

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

© Copyright by Hooyeon Lee 2017
All Rights Reserved

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

Scheduling an event for a group of agents is a challenging problem. It tends to be tedious and time-consuming. In practice, procrastination and strategic behavior of agents are often a problem, but even if we assume that agents are truthful, prompt, and indifferent among the possible outcomes, the group scheduling problem exhibits an interesting problem. We formally model this as an optimization problem, and provides an intuitive, efficient algorithm that solves the problem. We also discuss a generalization of this problem whose computational difficult is currently unknown.

Let us consider a different setting where the organizer wishes to assign agents to social activities. Agents may have preferences over activities as well as the number of participants in the activity they are assigned to (but it is assumed that agents have anonymous preferences in that they do not care who participates and who does not). For instance, agents may prefer more participants in social networking receptions, but wish to have not too many participants in a table tennis tournament. Naturally the organizer wishes to assign as many agents to activities as possible, but at the same time he wishes to ensure that every individual is satisfied with the assignment. We study the solution concept of individual rationality and stability which require that there is no single-agent deviation from an assignment. This problem is known to be NP-hard in general even if certain restrictions of preferences of agents are assumed. In this work we first show that the problem is still computationally difficult even if we seek a small, fixed-size solution in general. However, we also show that for a relaxed version of the stability requirement, the problem admits an efficient algorithm for finding a fixed-size, small solution.

Next we consider a special case of the assignment problem when there is only one activity, which is known to be in P. We introduce friends-and-enemies relationship among agents, and we show if the number of friends or enemies of an agent is large (namely, two or more), then the problem of finding a stable solution is NP-hard. Furthermore, finding a fixed-size, small solution also depends on the cardinality of the largest friend-set or enemy-set, which naturally categorizes the underlying problem into different complexity classes. We also show that the problem becomes

computationally easier if the friends and enemies relationship is symmetric. Lastly we consider strategic agents in this problem where agents may report false information to the event organizer. We show that in general finding a stable solution and strategy-proofness are incompatible in the activity selection problem, but we also provide a socially optimal, computationally efficient, and strategy-proof mechanism in the special case where there is only one activity and the preferences of agents align with the goal of the designer (so as to maximize the number of participants).

Acknowledgments

Contents

Abstract	iv
Acknowledgments	vi
1 Introduction	1
1.1 Organization of the Thesis	1
1.2 Preliminaries and Notation	1
2 Batched Doodle Problem	2
2.1 Notation and Definitions	5
2.2 Main Algorithm for the Batched Doodle Problem	10
2.3 Experimental Results	14
2.4 Discussion	18
3 Probabilistic Matrix Inspection Problem	20
3.1 Formal Model	21
3.2 Technical Results and Algorithm	24
3.3 Discussion and Future Work	31
4 Complexity of Group Activity Selection Problem	32
4.1 Definitions and Known Results	35
4.2 Parameterized Complexity	38
4.3 Discussion and Future Work	54
5 Complexity of Stable Invitations Problem	56
5.1 Definitions and Notation	57

5.2	Classical Complexity	58
5.3	Parameterized Complexity	58
5.4	Symmetric Social Relationship	66
5.5	Discussion and Future Work	73
6	Incentive Compatibility in Group Assignment Problems	74
6.1	Introduction and Related Work	74
6.2	Definitions and Known Results	74
7	Discussion and Open Problems	78
	Bibliography	79

List of Tables

2.1	Critical point S^* for various (n, p) when $f = 1, \alpha = 2$. Entries in boldface emphasize that $S^* \leq 15$	15
2.2	Efficiency of Doodle e_D for various (n, p) when $f = 1, \alpha = 2, s = 15$. Entries in boldface emphasize that $e_D < .750$	16
2.3	Efficiency of Doodle e_D for various (n, p) when $f = .7, \alpha = 2, s = 15$. Entries in boldface emphasize that $e_D < .750$	16
2.4	Efficiency of Doodle e_D for various (n, p) when $f = .7, \beta = 2, s = 15$. Entries in boldface emphasize that $e_D < .750$	17
2.5	Efficiency of Doodle e_D for various (n, p) when $f = .7, \gamma = 1.1, s = 15$. Entries in boldface emphasize that $e_D < .750$	18
4.1	Summary of results on parameterized complexity of GASP.	38
5.1	Complexity of k -IR-Invitation and k -Stable-Invitation.	59
5.2	Complexity of k -IR-Invitation and k -Stable-Invitation. $f(k)$ can be an arbitrary function of k that only depends on k . All entries other than “P” imply NP-completeness. “W[1]-C” and “W[2]-C” mean W[1]-completeness and W[2]-completeness, respectively. Note that P and NP-completeness results were presented in Section xxx.	59
5.3	Complexity of SIP with symmetric social relationships. $f(k)$ can be an arbitrary function of k that only depends on k . All entries other than “P” imply NP-completeness. “W[1]-C” and “W[2]-C” mean W[1]-completeness and W[2]-completeness, respectively. All results are original (including classical complexity results).	67

List of Figures

2.1	A scatter plot of B-Doodle Mechanisms given four invitees and six options.	3
2.2	Pareto-frontier and objective function.	4

Chapter 1

Introduction

1.1 Organization of the Thesis

To be added.

1.2 Preliminaries and Notation

To be added.

Chapter 2

Batched Doodle Problem

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. First, an event organizer selects a candidate set of date/time options, and asks invitees to respond with their availability. Given the responses, the organizer then 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 on all options proposed by the organizer. 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 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

¹<http://www.doodle.com>

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

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 – in a way, their ‘free time’ is being hostages until the final schedule is set. To quantify this inconvenience, we use the expected number of time slots floated by the organizer as a proxy.

In order to avoid incurring much inconvenience, the organizer can select just a handful number of 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 polling 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.



Figure 2.1: A scatter plot of B-Doodle Mechanisms given four invitees and six options.

We can clearly observe a time-inconvenience Pareto frontier in Figure 2.1. In addition, if there is an overall cost function combining time and inconvenience, one can identify an optimal B-Doodle mechanism along this frontier as illustrated in Figure 2.2. If we assume that the overall cost is defined as $3 \cdot \text{Time} + \text{Inconvenience}$ (which is a linear combination of the two), 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 finding an 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 an optimal polling mechanism for a broad class of objective functions.

In addition we also 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.³ 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.

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 agent during the process. In this section we formally define the class of B-Doodle mechanisms, the Batched Doodle Problem (BDP), and other technical terms and notations.

2.1.1 Problem Setting

Consider n invitees denoted by $\{a_1, \dots, a_n\}$ and s time slots denoted by $\{1, 2, \dots, s\}$. For each agent a_i and each time slot t , there is a known prior probability, $p_{i,t}$, such that the agent is available at time t with probability $p_{i,t}$ and unavailable with probability $(1 - p_{i,t})$. While the organizer knows all such priors (i.e. all $p_{i,t}$ ’s), she does not know the realization of availability of agents – it is private information of each agent. The organizer is trying to find a *feasible* time slot by asking invitees to reveal their availability; we assume that the organizer has a *feasibility threshold*, f , such that a feasible time slot requires at least $\lceil f \cdot n \rceil$ agents be available. The organizer iteratively sends out a batch of time slots until a feasible time slot 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 time slots that have been sent out by the organizer.

³<http://en.blog.doodle.com/2012/05/23/mondays-for-planning-busy-weekends/>

This setting is general enough to capture many realistic scenarios of group scheduling. For instance in a corporate setting, a busy manager may have low probability of being available for all time slots while other employees may be less busy and have high probability of being available. Or, if students are trying to decide on the meeting time for a group project, their availability is expected to be equally likely because the schedules of their classes are similar.

Let us define a class of B-Doodle mechanisms that describe how the organizer sends out time slots in each iteration.

Definition 1 (B-Doodle Mechanism). Let $S = \{1, 2, \dots, s\}$ be a set of s time slots. We define a B-Doodle mechanism for S as an ordered partition of S . Let $B = \langle S_1, S_2, \dots, S_m \rangle$ 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 time slots in S_1 during the first iteration. If a feasible time slot 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 fixed S (exponential in cardinality of S). Doodle is a special case of B-Doodle mechanism and can be described as $\langle S \rangle$.

Our objective 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 on the number of iterations and batch sizes.

Definition 2 (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 time slots during the j -th iteration. We assume that cost is additive so that the overall cost of executing the first j iterations of $B_m = \langle S_1, S_2, \dots, S_m \rangle$ 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 agents.

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 chapter 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 .

θ -simple cost functions form a broad class of cost functions, and can model many different settings. For instance, when $\theta = 1$, it means that the cost of sending out a batch of a certain size is the same regardless of during which iteration it is being sent. When $\theta > 1$, it means that a later iteration costs more than an earlier iteration if the size of a batch is the same. Lastly, when $\theta \in (0, 1)$, it is the opposite in that an earlier iteration costs more than a later iteration given the same batch size – this last case may be rare in real life, but our definition naturally allows for such scenarios. The following example shows this by various simple cost functions.

Example 1 (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.

$$c_\alpha(j, b_j) = \alpha + b_j, \quad C_\alpha(j, B_m) = (\alpha \cdot j) + \left(\sum_{k=1}^j b_k \right). \quad (2.1)$$

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. This cost function is β -simple.

$$c_\beta(j, b_j) = \beta^j \cdot b_j, \quad C_\beta(j, B_m) = \left(\sum_{k=1}^j \beta^k \cdot b_k \right). \quad (2.2)$$

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.

$$c_\gamma(j, b_j) = \gamma^{b_j}, \quad C_\gamma(j, B_m) = \left(\sum_{k=1}^j \gamma^{b_k} \right). \quad (2.3)$$

We now formally define the Batched Doodle Problem.

Definition 3 (Batched Doodle Problem). An instance of the Batched Doodle Problem (BDP) is a tuple (N, S, f, c, P) where $N = \{a_1, a_2, \dots, a_n\}$ is a set of n agents, $S = \{1, 2, \dots, s\}$ is a set of time s slots, f is the feasibility threshold ($0 \leq f \leq 1$), c is a cost function, and P is a probability distribution matrix of availability ($P = \{p_{i,t}\}$ with $0 \leq p_{i,t} \leq 1$ for all i, t).

The objective in BDP is to find an optimal partition of S , B^* , among all B-Doodle mechanisms that partition S , such that B^* minimizes the expected cost of the scheduling process (expectation with respect to P), given (N, S, f, c, P) .

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.

2.1.2 Technical Assumptions

As we briefly mentioned in the intro, we assume that the availability of agents for each time slot is independent of their availability at other time slots, and of the availability of other agents. That is, the set of events in which agents are available for certain time slots is mutually independent. We discuss the relaxation of this assumption in Section 2.4. The independence assumption allows us to compute the likelihood of each time slot being feasible, as the following lemma states.

Lemma 1. *Given (N, S, f, P) , let q_t for each $t \in S$ be the probability that time slot t is f -feasible. Under the independence assumption, q_t can be computed in polynomial time.*

Proof. For any fixed t , let us define $V_t(i, z)$ to be the probability that among the agents $\{a_1, a_2, \dots, a_i\}$ exactly z agents are available at time slot t . $V_t(i, z)$ 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:

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

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 $0 \leq z \leq i$:

$$V_t(i, z) = V_t(i-1, z-1) \cdot p_{i,t} + V_t(i-1, z) \cdot (1 - p_{i,t}) \quad (2.4)$$

It is easy to verify that the recurrence relation is correct. For the degenerate cases (or base cases) when $i = z = 0$ it is always (with probability 1) the case that no agents are available. When $z = -1$ or $(i = 0 \wedge z > 0)$, it is never (with probability 0) the case that z agents are available. For the main recurrence relation, agent a_i is either available or unavailable, 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 $O(sn^2)$ 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:

$$q_t = \sum_{z=\lceil f \cdot n \rceil}^n V_t(n, z). \quad (2.5)$$

□

Lemma 1 allows us to describe a B-Doodle mechanism in a simpler form, along with the following theorem.

Theorem 1. *Consider a B-Doodle mechanism B described by $\langle S_1, S_2, \dots, S_m \rangle$. Suppose that there is some time slot $t \in S_l$ 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 another B-Doodle mechanism, B^* , as follows: $B^* = \langle S_1^*, S_2^*, \dots, S_m^* \rangle$, which 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 time slots 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 \mathbb{E}_B be the expected cost of B and \mathbb{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:

$$\mathbb{E}_B - \mathbb{E}_{B^*} = (w_l - w_l^*)C(l, B) + \sum_{j=l+1}^{l'} (w_j - w_j^*)C(j, B) \quad (2.6)$$

$$= \sum_{j=l+1}^{l'} (w_j - w_j^*)(C(j, B) - C(l, B)) \quad (2.7)$$

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 $\mathbb{E}_B > \mathbb{E}_{B^*}$. \square

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 time slots 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 time slots in each batch. Due to Lemma 1 we can compute q_t for each t in polynomial time, and thus we can simply sort time slots 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:

Definition 4 (Simplified B-Doodle Mechanism). Let $S = \{1, 2, \dots, s\}$ be a set of s time slots. A simplified B-Doodle mechanism B for S is a vector of integers, described as $B = \langle b_1, b_2, \dots, b_m \rangle$ where $(b_j \geq 1 \text{ for all } j \leq m)$ and $(\sum_{k=1}^m b_k = s)$. We write B_m to emphasize that B has m batches.

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 Main Algorithm for the Batched Doodle Problem

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 (N, S, f, c, P) , consider some B-Doodle mechanism $B_m = \langle b_1, b_2, \dots, b_m \rangle$ with $\sum_{k=1}^m b_k = |S|$. Let $Pr(j)$ be the probability that the scheduling process ends after j -th iteration. Let

$\mathbb{E}[B_m|c]$ denote the expected cost of B_m given cost function c , which can be expressed as follows:

$$\mathbb{E}[B_m|c] = \sum_{j=1}^m Pr(j) \cdot C(j, B_m) = \sum_{j=1}^m Pr(j) \cdot \left(\sum_{k=1}^j c(k, b_k) \right). \quad (2.8)$$

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 $\mathbb{E}[B_m|c]$ in a recursive manner.

Lemma 2. *Consider any mechanism $B_m = \langle b_1, b_2, \dots, b_m \rangle$. 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. $\hat{B}_{m-1} = \langle b_2, b_3, \dots, b_m \rangle$). For each time slot $t \in S$, let q_t be the probability that t is f -feasible.*

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

$$\mathbb{E}[B_m|c] = c(1, b_1) + \left(\prod_{t=1}^{b_1} (1 - q_t) \right) \cdot \theta \cdot \mathbb{E}[\hat{B}_{m-1}|c] \quad (2.9)$$

Proof. Let us first provide an intuitive way to understand the recurrence relation given by Equation 2.9. 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 time slot 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 $j \in \{1, 2, \dots, m\}$, let r_j be the probability that the j -th batch has at least one f -feasible time slot. Let us define $v_0 = 0$ and $v_j = \sum_{k=1}^j b_k$ (i.e. v_j is the number of time slots in batches 1 through j). We can express r_j as:

$$r_j = 1 - \prod_{t=v_{j-1}+1}^{v_j} (1 - q_t) \quad (2.10)$$

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

$$Pr(j) = r_j \prod_{k=1}^{j-1} (1 - r_k) \quad (2.11)$$

Recall that $\hat{B}_{m-1} = \langle b_2, b_3, \dots, b_m \rangle$. For clarity, let us express it as $\hat{B}_{m-1} = \langle \hat{b}_1, \hat{b}_2, \dots, \hat{b}_{m-1} \rangle$ with $\hat{b}_j = b_{j+1}$ for all $j \geq 1$. Let \hat{r}_j be the probability that a feasible time slot exists in j -th batch of \hat{B}_{m-1} , which is equal to r_{j+1} for all $j \geq 1$. Let $\hat{Pr}(j)$ be the probability that scheduling ends after the j -th iteration of \hat{B}_{m-1} :

$$\hat{Pr}(j) = \hat{r}_j \prod_{k=1}^{j-1} (1 - \hat{r}_k) = r_{j+1} \prod_{k=2}^j (1 - r_k) \quad (2.12)$$

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

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

The first equality holds by definition. The second equality is due to Equation 2.12 and because $\hat{b}_k = b_{k+1}$. The third equality is by changing the range of j in the summation.

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

$$(1 - r_1)\theta \mathbb{E}[\hat{B}_{m-1}|c] = \sum_{j=2}^m r_j \left(\prod_{k=1}^{j-1} (1 - r_k) \right) \left(\sum_{k=2}^j c(k, b_{k+1}) \right) \quad (2.13)$$

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

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

The first equality holds by definition of $\mathbb{E}[B_m|c]$. 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 2.11. The last two inequalities hold due to Equation 2.13 and 2.10, respectively. The last expression exactly matches Equation 2.9 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 time slots are sorted in decreasing order of q_t (i.e. $q_1 \geq q_2 \geq \dots \geq q_s$); therefore we simply refer to the time slot by its index (i.e. $t = 3$ refers to the third time slot in the sorted list of time slots).

Given $1 \leq x \leq s$, $Rec(x)$ returns optimal B-Doodle (B^*) that consists of time slots $\{x, x+1, \dots, s\}$ 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  $b := 1, 2, \dots, s - x$  do
3:    $(B, EC) \leftarrow Rec(x + b)$ 
4:    $EC_b \leftarrow c(1, b) + (\prod_{t=x}^{x+b-1} (1 - q_t)) \cdot \theta \cdot EC$ 
5:   if  $EC^* > EC_b$  then
6:      $EC^* \leftarrow EC_b$ 
7:      $B^* \leftarrow \langle b, B \rangle$ 
8:   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 time slots $\{x, x+1, \dots, s\}$, it iteratively considers the case of sending out b time slots 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 time slots 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 time slots (namely, $\{x+b, x+b+1, \dots, s\}$) 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 time slots 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 $(s - k + 1)$ time slots that are to be sent, and the first batch can contain any number of time slots 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 subsequent 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. \square

2.3 Experimental Results

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 time slots (i.e., s is large) because it causes much inconvenience for agents to examine the options. Reasoning further, we can also see that inefficiency of Doodle depends on the probability of each time slot being feasible (i.e. q_t values), which in turn depends

on $p_{i,t}$ values. If agents are relatively free, then a small number of time slots would be sufficient for finding a feasible time slot, 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 agent is available for each time slot 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 time slots, 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 time slots being considered.

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 $S^* \leq 15$.

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: $e_D(n, s, p) = C^*(n, s, p)/C^D(n, s, p)$.

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 agents 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 entries 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 agents 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.

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 $e_D < .750$.

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

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 agents ($n \leq 10$).
- There is a large number of time slots ($s \geq 15$).
- Agents 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 time slot 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 time slots 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

We identified and formally defined two dimensions of optimality in group scheduling : Time and Inconvenience. We generalized the popular Doodle mechanism to a class of B-Doodle mechanisms that partition time slots 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 agents. We showed in

simulations that optimal B-Doodle mechanism is superior to Doodle in realistic situations, sometimes greatly so.

While useful in and of itself, our work described in this chapter leaves open many questions. Here we focused on a class of B-Doodle mechanisms which allow the event organizer to poll all invitees on a batch of date/time options. In some settings, it may be more desirable for the organizer to poll some invitees on some date/time options – for instance, when scheduling a defense of dissertation in the computer science department of Stanford University, it is desirable to poll availability of advisor and committee chair before polling other participants because the advisor and committee chair must be physically present while others can join remotely. In the very next chapter, we generalize the Batched Doodle Problem by considering such settings.

In addition, we implicitly assumed that agents prefer all time slots equally (by allowing agents to specify their availability but not preferences), but people often have specific preferences. Thus it will be important to study the trade-off between optimizing the cost of scheduling process and finding a ‘good’ schedule.

Finally, we assumed honesty, promptness, and trust of invitees, but these are strong assumptions. In particular, promptness assumes that invitees do not procrastinate and do not deliberately delay responses. It is arguably true that if the scheduling process is efficient then it will indeed reduce the procrastination of invitees. However in practice an invitee may have an incentive to delay her response or to lie about her availability or preferences. Practical mechanisms should not be vulnerable to such strategic behavior of invitees, and this is yet another direction for future work.

Chapter 3

Probabilistic Matrix Inspection Problem

In Chapter 2, we assumed that an event organizer is only allowed to inspect a group of columns (as a batch of date/time options) by polling all invitees at once. In this chapter, we generalize the notion of “inspection” by the event organizer, 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 at a unit cost). Consequently, Time and Inconvenience are 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.

In classic Doodle, an inspection is performed over the entire random matrix by querying all invitees (rows) about all date/time options (columns). In the Batched Doodle Problem, we considered a class of B-Doodle mechanisms in which all invitees are queried about a subset of date/time options. In this chapter, we study the Probabilistic Matrix Inspection Problem in which an individual invitee is queried over a single date/time option. In practice, an organizer may want to ask the most important (and/or busiest) invitee first about a couple of date/time options, to ensure that the person is available, before asking other invitees. For instance, when a doctoral graduate student is scheduling his/her university oral examination at Stanford University (which is usually the defense of dissertation in computer science), it is required that both the student’s primary advisor and the exam chair must be physically present while other examiners may participate virtually. In such cases, the student can ask the advisor and exam chair to rule out certain date/time options that do not work for them, and then ask other examiners about their availability.

We model this setting as the problem of inspecting a random matrix where each entry is a random variable (representing availability in group scheduling settings). Consider a matrix of mutually independent Bernoulli random variables where rows represent agents and columns represents the set of potential outcomes (such as date/time options). Given probability distribution of the matrix,

the organizer can “inspect” an entry of the matrix at unit cost to know of the realization of it, and wishes to determine with certainty whether the matrix contains a column consisting only of 1’s at minimum number of inspections possible. In our example, this corresponds to querying an agent for her availability, and seeking an agreeable outcome. By abstracting away the details of group scheduling, we are left with a mathematical model that represents the scheduling process, and in this work we study the properties of an optimal inspection policy and design an algorithm that finds it in polynomial time. As we later discuss, this model is applicable to other real-life settings as well.

This chapter is organized as follows. In Section 3.1, we introduce notation and formally define the problem. In Section 3.2, we present our technical results. In Section 3.3, we discuss how our technical results can be used for group scheduling problems and other real-life scenarios.

3.1 Formal Model

Let us introduce mathematical notation we use in this chapter, and formally define the Probabilistic Matrix Inspection Problem (PMIP), followed by an example to help the reader understand the setup.

3.1.1 Notation and Definitions

In the Probabilistic Matrix Inspection Problem, the goal is to determine whether a given random matrix contains a “feasible” column or not.

Definition 5 (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*).

We do not know of the values (or realizations) of the entries of A , as they are Bernoulli random variables, but we assume that we know of probability distribution of each entry. An input to the problem is this probability distribution as a matrix.

Definition 6 (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 7 (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, $\mathbb{E}[C(\pi)]$. 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.

3.1.2 Example

Consider the 2-by-2 matrix 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 to inspect the matrix, it will first inspect $a_{1,1}$ and learn its realization. Because $a_{1,1} = 1$, 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 $a_{2,2}$ as it is not necessary. 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 $\mathbb{E}[C(\pi)]$ in this example.

Another way to compute $\mathbb{E}[C(\pi)]$ 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: $\mathbb{E}[N(\pi, c)] = 1 \cdot \mathbf{P}[a_{1,c} = 0] + 2 \cdot \mathbf{P}[a_{1,c} = 1]$ for $c \in \{1, 2\}$. To express $\mathbb{E}[C(\pi)]$ in $N(\pi, c)$'s, we need to take conditional probability into account, as column 2 is inspected only if column 1 is infeasible: $\mathbb{E}[C(\pi)] = \mathbb{E}[N(\pi, 1)] + \mathbb{E}[N(\pi, 2)](1 - s_1) = 2.382$. Recall that s_c is the probability of success for column c .

There is a trivial, brute-force algorithm that can find an optimal inspection policy: Consider every inspection policy (there are $(nm)!$ of them) and for each inspection policy compute its expected number of inspections (by considering all 2^{nm} realization of A). This algorithm's runtime is exponential in the size of an input matrix, and we shall design an efficient algorithm in this chapter.

3.2 Technical Results and Algorithm

We first consider two special cases of the Probabilistic Matrix Inspection problem when $n = 1$ or $m = 1$, both of which admit intuitive, greedy algorithms. We then discuss 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:

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

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

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

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. \square

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 $\mathbb{E}[C(\pi)] - \mathbb{E}[C(\pi')]$ 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 $b_1 < b_2 < \dots < b_m$ (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 $b_1 < b_2 < \dots < b_m$, 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 $\mathbb{E}[C(\pi')] < \mathbb{E}[C(\pi)]$.

Let us define a set of new random variables which can be used to express $C(\cdot)$, as we did in Section 3.1.2 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 $\mathbb{E}[C(\pi)]$ and $\mathbb{E}[C(\pi')]$ as follows:

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

and

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

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 $\mathbb{E}[C(\pi)] \geq \mathbb{E}[C(\pi')]$. 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 $\mathbb{E}[C(\pi)] > \mathbb{E}[C(\pi')]$.

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 $\mathbb{E}[C(\pi)] > \mathbb{E}[C(\pi')]$.

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 $\mathbb{E}[C(\pi')] < \mathbb{E}[C(\pi)]$, 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 $\mathbb{E}[C(\pi)]$ and $\mathbb{E}[C(\pi')]$ 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 $\mathbb{E}[N(\pi, c)] = \mathbb{E}[N(\pi', c)]$ when $c \neq c^*$.

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

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

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

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

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

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

$$\pi = (a_{1,1} \ a_{2,1} \ a_{1,2} \ a_{2,2}).$$

Its expected cost, $\mathbb{E}[C(\pi)]$, 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 $\mathbb{E}[N(\pi, c)]$ in terms of s_c and μ_c as follows.

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

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 $\mathbb{E}[C(\pi)] - \mathbb{E}[C(\pi')]$ as follows.

$$\mathbb{E}[C(\pi)] - \mathbb{E}[C(\pi')] = \left(\frac{\mathbb{E}[N(\pi, c^*)]}{s_{c^*}} - \frac{\mathbb{E}[N(\pi, c^* + 1)]}{s_{c^*+1}} \right) \cdot \left(\prod_{j=1}^{c^*-1} (1 - s_j) \right) s_{c^*} s_{c^*+1} \quad (3.6)$$

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.

$$\frac{\mathbb{E}[N(\pi, c^*)]}{s_{c^*}} > \frac{\mathbb{E}[N(\pi, c^* + 1)]}{s_{c^*+1}} \Leftrightarrow \frac{\mu_{c^*}(1 - s_{c^*})}{s_{c^*}} > \frac{\mu_{c^*+1}(1 - s_{c^*+1})}{s_{c^*+1}}$$

By definition of c^* , the second inequality above holds, which implies the first. Hence, 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 which generalizes the Batched Doodle Problem. 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 or universities as in the example of university oral exams at Stanford University.

Lastly, the Probabilistic Matrix Inspection problem has many other applications than group scheduling. For instance, suppose that some research lab is hiring a new researcher among many candidates. After reading each candidate's curricular vitae and research statement, each member of the lab can estimate the likelihood of him/her approving the candidate, yet it is necessary to interview a candidate to make a decision. In this setting, the problem of finding a candidate who is approved by every member of the lab is equivalent to the problem of finding a feasible date/time option. Clearly, assuming that the lab is happy to hire any of the candidates whom all of the lab members approve, it is rational to minimize the expected number of in-person interviews until someone is hired or every candidate is rejected. Another example is a search quest for housing or childcare facility. Each house or facility is a candidate, and there is a set of requirements for the candidate being considered. Making inquiries about and taking a tour of the property or facility involves much effort – even though much information is available through brochures or booklets, people in this situation would wish to minimize the time and effort they need to spend until they declare a winner.

Chapter 4

Complexity of Group Activity Selection Problem

In Chapters 2 and 3, we considered settings where an event organizer is trying to choose a date/time option at minimal cost for a group of agents given their likelihood of availability. In this chapter we consider a different setting where the event organizer is trying to assign agents to different activities given constraints and preferences of agents.

Imagine an event in which several activities are to take place concurrently. A group of agents are willing to participate and announce their activity preferences to the event organizer who is to assign agents to activities. In many settings, the agent preferences include not only which activities the agent is willing to participate in, but also the number(s) of participants in each activity that are acceptable to the agent. For example, agents may wish to have enough participants in certain activities (such as group bus tour) to split the cost associated with it, whereas they may wish to have just few participants in activities with limited resources (such as a demo booth with a limited number of devices). We assume that an agent can be assigned to at most one activity (which naturally occurs when activities are run concurrently).

Given the preferences of agents, the organizer wishes to find a “good” assignment subject to certain rationality and/or stability conditions. The first condition is *individual rationality*: everyone who is assigned to some activity is willing to participate. In addition to individual rationality, the organizer may want to ensure that agents who are not assigned to any activity do not prefer to deviate from their assignment by joining an activity (*Nash stability*). Other concepts include *envy-freeness* that asserts that no unassigned agent would prefer to take the place of an assigned agent, and *perfection* in which all agents must be assigned.

To model this setting, in WINE'12, Darmann et al. [14] proposed the Group Activity Selection Problem (GASP) and defined three solution concepts (individual rationality, stability, and perfection). In addition, they also provided many computational complexity results, including NP-hardness results for GASP, even under various restrictions on agent preferences.

These hardness results essentially argue that it is hard to find an assignment that maximizes the number of participants in activities. Suppose, however, that we are satisfied if we can assign a small number k out of the n participants to some (up to k) of the p activities while satisfying our rationality criteria. There is a brute-force solution: try all possible $O(p^k)$ ordered choices of up to k activities, all $O(k^k)$ choices for the number of participants in each activity, and all $O(n^k)$ ordered choices of k participants; then check whether the desired criterion is satisfied by the induced assignment. This brute-force solution runs in $O((pnk)^k)$ time. While this running time is polynomial for any fixed k , it is not very desirable. A much better running time would be one of the form $O(f(k) \cdot (p+n))$ - such a runtime would be *linear*, regardless of the constant k , and the function f . More generally, on input size N , one would like an algorithm with runtime $f(k) \cdot N^c$, where c is independent of k . Such an algorithm is known as fixed parameter tractable (FPT), and the problems that admit such algorithms are said to be in the class FPT. Developing FPT algorithms, especially linear time ones, greatly mitigates the NP-hardness of problems as it shows that these problems are actually quite tractable for many instances.

The field of parameterized complexity strives to classify NP-hard problems on a finer scale by analyzing time complexity in terms of both the input size and an additional parameter. The FPT problems are viewed as the most tractable problems in NP (of course, all polytime problems are naturally in FPT). There is a hierarchy of complexity classes under parameterization (called the W-hierarchy), including FPT, W[1], W[2], etc. Hierarchy theorems show that FPT is contained in W[1] which is contained in W[2], and so on (see [17] for more details). It is believed that $\text{FPT} \neq \text{W}[1]$ and $\text{W}[2] \neq \text{W}[1]$, so that the NP-hard parameterized problems in W[2] are believed to be harder than those in W[1] that are themselves believed to be harder than the FPT problems. Furthermore, popular hardness assumptions such as the Exponential Time Hypothesis (ETH) of Impagliazzo and Paturi [28] can often be used to show that particular W[1]-hard problems cannot be solved in $N^{o(k)}$ time, giving concrete runtime lower bounds.

The goal of this work is to investigate how different solution concepts and different restrictions on agent preferences influence the difficulty of GASP. We focus on GASP with the size of the solution as our parameter, i.e. the number of assigned agents, or in the case of perfection, the number of used activities. We place different NP-hard versions of GASP under this parameterization into different

parts of the W-hierarchy. Our classification is nearly complete, as seen in Table 4.1 (in Section 4.2). We show that GASP for individual rationality is in FPT, whereas for the other solution concepts the problem is W[1]-complete or W[2]-hard even for restricted types of agent preferences.

The restrictions on preferences that we consider are quite natural – we consider the cases where agents have thresholds for each activity such that they are willing to participate if the number of participants is above or below the thresholds. In the case of *increasing* preferences, every agent i approves an activity a_j as long as at least $l_i(a_j)$ participants are assigned to it (hence the threshold is a lower-bound on the number of participants). In the case of *decreasing* preferences, agent i approves an activity a_j if at most $u_i(a_j)$ agents are assigned to it. Surprisingly to us, the case of decreasing preferences is actually easier than that of increasing preferences as it is FPT even when a stable assignment is to be found, whereas GASP for increasing preferences is W[1]-complete for all concepts except for individual rationality.

We also consider the special case of GASP in which all activities are equivalent. Here the preferences of the agents can vary but for each particular agent the preferences are the same for all copies of the activity. This special case is natural and captures many applications. We show that even though the problem is still NP-hard in this case [14], all parameterized versions of it (except possibly perfection) are FPT.

Related Work. Darmann et al. [14] defined GASP, and provided a number of computational complexity results – mostly, NP-completeness results. In this work, we adopt their definitions of the problem and solution concepts, but we also consider the new solution concept of envy-freeness, while we analyze the parameterized complexity of the problem. As Darmann et al. point out, it is worth noting that GASP is closely related to Hedonic Games (see Section 2.2 of [14] and Section 2 of [31] for more details); in fact, GASP can be viewed as a class of hedonic coalition games with concise representation of preferences of agents (linear in the number of activities and players as opposed to all subsets of players in the general setting). Much work has been devoted to analyzing solution concepts in hedonic coalition games such as stability and Pareto-optimality [8, 18, 4]. Ballester [5] provides a number of computational complexity results (in fact, hardness results) for finding a core-stable, Nash-stable, or individually rational outcome in hedonic games and anonymous hedonic games. These hardness results do not imply similar hardness results for GASP, however, because of the concise representation of an input to GASP. Lastly, Darmann [13] considered a different setting of GASP where agents are assumed to have strict, ordinal preferences over the outcomes, whereas both our work and [14] assumes that agents have a dichotomic preference over outcomes in the

sense that each agent is indifferent among all outcomes that she approves of. It is an interesting future problem to consider how our results in this work can be extended to the ordinal setting that Darmann [13] considered.

The rest of this chapter is structured as follows. In Section 4.1 we formally define GASP, and summarize the known complexity results. In Section 4.2 we provide our main technical results. In Section 4.3 we summarize our findings, and describe several open problems.

4.1 Definitions and Known Results

To make this work self-contained, we begin by introducing the formal definitions proposed by Darmann et al. in their work [14], yet we make slight modifications to notation for readability and consistency in this chapter.

Definition 8. In the Group Activity Selection Problem (GASP), we are given a set of agents $N = \{1, 2, \dots, n\}$, a set of non-void activities $A^* = \{a_1, a_2, \dots, a_p\}$, and the *void activity* (a_\emptyset) which refers to the case when an agent does not participate in any of the activities in A^* . An *outcome* is a pair $(a_j, x) \in A^* \times [n]$ which is interpreted as x agents participating in non-void activity a_j . For each agent i we are given a set S_i of outcomes (called *approval set*) such that the outcomes in S_i are equally liked and strictly preferred to a_\emptyset , where $S_i \subseteq A^* \times [1, n]$. We write $S_i(a_j) = \{x : (a_j, x) \in S_i\}$ to refer to the set of sizes which agent i approves for activity a_j .

Similarly to the work of [14], in this work we assume that each agent is indifferent among the outcomes in S_i ; that is, the void-activity (a_\emptyset) draws the line between which outcomes are approved and which ones are not by the agent. While this is a simplifying assumption, note that hardness results immediately imply the same hardness for the general case without this assumption. By abusing notation, we also use a_\emptyset to refer to the outcome in which an agent is not assigned to any non-void activity.

Example 2. Consider $N = \{1, 2, 3\}$ and $A^* = \{a_1, a_2\}$. There are six outcomes (besides a_\emptyset) in $A^* \times [3]$. Suppose $S_1 = \{(a_1, 1), (a_1, 2), (a_1, 3)\}$, $S_2 = \{(a_1, 2), (a_2, 2), (a_2, 3)\}$, and $S_3 = \{(a_1, 1), (a_2, 1), (a_2, 2)\}$. In particular, agent 1 approves a_1 for any size (i.e., unconditional approval) while does not approve a_2 for any size (i.e., unconditional refusal). Using our notation, $S_1(a_1) = \{1, 2, 3\}$ and $S_1(a_2) = \emptyset$. If we assign all agents to a_1 , then $(a_1, 3)$ is the outcome realized by all agents – notice that only agent 1 approves it (and thus is willing to participate) while agents 2 and 3 do not (and thus are unwilling to participate). Naturally, this assignment induces instability.

Let us define what constitutes a solution to GASP, and define four solution concepts.

Definition 9. Let $A = A^* \cup \{a_\emptyset\}$. An assignment in GASP is a mapping $\pi : N \rightarrow A$ where $\pi(i) = a_\emptyset$ means that agent i is not assigned to any non-void activity. Each assignment naturally partitions the agents into at most $|A|$ groups. We define $\pi^0 = \{i : \pi(i) = a_\emptyset\}$ and $\pi^j = \{i : \pi(i) = a_j\}$ for $j = 1, \dots, p$, so that $|\pi^j|$ refers to the number of agents assigned by π to a specific activity (including the void activity). Let us define the size of an assignment, denoted by $|\pi|$, as the number of agents that are assigned to non-void activities; that is, $|\pi| = \sum_{j=1}^p |\pi^j|$.

Note that π induces an outcome for each agent: If $\pi(i) = a_\emptyset$, then agent i does not participate in any non-void activity (and thus outcome a_\emptyset is induced), and if $\pi(i) = a_j \in A^*$, then $(a_j, |\pi^j|)$ is the induced outcome for agent i .

The objective in GASP is to find a “good” assignment of maximum size thereby assigning as many agents to activities as possible. We define four solution concepts which require different levels of rationality/stability in an assignment.

Definition 10. An assignment $\pi : N \rightarrow A$ is said to be *individually rational* (IR) if for every $j \in [p]$ and every agent $i \in \pi^j$ it holds that $(a_j, |\pi^j|) \in S_i$.

An assignment $\pi : N \rightarrow A$ is said to be *(Nash) stable* if it is individually rational and for every agent $i \in N$ such that $\pi(i) = a_\emptyset$ and every $a_j \in A^*$ it holds that $(a_j, |\pi^j| + 1) \notin S_i$.

An assignment $\pi : N \rightarrow A$ is said to be *envy-free* (EF) if it is individually rational and for every agent $i \in N$ such that $\pi(i) = a_\emptyset$ and every agent $i' \in N$ such that $\pi(i') = a_j \in A^*$, it holds that $(\pi(i'), |\pi^j|) \notin S_i$.

An assignment $\pi : N \rightarrow A$ is said to be *perfect* if it is individually rational and $\pi(i) \neq a_\emptyset$ for all $i \in N$.

Individual rationality requires that every agent who's assigned to a non-void activity is willing to participate because participation is preferred to non-participation. Therefore everyone who is assigned has no incentive to deviate from the assignment. Stability further requires that every agent who is assigned to the void activity is also unwilling to deviate (unilaterally, without permission of other agents) from the given assignment. Envy-freeness requires that every agent who is assigned to the void activity is not envious of someone else who is assigned to a non-void activity.¹ Lastly, a perfect assignment is the strongest solution concept which implies individual rationality, stability and envy-free by definition.

¹While one can naturally define another solution concept which requires both stability and envy-freeness, it is not considered in this work.

As Darmann et al. showed in their work [14], finding a solution in GASP is NP-hard even under various restrictions on inputs. Yet we shall see that some of such restrictions make the problem less complex under parameterization. First we define two natural restricted domains of preferences of agents, called *increasing* and *decreasing* preferences.

Definition 11. We say that agent i has an *increasing* preference for activity a_j if there exists a threshold $l_i(a_j) \in \{1, 2, \dots, n+1\}$ such that $S_i(a_j) = [l_i(a_j), n]$ (where $[n+1, n] = \emptyset$). Similarly, we say that agent i has a *decreasing* preference for activity a_j if there exists a threshold $u_i(a_j) \in \{0, 1, \dots, n\}$ such that $S_i(a_j) = [1, u_i(a_j)]$ (where $[1, 0] = \emptyset$).

A natural example that accounts for an increasing preference is when an activity is associated with a cost that is to be split by the participants (e.g., group bus tourism in the city). For decreasing preferences, participants in some activity may need to share limited resources (e.g., a trial-demo of new wearable devices).

Next, we consider a restriction on the activities when there may exist multiple “copies” of the same activity (e.g., chess matches with multiple chessboards). We define equivalence of activities, and consider a special case of the problem in which all activities are equivalent.

Definition 12. We say that two activities a_j and $a_{j'}$ are equivalent if for every agent $i \in N$, $S_i(a_j) = S_i(a_{j'})$.

Let us re-visit the problem instance from Example 2, and relate it to various definitions and concepts we have defined in this section.

Example 3. In Example 2, note that agent 2 has an increasing preference for a_2 with $l_2(a_2) = 2$ while agent 3 has a decreasing preference for a_2 with $u_3(a_2) = 2$. Agent 1 has (degenerate) increasing/decreasing preferences for both a_1 and a_2 with $l_1(a_1) = 1, u_1(a_2) = 3$ and $l_1(a_2) = 4, u_1(a_2) = 0$.

Consider an assignment π with $\pi(1) = \pi(2) = a_1$ and $\pi(3) = a_2$; under the assignment π , agents 1, 2 realize the outcome $(a_1, 2)$ and agent 3 realizes $(a_2, 1)$. It is easy to check that π is prefect (and thus IR, stable, and EF). Consider another assignment π' with $\pi'(1) = a_1$ and $\pi'(2) = \pi'(3) = a_0$; under π' , agent 1 realizes the outcome $(a_1, 1)$ and agents 2, 3 realize a_0 . π' is individually rational (as $(a_1, 1) \in S_1$), but it is not stable (as $(a_1, 2) \in S_2$) or envy-free (as $(a_1, 1) \in S_3$).

Darmann et al. proved many hardness results of the Group Activity Selection Problem even under restrictions on inputs of GASP [14]. Here we mention the most relevant results of theirs to this work.

Theorem 6. *Finding a perfect assignment is NP-hard. It remains to be NP-hard even if all agents have increasing preferences for all activities, even if all agents have decreasing preferences for all activities, or even if all activities are equivalent (Theorems 4.1, 4.2, 4.3, and 4.4 of [14]).*

As corollaries, the following four problems are also NP-hard even under any of the three restrictions mentioned in Theorem 6. Note that the first three problems parameterize the size of an assignment while the last problem parameterizes the number of used activities in a perfect assignment.

- k -IR-GASP: Does there exist an individually rational assignment of size k ?
- k -Stable-GASP: Does there exist a stable assignment of size k ?
- k -EF-GASP: Does there exist an envy-free assignment of size k ?
- k -Perfect-GASP: Does there exist a perfect assignment using k non-void activities?

In what follows, we show that the parameterized complexity of k -GASP varies with different solution concepts and under different restrictions on inputs to GASP.

4.2 Parameterized Complexity

In this section, we provide parameterized complexity of the four problems mentioned in previous section: k -IR-GASP, k -Stable-GASP, k -EF-GASP, and k -Perfect-GASP. Our main contributions are summarized in Table 4.1. Note that all problems considered in this work are known to be NP-hard (and NP-complete) due to Darmann et al. [14]. The “general case” refers to the case where approval set of each agent can be any set of outcomes (i.e., no restrictions on preferences of agents). “Increasing (decreasing) preferences” refer to the case where all agents have increasing (decreasing) preferences for all activities. “Equivalent activities” refer to the case where all activities are (pairwise) equivalent (i.e., they are copies of one kind of an activity).

	k -IR-GASP	k -Stable-GASP	k -EF-GASP	k -Perfect-GASP
General case	FPT	$W[1]$ -hard	$W[1]$ -complete	$W[2]$ -hard
Increasing preferences	FPT	$W[1]$ -complete	$W[1]$ -complete	$W[2]$ -hard
Decreasing preferences	FPT	FPT	$W[1]$ -complete	$W[2]$ -hard
Equivalent activities	FPT	FPT	FPT	Unknown

Table 4.1: Summary of results on parameterized complexity of GASP.

4.2.1 k -IR-GASP: Finding Individually Rational Assignments in GASP

Recall that k -IR-GASP is the problem of finding an IR assignment of size k . We show that k -IR-GASP is in FPT. As seen below, our algorithm runs in $\exp knp \log n$ time, where n is the number of agents and p the number of activities. The input size to GASP is $\Theta(n^2 p)$ as the number of possible outcomes is np and each of the n agents needs to specify a subset of approved ones. As we only care about solutions of size k , we are only interested in those preferences of the agents for numbers of activity participants that are at most k . We can hence prune the input to size $\Theta(nkp)$ (in about that much time, assuming random access to preferences). In general, we cannot prune the input more, and for any constant k , our FPT algorithm below runs in near linear time in the input size!

Theorem 7. *k -IR-GASP is in FPT, and can be solved in time $2^{O(k)}(np \log n)$ where $n = |N|$ and $p = |A^*|$.*

Proof. We use “Color Coding” to design a randomized (Monte Carlo) algorithm, which can easily be de-randomized using a family of k -perfect hash functions as shown in the work [2].

Recall that $N = \{1, 2, \dots, n\}$ is the set of agents, $A^* = \{a_1, a_2, \dots, a_p\}$ is the set of non-void activities, and S_i is the set of approved outcomes for agent i . We first color each agent using k colors independently and uniformly at random. We seek to assign exactly one agent of each color to some activity such that the resulting assignment is IR and of size k .

Let $c(i)$ denote the color of agent i where $c(i) \in [k]$. For each activity $a_j \in A^*$ and every subset C of colors (i.e., $C \subseteq [k]$), we will first determine whether it is possible to assign to activity a_j exactly $|C|$ agents with distinct colors specified by C while satisfying the IR constraint for each agent; we refer to this subproblem by $T(C, j)$ for every set of colors C and activity a_j . For any fixed a_j and C , we can check for every color $d \in C$ whether there exists an agent i with $c(i) = d$ and $(a_j, |C|) \in S_i$ in time $O(n)$ by iterating over the set of agents and look up her approval set, S_i . If the test is affirmative, we conclude that we can assign exactly $|C|$ agents with distinct colors specified by C to activity a_j . We perform this sub-routine for every activity $a_j \in A^*$ and every subset of colors, which can be done in time $O(n \cdot p \cdot 2^k)$ overall.

Next, we solve another type of sub-problems (which we call $R(C, j)$) to check if it is possible to assign $|C|$ agents of distinct colors in C to activities in $A_j = \{a_1, a_2, \dots, a_j\}$ for every $j \leq p$ and $C \subseteq [k]$. When $j = 1$, the sub-problems $R(C, j)$ and $T(R, j)$ are equivalent, and thus we simply use the result of $T(R, j)$ to solve $R(C, j)$. To solve $R(C, j)$ when $j > 1$, we enumerate over every subset $C' \subseteq C$, and solve $R(C', j-1)$ and lookup the result of $T(C \setminus C', j)$. If both subproblems $R(C', j-1)$ and $T(C \setminus C', j)$ are affirmative for some $C' \subset C$, then we conclude that $R(C, j)$ is also affirmative.

If $R([k], p)$ is affirmative, then we can find an IR assignment of size k where those k agents are distinctly colored. There are at most $O(2^k \cdot p)$ subproblems in the form of $R(C, j)$ to be solved, and each subproblem can be solved in time $O(2^k)$ (as we enumerate over all subsets of C).

Therefore the overall running time of this algorithm is bounded by $O(4^k \cdot p + 2^k \cdot (np)) = 2^{O(k)}(np)$; in particular, it is exponential only in k . It is easy to confirm that this algorithm is a Monte Carlo algorithm (i.e., if it finds a solution, it is guaranteed to be IR). On the other hand, if there exists an IR assignment of size k in the original instance, there is a chance that this algorithm does not find it when the k agents are not colored properly. The probability of a proper coloring (i.e., the k agents in the assumed assignment are colored distinctly) is at least $k!/k^k > 1/e^k$, and this is only exponentially small in k . Therefore one can repeat this randomized algorithm $e^k \ln n$ times to increase the probability of success to $1 - 1/n$ (with overall runtime $2^{O(k)}(np \log n)$).

To de-randomize this algorithm, one can use a k -perfect family of hash functions from N to $[k]$. Specifically, if we have a list of colorings of agents N such that for every subset $N' \subseteq N$ of size $|N'| = k$ there exists a coloring in the list that gives each agent in N' a distinct color, then we can simply enumerate over this list of colorings. This is precisely what a k -perfect family of hash functions from N to $[k]$ is, and it is known that the size of the family can be specified using $2^{O(k)} \log n$ bits (for details please see [2]). Therefore we can obtain a deterministic algorithm whose runtime is bounded by $2^{O(k)}(np \log n)$, and conclude that k -IR-GASP is in FPT. \square

Individual rationality is the weakest solution concept among the four we consider, and naturally k -IR-GASP is the least complex problem. On the other hand, we will show that other problems are $W[1]$ - or $W[2]$ -hard unless additional restrictions are assumed. Note that Theorem 7 proves an easiness result, and therefore it is implied that k -IR-GASP is in FPT under any of the three restrictions on inputs we consider (see the k -IR-GASP column in Table 4.1).

4.2.2 k -Stable-GASP: Finding Stable Assignments

Recall that k -Stable-GASP is the problem of finding a stable assignment of size k . Stability is a stronger solution concept than individual rationality, and the problem of finding a stable assignment is harder than that of finding an IR assignment. This relationship is not apparent under the classic complexity hierarchy as both problems are NP-complete. However, under parameterization, k -IR-GASP is FPT whereas k -Stable-GASP is $W[1]$ -hard.

Theorem 8. *k -Stable-GASP is $W[1]$ -hard. The problem remains to be $W[1]$ -hard even if each agent approves at most one size per activity (i.e., $|S_i(a_j)| \leq 1$ for all $i \in N$ and $a_j \in A^*$).*

Proof. The k -Clique problem is known to be $W[1]$ -hard, and we reduce the k -clique problem to k -Stable-GASP.

Construction of GASP instance. Consider an instance of the k -Clique problem, $G = (V, E)$ and a parameter k where $V = \{v_1, v_2, \dots, v_n\}$. Let us create an instance of GASP as follows: Let $N = V \cup \{w_{i,x} : (1 \leq i \leq n) \wedge (1 \leq x \leq k-1)\}$; that is, we create n node-agents v_i 's (by abusing notation) and $(k-1)$ copies of neighbor-agents ($w_{i,x}$'s) for each v_i . The neighbor-agents will be used to “select” the $k-1$ edges incident to each node if the node is to be included in a clique we are seeking. Let $A^* = \{a_1, \dots, a_k\} \cup \{e_{i,j} : 1 \leq i < j \leq n\}$; we create k clique-activities (which are used to determine membership of a node in a clique) and $\binom{n}{2}$ edge-activities $e_{i,j}$ (where $i < j$). For each node-agent v_i , we set its approval set $S_{v_i} = \{(a_j, 1) : 1 \leq j \leq k\} \cup \{(e_{i,j}, 3) : i \neq j\}$. For each neighbor-agent $w_{i,x}$, we set its approval set $S_{w_{i,x}} = \{(e_{i,j}, 2) : (v_i, v_j) \in E\}$. Finally we set the parameter k' of GASP (to distinguish from k in the Clique problem) to $k' = k + 2\binom{k}{2}$. This is a valid fpt-reduction as k' depends only on k but not on n , and the size of our instance of GASP is bounded by $O(nk^3)$ as there are $O(nk)$ agents and $O(k^2)$ activities in the instance.

Let us first describe how a clique in the original instance and a stable assignment in the GASP instance we created are related. A node-agent is assigned to a clique-activity if and only if its corresponding node belongs to a (corresponding) clique. For each node-agent, there exists $k-1$ neighbor-agents, and these neighbor-agents must be assigned properly to edge-activities in order to ensure that the resulting set of nodes is indeed a clique. We claim that there exists a clique of size k in the original instance if and only if there exists a stable assignment of size k' in the GASP instance we constructed.

Proof of equivalence between instances. First suppose there exists a clique C of size k in G , and without loss of generality assume $C = \{v_1, v_2, \dots, v_k\}$. Consider the following assignment π :

$$\pi(v_i) = \begin{cases} a_i & i \leq k \\ a_\emptyset & i > k \end{cases} \quad \text{and} \quad \pi(w_{i,x}) = \begin{cases} e_{i,x+1} & i \leq k \wedge i \leq x \\ e_{x,i} & i \leq k \wedge i > x \\ a_\emptyset & i > k \end{cases}$$

That is, node-agents are assigned to the clique-activities and their associated neighbor-agents are assigned to the edge-activities; all other agents are assigned to the void activity. Clearly π assigns exactly $k + 2\binom{k}{2} = k'$ agents to non-avoid activities. While the details are omitted, it is easy to verify

that π is indeed a stable assignment.

Conversely, suppose there is a stable assignment π of size $k' = k + 2\binom{k}{2}$, and we want to show that a clique of size k exists in G . First notice that for each edge-activity $e_{i,j}$ there are precisely two agents who approve the outcome $(e_{i,j}, 3)$ – namely, v_i and v_j . Therefore if π is stable, it cannot assign any node-agents (of the form v_i) to any edge-activity (of the form $e_{i,j}$). In other words, for each v_i , $\pi(v_i) \in \{a_\emptyset\} \cup \{a_1, \dots, a_k\}$. Let $C = \{v_i : \pi(v_i) \neq a_\emptyset\}$; since there are k clique-activities, $|C| \leq k$. We claim that $|C| = k$ if π is stable; if $|C| < k$, then there exists some a_j such that no agent is assigned to it; since $k \leq n$, there must be some v_i such that $\pi(v_i) = a_\emptyset$. This implies that π is not stable because $(a_j, 1) \in S_{v_i}$ while $\pi(v_i) = a_\emptyset$; hence $|C| = k$ must hold. Without loss of generality, we now assume that $\pi(v_i) = a_i$ if $i \leq k$ and $\pi(v_i) = a_\emptyset$ if $i > k$ (by re-labeling), and we claim that $C = \{v_1, v_2, \dots, v_k\}$ is a k -clique in the original instance.

We argued earlier that π never assigns node-agents to any edge-activities if it is stable. This implies that, if π assigns any agent to an edge-activity, it must be the case that π assigns exactly two neighbor-agents (of the form $w_{i,x}$) to it (due to the construction of $S_{w_{i,x}}$'s). If π is stable, then π must assign no neighbor-agents to $e_{i,j}$ if $i > k$ or $j > k$ and exactly two neighbor-agents to $e_{i,j}$ if $i \leq k$ and $j \leq k$. To prove the first claim, suppose that π assigns two neighbor-agents to $e_{i,j}$ where $i > k$ (and recall that $\pi(v_i) = a_\emptyset$ when $i > k$). Then π is not stable because $(e_{i,j}, 3) \in S_{v_i}$, and thus v_i wishes to join $e_{i,j}$, and this is a contradiction. Similarly one can prove the claim in the case where $j > k$. To prove the second part, recall that $|\pi| = k' = k + 2\binom{k}{2}$. Since π assigns exactly k node-agents to non-void activities, it must assign $k(k-1) = 2\binom{k}{2}$ neighbor-agents to $\binom{k}{2}$ edge-activities from $\{e_{i,j} : i, j \leq k\}$. By the Pigeon Hole principle, π must assign two agents to each of the edge-activities in $\{e_{i,j} : i, j \leq k\}$. This implies that there is an edge between v_i and v_j in the original instance if $i, j \leq k$. Otherwise, if $(v_i, v_j) \notin E$ where $i, j \leq k$, then there is no neighbor-agents who can be assigned to $e_{i,j}$, which contradicts the assumption that π is of size k' . This completes the proof of the claim that if a stable assignment of size k' exists, then a clique of size k exists, and one can construct a clique by choosing the corresponding nodes to the node-agents that are assigned to one of the clique-activities.

Note that in our reduction each agent approves at most one size per activity, proving the second statement in the theorem. \square

Because our reduction increases the parameter quadratically (i.e., $k' = k + 2\binom{k}{2} = k^2$), the following corollary follows immediately.

Corollary 1. *Unless the Exponential Time Hypothesis (ETH) fails, k -Stable-GASP cannot be solved*

in time $(np)^{o(\sqrt{k})}$.

We now consider k -Stable-GASP with the restriction that all agents have increasing preferences for all activities.

Theorem 9. *k -Stable-GASP is $W[1]$ -complete when all agents have increasing preferences for all activities.*

Proof. We show $W[1]$ -hardness of the problem by reducing from the k -Clique problem, but we omit the proof of completeness which can be shown by reducing k -Stable-GASP to the k -Clique problem.

Construction of GASP instance. Let $G = (V, E)$ be a graph instance of the k -Clique problem. For each vertex $v_i \in V$, we create k^2 copies of v_i as agents (call them copies of v_i) and create an activity a_i ; this creates $k^2|V|$ agents and $|V|$ activities. For each edge $e_{i,j} = (v_i, v_j) \in E$, we create two copies of $e_{i,j}$ as agents (call them copies of $e_{i,j}$) and create an activity $w_{i,j}$; this creates $2|E|$ agents and $|E|$ activities. Let $k' = k^3 + k^2 - k$, and we create $k' + 1$ copies of dummy agents (call them copies of z). For each of the k^2 copies of v_i agents, we set its approval set such that $l_{v_i}(a_i) = k^2$ (i.e., approves any outcome with a_i and size k^2 or larger) and $l_{v_i}(w_{i,j}) = 3$ if $(v_i, v_j) \in E$ and $l_{v_i}(\cdot) = n + 1$ for all other activities (where $n = k^2|V| + 2|E| + k' + 1$ is the total number of agents we create). For each of the two copies of $e_{i,j}$ agents, we set its approval set such that $l_{e_{i,j}}(w_{i,j}) = 2$. For each of the $k' + 1$ copies of z agents, we set its approval set such that $l_z(w_{i,j}) = 4$ for all (i, j) where $(v_i, v_j) \in E$. We claim that a clique of size k exists in G if and only if a stable assignment of size k' exists in the GASP instance we created.

Proof of equivalence between instances. First, suppose that $C = \{v_1, v_2, \dots, v_k\}$ is a clique of size k in G . We can construct a stable assignment of size k' as follows: (a) For k^2 copies of v_i , we assign them to a_i if $v_i \in C$ and to a_\emptyset otherwise, (b) for two copies of $e_{i,j}$, we assign them to $w_{i,j}$ if $v_i \in C$ and $v_j \in C$ and to a_\emptyset otherwise, and (c) copies of z are assigned to a_\emptyset . Note that this assignment assigns exactly $k^3 + 2\binom{k}{2} = k^3 + k(k-1) = k'$ agents to non-void activities. It is easy to verify that π is IR and stable, proof of which is created to due space.

Conversely, now suppose that π is a stable assignment of size k' , and we show that there exists a clique of size k in G . If π assigns three or more agents to any $w_{i,j}$, then π must assign all copies of z to some activity (possibly $w_{i,j}$) or π would not be stable; yet we know that π is of size k' and there are $k' + 1$ copies of z , and therefore π can only assign two or fewer agents to each $w_{i,j}$. If π assigns two agents to some $w_{i,j}$, then those two agents must be the two copies of $e_{i,j}$ because no

other agent approves the outcome $(w_{i,j}, 2)$. Furthermore, if π assigns the two copies of $e_{i,j}$ to $w_{i,j}$, then π must assign all k^2 copies of v_i to a_i and all k^2 copies of v_j to a_j – otherwise, π would not be stable. Let W be the set of activities of the form $w_{i,j}$ such that π assigns exactly two agents to $w_{i,j}$; if $|W| > \binom{k}{2}$, then there must be at least $k+1$ indices that appear in elements of W , which implies that π must assign agents to at least $k+1$ activities of the form a_i . This is a contradiction because π is of size k' but $(k+1)k^2 > k'$. Therefore, $|W| \leq \binom{k}{2}$. Now suppose $|W| < \binom{k}{2}$ instead. As argued earlier, π can assign to at most k activities of the form a_i , but $k^3 + 2|W| < k'$, which implies that π cannot be of size k' if $|W| < \binom{k}{2}$. Lastly, suppose $|W| = \binom{k}{2}$ (and from previous arguments, it is clear that the number of the indices that appear in the elements of W must be exactly k); without loss of generality, assume $W = \{w_{i,j} : 1 \leq i < j \leq k\}$ (by re-labeling) – this implies that π assigns k^2 copies of v_l to a_l if $1 \leq l \leq k$, but more importantly, it implies that $(v_i, v_j) \in E$ because we create $w_{i,j}$ if and only if there is an edge between v_i and v_j . That is, $C = \{v_1, v_2, \dots, v_k\}$ is a clique in the original instance. This completes the proof of W[1]-hardness of the problem. \square

Unlike the case of increasing preferences, if all agents have decreasing preferences the problem admits an FPT algorithm.

Theorem 10. *k -Stable-GASP is in FPT when all agents have decreasing preferences for all activities.*

Proof. We use Color Coding to reduce the k -Stable-GASP with decreasing preferences to a variant of the Vertex Cover problem. With probability which is exponentially small only in k , we color agents and activities “properly”, and given a proper coloring we can find a stable assignment of size k in polynomial time in n, p yet exponential only in k .

Preliminaries. Suppose that a stable assignment of size k exists, and without loss of generality we know that it assigns k agents to l distinct activities (where $l \in [1, k]$), which can be done by checking every value in $[1, k]$. We first color agents and activities using l colors 1 through l , uniformly and independently at random (let $c(i)$ denote the color of agent i and $c(a_j)$ the color of activity a_j), and then fix the value of k_d for each $d \in [1, l]$ such that $\sum_{d \in [1, l]} k_d = k$. We say that the coloring c (together with l and k_d 's) is compatible with a stable assignment π of size k (using l activities) if π assigns exactly k_d agents to an activity of color d for every $d \in [1, l]$. Given some coloring c , our algorithm will find a stable assignment compatible with c or determine that no such stable assignment exists. It is clear that any stable assignment (of size k) has at least one compatible coloring. With probability at least $(1/l)^{l+k}$ (which is exponentially small only in k), our randomized coloring is a compatible

coloring of a stable assignment of size k (provided that it exists); the algorithm can be easily be de-randomized using a family of k -perfect hash functions as shown in the work [2].

FPT Algorithm. We now proceed with fixed values of l and k_d 's as well as some coloring c as described earlier. We will use the special color $l + 1$ to mark the agents and activities that cannot be assigned/used in any stable assignment that is compatible with the given coloring c . Define $N_d = \{i \in N : c(i) = d\}$ and $A_d^* = \{a_j \in A^* : c(a_j) = d\}$ where $d \in [1, l + 1]$; these subsets naturally partition N and A^* into $l + 1$ subsets by their colors (at first N_{l+1} and A_{l+1}^* are empty, but we may re-color some agents and activities during the course of the algorithm). Let $N(a_j) = \{i \in N_{c(a_j)} : u_i(a_j) \geq k_{c(a_j)}\}$, which is the set of agents who have the same color as a_j and approve the size $k_{c(a_j)}$ for activity a_j (recall that agents have decreasing preferences, so we only need to check their upper-bound $u_i(a_j)$ for a given activity a_j). If $|N(a_j)| > k_{c(a_j)}$, then we label the activity a_j as “popular” because any compatible assignment must assign $k_{c(a_j)}$ agents of the same color to a_j , but more than $k_{c(a_j)}$ agents approve a_j for size $k_{c(a_j)}$. If any color $d \in [1, l]$ contains two or more popular activities, we reject the coloring because there is no stable assignment compatible with this coloring. To see why, if no agents are assigned to a popular activity of some color d , then due to compatibility there must exist at least one agent of the same color who is assigned to the void activity but approves the popular activity for size 1. Therefore, any stable, compatible assignment must assign k_d agents to a popular activity for color d (if any), but if there exist multiple popular activities of the same color, then no compatible assignment is stable. Without loss of generality (by re-coloring) let us assume that colors $[1, q]$ contain exactly one popular activity and colors $[q + 1, l]$ contain non-popular activities (it is possible that $q = 0$ or $q = l$). To emphasize, we shall refer to colors in $[1, q]$ as “popular” colors and in $[q + 1, l]$ as “unpopular” colors.

Let us now examine each color to decide whether we should reject the coloring or whether we can exclude some agents and/or activities from consideration (by re-coloring them as the special color, $l + 1$). First, for each popular color $d \in [1, q]$ with a popular activity a_{j_d} (recall that there is exactly one popular activity for each popular color), we re-color all agents in $(N_d \setminus N(a_{j_d}))$ and all activities in $(A_d^* \setminus \{a_{j_d}\})$ as the special color $(l + 1)$ because they cannot be assigned/used in any stable assignment compatible with c as we argued earlier. Next, for each unpopular color $d \in [q + 1, l]$, if there exist two distinct activities $a_j, a_{j'} \in A^*(d)$ such that $N(a_j) \neq N(a_{j'})$, then we reject the coloring; any compatible assignment must assign no agents to at least one of these two activities (assume that a_j is such activity), but at least one agent in $N(a_j)$ approves $(a_j, 1)$ (due to decreasing preferences) while she must be assigned to the void activity, which implies instability

of the assignment. If the coloring is not rejected after these conditions are checked, then we have $N(a_j) = N(a_{j'})$ for all $a_j, a_{j'} \in A^*(d)$ where $d \in [q+1, l]$. Let us re-color all agents in $N(d) \setminus N(a_j)$ as $l+1$ where a_j is any activity in $A^*(d)$ for all $d \in [q+1, l]$. We then check for another condition for each unpopular color $d \in [q+1, l]$. Let $A'(d) = \{a_j \in A^*(d) : \exists i \in N_{l+1}, u_i(a_j) \geq 1\}$. If $A'(d)$ contains two or more activities, it is clear that the coloring must be rejected because agent i (who cannot be assigned to any activity under the given coloring) approves size 1 for the activities in $A'(d)$ but the assignment can only choose one activity from $A^*(d)$. Therefore, if $|A'(d)| \geq 2$ then we reject the coloring; otherwise, if $|A'(d)| = 1$, then we re-color all activities in $A^*(d) \setminus A'(d)$ as $l+1$ (as the only one in $A'(d)$ must be used for color d). If $|A'(d)| = 0$, this step has no effect for this color. Lastly, we now consider the agents of color $l+1$ (who must be assigned to the void activity by any stable assignment compatible with the given coloring). Let us define $k_{l+1} = 0$ for convenience (i.e., we do not assign any agents of color $l+1$ to any activities). For each color $d \in [1, l+1]$, if there exists some activity $a_j \in A^*(d)$ and some agent $i \in N(l+1)$ with $u_i(a_j) \geq k_d + 1$, then we reject the coloring because agent i is to be assigned to the void-activity, but she approves the outcome $(a_j, k_d + 1)$ as well as $(a_j, 1)$ (due to decreasing preferences), which means that regardless of whether a_j is used or not, no assignment would not be stable and compatible at the same time due to agent i . If the coloring has not been rejected, then we can now safely ignore all agents in $N(l+1)$ (as if they are non-existent) because stability constraint would not be violated by those agents.

We now proceed with the assumption that the coloring has not been rejected by our algorithm. Recall that we need to choose k_d agents among N_d where $d \in [1, q]$ to be assigned to the popular activity a_{j_d} while we know exactly which $k_{d'}$ agents must be assigned to one of the activities in $A_{d'}^*$ where $d' \in [q+1, l]$. For each popular color $d \in [1, q]$ define $N'_d = \{i \in N_d : \exists d' \in [1, l+1], u_i(a_j) \geq k_{d'} + 1 \text{ where } a_j \in A_{d'}\} \cup \{i \in N_d : \exists d' \in [q+1, l], |\{a_j \in A_{d'} : u_i(a_j) \geq 1\}| \geq 2\}$. Any stable assignment compatible with c must assign all agents in N'_d to the popular activity a_{j_d} . Otherwise, if some agent i in N'_d is assigned to the void-activity instead, then the resulting assignment cannot be stable; if i is contained in the first set (on the right-hand-side of definition of N'_d) above, then i approves sizes of both $k_{d'} + 1$ and 1 for some activity a_j , which implies that regardless of whether a_j is used or not, i would wish to join a_j instead of a_\emptyset , while if i is contained in the second set (on the right-hand-side of definition of N'_d), then i approves size 1 for at least two non-popular activities of the same color which implies that i would wish to join one of them that is not used. Therefore, if $|N'_d| > k_d$ for some $d \in [1, q]$ we must reject the coloring, and otherwise we must assign all agents in N'_d to a_{j_d} . Without loss of generality we can assume that $N'_d = \emptyset$ for all $d \in [1, q]$ (provided that the coloring is not rejected by this point) by assigning all such agents to the appropriate popular activity

and then decreasing k_d by $|N'_d|$ before we proceed to the next step.

Now suppose that for some color $d \in [1, q]$ and agent $i \in N_d$ and some color $d' \in [q+1, l]$ and some activity $a_j \in A_{d'}^*$, we have $u_i(a_j) \geq 1$. If i is assigned to a_\emptyset (instead of a_{d_j}) and a_j is not used (no agents is assigned to it), then the assignment cannot be stable as i approves $(a_j, 1)$. That is, any compatible, stable assignment must assign i to a_{d_j} and/or use activity a_j . If we consider agents in $X = \cup_{d \in [1, q]} N_d$ and activities in $Y = \cup_{d' \in [q+1, l]} A_{d'}^*$ as vertices and there is an edge between (i, a_j) if and only if $u_i(a_j) \geq 1$ where $i \in X$ and $a_j \in Y$ (as a bipartite graph), finding a compatible assignment is equivalent to finding a vertex cover such that it chooses exactly k_d vertices from each N_d with $d \in [1, q]$ and exactly 1 vertex from each $A_{d'}^*$ with $d' \in [q+1, l]$. Because the total number of vertices to be selected is bounded above by $k+l$, one can use a bounded search tree to determine whether a vertex cover of a small size exists or not in FPT time (i.e., exponential only in k but polynomial in n, p). If vertex i from X is chosen then we assign i to the popular activity of the same color and if vertex a_j from Y is chosen then we assign the agents of the same color to it. It is easy to verify that a compatible, stable assignment exists if and only if a vertex cover (with the aforementioned constraints) exists in this bipartite graph.

While we omit the details, it is straightforward to verify that our algorithm would not reject any coloring c which is compatible with at least one stable assignment. \square

Lastly, we consider another special case of GASP when all activities are (pairwise) equivalent. In this case, the problem of finding a stable assignment becomes the problem of partitioning agents into groups where only group sizes matter (as all of the activities are identical to every agent). This restriction allows the problem for an FPT algorithm

Theorem 11. *k -Stable-GASP is in FPT if all (non-void) activities are equivalent.*

Proof. Let $p = |A^*|$ be the number of non-void activities which we assume are all equivalent. We use Color Coding to design a randomized FPT algorithm, which can easily be de-randomized using a family of k -perfect hash functions as shown in the work [2].

We can assume that $p \leq k+1$ because k agents can be assigned to at most k copies and having more than one extra copy to which no agents is assigned does not change the problem (this is because we are seeking a solution of size exactly k). For brevity, we only prove the claim when $p = k+1$ in this work, but it can be easily extended to the cases when $p < k+1$.

Preliminaries. Let $N = \{1, 2, \dots, n\}$ be the set of n agents and $A^* = \{a_1, a_2, \dots, a_p\}$ be the set of p copies of the only activity when $p = k+1$. Recall that by definition of equivalent activities, every

agent i has $S_i(a_j) = S_i(a_{j'})$ for all j, j' . We first fix l (the number of copies of the activity to be used by a stable assignment) which must be between 1 and k , and k_1, k_2, \dots, k_l which is the number of agents assigned to each of the l copies; for convenience we define $k_{l+1} = 0$ as there is at least one extra copy that would not be used by the assignment. The total number of possible values for l and k_d 's are bounded above by $O(k^k)$, which is exponential only in k . After we fix l , we color all agents uniformly and independently at random using colors 1 through l ; let $c(\cdot)$ be this coloring scheme and $c(i)$ denote the color of agent i . We say that coloring c (together with l and k_d 's) and a stable assignment π of size k are *compatible* if π assigns exactly k_d agents of color d to activity a_d for $d \in [1, l]$.

FPT Algorithm. Our algorithm will find a stable assignment compatible with c or determine that no such stable assignment exists. Note that any stable assignment of size k has at least one compatible coloring, and the probability that a random coloring we choose is compatible with some fixed stable assignment of size k is at least $(1/l)^k$ (as we must color those k agents correctly) which is exponentially small only in k .

Our algorithm first partitions agents of each color into several subsets, and check several necessary conditions for the coloring to be compatible with at least one stable assignment; if any of the conditions is not met, the coloring will be rejected by the algorithm (as there is no stable assignment compatible with the given coloring). For each color $d \in [1, l]$, the algorithm computes three subsets: $N_d = \{i \in N : c(i) = d\}$, $N_d^{\text{IR}} = \{i \in N_d : (a_d, k_d) \in S_i\}$, and $N_d^{\text{IN}} = \{i \in N_d : \exists d' \in [1, l+1] \text{ s.t. } (a_{d'}, k_{d'} + 1) \in S_i\}$ (recall $k_{l+1} = 0$). If $|N_d^{\text{IR}}| < k_d$ for any $d \in [1, l]$, no stable assignment is compatible with c because assigning k_d agents to a_d would not be individually rational (i.e., not enough agents approve the outcome), so the coloring should be rejected in this case. If for some $d \in [1, l]$ the set $N_d^{\text{IN}} - N_d^{\text{IR}}$ is not empty but contains some agent i , then no stable assignment is compatible with c because a stable assignment cannot assign i to a_d (because $i \notin N_d^{\text{IR}}$) but i would wish to join $a_{d'}$ for some $d' \in [1, l+1]$ which would make the assignment not stable; hence the coloring should be rejected in this case. If $|N_d^{\text{IN}}| > k_d$, then at least one agent i in N_d^{IN} should be assigned to the void activity, but i would wish to join $a_{d'}$ for some $d' \in [1, l+1]$ which would make the assignment not stable; hence the coloring should be rejected. If the coloring is not rejected by any of the cases mentioned earlier, then we have the following three conditions for every color $d \in [1, l]$: (a) $|N_d^{\text{IR}}| \geq k_d$, (b) $|N_d^{\text{IN}}| \leq k_d$, and (c) $N_d^{\text{IR}} \subseteq N_d^{\text{IN}}$. Let us define X_d for each $d \in [1, l]$ as follows: X_d contains an arbitrary set of k_d agents from N_d^{IR} such that every agent in N_d^{IN} is contained in X_d . Note that this is always possible due to the three conditions mentioned above. We claim that

an assignment π which assigns agents in X_d to a_d and all other agents to a_\emptyset is a stable assignment compatible with c . To prove compatibility, all agents in X_d are by definition of color d and $|X_d| = k_d$ for all $d \in [1, l]$. To prove stability, first consider any agent i who is assigned to the void activity and suppose $d = c(i)$. Since $i \notin X_d$, we know that $i \notin N_d^{\text{IN}}$ by definition, and therefore there is no $d' \in [1, l+1]$ such that $(a_{d'}, k_{d'} + 1) \in S_i$. Now consider any agent i who is assigned to a_d by π (thus $c(i) = d$). By definition $i \in X_d$ and thus $i \in N_d^{\text{IR}}$, which implies that $(a_d, k_d) \in S_i$. Therefore π is a stable assignment of size k , compatible with c .

Let us now prove that if there is at least one stable assignment that is compatible with c , then the algorithm does not reject the coloring. Let π be one such assignment and let X_d be the set of agents assigned to a_d by π . Due to compatibility we have $|X_d| = k_d$ and $c(i) = d$ for all $i \in X_d$ for all $d \in [1, l]$; in particular, by definition $X_d \subseteq N_d^{\text{IR}}$ and thus the first condition (a) above holds for all $d \in [1, l]$. Due to stability of π , every agent i with $\pi(i) = a_\emptyset$ satisfies that $\nexists d' \in [1, l+1]$ such that $(a_{d'}, k_{d'} + 1) \in S_i$. Therefore the conditions (b) and (c) above hold for all $d \in [1, l]$, which proves that the coloring c would not be rejected by the algorithm.

We have shown that if we begin with a coloring c (together with l and k_d 's) that is compatible with at least one stable assignment, then our algorithm would find a stable assignment compatible with the coloring and that if no such assignment exists the coloring would be rejected. This is a Monte Carlo algorithm with probability of success at least $(1/k)^k$ and runtime bounded by $O((k^k)nk)$ (as our algorithm must enumerate all possible values of l and k_d 's), which is polynomial in n but exponential only in k . \square

4.2.3 k -EF-GASP: Finding Envy-free Assignments

Recall that k -EF-GASP is the problem of finding an envy-free assignment of size k . Similarly to how we showed that finding a stable assignment is computationally harder than finding an IR assignment under parameterization, we can show that finding an envy-free (EF) assignment is also computationally harder than finding an IR assignment under parameterization; again, this relationship is not apparent under the classic complexity classes (i.e., NP-hardness).

Theorem 12. *k -EF-GASP is $W[1]$ -complete. The problem remains to be $W[1]$ -complete even if each agent approves at most one size per activity.*

Proof. We reduce from the k -Clique problem. Let $G = (V, E)$ be an undirected graph and k be a parameter of the k -Clique instance.

Construction of GASP instance. Let $V = \{v_1, v_2, \dots, v_n\}$, and for each vertex $v_i \in V$, we create activity a_i and k^2 copies of v_i as agents. For each edge $(v_i, v_j) \in E$, we create an activity $e_{i,j}$ and an agent $w_{i,j}$. This creates $|V| + |E|$ activities and $k^2|V| + |E|$ agents overall. For each copy of v_i , we set its approval set as $S_{v_i} = \{(a_i, k^2)\} \cup \{(e_{i,j}, 1) : (v_i, v_j) \in E\}$ and for each agent $w_{i,j}$ we set its approval set as $S_{w_{i,j}} = (e_{i,j}, 1)$. Let $k' = k^3 + \binom{k}{2} = k^3 + k(k-1)/2$. We claim that a clique of size k exists in G if and only if an envy-free assignment of size k' exists in the GASP instance we created.

Proof of equivalence between instances. Suppose that π is an envy-free assignment of size k' . If π assigns any copy of v_i to some activity $e_{i,j}$, then π cannot be envy-free because $w_{i,j}$ wishes to be assigned to $e_{i,j}$ in place of the copy of v_i ; furthermore, π cannot assign more than one agent to any $e_{i,j}$ as no other agent approves the activity with any other size than 1. If π assigns any copy of v_i to a_i , then it must assign all k^2 copies of v_i to a_i as those agents only approve a_i with size k^2 . Because π is of size k' , it is clear that π can only assign agents to at most k different activities of the form a_i . Now suppose π assigns some $w_{i,j}$ to $e_{i,j}$; due to envy-freeness, all k^2 copies of v_i must be assigned to a_i and all k^2 copies of v_j must be assigned to a_j ; this implies that π can assign at most $\binom{k}{2}$ agents of the form $w_{i,j}$ to activities of the form $e_{i,j}$ (otherwise, π cannot be of size k' because $k^2(k+1) > k'$). Therefore, we conclude that π assigns k^3 agents of the form v_i to k activities of the form a_i (without loss of generality, assume those activities are $\{a_1, a_2, \dots, a_k\}$) and that π assigns $w_{i,j}$ to $e_{i,j}$ if and only if $1 \leq i, j \leq k$ (all other agents are assigned to the void activity). This implies that the original instance contains a clique $C = \{v_1, v_2, \dots, v_k\}$ as there is an edge (v_i, v_j) if $1 \leq i, j \leq k$.

To prove the converse, suppose that $C = \{v_1, v_2, \dots, v_k\}$ is a clique in the original instance. Let π be an assignment such that π assigns k^2 copies of v_i to a_i if $i \leq k$ and $w_{i,j}$ to $e_{i,j}$ if $1 \leq i, j \leq k$ and assigns all other agents to the void activity. Clearly π is individually rational by the construction of approval sets; π is also envy-free because no copy of v_i with $i > k$ or $w_{i,j}$ with $i > k$ or $j > k$ wishes to replace any other agent who is assigned to a non-void activity. This completes the proof of $W[1]$ -hardness.

Note that in our reduction each agent approves at most one size per activity, proving the second statement in the theorem.

To show completeness, one can reduce k -EF-GASP to the colored k -clique problem which is known to be $W[1]$ -complete (we omit the details). \square

Although k -EF-GASP is $W[1]$ -complete, the problem may admit FPT algorithms if we assume that preferences of agents are restricted to increasing preferences or decreasing preferences. Unlike

the case of stable assignments, we show that k -EF-GASP remains to be $W[1]$ -complete even if all agents have increasing preferences or all agents have decreasing preferences.

Theorem 13. *k -EF-GASP is $W[1]$ -complete even if all agents have increasing preferences.*

sketch. A slight modification to the reduction we used in proof of Theorem 12 shows that k -EF-GASP is $W[1]$ -hard even if all agents have increasing preferences. We construct the same instance, but we change the approval set of each agent such that if agent i approves an outcome (a, x) for some activity a and size x , then we let the agent approve all outcomes (a, x') with $x < x' \leq |N|$, which ensures that all agents have increasing preferences. In addition we create $k' + 1$ copies of a dummy agent z such that z approves all outcomes $(e_{i,j}, x)$ with $2 \leq x \leq |N|$ for all activities $e_{i,j}$ we create.

It is easy to see that if π is an envy-free assignment of size k' , then π cannot assign any copy of z to any activity because if π assigns any copy of z to some $e_{i,j}$, then π must assign all copies of z to some activity (to avoid envy-freeness with respect to those copies) which results in the size of π being at least $k' + 1$. This in turn implies that π cannot assign more than one agent to any $e_{i,j}$; if two or more agents are assigned to $e_{i,j}$, then all copies of z must be assigned to some activity (or they would be envious) due to increasing preferences. Lastly, this implies that π cannot assign any v_i to any $e_{i,j}$ because such assignment implies $w_{i,j}$ must also be assigned to $e_{i,j}$ (due to envy-freeness and increasing preferences), resulting in more than one agent being assigned to $e_{i,j}$. With this observation, the rest of the proof follows in a straightforward manner, and we omit details. Note that completeness follows from the fact that k -EF-GASP is in $W[1]$. \square

Theorem 14. *k -EF-GASP is $W[1]$ -complete even if all agents have decreasing preferences.*

sketch. A slight modification to the reduction we used in proof of Theorem 12 shows that k -EF-GASP is $W[1]$ -hard even if all agents have decreasing preferences. We construct the same instance, but we change the approval set of each agent such that if agent i approves an outcome (a, x) for some activity a and size x , then we let the agent approve all outcomes (a, x') with $1 \leq x' < x$, which ensures that all agents have decreasing preferences. The rest of the proof follows in a straightforward manner, and we omit details. Note that completeness follows from the fact that k -EF-GASP is in $W[1]$. \square

Theorem 15. *k -Stable-GASP is in FPT if all (non-void) activities are equivalent.*

Proof. Let $p = |A^*|$ and let all non-void activities be equivalent. We use Color Coding to design a randomized FPT algorithm, which can easily be de-randomized using a family of k -perfect hash

functions as shown in the work [2]. Because there are only n agents we can assume that $p \leq k$ because k agents can be assigned to at most k copies (unlike the case of stability, extra copies have no effect in envy-freeness as agents are only envious of others who participate in some activity). For simplicity we only prove the claim when $p = k$ but it can be easily extended to the cases when $p < k$.

Preliminaries. Let $N = \{1, 2, \dots, n\}$ be the set of n agents and $A^* = \{a_1, a_2, \dots, a_p\}$ be the set of p copies of the only activity when $p = k$. Recall that by definition of equivalent activities, every agent i has $S_i(a_j) = S_i(a_{j'})$ for all j, j' . We first fix l (the number of copies of the activity to be used by an EF assignment) which must be between 1 and k , and k_1, k_2, \dots, k_l which is the number of agents assigned to each of the l copies. The total number of possible values for l and k_d 's are bounded above by $O(k^k)$, which is exponential only in k . After we fix l , we color all agents uniformly and independently at random using colors 1 through l ; let $c()$ be this coloring scheme and $c(i)$ denote the color of agent i . We say that coloring c (together with l and k_d 's) and an EF assignment π of size k are *compatible* if π assigns exactly k_d agents of color d to activity a_d for $d \in [1, l]$.

Our algorithm will find an EF assignment compatible with c or determine that no such EF assignment exists. Note that any EF assignment of size k has at least one compatible coloring, and the probability that a random coloring we choose is compatible with some fixed EF assignment of size k is at least $(1/l)^k$ (as we must color those k agents correctly) which is exponentially small only in k .

FPT algorithm. Our algorithm first partitions agents of each color into several subsets, and check several necessary conditions for the coloring to be compatible with at least one EF assignment; if any of the conditions is not met, the coloring will be rejected by the algorithm (as there is no EF assignment compatible with the given coloring). For each color $d \in [1, l]$, the algorithm computes three subsets: $N_d = \{i \in N : c(i) = d\}$, $N_d^{\text{IR}} = \{i \in N_d : (a_d, k_d) \in S_i\}$, and $N_d^{\text{IN}} = \{i \in N_d : \exists d' \in [1, l] \text{ s.t. } (a_{d'}, k_{d'}) \in S_i\}$. If $|N_d^{\text{IR}}| < k_d$ for any $d \in [1, l]$, no EF assignment is compatible with c because assigning k_d agents to a_d would not be individually rational (i.e., not enough agents approve the outcome), so the coloring should be rejected in this case. If for some $d \in [1, l]$ the set $N_d^{\text{IN}} - N_d^{\text{IR}}$ is not empty but contains some agent i , then no EF assignment is compatible with c because an EF assignment cannot assign i to a_d (because $i \notin N_d^{\text{IR}}$) but i would wish to join $a_{d'}$ for some $d' \in [1, l]$ which would make the assignment not envy-free; hence the coloring should be rejected in this case. If $|N_d^{\text{IN}}| > k_d$, then at least one agent i in N_d^{IN} should be assigned to the void activity, but i would wish

to join $a_{d'}$ for some $d' \in [1, l]$ which would make the assignment not envy-free; hence the coloring should be rejected. If the coloring is not rejected by any of the cases mentioned earlier, then we have the following three conditions for every color $d \in [1, l]$: (a) $|N_d^{\text{IR}}| \geq k_d$, (b) $|N_d^{\text{IN}}| \leq k_d$, and (c) $N_d^{\text{IR}} \subseteq N_d^{\text{IN}}$. Let us define X_d for each $d \in [1, l]$ as follows: X_d contains an arbitrary set of k_d agents from N_d^{IR} such that every agent in N_d^{IN} is contained in X_d . Note that this is always possible due to the three conditions mentioned above. We claim that an assignment π which assigns agents in X_d to a_d and all other agents to a_\emptyset is an EF assignment compatible with c . To prove compatibility, all agents in X_d are by definition of color d and $|X_d| = k_d$ for all $d \in [1, l]$. To prove stability, first consider any agent i who is assigned to the void activity and suppose $d = c(i)$. Since $i \notin X_d$, we know that $i \notin N_d^{\text{IN}}$ by definition, and therefore there is no $d' \in [1, l]$ such that $(a_{d'}, k_{d'}) \in S_i$. Now consider any agent i who is assigned to a_d by π (thus $c(i) = d$). By definition $i \in X_d$ and thus $i \in N_d^{\text{IR}}$, which implies that $(a_d, k_d) \in S_i$. Therefore π is an EF assignment of size k , compatible with c .

Let us now prove that if there is at least one EF assignment that is compatible with c , then the algorithm does not reject the coloring. Let π be one such assignment and let X_d be the set of agents assigned to a_d by π . Due to compatibility we have $|X_d| = k_d$ and $c(i) = d$ for all $i \in X_d$ for all $d \in [1, l]$; in particular, by definition $X_d \subseteq N_d^{\text{IR}}$ and thus the first condition (a) above holds for all $d \in [1, l]$. Due to stability of π , every agent i with $\pi(i) = a_\emptyset$ satisfies that $\nexists d' \in [1, l]$ such that $(a_{d'}, k_{d'}) \in S_i$. Therefore the conditions (b) and (c) above hold for all $d \in [1, l]$, which proves that the coloring c would not be rejected by the algorithm.

We have shown that if we begin with a coloring c (together with l and k_d 's) that is compatible with at least one EF assignment, then our algorithm would find an EF assignment compatible with the coloring and that if no such assignment exists the coloring would be rejected. This is a Monte Carlo algorithm with probability of success at least $(1/k)^k$ and runtime bounded by $O((k^k)nk)$ (as our algorithm must enumerate all possible values of l and k_d 's), which is polynomial in n but exponential only in k . \square

4.2.4 k -Perfect-GASP: Finding Perfect Assignments

Recall that k -Perfect-GASP is the problem of finding a perfect assignment that uses k activities out of p activities. Under this parameterization, GASP is $W[2]$ -hard as the following theorem shows.

Theorem 16. *k -Perfect-GASP is $W[2]$ -hard. The problem remains to be $W[2]$ -hard even if all agents have increasing preferences or all agents have decreasing preferences.*

Proof. Consider an instance of the parameterized Dominating Set problem which consists of a graph

$G = (V, E)$ and a parameter k , and asks whether there exists a dominating set D of size k ($D \subseteq V$ is a dominating set if for every node $v \in V$ either $v \in D$ or v has a neighbor in D). This problem is known to be $W[2]$ -complete.

Let us create an instance of GASP as follows: Let $N = \{1, 2, \dots, n\}$ and $A^* = \{a_1, a_2, \dots, a_n\}$ where $n = |V|$. For each agent i , define $S_i = \{(a_j, x) : ((v_i, v_j) \in E) \wedge (1 \leq x \leq n)\} \cup \{(a_i, x) : 1 \leq x \leq n\}$. Note that in this instance agents do not care about the number of participants, but the activities only. Finally we set the parameter of GASP to be equal to the parameter of the Dominating Set instance. Note that the size of the GASP instance we create is polynomial in n and k .

Let D be a dominating set of size k in the original instance. Let us construct a perfect assignment π as follows: For each agent i , if $v_i \in D$, then let $\pi(i) = a_i$; if $v_i \notin D$, then there exists some $v_j \in D$ such that $(v_i, v_j) \in E$ because D is a dominating set, and let $\pi(i) = a_j$. Clearly π is a perfect assignment that uses only k activities. Conversely, suppose that a perfect assignment π exists which uses exactly k activities. Let A' be the set of k activities to which at least one agent is assigned under π (note that $|A'| = k$). Let $D = \{v_i : a_i \in A'\}$, and we claim that D is a dominating set in G . For any node $v_i \notin D$, we know that π assigns agent i to some activity $a_j \in A'$ where $a_j \neq a_i$, and thus $v_j \in D$. Since π is individually rational, it implies that $(v_i, v_j) \in E$, and therefore D is a dominating set.

Note that in our reduction all agents have increasing preferences as well as decreasing preferences, proving the second statement of the theorem. \square

It is not surprising that k -Perfect-GASP is the most complex problem being considered, as it pertains to the strongest solution concept. We do not know the exact complexity of k -Perfect-GASP when all activities are equivalent, and it remains to be an open problem (yet the problem is known to be NP-complete by Darmann et al. [14]).

4.3 Discussion and Future Work

In this work we investigated the parameterized complexity of the Group Activity Selection Problem (GASP) when parameterized by the size of the solution, for four different solution concepts and under various restrictions on inputs. Despite the fact that all problems being considered are NP-hard, we showed that some special cases of the problem admit efficient FPT algorithms. Our results indicate that the computational complexity of GASP varies when its input is restricted (imposed by special preferences of agents or uniformity of activities) or the solution concept changes, which is not distinguishable under the classic complexity. Our work leaves a few interesting open problems

for future work. First, we do not know the exact complexity of k -Perfect-GASP with equivalent activities besides its NP-completeness. It would be intriguing if the problem is FPT. Second, the focus in this chapter is to exhibit any FPT algorithm; we have not tried hard to optimize the dependence on k . It would be interesting to do this and to show conditional lower bounds on how the runtime should depend on k , especially in the case of k -IR-GASP. Lastly, one can consider a different setting where agents have a strict ordering over the set of outcomes, instead of having approved outcomes that are equally preferred. Darmann recently proved several easiness and hardness results under this setting [13], but no parameterized complexity results are known yet.

Chapter 5

Complexity of Stable Invitations Problem

In Chapter 4 we considered the setting where an event organizer is to assign agents to multiple activities that take place concurrently. In this chapter, we consider a slightly different setting in which there is only one activity – and thus the organizer is to choose a subset of agents to be “invited” to participate in the activity, while agents may exhibit friend or enemy relationships. On the one hand, this is a special case of the Group Activity Selection Problem in that there is only one activity. On the other hand, it is a generalization of this special case in that agents now have non-anonymous preferences – they do not only care about the number of participants in the activity, but they also care about who those participants are.

Let us consider a concrete example to understand the setting. Suppose that an event organizer trying to convene an event – for example, a fundraiser. We assume that the time and venue for the event are fixed, and that the only remaining decision for the organizer to make is whom to invite among a set of agents. An *invitation* is simply defined to be a subset of agents. The goal of the organizer is to maximize attendance (for example, in order to maximize donations), but the potential invitees have their own preferences over how many attendees there should be at the event and possibly also who the potential attendees should be. For example, a given donor may not want to attend if too few attendees show up, but she may not want the event to be overly crowded. Another donor may want to attend the event only if her friends attend and her business competitor does not.

Clearly, this can be viewed as an instance of the Group Activity Selection Problem (GASP) with the exception that agents have non-anonymous preferences. We call this the Stable Invitations Problem (SIP), in which each agent can specify a set of friends and a set of enemies (in addition to

her preference over sizes as in GASP). An agent is willing to attend only if all of her friends attend, none of her enemies attends, and the number of attendees is acceptable to her.

Not surprisingly, complexity of SIP depends highly on the cardinality of friend-sets and enemy-sets, as friends-and-enemies relationship introduces combinatoric complexity in the problem of finding a good solution. In this chapter we provide a complete analysis of complexity results on SIP; we consider individual rationality (IR) and Nash stability as we did for GASP, and we also consider both asymmetric and symmetric friends-and-enemies relation.

5.1 Definitions and Notation

Let us first define the Stable Invitations Problem and related solution concepts.

Definition 13. An instance of the Stable Invitations Problem (SIP) is given by a set of agents $N = \{a_1, a_2, \dots, a_n\}$, and an *approval set* $S_i \subseteq [1, n]$, a *friend set* $F_i \subseteq N$, and an *enemy set* $E_i \subseteq N$ for each agent $a_i \in N$. We interpret that agent a_i is willing to attend if all friends in F_i attend, no one in E_i attends, and the number of attendees (including a_i) is acceptable (i.e., it is contained in S_i).

Definition 14. An *invitation* I in SIP is a subset of agents.

We say that an invitation I is *individually rational* (IR) if for every agent $a_i \in I$, $|I| \in S_i$, $F_i \subseteq I$, and $R_i \cap I = \emptyset$.

We say that an invitation I is (*Nash*) *stable* if it is individually rational, and if for every agent $a_j \notin I$, $|I_j| \notin S_j$, $F_j \not\subseteq I_j$, or $R_j \cap I_j \neq \emptyset$ where $I_j = I \cup \{a_j\}$.

Individual rationality (IR) requires every invited agent be willing to attend. Stability further requires those who are not invited be unwilling to participate (without permission of others) because not all of her friends are attending, some of her enemies are attending, or the number of attendees would be unacceptable. We consider the following two problems of finding invitations of size k :

- k -IR-Invitation: \exists IR invitation of size k ?
- k -Stable-Invitation: \exists stable invitation of size k ?

As we shall see, both problems are NP-hard even under some restrictions on inputs of the problems. Yet, as we mentioned in Chapter 4, NP-hardness is not the end of hardness, and we focus on parameterized complexity of these problems in this chapter.

There are two kinds of restrictions on inputs we will consider in this work. First, we consider inputs in which the sizes of largest friend sets and enemy sets is bounded above by constants;

intuitively, the smaller the constants are, the easier the problems become. Second, we consider the cases where friend relationship and enemy relationship are symmetric (i.e., A is a friend of B if and only if B is a friend of A and X is an enemy of Y if and only if Y is an enemy of X).

We first consider restrictions on inputs by limiting the size of largest friend-sets and enemy-sets, respectively. For integer constants α and β , if an instance of SIP satisfies $|F_i| \leq \alpha$ and $|E_i| \leq \beta$ for all $a_i \in N$, we call it an (α, β) -instance of SIP. We show that that SIP can be solved in polytime only if α and β are small enough, but the problems are NP-hard in general. We then investigate parameterized complexity of the problem under the same restrictions when the size of a solution is parameterized. Lastly, we will consider a special case of SIP where agents have symmetric social relationships.

Definition 15. Given an instance of SIP, we say that agents have *symmetric social relationships* if $a_j \in F_i$ if and only if $a_i \in F_j$ and $a_l \in E_i$ if and only if $a_i \in E_l$ for every a_i .

Theorem 17 summarizes the most relevant results of Darmann et al. [14] and Lee and Shoham [31] on complexity of SIP.¹

Theorem 17. *[[31, 14]] k -IR-Invitation and k -Stable-Invitation can be solved in polynomial time if $(\max_{a_i \in N} |F_i|) + (\max_{a_i \in N} |E_i|) \leq 1$ (i.e., $\alpha + \beta \leq 1$). In other cases, both problems are NP-hard.*

Note that k -IR-Invitation and k -Stable-Invitation are of the same classical complexity, even though stability is a stronger solution concept. Under parameterization, however, these two problems are contained in different complexity classes in the W-hierarchy (see Table 5.2). In what follows, we show that the parameterized complexity of these problems varies with different solution concepts and under different restrictions on inputs to SIP.

5.2 Classical Complexity

5.3 Parameterized Complexity

In this section, we study parameterized complexity of k -IR-Invitation and k -Stable-Invitation. Our main contributions are summarized in Table 5.2. For instance, finding an IR invitation of size k is in FPT when $\alpha = 1$ and β is a positive constant (bounded above by some function of k), but finding a stable invitation in the same cases is W[1]-complete.

¹Darmann et al. [14] showed easiness when $\alpha = \beta = 0$, while Lee and Shoham [31] proved easiness and hardness in all other cases.

	k -IR-Invitations			k -Stable-Invitations		
	$\beta = 0$	$\beta = 1$	$2 \leq \beta$	$\beta = 0$	$\beta = 1$	$2 \leq \beta$
$\alpha = 0$	P	P	NP-C	P	P	NP-C
$\alpha = 1$	P	NP-C	NP-C	P	NP-C	NP-C
$\alpha \geq 2$	NP-C	NP-C	NP-C	NP-C	NP-C	NP-C

Table 5.1: Complexity of k -IR-Invitation and k -Stable-Invitation.

	k -IR-Invitations				k -Stable-Invitations			
	$\beta = 0$	$\beta = 1$	$2 \leq \beta \leq f(k)$	unbounded β	$\beta = 0$	$\beta = 1$	$2 \leq \beta \leq f(k)$	unbounded β
$\alpha = 0$	P	P	FPT	W[1]-C	P	P	FPT	W[2]-C
$\alpha = 1$	P	FPT	FPT	W[1]-C	P	W[1]-C	W[1]-C	W[2]-C
$\alpha \geq 2$	W[1]-C	W[1]-C	W[1]-C	W[1]-C	W[1]-C	W[1]-C	W[1]-C	W[2]-C

Table 5.2: Complexity of k -IR-Invitation and k -Stable-Invitation. $f(k)$ can be an arbitrary function of k that only depends on k . All entries other than “P” imply NP-completeness. “W[1]-C” and “W[2]-C” mean W[1]-completeness and W[2]-completeness, respectively. Note that P and NP-completeness results were presented in Section xxx.

5.3.1 k -IR-Invitation

Recall that k -IR-Invitation is the problem of finding an IR invitation of size k . When $\alpha + \beta > 1$, the problem is known to be NP-hard (Theorem 17). We first present easiness results: k -IR-Invitation is in W[1] in general, and it is in FPT if $\alpha \leq 1$ and β is bounded by some function $f(k)$ of k . We then present hardness results by showing that k -IR-Invitation is W[1]-hard when $\alpha \geq 2$ and/or β is unbounded.

Theorem 18. k -IR-Invitation is in W[1].

Proof. We reduce k -IR-Invitation to the weighted circuit SAT (WCSAT) of constant depth and of weight at most 1.

Details of the proof can be found in proof of Theorem 22 which appears later in this chapter. \square

Theorem 19. k -IR-Invitation is in FPT if $\alpha \leq 1$ and $\beta \leq f(k)$ where $f(k)$ can be an arbitrary function of k .

Proof. Without loss of generality, assume that $k \in S_i$ for all $a_i \in N$. Otherwise, we can remove a_i from the input instance as no IR invitation of size k can contain a_i . If a_i is removed, and there is some a_j with $a_i \in F_j$, we remove a_j as well for the same reason. We repeat this removal process until no such agent remains (this can be done in linear time).

Let \mathcal{A} be some polytime algorithm that solves k -IR-Invitation if $\alpha \leq 1$ and $\beta = 0$ (it exists due to Theorem 17). We will use \mathcal{A} as a sub-routine in our FPT algorithm. Consider any coloring c

which colors agents using two colors $\{0, 1\}$; let $c(i) \in \{0, 1\}$ be the color of agent a_i . We say that coloring c and IR invitation I of size k are *compatible* if the following holds: For every agent $a_i \in I$, $c(i) = 1$ and for every agent $a_j \in \cup_{i \in I} E_i$, $c(j) = 0$. Note that coloring c may be compatible with any number of IR invitations of size k (possibly none), and any IR invitation of size k may be compatible with many colorings (but it is compatible with at least one coloring).

Given some arbitrary coloring c , we can find an IR invitation of size k that is compatible with c or determine that no compatible IR invitation exists in FPT time as follows. First, we re-color every agent a_i with $c(i) = 1$ to color 0 such that $\exists a_j \in F_i$ with $c(j) = 0$ or $\exists a_l \in E_i$ with $c(l) = 1$ (order in which we re-color agents does not matter). Notice that this process does not re-color any agent $a_i \in I$ if I is compatible with c . After the re-coloring step, let $N_1 = \{a_i \in N : c(i) = 1\}$, and we run the algorithm \mathcal{A} on N_1 as input. Suppose that \mathcal{A} finds an IR invitation I of size k given N_1 . I is individually rational because its friend constraints are satisfied (due to correctness of \mathcal{A}) and its enemy constraints are satisfied because no agent with color 0 is included in N_1 (enforced by coloring). Now suppose that \mathcal{A} reports that no IR invitation I of size k exists among the agents in N_1 . Then there is no IR invitation of size k that is compatible with c ; if such invitation $I' \subseteq N_1$ exists, then I' satisfies the friend constraints (because it is IR) and therefore \mathcal{A} should find it, which is a contradiction. Hence if our algorithm begins with coloring that is compatible with some IR invitation(s), it will find one.

If we color agents uniformly and independently at random, then the probability of success of our algorithm is at least $1/2^{(k+1)\beta}$ (because, with respect to some fixed IR invitation I^* , we must color all agents in I^* as 1 and the union enemies of agents in I^* as 0, to start with compatible coloring). If we run this algorithm $2^{(k+1)\beta} \ln n$ times, the probability of success is at least $1 - 1/n$. Our FPT algorithm's runtime depends on the runtime of \mathcal{A} . The algorithm can be de-randomized using a family of k -perfect hash functions as shown in the work by Alon et al. [2]. \square

Theorem 20. *k -IR-Invitation is $W[1]$ -complete if β is not bounded above by any function $f(k)$.*

Proof. We reduce from the k -Independent-Set problem which is known to be $W[1]$ -complete. Given an arbitrary graph $G = (V, E)$ and a parameter k , we create agents $N = V = \{v_1, v_2, \dots, v_n\}$. For each v_i , define $S_{v_i} = \{k\}$, $F_{v_i} = \emptyset$, and $E_{v_i} = \{v_j : (v_i, v_j) \in E\}$ (hence β is equal to the max-degree of nodes in G). If $I \subset V$ is an independent set of size k , then I is an IR invitation in the instance we created: For all $v_i \in I$, we have $|I| = k \in S_{v_i}$, $F_{v_i} = \emptyset \subset I$, and $E_{v_i} \cap I = \emptyset$ because I is an independent set in the original graph. Conversely, suppose I is an IR invitation of size k in the instance we created. Then I is an independent set because no two agents in I are enemies of each other, and

thus their corresponding nodes in the graph are not neighbors of each others. This reduction proves W[1]-hardness, and W[1]-completeness follows from Theorem 18. \square

Theorem 21. *k -IR-Invitation is W[1]-complete if $\alpha \geq 2$.*

Proof. We reduce from the k -Clique problem. Given an arbitrary graph $G = (V, E)$ and a parameter k , we create a set of agents N as follows. For each node $v_i \in V$, we create k^2 node-agents that are labeled as $w_{i,x}$ where $x \in [k^2]$. For each node-agent $w_{i,x}$ we define $F_{w_{i,x}} = \{w_{i,x+1}\}$ (where w_{i,k^2+1} is understood as $w_{i,1}$) and $E_{w_{i,x}} = \emptyset$. Note that an IR invitation must include all or none of the $w_{i,x}$'s for each i because of their friend sets. Next, for each edge $(v_i, v_j) \in E$, we create an edge-agent $e_{i,j}$ with $F_{e_{i,j}} = \{w_{i,1}, w_{j,1}\}$ and $E_{e_{i,j}} = \emptyset$. Note that if an IR invitation includes $e_{i,j}$, then it must also include all $2k^2$ node-agents of the form $w_{i,x}$ and $w_{j,x}$ with $x \in [k^2]$ (due to friend sets).

Finally, define $k' = k^3 + \binom{k}{2}$ to be the parameter for the k -IR-Invitations we created, and define approval sets of all agents to contain k' . Clearly the instance we created satisfies $\alpha = 2$ and $\beta = 0$. The number of agents we created is $k^2|V| + |E|$, polynomial in the size of the original instance. It remains to show that a clique of size k exists if and only if an IR invitation of size $k' = k^3 + \binom{k}{2}$ exists.

If C is a clique of size k in the original graph, define $I = \{w_{i,x} : v_i \in C, x \in [k^2]\} \cup \{e_{i,j} : v_i, v_j \in C\}$. Clearly, $|I| = k^3 + \binom{k}{2}$ because $|C| = k$. To see that I is individually rational, every agent in I approves size $k^3 + \binom{k}{2}$, every agent included in I has their friends included in I .

Conversely, suppose that I is an IR invitation of size $k^3 + \binom{k}{2}$ in the instance we created. Let us define $G_i = \{w_{i,x} : x \in [k^2]\}$ to be the set of node-agents corresponding to v_i in the original instance (informally, G_i is a cycle of length k^2 representing node v_i). Recall that I must contain all node-agents or none in G_i for each i if I is individually rational. If I contains more than k such sets, then $|I| \geq (k+1)k^2 > k^3 + \binom{k}{2}$, and therefore I can contain at most k such sets, which means that I must contain at least $\binom{k}{2}$ edge-agents (because $|I| = k^3 + \binom{k}{2}$). Now suppose that I contains more than $\binom{k}{2}$ edge-agents. Then I must include at least $k+1$ of G_i 's because each edge-agent $e_{i,j}$ in I requires both G_i and G_j to be included in I , and containing more than $\binom{k}{2}$ edge-agents in I implies that at least $k+1$ sets of node-agents are included in I which we argued leads to a contradiction. That is, I can contain at most $\binom{k}{2}$ edge-agents. Together with the previous argument, this means that I contains exactly $\binom{k}{2}$ edge-agents and exactly k sets of node-agents. Because $e_{i,j} \in I$ implies that there is an edge (v_i, v_j) in the original instance, this implies that a clique of size k exists in the original instance.

Note that W[1]-completeness follows from Theorem 18. \square

5.3.2 k -Stable-Invitation

k -IR-Invitation and k -Stable-Invitation have the same classical complexity for all values of α and β , but parameterization indicates that k -Stable-Invitation is a more difficult problem than k -IR-Invitation. This is not surprising because a stable invitation requires that everyone (whether invited or not) be satisfied with the invitation.

Theorem 22. *k -Stable-Invitation is in $W[2]$. When β is bounded above by some function $f(k)$, it is in $W[1]$.*

Proof. Assume unbounded α and β , and let us reduce the k -Stable-Invitation problem to the weighted circuit SAT (WCSAT) of constant depth and of weft at most 2. This shows that the problem is in $W[2]$; along the way, we prove the other statements as well.

Given an instance of the k -Stable-Invitation problem with n agents ($N = \{a_1, a_2, \dots, a_n\}$), we create n input nodes which represent each agent being invited or not; let x_i be the input node for agent a_i ($x_i = 1$ means that agent a_i is included in an invitation). For convenience we also create a node y_i for each x_i where y_i is the NOT gate attached to x_i . We create the output AND gate denoted by x_o .

For each agent a_i , if $k \notin S_i$ and/or $|F_i| \geq k$, then agent i cannot be included in a stable invitation of size k ; for all such a_i , we create a path from the node x_i to y_i (recall that y_i simply negates x_i) to the output node x_o . For each agent a_i with $k \in S_i$ and $|F_i| < k$, we wish to ensure that if agent a_i is included, then everyone in F_i is included and no one in R_i is included. For each agent $j \in F_i$, we create a new node $f_{i,j}$ with the OR gate whose input is y_i and x_j , and the output node x_o takes $f_{i,j}$ as input. Note that a solution to the WCSAT instance must have $f_{i,j} = 1$ for all i, j (if the node $f_{i,j}$ is created), which holds if and only if $(x_i, x_j) \in \{(0, 0), (0, 1), (1, 1)\}$. This ensures that if a_i is selected, all agents in F_i are also selected. Note that each path containing $f_{i,j}$ to the output node has a constant depth and a weft of 1 (due to the output node). Similarly, for each $j \in R_i$ with $k \in S_i$ and $|F_i| < k$, we create a new node $r_{i,j}$ with the OR gate whose input is x_i and y_j , and the output node x_o takes $r_{i,j}$ as input. So far we have implemented a circuit such that any solution to the WCSAT instance corresponds to an IR invitation of size k . Every path from any input node to the output node has constant depth and weft 1 (due to the output node), and this shows that k -IR-Invitation is in $W[1]$.

To ensure stability in addition to individual rationality, we need to add more gates and paths. For each agent a_i with $k+1 \in S_i$, if agent a_i is not included in an invitation, then the invitation is stable only if at least one agent in F_i is not included and/or at least one agent in R_i is included. Note

that if $|F_i| > k$, then every solution to the WCSAT leaves out at least one agent from F_i , and therefore agent a_i (even if excluded) would not be willing to join the invitation. Therefore, we only need to ensure stability for those agents with $k+1 \in S_i$ and $|F_i| \leq k$. For each such agent a_i , let us create a new node s_i with the OR gate whose input consists of all nodes y_j where $j \in F_i$, all nodes x_j where $j \in R_i$, and x_i itself. Note that $s_i = 1$ if $x_i = 1$ (when agent a_i is selected) or if $x_i = 0$ and at least one friend (enemy) of agent a_i is not included (included). Notice that any path containing s_i has a constant depth but a weft of size 2 due to s_i and the output node. This shows that k -Stable-Invitation is in $W[2]$.

When β is bounded by $f(k)$ (any constant not depending on n), we can create a chain of nodes to implement s_i . Specifically, instead of having s_i to take $(|F_i| + |R_i| + 1)$ inputs, we create a chain of nodes with the OR gate, each of which takes only two inputs. Although this increases the depth of a path containing these nodes to $(|F_i| + |R_i| + 1)$, the depth is bounded by $(k + f(k) + 1)$ because $|F_i| \leq k$. Therefore, when β is bounded, the k -Stable-Invitation is in $W[1]$. \square

Theorem 23. *k -Stable-Invitation is in FPT when $\alpha = 0$ and $\beta \leq f(k)$ where $f(k)$ can be an arbitrary function of k .*

Proof. The main idea is similar to that of our proof of Theorem 19.

Consider any coloring c which colors agents using two colors $\{0, 1\}$; let $c(i) \in \{0, 1\}$ be the color of agent a_i . Let I be a stable invitation of size k . We say that c and I are *compatible* if the following holds: For every agent $a_i \in I$, $c(i) = 1$ and for every agent $a_j \in \cup_{i: a_i \in I} R_i$, $c(j) = 0$; other agents can be of any color. Given an arbitrary coloring c , we can find a stable invitation of size k that is compatible with c or determine that no compatible invitation exists in FPT time.

Given c , we first re-color some of the agents if necessary as follows. If there is some agent a_i with $c(i) = 1$ and $k+1 \notin S_i$, then we re-color agent i as $c(i) = 0$ as no stable invitation can include i . If there is some agent i with $c(i) = 1$ and $\exists j \in R_i$ such that $c(j) = 1$, then we re-color agent i as $c(i) = 0$ as no compatible invitation I can include i in it. We repeat these re-coloring steps until no such agents exist (which happens at most $O(|N|)$ times).

After all re-coloring steps are completed, we partition the set of agents into four subsets as follows. Define $X_d = \{a_i \in N : c(i) = d \wedge k+1 \in S_i\}$ and $Y_d = \{a_i \in N : c(i) = d \wedge k+1 \notin S_i\}$ where $d \in \{1, 2\}$. For any I compatible with c , it must hold that $X_1 \subseteq I$ (because I is stable) and that $I \subseteq X_1 \cup Y_1$ (because c and I are compatible). Therefore if $|X_1| > k$ or $|X_1| + |Y_1| < k$, then we conclude that no stable invitation of size k is compatible with c and terminate. Because $X_1 \subseteq I$ for all $I \in \mathcal{I}_c$, let us assume that $X_1 = \emptyset$ without loss of generality.

Then, $I \subseteq Y_1$ for every stable invitation I that is compatible with c . We cannot do a naive exhaustive search on Y_1 to find compatible I because the search space can be as large as $O(n^k)$. Yet we can reduce the search space to $O(\beta^k)$. Let I' be any subset of Y_1 of size k ; I' is individually rational due to the proper coloring constraint. Therefore its stability only depends on the existence of some agent $a_i \in X_0$ such that $R_i \cap I' = \emptyset$. Let $J = \emptyset$, and let us construct an invitation of size k . If there is some agent $a_i \in X_0$ such that $R_i \cap J = \emptyset$, then we try adding one agent from $R_i \cap Y_1$ to J , thereby increasing the size of J by one. We repeat this process in an exhaustive manner until J contains k agents or there is no such agent $i \in X_0$ such that $R_i \cap J = \emptyset$. If the former occurs, we check whether J is a stable invitation of size k in poly-time; if J is stable, then we output J , otherwise we continue exhaustive search. If the latter occurs, we can choose an arbitrary invitation I with $J \subseteq I \subseteq Y_1$ because any such I is guaranteed to be stable. Note that the search space is $O(\beta^k)$ because we choose up to k agents (as $|J| \leq k$ at all times) and at each step we have at most β agents to choose from (i.e., $|R_i \cap J| \leq |R_i| \leq \beta$).

Therefore, this algorithm finds a stable invitation of size k if it is given a proper coloring of any stable invitation of size k in time $O(\beta^k n^O(1))$. We can de-randomize this algorithm by de-randomizing the coloring step (for instance, one can use k -perfect family of hash functions as in the work by [2]). \square

Theorem 24. *k -Stable-Invitation is $W[1]$ -complete if $\alpha, \beta \geq 1$ and β is bounded above by some function $f(k)$.*

Proof. We reduce from the k -Clique problem to show $W[1]$ -hardness. Let $G = (V, E)$ be an arbitrary graph for the k -Clique problem with parameter k . Let us define $k' = 2(k^3 + \binom{k}{2})$ which is the parameter for k -Stable-Invitation. For each node $v_i \in V$, we first create a group of $2k^2$ node-agents (call them G_i) such that $G_i = \{w_{i,x} : x \in [2k^2]\}$, and define $F_{w_{i,x}} = \{w_{i,x+1}\}$ (where $w_{i,2k^2+1}$ is understood as $w_{i,1}$) and $S_{w_{i,x}} = \{k'\}$. For each edge $(v_i, v_j) \in E$, we create four edge-agents $e_{i,j}, e'_{i,j}, f_{i,j}$, and $f'_{i,j}$. Define $F_{e_{i,j}} = \{w_{i,1}\}$, $F_{e'_{i,j}} = \{w_{j,1}\}$, and $S_{e_{i,j}} = S_{e'_{i,j}} = \{k'\}$. Define $F_{f_{i,j}} = \{e_{i,j}\}$, $F_{f'_{i,j}} = \{e'_{i,j}\}$, and $S_{f_{i,j}} = \{k' + 1\}$. Define $F_{f'_{i,j}} = \{e'_{i,j}\}$, $E_{f'_{i,j}} = \{e_{i,j}\}$, and $S_{f'_{i,j}} = \{k' + 1\}$. We have created $2k^2n$ node-agents and $4|E|$ edge-agents, whose size is polynomial in n, k , and each agent we created has at most one friend and at most one enemy (thereby satisfying $\alpha = \beta = 1$).

Suppose C is a clique of size k . Without loss of generality, suppose $C = \{v_1, v_2, \dots, v_k\}$, and define $I = (\cup_{i=1}^k G_i) \cup \{e_{i,j}, e'_{i,j} : 1 \leq i < j \leq k\}$. Note that $|I| = 2k^3 + 2\binom{k}{2} = k'$, and we claim that I is a stable invitation. It is clear that I is individually rational, as each agent in I approves the size k' and her friends are all included in I . To see why I is stable, first notice that any node-agent $v_j \notin I$

and any edge-agents $e_{i,j}, e'_{i,j} \notin I$ does not approve the size $k+1$. Now consider any edge-agent $f_{i,j}$. Since I contains both or neither of $e_{i,j}$ and $e'_{i,j}$, $f_{i,j}$ does not wish to join I because $f_{i,j}$ has one of them in friend-set and the other in enemy-set. Similarly, no edge-agent $f'_{i,j}$ wishes to join I . Therefore, I is a stable invitation of size k' .

Conversely, suppose that I is a stable invitation of size k' . Because I must contain all or none of G_i for each i , I can contain at most k such groups of node-agents (otherwise, if it contains more than k such groups, the size of the invitation would exceed k'). Because I is stable, it cannot contain any of the edge-agents $f_{i,j}$ or $f'_{i,j}$ as they do not approve the size k' . Finally, for every pair of edge-agents $e_{i,j}$ and $e'_{i,j}$, I must contain both or neither of them; if only one of them is included in I , then either $f_{i,j}$ or $f'_{i,j}$ would wish to join I which contradicts the stability of I . Therefore, because we earlier argued that I can contain at most k groups of node-agents, we conclude that I must contain at least $2\binom{k}{2}$ edge-agents of the form $e_{i,j}$ and $e'_{i,j}$. Furthermore, because $e_{i,j}, e'_{i,j} \in I$ implies that $G_i \subset I$ and $G_j \subset I$, if I contains more than $2\binom{k}{2}$ edge-agents, it must also contain at least $k+1$ groups of node-agents (which we argued earlier leads to a contradiction). In summary, if I is a stable invitation of size k' , it must contain exactly k groups of node-agents (without loss of generality, assume that I contains G_1, G_2, \dots, G_k) and exactly $2\binom{k}{2}$ edge-agents that are precisely $\{e_{i,j}, e'_{i,j} : 1 \leq i < j \leq k\}$. Then, $C = \{v_1, v_2, \dots, v_k\}$ is a clique in the original instance because there exists an edge between v_i and v_j for every pair $v_i, v_j \in C$ (the fact that edge-agents $e_{i,j}$ exist implies this).

This shows W[1]-hardness, and W[1]-completeness follows from Theorem 22. □

Theorem 25. *k -Stable-Invitation is W[1]-complete if $\alpha \geq 2$ and β is bounded above by some function $f(k)$.*

Proof. We reduce from the k -Independent-Set problem to show W[1]-hardness. Let $G = (V, E)$ be an arbitrary instance of the k -Independent-Set problem with parameter k . For each node $v \in V$ we create a node-agent v with approval set $S_v = \{k\}$ and friend set $F_v = \emptyset$. For each edge $(v, w) \in E$ we create an edge-agent $e_{v,w}$ with friend set $F_{e_{v,w}} = \{v, w\}$ and approval set $S_{e_{v,w}} = \{k+1\}$.

Suppose $S \subseteq V$ is an independent set of size k and let $I = S$. I is individually rational as for every node-agent $v \in I$, $|I| = k \in S_v$ and $F_v \subseteq I$. To see why I is stable, for every node-agent $w \notin I$, $k+1 \notin S_w$. For every edge-agent $e_{v,w}$, we know that $v \notin I$ or $w \notin I$ because if $v, w \in I$ then $v, w \in S$ and $(v, w) \in E$, which contradicts the fact that S is an independent set (note that the edge-agent $e_{v,w}$ is created only if $(v, w) \in E$). Therefore I is a stable invitation of size k .

Conversely, suppose that I is a stable invitation of size k . Then I cannot contain any edge-agent because every edge-agent only approves the size $k + 1$. Let $S = I$ and we claim that S is an independent set. If there is a pair of nodes $v, w \in S$ with $(v, w) \in E$, then there must exist an edge-agent $e_{v,w}$. However, $F_{e_{v,w}} = \{v, w\}$ and $e_{v,w} \notin I$, which implies that I is not stable. This is a contradiction, and therefore S is an independent set of size k .

This shows W[1]-hardness, and W[1]-completeness follows from Theorem 22. \square

Theorem 26. *k -Stable-Invitation is W[2]-complete if β is not bounded above by any function of k .*

Proof. We reduce from the k -Dominating-Set problem which is known to be W[2]-hard. Given an arbitrary graph $G = (V, E)$ and a parameter k , we create $2n$ node-agents by creating x_i and y_i for each $v_i \in V$. We define their approval sets and enemy sets as follows: $S_{x_i} = \{k\}$ and $E_{x_i} = \emptyset$ for all x_i while $S_{y_i} = \{k + 1\}$ and $E_{y_i} = \{x_i\} \cup \{x_j : (v_i, v_j) \in E\}$ for all y_i . Note that a stable invitation cannot contain any of y_i 's because of their approval sets.

Let D be a dominating set of size k in the original instance, and let $I = \{x_i : v_i \in D\}$. We claim that I is a stable invitation of size k . For every node-agent $x_i \in I$, $|I| = k \in S_{x_i}$ and $E_{x_i} \cap I = \emptyset$, and thus I is individually rational. For every node-agent $x_j \notin I$, $k + 1 \notin S_{x_j}$, and thus x_j has no incentive to join I . For every node-agent y_i , either $x_i \in I$ or there is some $x_j \in I$ such that $x_j \in E_{y_i}$ because D is a dominating set; that is, if $v_i \notin D$, then there exists some $v_j \in D$ where $(v_i, v_j) \in E$, which implies that $\exists x_j \in E_{y_i} \cap I$ if $y_i \notin I$.

Conversely, suppose that there exists a stable invitation I of size k . Clearly I can only contain node-agents of the form x_i 's (due to approval sets of y_i 's), and we claim that $D = \{v_i : x_i \in I\}$ is a dominating set. Pick any node-agent $x_i \notin I$. Since I is stable and $x_i \notin I$, we know that there exists some node-agent $x_j \in I$ such that $x_j \in E_{y_i}$. This implies that $(v_i, v_j) \in E$, and therefore D is a dominating set.

This shows W[2]-hardness, and W[2]-completeness follows from Theorem 22. \square

5.4 Symmetric Social Relationship

Recall the definition of “symmetric social relationships” from Definition 15. Under symmetric social relationships, both k -IR-Invitation and k -Stable-Invitation admit efficient FPT algorithms when β is bounded, as shown in Table 5.3, although their complexity does not change when β is unbounded (W[1]-complete and W[2]-complete, respectively).

	k -IR-Invitations (symmetric social relationships)					k -Stable-Invitations (symmetric social relationships)				
	$\beta = 0$	$\beta = 1$	$\beta = 2$	$3 \leq \beta \leq f(k)$	unbounded β	$\beta = 0$	$\beta = 1$	$\beta = 2$	$3 \leq \beta \leq f(k)$	unbounded
$\alpha = 0$	P	P	P	FPT	W[1]-C	P	P	P	FPT	W[2]-C
$\alpha = 1$	P	P	FPT	FPT	W[1]-C	P	P	FPT	FPT	W[2]-C
$\alpha \geq 2$	P	FPT	FPT	FPT	W[1]-C	P	FPT	FPT	FPT	W[2]-C

Table 5.3: Complexity of SIP with symmetric social relationships. $f(k)$ can be an arbitrary function of k that only depends on k . All entries other than “P” imply NP-completeness. “W[1]-C” and “W[2]-C” mean W[1]-completeness and W[2]-completeness, respectively. All results are original (including classical complexity results).

When we compare results in Table 5.2 and Table 5.3, two interesting observations can be made. First, k -IR-Invitations and k -Stable-Invitations have the same classical complexity even under symmetric social relationships. Second, both problems now admit efficient FPT algorithms for broader domains of inputs – as long as β is bounded. Note that our classical complexity results are original (i.e., not implied by Theorem 17) because Lee and Shoham [31] did not consider the special case of symmetric social relationships.

5.4.1 Symmetric k -IR-Invitations

We first present classical complexity results for k -IR-Invitations under symmetric social relationships, followed by parameterized complexity results.

Theorem 27. *When agents have symmetric social relationships, k -IR-Invitations can be solved in polynomial time if (i) $\beta = 0$, (ii) $\beta = 1$ and $\alpha \leq 1$, or (iii) $\beta = 2$ and $\alpha = 0$. Otherwise, the problem is NP-hard.*

Proof. Let us consider case (ii) in the statement. As before, without loss of generality assume that all agents accept the size k (i.e., $k \in S_i$ for all $a_i \in N$).

We first construct an *enemy graph* in which nodes represent agents, and we create an edge between two nodes if their corresponding agents are enemies of each other. For every pair of friends, we merge their nodes in this graph into a meta-node of weight 2 (if they are also enemies of each other, then we simply remove them from the graph); let us call the resulting graph a *friend graph*. Now finding an IR invitation of size k is equivalent to finding an independent set of total weight k in the friend graph. Although finding an independent set (of an arbitrary size) is NP-hard, all nodes in the friend graph have at most two edges, and thus each connected component in the friend graph is either a path or a cycle. A dynamic programming algorithm can solve this problem. For each connected component (which is either a path or a cycle), we can check for every integer x

between 0 and k , inclusive, whether it is possible to choose exactly k nodes (agents) from the component, while satisfying individual rationality (provided that we will choose $k - x$ agents from other components, thereby making the invitation's size equal to k). This check can be done trivially: If a path contains y nodes in total, we can choose up to $\lceil y/2 \rceil$ agents from the component (which should be an independent set in the enemy graph) and if a cycle contains $z \geq 3$ nodes in total, we can choose up to $\lfloor z/2 \rfloor$ nodes. After this check for each component and each integer, we can solve the main problem using a knapsack-like algorithm to find an IR invitation of size k .

Case (i) can be solved in polytime as follows (again, assume $k \in S_i$ for every agent a_i). Because $\beta = 0$, agents only have friends. Again we construct a friendship graph in which nodes are agents and there is an edge between two nodes if they are friends of each other in SIP. Now we find all connected components in this graph; everyone in the same component must all be invited or all be uninvited. This reduces to a knapsack problem where we seek a subset of components whose total weight (the number of nodes in them) is equal to k .

Lastly, Case (iii) can be solved in polytime as follows (again, assume $k \in S_i$ for every agent a_i). Since $\alpha = 0$, no agent has friends. If we construct an enemy graph in which nodes are agents and there is an edge between two nodes if the corresponding agents are enemies, then finding an IR invitation of size k is equivalent to finding an independent set of size k in the enemy graph we constructed. Furthermore, each node in this graph has at most two edges (because $\beta = 2$), and therefore every connected component in this graph is either a path or a cycle. Again, we can use a dynamic programming algorithm (similar to the one we used for Case (i) above) to solve this problem in polytime.

Let us now prove that the problem is NP-hard if none of the three conditions in the statement holds. It is known that the Independent Set problem is NP-hard even if every node has degree at most 3 [22]. Given an instance of this problem, we can create an instance of k -IR-Invitations as follows. For each node, we create an agent a_i with $S_i = \{k\}$ (agent only approves size k). If there is an edge between two nodes, we make their corresponding agents enemies of each other. The resulting instance is a valid instance (with symmetric social relationships) of k -IR-Invitations with $\alpha = 0$ and $\beta = 3$ (because each node in the original instance has at most three neighbors). This shows NP-hardness of k -IR-Invitations with symmetric social relationships when $\alpha = 0$ and $\beta \geq 3$.

We can modify our reduction to show that the problem is NP-hard when $\alpha = 2$ and $\beta = 1$. First, given an instance of the Independent Set problem with max-degree 3, we create three agents for each vertex and make the three agents friends with one another; these three agents together represent a vertex in the original instance. For each edge in the original instance, pick one agent

from each component representing the either end-point of the edge, and make them enemies of each other; because each node has at most three edges in the original instance, we can manage to keep each agent's enemy set to contain at most one agent. Lastly, we set approval sets of agents such that everyone approves of size k . The resulting instance is a valid instance of SIP with symmetric social relationships with $\alpha = 2$ and $\beta = 1$.

Lastly, we need a similar modification to our reduction for the case when $\alpha = 1$ and $\beta = 2$. First, given an instance of the Independent Set problem with max-degree 3, we create two agents for each vertex and make the two agents friends with each other; these two agents together represent a vertex in the original instance. For each edge in the original instance, pick one agent from each component representing the either end-point of the edge, and make them enemies of each other; because each node has at most three edges in the original instance, we can manage to keep each agent's enemy set to contain at most two agents (and each component can have at most four edges). Lastly, we set approval sets of agents such that everyone approves of size k . The resulting instance is a valid instance of SIP with symmetric social relationships with $\alpha = 1$ and $\beta = 2$.

Note that NP-hardness for $\alpha = a$ and $\beta = b$ implies NP-hardness for $\alpha \geq a$ and $\beta \geq b$. Hence, our proofs provide a complete analysis as stated in the theorem. □

Theorem 28. *When agents have symmetric social relationships, k -IR-Invitations is in FPT if $\beta \leq f(k)$ where $f(k)$ can be an arbitrary function of k .*

Proof. As before, without loss of generality, assume $k \in S_i$ for all $a_i \in N$. We first create a *friend graph* in which nodes represent agents, and we create an edge between two nodes if their corresponding agents are friends. Clearly, subsets of nodes in this graph and invitations have one-to-one correspondence. In the friend graph, it is clear that all or none agents in each component should be chosen to form an IR invitation. Thus, if any connected component contains two nodes whose corresponding agents are enemies of each other, then we can safely remove the component from the graph (as it cannot be included in any IR invitation). Likewise, if any component contains more than k agents, we can remove the component as well.

We then create an *enemy graph* in which nodes represent connected components in the friend graph. Each node in the enemy graph has a weight that is equal to the size of the component it represents, and we create an edge between two nodes if their corresponding components contain a pair of enemies (one agent in each component). Because each agent has at most β enemies, each node in the enemy graph has at most $k \cdot \beta$ edges. Notice that an independent set in the enemy graph

represents an IR invitation in the original instance.

Similarly to the FPT algorithm given in proof of Theorem 19, we use Color Coding to color each node in the enemy graph as $\{0, 1\}$ with equal probability. If there is any edge in the enemy graph whose both end-points (components) are of color 1, then we re-color both of them as 0. We repeat this process until no such pair exists (which can be done in linear time by scanning through the edges). After this step, it is clear that all nodes of color 1 form an independent set; we can easily determine if a subset of nodes whose weight is k exists, using a knapsack-like algorithm.

Provided that an IR invitation of size k exists, this algorithm's probability of success is at least $(1/2^k) \cdot (1/2^{k\beta}) \geq 1/2^{k(1+f(k))}$. For any fixed IR invitation I^* of size k , we color all agents in I^* as color 1 with probability $1/2^k$, and with probability at least $1/2^{k\beta}$ we color the union of enemies of all agents in I^* as color 0. Regardless of coloring of all other agents, this coloring will ensure that all agents in I^* remain to be of color 1 in the enemy graph, and thus our algorithm can find I^* (or some other solution).

The overall runtime of our algorithm is $O(f(k)n)$ as all sub-routines can be implemented in linear time in size of each graph and each graph contains at most $O(n)$ nodes and $O(f(k)n)$ edges. We can repeat this randomized algorithm $2^{k(1+f(k))} \ln n$ times to increase the probability of success to $1 - 1/n$ (with overall runtime $2^{k(1+f(k))} (f(k)n \ln n)$). This algorithm can also be de-randomized using a family of k -perfect hash functions [2]. \square

Lastly we show that k -IR-Invitations remains to be $W[1]$ -complete, even under symmetric social relationships, when β is not bounded. Proof of $W[1]$ -hardness is similar to that of Theorem 20, and completeness follows from Theorem 18. We omit proof of Theorem 29.

Theorem 29. *When agents have symmetric social relationships, k -IR-Invitations is $W[1]$ -complete if β is not bounded above by any function of k .*

5.4.2 Symmetric k -Stable-Invitations

Interestingly, complexity of k -Stable-Invitations and that of k -IR-Invitations are identical, except when β is unbounded, if we assume symmetric social relationships. This implies that the combinatoric complexity due to social relationships plays an important role in SIP, and restrictions on social relationships (such as symmetry) can substantially reduce the complexity. Yet we emphasize that both polytime and FPT algorithms for k -Stable-Invitations are much more complicated than those for k -IR-Invitations, and much of its complexity is due to the additional requirement that uninvited agents must not be willing to attend.

We first present classical complexity results for k -Stable-Invitations under symmetric social relationships, followed by parameterized complexity results.

Theorem 30. *When agents have symmetric social relationships, k -Stable-Invitations can be solved in polynomial time if (i) $\beta = 0$, (ii) $\beta = 1$ and $\alpha \leq 1$, or (iii) $\beta = 2$ and $\alpha = 0$. Otherwise, the problem is NP-hard.*

Proof. Let us first consider the case (i) when $\beta = 0$. We construct a friend graph as before, and find connected components in this graph. Any stable invitation must contain all or none of nodes in each connected component. For each connected component, we check two things: Whether a stable invitation can contain all of nodes in the component and whether it can contain none of nodes in it. To check the first, we simply check if the component is of size k or less and if everyone in the component approves size k . To check the second, we check if the component contains two or more nodes (then we can leave them out) or if it is a singleton component but the only agent in it does not approve size k (then we can leave the agent out). All of these checks can be done in linear time. Now we can use a dynamic programming algorithm to determine whether a subset of connected components that contains k nodes over all such that every component (whether selected or not) does not violate the stability conditions (which can be easily checked by the two conditions we processed earlier).

Let us now consider case (ii) where $\alpha \leq 1$ and $\beta = 1$. We first construct a friend graph, and merge any connected component into a meta-node with weight 2. The resulting graph would only have nodes but no edges, and each node is either of weight 1 or 2. Now we construct an enemy graph on top of this by creating an edge between nodes if their corresponding agents are enemies of each other; if we create self-loops, then it means two agents are both friends and enemies at the same time, and we just remove them from our graph (and input instance) with no harm. Now, after all edges are created, each node of weight 1 has at most one edge and each node of weight 2 has at most two edges. This implies that all connected components in this graph are either a path or a cycle. We can use the same algorithm for case (iii) below to solve this problem.

Finally let us consider case (iii) where $\alpha = 0$ and $\beta = 2$. Let us construct an enemy graph by creating nodes for agents and an edge between nodes if the agents are enemies of each other. Each node can have at most two edges because $\beta = 2$. Therefore, every connected component in the enemy graph is either a path or a cycle. For each connected component and every integer x between 0 and k , we check if it is possible to choose exactly x nodes from the component such that inviting their corresponding agents and not inviting other agents (in the same component) do not

violate stability conditions. This is similar to finding an independent set on a path or a cycle, but with extra conditions: the nodes not being chosen must still satisfy the stability requirements (by not approving size $k + 1$ or by having one of its neighbors included in the set). We omit the details, but mention that this can be solved by a dynamic programming algorithm in polytime. After this step, we can solve the main problem using a knapsack algorithm in a straightforward manner.

Our reductions for k -IR-Invitations in proof of Theorem 27 show NP-hardness for k -Stable-Invitations as well, because agents only approve invitations of size k in our reduction; it ensures that any uninvited agent would be unwilling to attend due to the size of an invitation.

□

Theorem 31. *When agents have symmetric social relationships, k -Stable-Invitations is in FPT if $\beta \leq f(k)$ where $f(k)$ can be an arbitrary function of k .*

Proof. The main is similar to that of our proof of Theorem 28, but details entail more complicated processes due to stability conditions on uninvited agents.

We first create a *friend graph* in which nodes represent agents, and we create an edge between two nodes if they are friends of each other. Any stable invitation must contain all or none of agents in the same connected component; if a connected component contains more than k agents or it contains enemies within itself, then we can safely remove the components (or agents in them) from input instance. Also, if a connected component of size at least 2 contains an agent who does not approve size k , then we remove the entire connected component; if it is just one agent who does not approve size k , then we remove it only if it does not approve size $k + 1$ as well (this is because of stability conditions). After the removal process, we create an *enemy graph* by treating each connected component as a weighted node whose weight is equal to the size of the component, and creating an edge between nodes if they contain enemies (one in each).

Again, we use color coding approach by coloring nodes using two colors $\{0, 1\}$ uniformly and independently at random. If two adjacent nodes in the enemy graph are both colored as 1, then we re-color both of them to 0. We repeat the re-coloring step until no such nodes remain; at this point, all nodes of color 1 form an independent set, and any subset of them form an IR invitation. We will try to find a subset of nodes of color 1 whose weight is equal to k that forms a stable invitation. We can disregard any node of color 0 and weight greater than 1, because the agents contained in those nodes would not block stability (because we only consider single-agent deviation). On the other hand, singleton nodes of color 0 may block an invitation when excluded, if those node approve size $k + 1$ and no enemy of them are invited. To avoid this, we are going to choose at least one enemy

(of color 1) of each singleton node of color 0 who approves size $k + 1$ in brute-force manner; the search space is bounded because we can only choose up to k agents and each agent has at most $f(k)$ enemies (i.e., the search space is $O((f(k))^k)$). After we pre-select a subset of agents of color 1 in this manner, we no longer need to worry about stability condition being violated by any agent of color 0. Next, we check if there is any singleton node of color 1 who approves size $k + 1$ and is not pre-selected in the previous step; those nodes must also be included so as not to violate stability condition. Suppose that we have so far selected x agents in this manner; if $x > k$, then our algorithm will (possibly incorrectly) report that no stable invitation of size k exists. Otherwise, among the remaining nodes of color 1, we will check if an independent set of weight $k - x$ exists (via a knapsack algorithm), to solve the main problem.

Overall, this randomized algorithm runs in FPT time ($O((f(k))^k \cdot (f(k)n))$), and its probability of success is at least $(1/2^k) \cdot (1/2^{k\beta}) \geq 1/2^{k(1+f(k))}$ (similar to the algorithm shown in Theorem 28).

□

5.5 Discussion and Future Work

In this work we investigated the parameterized complexity of the Stable Invitations Problem (SIP) for two different solution concepts – individual rationality and (Nash) stability, when the size of a solution is parameterized. We considered restrictions on inputs by limiting the number of friends and enemies each agent can have, and also studied the special case in which all agents have symmetric social relationships. Despite the fact that the majority of the problems we consider in this work are NP-hard, we showed that many special cases of the problem admit efficient FPT algorithms. Our results indicate that the computational complexity of SIP varies when its input is restricted or the solution concept changes, which is not distinguishable under the classic complexity. Our work leaves a few interesting open problems for future work. Lee and Shoham [31] considered another solution concept in which agents who are not invited are not envious of those who are invited (motivated by ‘envy-freeness’). It would be interesting to analyze the parameterized complexity of finding an envy-free invitation of size k , and compare the results with what we have in this work. In addition, analyzing the parameterized complexity of the Group Activity Selection Problem [14] is another interesting direction for future work.

Chapter 6

Incentive Compatibility in Group Assignment Problems

6.1 Introduction and Related Work

6.2 Definitions and Known Results

6.2.1 Main Results

In the strategic case of ASIP, we assume that agents may act strategically in reporting their preferences to the event organizer. Example 4 shows how agents may have an incentive to act strategically.

Example 4 (An agent may act strategically.). Let us consider an example with two agents with identical preferences:

$$P_1 : 1 \succ 0 \succ 2, \quad P_2 : 1 \succ 0 \succ 2$$

Note that both agents have DEC-preferences (with $h_1 = h_2 = 1$). The two stable invitations are $S_1 = \{a_1\}$ and $S_2 = \{a_2\}$. The empty invitation (\emptyset) is not EF while the full invitation (N) is not IR.

Hence $S_1 = \{a_1\}$ and $S_2 = \{a_2\}$ are the only two stable invitations provided that both agents are truthful. If S_1 were to be chosen by the organizer, a_2 would have an incentive to act strategically, by reporting $\hat{P}_2(1 \succ 2 \succ 0)$ instead of P_2 . Given (P_1, \hat{P}_2) , the only stable invitation is S_2 (S_1 is no longer EF due to a_2). Notice that a_2 strictly prefers S_2 over S_1 (because $1 \succ_2 0$) and has an incentive to misreport in this example. By symmetry, a_1 may act strategically if S_2 were to be chosen.

We first provide a formal definition of a mechanism in the context of ASIP with strategic agents.

We then state several impossibility results for general cases of ASIP, and also provide a strategy-proof mechanism for special cases of ASIP. Although we only consider deterministic mechanisms here, we discuss how one can generalize to randomized mechanisms at the end of this section.

Definition 16. Given an instance (N, P) of ASIP, we define V_i (the set of available actions to a_i) to be the set of all preferences over X where $X = \{0, 1, 2, \dots, n\}$ is the set of outcomes. A (deterministic) *mechanism* is a pair (V, Z) where $V = (V_1 \times \dots \times V_n)$ is the set of action profiles of all agents (i.e., V_i is a subset of total preorder on X) and $Z : V \mapsto U$ is a mapping from each action profile to an invitation in U where $U = 2^N$. Let $V_{-i} = (V_1 \times \dots \times V_{i-1} \times V_{i+1} \times \dots \times V_n)$ be the set of action profiles available to all agents but agent a_i . A mechanism (V, Z) is said to be *strategy-proof* if for all $a_i \in N$ it holds that $Z(P_i, v_{-i}) \succeq_i Z(v_i, v_{-i})$ for all $v_i \in V_i$ and $v_{-i} \in V_{-i}$.

We now formally state our first impossibility result.

Theorem 32. *No strategy-proof mechanism can find a stable invitation, even if it exists, for arbitrary instances of ASIP.*

Proof. Example 4 can serve as a proof; no strategy-proof mechanism can find a stable invitation for this instance. \square

Intuitively this result is due to the conflicting interests of the organizer and agents – the organizer is trying to maximize attendance while the agents (with DEC-preferences) to minimize. Since no strategy-proof mechanisms can find a stable invitation, one can instead seek to design a strategy-proof mechanism that can find a non-empty IR invitation by dropping the requirement of envy-freeness. We show that this is also impossible as Theorem 33 states.

Theorem 33. *No strategy-proof mechanism can find a non-empty individually rational (IR) invitation, even if it exists, for arbitrary instances of ASIP.*

Proof. Consider three agents with preferences as follows:

$$P_1 : 3 \succ 0 \succ 2 \sim 1, \quad P_2 : 2 \succ 3 \succ 2 \succ 1, \quad P_3 : 3 \succ 2 \succ 0 \succ 1$$

There are two non-empty IR invitations: $S_1 = \{a_1, a_2, a_3\}$ and $S_2 = \{a_2, a_3\}$. If a mechanism chooses S_1 given (P_1, P_2, P_3) , a_2 can report $(2 \succ 0 \succ 3 \sim 1)$ and make S_2 the only non-empty IR invitation. Similarly, if a mechanism chooses S_2 given (P_1, P_2, P_3) , a_3 can report $(3 \succ 0 \succ 2 \sim 1)$ so as to make S_3 the only non-empty IR invitation. The rest of the proof is similar to that of Theorem 32. \square

Earlier we emphasized that the conflict between agent(s) and the organizer is the main factor that leads to an impossibility result. Indeed, in the example we used in the proof of Theorem 32, both agents have DEC-preferences while the organizer's goal is to maximize attendance.

We now consider a special case of ASIP in which all agents have INC-preferences. We obtain a positive result in this case as Theorem 34 states.

Theorem 34. *There is a strategy-proof mechanism for INC-instances of ASIP, which can also find a maximum stable invitation in linear time (after sorting).*

Proof sketch. For simplicity let us assume that each agent reports her threshold value (i.e., l_i as defined in Definition (to be added)) as L_i (L_i may differ from l_i). Our mechanism then chooses the largest k such that $L_k \leq k$ holds and chooses the set of k agents with largest threshold values (if no such k exists, mechanism chooses the empty invitation). \square

Although our mechanism is simple, proof of Theorem 34 is not trivial. First, even though all agents have INC-preferences, a full invitation is not necessarily stable (if at least one agent is unwilling to attend at all). Second, an agent may have an incentive to under-report that results in having the organizer to invite more agents than when the agent had truthfully reported. It is indeed possible for an agent to under-report ($L_i < l_i$) and lead to a larger invitation, but we show that the size of the larger invitation would still be less than l_i (see Appendix for a proof).

6.2.2 Extensions

Although we have so far only discussed deterministic mechanisms, our impossibility results can be extended to randomized mechanisms. First we define Z to be a mapping from V to $\Pi(U)$ where $\Pi(U)$ denotes the set of all probability distributions over U . The definition of a strategy-proof mechanism must change accordingly – we do this by adopting the axioms in the von Neumann-Morgenstern utility theorem [44]. We introduce lotteries over invitations and define preferences of agents over lotteries. Given a probability distribution over invitations, one can compute the expected cardinal utility of lotteries. We then define a strategy-proof mechanism analogously to Definition 16: for each a_i , it must hold that the expected utility of $Z(P_i, v_{-i})$ is no less than the expected utility of $Z(v_i, v_{-i})$ for all $v_i \in V_i$ and for all $v_{-i} \in V_{-i}$. The impossibility result given by Theorem 32 still holds: If (V, Z) is a strategy-proof mechanism, then $Z(P_1, P_2)$ must assign zero probability to both $\{a_1\}$ and $\{a_2\}$, yet these are the only two stable invitations. All other impossibility results we mention in this work can be extended in this manner.

We can extend our results in a different direction by considering multiple time-alternatives for the event. In such settings, agents may have preferences over time-alternatives for the event, in addition to size of invitations. In the non-strategic case, our easiness result is still applicable: One can run the algorithm used in Theorem (to be added GT:thm:asip:algo) iteratively for each time-alternative, and choose the maximum stable invitation among all. In the strategic case, our impossibility results immediately imply the same negative results. For INC-instances of ASIP, we obtain a similar impossibility result even if there is only two time-alternatives. The intuition is that over-reporting ($L_i > l_i$) can give the veto power to an agent, which prevents us from designing a strategy-proof mechanism even for INC-instances of ASIP. Note that all of our impossibility results can be naturally extended to the Group Activity Selection Problem by Darmann et al. [14] since ASIPs is a sub-class of GASP with a single activity.

Chapter 7

Discussion and 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.