manner one can consider all $2^{2\cdot 2} = 16$ possible realizations of A, and compute $\mathbf{E}[C(\pi)]$ in this example.

Another way to compute $\mathbf{E}[C(\pi)]$ is by de-coupling $C(\pi)$ into two random variables $N(\pi, c)$ with $c \in \{1,2\}$ where $N(\pi, c)$ denotes the number of inspections (on column c) performed by π conditioning on the event that (at least one element of) column c is inspected. We can efficiently compute these: $\mathbf{E}[N(\pi, c)] = 1 \cdot \mathbf{P}[a_{1,c} = 0] + 2 \cdot \mathbf{P}[a_{1,c} = 1]$ for $c \in \{1,2\}$. To express $\mathbf{E}[C(\pi)]$ in $N(\pi, c)$'s, we need to take conditional probability into account, as column 2 is inspected only if column 1 is infeasible: $\mathbf{E}[C(\pi)] = \mathbf{E}[N(\pi,1)] + \mathbf{E}[N(\pi,2)](1-s_1) = 2.382$. Recall that s_c is the probability of success for column c.

3.2 Technical Results and Algorithm

We first consider two special cases (1-row or 1-column matrices) of the Probabilistic Matrix Inspection problem, which admit intuitive, greedy algorithms. We then discuss a couple of interesting properties of an optimal inspection policy, which leads to our main result and algorithm.

3.2.1 1-Row Matrix and 1-Column Matrix

Let us first consider the case where an input matrix A has only one row (i.e., $n = 1$). In this case it is natural to inspect entries with largest probability first because we can stop as soon as we find an entry whose value is 1 – which makes its column and A feasible. This intuition is exactly what an optimal policy should do in the single row case.

Lemma 3 (1-Row Matrix)**.** *When $n = 1$, an inspection policy π is optimal if and only if it inspects the entries in non-increasing order of their associated probabilities.*

Proof. Without loss of generality let us assume that a policy π inspects the entries in increasing order of their column index; that is, $\pi(i) = a_{1,i}$. Recall that $C(\pi)$ is a random variable that denotes the number of inspections π incurs. We can express the expectation of $C(\pi)$ in terms of $p_{1,c}$'s as follows:

$$\mathbf{E}\left[C(\pi)\right] = m\left(\prod_{k=1}^{m-1}(1-p_{1,k})\right) + \sum_{j=1}^{m-1} j\left(p_j\prod_{k=1}^{j-1}(1-p_{1,k})\right). \tag{3.1}$$

Suppose that there exists some c^* such that $p_{1,c^*} < p_{1,c^*+1}$ (if no such c^* exists, then π is an inspection policy that inspects the entries in non-increasing order of probabilities). Let π' be the same policy as π except we swap the order of a_{1,c^*} and a_{1,c^*+1}. That is, π' is defined as follows.

$$\pi'(j) = \begin{cases} \pi'(j) = \pi(c^*+1) & \text{if } j = c^* \\ \pi'(j) = \pi(c^*) & \text{if } j = c^*+1 \\ \pi'(j) = \pi(j) & \text{if } j \neq c^* \wedge j \neq c^*+1 \end{cases}$$

After expressing $\mathbf{E}[C(\pi)]$ and $\mathbf{E}[C(\pi')]$ as in Equation 3.1, one can re-arrange the terms to obtain the following:

$$\mathbf{E}\left[C(\pi)\right] - \mathbf{E}\left[C(\pi')\right] = \left(\prod_{j=1}^{c^*-1}(1-p_{1,j})\right)(p_{1,c^*+1} - p_{1,c^*}). \tag{3.2}$$

This quantity is positive if $p_{1,c^*+1} > p_{1,c^*}$ (recall that $p_{1,j} \in (0,1)$ for all j as mentioned in Section 3.1).

This proves the lemma because any policy that inspects an entry with smaller probability before another entry with higher probability is suboptimal, and therefore an optimal policy must inspect entries in non-increasing order of their associated probabilities. □

Although the proof of Lemma 3 is simple, it confirms correctness of our intuition. Equation 3.2 illustrates this intuition; conditioning on the event that the first $c^* - 1$ inspections fail (whose probability is the product term in Equation 3.2), the difference $\mathbf{E}[C(\pi)] - \mathbf{E}[C(\pi')]$ depends on the difference in the probabilities of success between the next-entry-to-be-inspected by π and π'.

We can also consider the case where an input matrix A has only one column. Intuitively, if we wish to minimize the expected number of inspections, we must inspect entries with smallest probability first because we can stop as soon as we determine that A is infeasible. Lemma 4 formally states this intuition about optimal policy, and we omit a proof of it as it can be easily done by following the proof of Lemma 3.

Lemma 4 (1-Column Matrix)**.** *When $m = 1$, an inspection policy π is optimal if and only if it inspects the entries in non-decreasing order of their associated probabilities.*

3.2.2 Inspection of Entire Column

Another interesting property of an optimal inspection policy is that once it inspects the first entry of a column, then it must commit to it and continue inspecting the remaining entries of the column until feasibility of the column is determined. Otherwise, if the policy switches to another column too soon, then it is not optimal.

Theorem 3 (Optimality of inspecting entire column)**.** *Consider any inspection policy π. Without loss of generality, let us assume that for each column c, π inspects $a_{n,c}$ the last among n entries of the column. Let b_c be the index of π such that $\pi(b_c) = a_{n,c}$. Without loss of generality, assume $b_1 < b_2 < \cdots < b_m$ (we can do this by re-labeling the columns of A). If there is some column c^* such that $b_{c^*} > n \cdot c^*$, then π is not optimal.*

Proof. First, note that $b_c \geq n \cdot c$ for all c because we assumed $b_1 < b_2 < \cdots < b_m$, and therefore the entries of previous columns must appear before the last entry of each column.

Let π be an inspection policy being considered in the theorem for which there exists some c with $b_c > n \cdot c$. Let us construct a different inspection policy π'. First, π' inspects all entries of column 1 in the same order π does. Then, π' inspects all entries of column 2 in the same order π does, and so on. In particular, π' inspects all entries of a column before inspecting another column, while preserving the original ordering of the entries within each column that is given by π. We will show that $\mathbf{E}[C(\pi')] < \mathbf{E}[C(\pi)]$.

Let us define a set of new random variables which can be used to express $C(\cdot)$, as we did in Section 3.1.1 when analyzing an example. Recall that $s_c = \prod_{r=1}^{n} a_{r,c}$ is the probability of success for column c. Let $N(\pi, c)$ ($N(\pi', c)$, respectively) be a random variable that denotes the number of entries of column c that is inspected by π (by π', respectively), conditioning on the event that column c is inspected (i.e., when the previous $c-1$ columns are infeasible). We can then express $\mathbf{E}[C(\pi)]$ and $\mathbf{E}[C(\pi')]$ as follows:

$$\mathbf{E}[C(\pi)] = \sum_{c=1}^{m} \mathbf{E}[N(\pi, c)] \left(\prod_{k=1}^{c-1} (1 - s_c) \right) \tag{3.3}$$

and

$$\mathbf{E}[C(\pi')] = \sum_{c=1}^{m} \mathbf{E}[N(\pi', c)] \left(\prod_{k=1}^{c-1} (1 - s_c) \right). \tag{3.4}$$

To prove the theorem we will first show that for any realization of A, $N(\pi, c) \geq N(\pi', c)$ holds for all c; this immediately implies $\mathbf{E}[C(\pi)] \geq \mathbf{E}[C(\pi')]$. We will then show that there exists at least one realization of A such that for some column c' the strict inequality $N(\pi, c') > N(\pi', c')$ holds. These two statements together imply that $\mathbf{E}[C(\pi)] > \mathbf{E}[C(\pi')]$.

Consider any realization of A with the condition that the first $m-1$ columns are infeasible (recall that m is the number of columns of A). Then $N(\pi, c) = N(\pi', c)$ for all c regardless of feasibility of column m. To see why, both π and π' would inspect the same set of entries in each of the first $m-1$ columns in the same order until the column is determined to be infeasible, and therefore $N(\pi, c) = N(\pi', c)$ if $c < m$. If column m is feasible, then both π and π' would inspect all n entries of it, and thus we have $N(\pi, m) = N(\pi', m) = n$. Otherwise, if column m is also infeasible (in which case A is infeasible), then π and π' would inspect the same set of entries of column m in the same order until the first infeasible entry of the column is found. Therefore if the first $m-1$ columns are infeasible we have $N(\pi, c) \geq N(\pi', c)$ for all c.

Now consider any realization of A with the condition that at least one of the first $m-1$ columns is feasible. Let c' be the smallest index of feasible columns of A. Because the columns from 1 to $c'-1$ are infeasible, $N(\pi, c) = N(\pi', c)$ for all $c < c'$ for the same reason we stated earlier for the other case. Since c' is feasible, $N(\pi, c') = N(\pi', c') = n$ as both policies would inspect all n entries of c'. By our construction of π' it is clear that $N(\pi', c) = 0$ for all $c > c'$; therefore we have $N(\pi, c) \geq N(\pi', c)$ for all $c > c'$. In summary $N(\pi, c) \geq N(\pi', c)$ holds for all c in this case as well.

So far we proved the first claim we stated earlier: for all realizations of A, we have $N(\pi, c) \geq N(\pi', c)$ for all c. Let us now prove the second claim. Let c^* be the smallest index c of columns such that $b_c > nc$ (note that $c^* < m$ because $b_m = nm$ by definition). Consider any realization of A with the condition that the first $c^* - 1$ columns are infeasible and column c^* is feasible (feasibility of other columns do not matter). Using the same arguments we used earlier, we can show that $N(\pi, c) = N(\pi', c)$ for all $c < c^*$, that $N(\pi, c^*) = N(\pi', c^*) = n$, and that $N(\pi', c) = 0$ for all $c > c^*$. However, because $b_{c^*} > n \cdot c^*$, there is at least one entry $a_{r', c'}$ with $c' > c^*$ which appears before b_{n, c^*} in π (otherwise, if no such entry exists, then b_{c^*} would be equal to $n \cdot c^*$). This implies that there exists some c' with $c' > c^*$ such that $N(\pi, c') > 0$. This proves the second claim that for some realization of A, there is some column c' for which $N(\pi, c') > N(\pi', c')$, and together with the first claim we proved earlier, this implies that $\mathbf{E}[C(\pi)] > \mathbf{E}[C(\pi')]$.

This proves the theorem: Any policy that does not inspect all entries of a column consecutively is suboptimal. $\square$

By Theorem 3, when seeking an optimal policy, it is sufficient to consider the set of policies that inspect an entire column before committing to another column. Lemma 4 hints that one should inspect the entries of each column in increasing order of probabilities, and this is what we prove next.

3.2.3 Optimal Ordering within Column

Lemma 4 states that an optimal policy must inspect the entries in increasing order of their probability of success, if A is a 1-column matrix. This argument can be generalized to the case where there is more than one column: If an optimal policy is to inspect an entry of some column c, it must inspect the entry with smallest probability of success first.

Theorem 4 (Optimal ordering within column)**.** *Consider any inspection policy π. If there exist two entries $a_{r_1,c}$ and $a_{r_2,c}$ from the same column such that $a_{r_1,c}$ appears before $a_{r_2,c}$ in π and $p_{r_1,c} > p_{r_2,c}$, then π is not optimal. In other words, when restricted to each column, an optimal policy must inspect the entries of the column in non-decreasing order of probabilities.*

Proof. Let π be an inspection policy being considered in the theorem. Because of Theorem 3 we can assume, without loss of generality, that π inspects all entries of column 1, followed by column 2, and so on. Further let us assume that π inspects the entries of each column in increasing order of their row index (we can do so by re-labeling the indices of entries). Precisely, $\pi(r+n(c-1)) = a_{r,c}$ defines π. Let p_{r,c^*} and p_{r+1,c^*} be the entries with $p_{r,c^*} > p_{r+1,c^*}$. Let us consider a different inspection policy π' that is the same as π except that π' inspects p_{r+1,c^*} before p_{r,c^*}, by swapping the ordering of them.

$$\pi'(j) = \begin{cases} \pi'(j) = a_{r+1,c^*} & \text{if } \pi(j) = a_{r,c^*} \\ \pi'(j) = a_{r,c^*} & \text{if } \pi(j) = a_{r+1,c^*} \\ \pi'(j) = \pi(j) & \text{otherwise} \end{cases}$$

We claim that $\mathbf{E}[C(\pi')] < \mathbf{E}[C(\pi)]$, which implies that π is not optimal.

Let us define new random variables $N(\pi,c)$ and $N(\pi',c)$ as we did in our proof of Theorem 3 (i.e., the number of inspections performed by the respective policy on column

c, conditioning on the event that the column is inspected). Then we can express $\mathbf{E}[C(\pi)]$ and $\mathbf{E}[C(\pi')]$ in terms of the new random variables and s_c's as we did in Equations 3.3 and 3.4.

Observe that $N(\pi,c) = N(\pi',c)$ for any realization of A if $c \neq c^*$. To see this, first note that column c would not be inspected by π or by π' if any of the previous columns (that is, columns 1 through $c-1$) is found to be feasible, in which case $N(\pi,c) = N(\pi',c) = 0$. Otherwise, if column c is inspected, both policies would inspect the entries of c in the very same order, so $N(\pi,c) = N(\pi',c)$ must hold. Therefore we conclude that $\mathbf{E}[N(\pi,c)] = \mathbf{E}[N(\pi',c)]$ when $c \neq c^*$.

We will now show that $\mathbf{E}[N(\pi,c^*)] > \mathbf{E}[N(\pi',c^*)]$ holds. This immediately implies $\mathbf{E}[C(\pi)] > \mathbf{E}[C(\pi')]$ due to Equations 3.3 and 3.4. Let us express $\mathbf{E}[N(\pi,c^*)]$ in terms of p_{r,c^*}'s.

$$\mathbf{E}[N(\pi,c^*)] = n\left(\prod_{k=1}^{n-1} p_{k,c^*}\right) + \sum_{j=1}^{n-1} j(1-p_{j,c^*})\left(\prod_{k=1}^{j-1} p_{k,c^*}\right)$$

Note that the event $N(\pi,c^*) = j$ occurs if the first $j-1$ entries are feasible while the j-th entry is not feasible when $j < n$, and $N(\pi,c^*) = n$ occurs if the first $n-1$ entries are feasible (but the n-th entry's feasibility does not matter).

We can express $\mathbf{E}[N(\pi',c^*)]$ in a similar manner, and simplify $\mathbf{E}[N(\pi,c^*)] - \mathbf{E}[N(\pi',c^*)]$ as follows:

$$\mathbf{E}[N(\pi,c^*)] - \mathbf{E}[N(\pi',c^*)] = r\left(\prod_{k=1}^{r-1} p_{k,c^*}\right)(p_{r,c^*} - p_{r+1,c^*}).$$

The quantity above is positive if $p_{r,c^*} > p_{r+1,c^*}$, which is the assumption we began with. This proves the theorem. □

Theorems 3 and 4 together tell us that in order to find an optimal policy we only need to decide the ordering of the columns. There are still $m!$ orderings of columns, and an exhaustive search algorithm would not be efficient. As we were able to generalize Lemma 4 to Theorem 4 by generalizing the optimal solution for 1-column case, it would be natural to consider generalizing Lemma 3 in a similar manner.

This idea leads to the following greedy algorithm: First we sort columns by their probability of success ($s_c = \prod_{r=1}^{n} p_{r,c}$) in decreasing order, and inspect the entries of each column in increasing order of their associated probabilities. However, as the following example shows, this algorithm is suboptimal.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}, P_A = \begin{bmatrix} 0.4459 & 0.2262 \\ 0.4459 & 0.8114 \end{bmatrix}$$

Here we have $s_1 = 0.199$ and $s_2 = 0.184$, and the greedy algorithm would produce $\pi = \begin{pmatrix} a_{1,1} & a_{2,1} & a_{1,2} & a_{2,2} \end{pmatrix}$. Its expected cost, $\mathbf{E}[C(\pi)]$, is 2.428, but if we inspect the second column first, then the expected cost is 2.407 which is optimal in this example. One can consider another greedy algorithm which inspects the columns in increasing order of their expected number of inspections (within column), but this algorithm turns out to be suboptimal as well.

3.2.4 Main Result and Algorithm

Let us present the main result that leads to an efficient algorithm for finding an optimal inspection policy.

Theorem 5. *Let s_c be the probability of success for column c as before. Let μ_c be the expected number of inspections that column c incurs if its entries are inspected in increasing order of their probability of success, conditioning on the event that column c is inspected and infeasible. An optimal policy must be a column-by-column policy (due to Theorem 3), must inspect the entries of each column in non-decreasing order of probabilities (due to Theorem 4), and must inspect the columns in non-decreasing order of $\mu_c(1-s_c)/s_c$.*

Proof. Consider a column-by-column inspection policy π which inspects the column 1 through m in increasing order of their index (we can assume this without loss of generality by re-labeling columns).

As before, let $N(\pi, c)$ be a random variable that denotes the number of inspections performed by π on column c, conditioning on the event that column c is inspected. Then we can express $\mathbf{E}[N(\pi, c)]$ in terms of s_c and μ_c as follows.

$$\mathbf{E}[N(\pi, c)] = s_c \cdot n + (1 - s_c) \cdot \mu_c \tag{3.5}$$

This equation holds because if the column is feasible (with probability s_c), it would require

n inspections, but if it is not (with probability $1-s_c$), it would require μ_c inspections in expectation. The equation above simply considers these two events, and calculates the expected value of $N(\pi,c)$.

Suppose that there is some column c^* such that $\mu_{c^*}(1-s_{c^*})/s_{c^*} > \mu_{c^*+1}(1-s_{c^*+1})/s_{c^*+1}$. Because π inspects column c^* before column c^*+1, it would not be inspecting the columns in increasing order of $\mu_c(1-s_c)/s_c$. Consider a different inspection policy π' which inspects the columns in the same order as π except that π' inspects column c^*+1 before c^* by swapping the inspection ordering of the two. We can relate $N(\pi,\cdot)$ to $N(\pi',\cdot)$ as follows.

$$N(\pi',c) = \begin{cases} N(\pi,c^*+1) & \text{if } c = c^* \\ N(\pi,c^*) & \text{if } c = c^*+1 \\ N(\pi,c) & \text{otherwise} \end{cases}$$

As we did in proofs of Theorems 3 and 4, we can use Equations 3.3 and 3.4, and simplify $\mathbf{E}[C(\pi)] - \mathbf{E}[C(\pi')]$ as follows.

$$\begin{aligned} \mathbf{E}[C(\pi)] - \mathbf{E}[C(\pi')] = &\left(\frac{\mathbf{E}[N(\pi,c^*)]}{s_{c^*}} - \frac{\mathbf{E}[N(\pi,c^*+1)]}{s_{c^*+1}}\right) \\ &\cdot \left(\prod_{j=1}^{c^*-1}(1-s_j)\right) s_{c^*}s_{c^*+1} \end{aligned} \tag{3.6}$$

The quantity in Equation 3.6 is positive if the difference of the weighted expected values (in the first parentheses) are positive. Using Equation 3.5 we obtain the following inequality.

$$\begin{aligned} &\frac{\mathbf{E}[N(\pi,c^*)]}{s_{c^*}} > \frac{\mathbf{E}[N(\pi,c^*+1)]}{s_{c^*+1}} \\ \Leftrightarrow &\mu_{c^*}(1-s_{c^*})/s_{c^*} > \mu_{c^*+1}(1-s_{c^*+1})/s_{c^*+1} \end{aligned}$$

By definition of c^*, the second inequality above holds, which implies $\mathbf{E}[C(\pi)] > \mathbf{E}[C(\pi')]$. Therefore, an optimal inspection policy must inspect the columns in non-decreasing order of $\mu_c(1-s_c)/s_c$. $\square$

Because there is a unique ordering of columns if we sort them by $\mu_c(1-s_c)/s_c$ (up to ties), Theorem 5 leads to the following algorithm: We inspect the columns in increasing order of $\mu_c(1-s_c)/s_c$, and in each column, we inspect the entries of it in increasing order of probabilities. This algorithm can easily be implemented to run in polynomial time.

3.3 Discussion and Future Work

In this work we defined the Probabilistic Matrix Inspection problem motivated by group scheduling and Doodle. We first considered two special cases, and discovered interesting properties of an optimal inspection policy which agree with our intuition. We then generalized our findings to design an efficient algorithm to solve the general case, and along the way we showed that two natural greedy algorithms fail to find an optimal solution. While we believe that our technical results make a great starting point for studying and optimizing a group scheduling process, there remain several open problems and future work to be done.

As we discussed in previous chapter, our model and algorithm rely on the assumption that probability estimates on availability of agents are available. We suggested several ideas motivated by previous work in the literature, but it will be important to deploy such ideas into a system, and integrate it with our algorithm. From a theoretical perspective, there remain several open problems. While we assumed that an inspection can be performed on a single entry at unit cost, one can generalize the cost model by allowing an inspection of any subset of entries whose cost depends on, for example, the number of entries being inspected. In the context of group scheduling, an inspection on many entries means querying multiple agents at the same time for one or many outcomes (but an inspection is not limited to the entries from the same column or row). This generalization is particularly useful when scheduling takes place in a hierarchical setting such as corporates. For instance the event organizer may feel that the cost of querying a supervisor is significantly different from that of querying a colleague.

Lastly, although we focused on relating our model to group scheduling, the Probabilistic Matrix Inspection problem has other applications. Finding the right childcare facility, for example, involves extensive inquiries as parents wish to gather more information about how they would handle certain situations, what benefits and environments they provide, and so on. Through advertisements or brochures parents may even be able to gauge the likelihood of a certain facility satisfying their needs. Yet they still need to inquire facilities for precise information, which can be modeled by our probabilistic matrix model where columns correspond to facilities and rows correspond to the needs of parents.

Chapter 4

Complexity of Group Activity Selection Problem

Chapter 5

Complexity of Stable Invitations Problem

Chapter 6

Incentive Compatibility in Group Assignment Problems

Chapter 7

Open Problems

Bibliography

[1] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, Massachusetts, 1985.

[2] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

[3] George E. Andrews. Euler's 'exemplum memorabile inductionis fallacis' and q-trinomial. *Journal of the American Mathematical Society*, 3(3):653–669, 1990.

[4] Haris Aziz and Florian Brandl. Existence of stability in hedonic coalition formation games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 763–770, 2012.

[5] Coralio Ballester. Np-completeness in hedonic games. *Games and Economic Behavior*, 49(1):1–30, 2004.

[6] Robert Baumgartner, Georg Gottlob, and Sergio Flesca. Visual information extraction with Lixto. In *Proceedings of the 27th International Conference on Very Large Databases*, pages 119–128, Rome, Italy, September 2001. Morgan Kaufmann.

[7] P. Berry, B. Peintner, K. Conley, M. Gervasio, T. Uribe, and N. Yorke-Smith. Deploying a personalized time management agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1564–1571. ACM, 2006.

[8] Anna Bogomolnaia and Matthew O. Jackson. The stability of hedonic coalition structures. *Games and Economic Behavior*, 38(2):201–230, 2002.

[9] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, April–June 1985.

[10] M.H. Chia, D.E. Neiman, and V.R. Lesser. Coordinating asynchronous agent activities in a distributed scheduling system. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98)*, 1998.

[11] Peter Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In Edmund Burke and Wilhelm Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin Heidelberg, 2001.

[12] E. Crawford and M. Veloso. Learning to select negotiation strategies in multi-agent meeting scheduling. *Progress in Artificial Intelligence*, pages 584–595, 2005.

[13] Andreas Darmann. Group activity selection from ordinal preferences. In Toby Walsh, editor, *Algorithmic Decision Theory*, volume 9346 of *Lecture Notes in Computer Science*, pages 35–51. Springer International Publishing, 2015.

[14] Andreas Darmann, Edith Elkind, Sascha Kurz, Jrme Lang, Joachim Schauer, and Gerhard Woeginger. Group activity selection problem. In PaulW. Goldberg, editor, *Internet and Network Economics*, volume 7695 of *Lecture Notes in Computer Science*, pages 156–169. Springer Berlin Heidelberg, 2012.

[15] J.C. de Borda. Mémoire sur les élections au scrutin. 1781.

[16] K. Decker and J. Li. Coordinated hospital patient scheduling. In *Multi Agent Systems, 1998. Proceedings. International Conference on*, pages 104–111. IEEE, 1998.

[17] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[18] Jacques H. Dreze and Joseph Greenberg. Hedonic coalitions: Optimality and stability. *Journal of the Econometric Society*, pages 987–1003, 1980.

[19] E. Ephrati, G. Zlotkin, and J.S. Rosenschein. A non-manipulable meeting scheduling system. In *Proceedings of the 13th international workshop on distributed artificial intelligence*, pages 105–125, 1994.

[20] M. S. Franzin, F. Rossi, E. C. Freuder, and R. Wallace. Multi-agent constraint systems with preferences: Efficiency, solution quality, and privacy loss. *Computational Intelligence*, 20(2):264–286, 2004.

[21] E.C. Freuder, M. Minca, and R.J. Wallace. Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents. In *Proc. IJCAI DCR*, pages 63–72, 2001.

[22] Michael R. Garey, David S. Johnson, and Larry Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[23] L. Garrido and K. Sycara. Multi-agent meeting scheduling: Preliminary experimental results. In *Proceedings of the Second International Conference on Multiagent Systems*, pages 95–102, 1996.

[24] Georg Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, June 1992.

[25] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, May 2002.

[26] M. Hannebauer and S. Müller. Distributed constraint optimization for medical appointment scheduling. In *Proceedings of the fifth international conference on Autonomous agents*, pages 139–140. ACM, 2001.

[27] A.B. Hassine, X. Defago, and T.B. Ho. Agent-based approach to dynamic meeting scheduling problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1132–1139. IEEE Computer Society, 2004.

[28] Russell Impagliazzo and Ramamohan Paturi. Complexity of k-sat. In *Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on*, pages 237–240. IEEE, 1999.

[29] N. R. Jennings, A. J. Jackson, and London E Ns. Agent-based meeting scheduling: A design and implementation, 1995.

[30] Hooyeon Lee and Yoav Shoham. Optimizing time and convenience in group scheduling. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1345–1346. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[31] Hooyeon Lee and Yoav Shoham. Stable invitations. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 965–971, 2015.

[32] Hector J. Levesque. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 23(2):155–212, July 1984.

[33] Hector J. Levesque. A logic of implicit and explicit belief. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 198–202, Austin, Texas, August 1984. American Association for Artificial Intelligence.

[34] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, 1994.

[35] Leon Mann, Mark Radford, Paul Burnett, Steve Ford, Michael Bond, Kwok Leung, Hiyoshi Nakamura, Graham Vaughan, and Kuo-Shu Yang. Cross-cultural differences in self-reported decision-making style and confidence. *International Journal of Psychology*, 33(5):325–335, 1998.

[36] T.M. Mitchell, R. Caruana, D. Freitag, J. McDermott, D. Zabowski, et al. Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91, 1994.

[37] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

[38] D.E. Neiman, D.W. Hildum, V.R. Lesser, T. Sandholm, et al. Exploiting meta-level information in a distributed scheduling system. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 394–394. JOHN WILEY & SONS LTD, 1994.

[39] Katharina Reinecke, Minh Khoa Nguyen, Abraham Bernstein, Michael Näf, and Krzysztof Z Gajos. Doodle around the world: Online scheduling behavior reflects cultural differences in time perception and group decision-making. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 45–54. ACM, 2013.

[40] S. Sen and E.H. Durfee. A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, 7(3):265–289, 1998.

[41] S.T.F. St. Satisfying user preferences while negotiating meetings. 1997.

[42] K. Sycara, S.F. Roth, N. Sadeh, and M.S. Fox. Distributed constrained heuristic search. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(6):1446–1461, 1991.

[43] T. Tsuruta and T. Shintani. Scheduling meetings using distributed valued constraint satisfaction algorithm. In *ECAI*, pages 383–387, 2000.

[44] John Von Neumann and Oskar. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 1947.

[45] J. Wainer, P.R. Ferreira, and E.R. Constantino. Scheduling meetings through multi-agent negotiations. *Decision Support Systems*, 44(1):285–297, 2007.

[46] James Zou, Reshef Meir, and David Parkes. Strategic voting behavior in doodle polls. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 464–472. ACM, 2015.