Abstract: We study the matching of workers to firms when workers have preferences over being matched with their colleagues. We extend existing results by considering that workers want to be matched in groups of friends, and these groups of friends partition the set of workers. Rather than agents having preferences over their own side of the market, they have a pre-determined set of agents with whom they want to be matched, and we want to keep these groups intact to whatever extent we can. We then propose several algorithms to match agents in groups, and we analyze the stability and other desirable properties of the matchings. We develop an algorithm to find a stable matching in the case of all workers in a group having the same preferences, all groups being the same size, and all firms having the same capacity. However, if any of these three conditions fail to hold, then there may not exist a stable matching for workers who want to be matched in groups. We include numerical results in the case of matchings that are almost stable. We examine an actual market to which we can apply our model and demonstrate how one of our algorithms does a better job of matching agents in groups.

Email: ginny6@stanford.edu

Keywords: market design, sororities, microeconomics, deferred acceptance, two-sided matching, coalition formation.

Acknowledgements: I am immensely grateful to Professor Fuhito Kojima, my advisor throughout the process. Without his expertise, support, and endless advice over the past year, I would have been unable to finish this thesis. I would like to thank Professor Rothwell for all the support through the honors seminar and college, as well as the rest of the Stanford Economics Department.
## Contents

1. Introduction 4  
2. Literature Review 5  
2.1. Stability and Preferences 6  
2.2. Many-to-Many 8  
2.3. Preferences over Colleagues 8  
2.4. Couples 9  
2.5. Coalition Formation 9  
2.6. Housing 10  
2.7. Roommates 10  
2.8. Fractional Assignments 11  
2.9. Preference Aggregation 11  
3. Our Model 12  
3.1. Problem Statement 12  
3.2. Firm Preferences 12  
3.3. Worker Preferences 13  
3.4. Group Preferences 15  
3.5. Solution Concepts 19  
4. Simple Examples for Unsplittable Groups 21  
5. Greedy Group Algorithm 22  
5.1. Greedy Group Algorithm 23  
5.2. Analysis of the Greedy Group Algorithm 25  
6. Same Size Algorithm 29  
6.1. Analysis of the Same Size Algorithm 35  
6.2. Numerics 41  
7. Varying Sizes Algorithm 43  
7.1. Algorithmic Analysis 48  
7.2. Numerics 49  
8. Sororities 49  
9. Conclusion 53  
References 54
Appendix A. Glossary of New Terms 57
Appendix B. Sequential Group Algorithm 58
Appendix C. Additional Numerical Results 60
Appendix D. Proof of Lemma 31 65
Appendix E. Proof of Lemma 36 66
1. INTRODUCTION

Imagine a group of four friends all hoping to join a sorority, a social organization for women popular in American colleges. There exist a number of models to match two sides of a market, and one of these many models does indeed match the four friends to a sorority in practice. However, consider the possibility that not only do these friends prefer to be matched with one another, they only want to be matched with one another. In this case, existing models are not sufficient to do the task of guaranteeing a stable outcome. The question of matching women to sororities is where we began our seeking an answer to the question: how do we match agents who prefer to be matched in groups?

A key omission from many previous matching models is one in which agents strongly prefer to be matched in units, and that is what we consider here. This is a variant of the many-to-one matching problem in which one side of the market would like to be matched in groups. We first state the problem, then discuss past models. We look at a few potential solutions and study their theoretical and empirical properties. We find an algorithm for a stable matching in the case of very specific preferences, and show counter-examples to stability in cases where those preference restrictions do not apply. We come full circle at the end and apply our algorithm to matching women in sororities.

To demonstrate the relevance of our model, we consider markets in which this group structure exists.

(1) As discussed above, sorority rush refers to the process in which women at a college or university are selected to join their school’s chapter of a particular sorority. Roth and Mongell discuss sorority matching in [41], and they note that a high percentage of women rushing sororities “suicide,” meaning they list only one sorority in their preference ranking. This suggests that women care a great deal about being with their friends if their friends are already in one particular sorority. In Section VII of [41], the first open question that the authors ask is how to extend the model to consider that rushees have preferences over groups of friends in the same sorority. Our idea of matching in groups offers a possible answer to this question, as girls have preferences over agents in the same group. We later explore the possibility that the lack of a group matching procedure accounts for high suicide rates among women rushing sororities.

(2) Members of a competitive middle school athletics team, such as a tryout-only travel team, are recruited by high schools in the area to play for their school team. As the middle school sports get more and more competitive (see [58]), high schools need to use different techniques to recruit top players. Not only would the athletes have preferences over remaining with a large fraction of their existing team, but they have developed a capacity to work together that makes them better as a
cohesive unit, so the schools derive more utility from having athletes come in groups. This would be a case in which not only the agents within the group have a preference to stay together, but also the agent matched to them would rather the agents stay with their original group. Their value as a unit is more than the aggregate value of the agents as individuals, and thus we would want to match them using a model for matching in groups.

(3) Grouper Social Club is an online dating site that seeks to ease the awkwardness of a blind date by matching members to go on a group date, rather than a 1-on-1 date [28]. While this could be a standard many-to-many matching problem, given the awkwardness of first dates, one or both sides of the market might want to apply for dates as a group. For example, three women might like to use the site to be matched on a group date together, and this would require a different model that incorporates matching in groups. Our model could be used to take in the preferences of groups of agents using the site and produce a matching that sets them up on group dates.

2. Literature Review

The goal of the market design literature is to create optimal markets or institutions in order to correct current market failures. Matching theory in particular focuses on matching mechanisms divided into one-sided matching problems, that match one set of agents to each other, and two-sided matching problems, that match two sides of a market in which each side has preferences over the other side.

The standard one-to-one matching problem is a marriage market in which a set of men have preferences over a set of women and vice versa. Here, each agent will be matched to exactly one other agent or remain unmatched. The many-to-one matching problem, which our problem is, matches many agents on one side of the market to one agent on the other. For example, the college admissions problem is a many-to-one matching problem because many students are matched to a single college.

We are going to use the terminology of firms and workers throughout. This is a two-sided many-to-one matching problem with a set of workers $W$ and a set of firms $F$. Assume each firm $f_i$ has capacity $k_i$. Following the existing market design literature, we define the following:

**Definition 1.** A matching $\mu$ is a function from the set $W \cup F$ into $F \cup 2^W$ such that:

1. $\mu(w) = \emptyset$ if $w \notin \mu(f_i)$ for all $f_i \in F$. We use $\emptyset$ to denote unemployment.
2. $|\mu(f_i)| \leq k_i$ for all $f_i \in F$.
3. $f \in \mu(w) \iff w \in \mu(f)$ for all $f \in F$ and $w \in W$. 
The most important algorithm that we use is the Deferred Acceptance algorithm of Gale and Shapley. The Deferred Acceptance algorithm is a matching algorithm in which men propose to women they would like to marry, and women are allowed to hold or reject the offer. Women do not accept anyone until the end when they have seen all their offers, and men continue to propose down their preference lists until they are accepted or rejected by all acceptable women. This algorithm is foundational because the designers proved that there does not exist a rogue pair, so the matching is stable and has become the basis for many future matching algorithms. We will use the deferred acceptance algorithm extensively in our paper.

We are not the first to consider generalizations to the standard one-to-one or many-to-one matching models, and we discuss many alternatives below. We first define precisely what it means for a matching to be stable, and how we can think about preferences of agents.

2.1. Stability and Preferences. We here look at common preference restrictions found in the market design literature. An agent \( i \) has preferences \( \succ_i \) over any set of other agents if \( a \succ_i b \) means agent \( i \) would be happier being matched to \( a \) (\( a \) can be either one agent or a set of agents) than to \( b \). We will sometimes drop the subscript if it is clear to whose preferences we are referring. We discuss how to use the idea of an individual’s preferences to create notions of a preferred subset.

**Definition 2.** The choice set \( Ch_i(S) \) for an agent \( i \) is the most preferred subset of \( S \), when \( S \) is chosen entirely from the other side of the market.

We now define several more types of preference restrictions.

**Definition 3.** An agent \( i \) has substitutable preferences if for any set \( w, w' \in S \) for agents \( S \) on the other side of the market, if \( w \) is the optimal choice for \( i \) from \( S \), then \( w \) is the optimal choice for \( i \) from \( S \setminus w' \). Firm \( f_i \) has strongly-substitutable preferences if hiring \( w \) is optimal when the set of available workers is \( S \cup w \), and \( S \succ_i S' \), then hiring \( w \) is optimal when the set of available workers is \( S' \cup w \).

We also want to borrow from preference restrictions about how agents want to be matched to each other, so we provide definitions of several more specific preference concepts.

**Definition 4.** In [25], Dutta and Masso define togetherness as a preference restriction on couples that states that for all possible matchings, the couples prefer all matchings to the same firm that match them together to all matchings to the same firm that match them apart, and are indifferent among all matchings
to the same firm that match them together. In [49], Revilla defines **group-togetherness**, which is similar to group-lexicographic (to be defined later) except that agents care about not being matched with an unacceptable coalition. Agents’ preferences satisfy group-togetherness if each agent prefers all matchings with an acceptable coalition (regardless of firm) to all matchings with an unacceptable coalition, but among matchings with acceptable coalitions, the agent prefers that in which he or she is matched to a more preferred firm.

We need to define solution concepts to evaluate how effective matchings are. We start with the more basic solution concepts initially encountered in one-to-one matchings.

**Definition 5.** Given an initial matching \( \mu \), another matching \( \eta \) is **individually rational** if is it weakly preferred to \( \mu \) for all agents. This means no single agent wants to defect from the matching. A matching is part of the **pairwise stable set** if there do not exist any two agents who would rather be matched to each other than their current partners, and the matching is individually rational. If a matching is not in the pairwise stable set, then either one or two individuals **block** the matching.

We want to define what it means for a whole coalition to block a matching, as we will be working with groups rather than just individuals.

**Definition 6.** We borrow from [26] in defining the **blocking coalition**. For \( F' \subseteq F \), \( W' \subseteq W \), and a matching \( \mu' \), we say the triple \((W', F', \mu')\) form a blocking coalition to a matching \( \mu \) if the following are true:

1. At least one agent is involved.
2. For all \( f \in F' \) and \( w \in W' \), \( \mu'(f) \in 2^W' \) and \( \mu'(w) \in F' \cup 2^W' \); the agents in the blocking coalition can be matched under \( \mu' \) even if no other agents are involved.
3. All \( c \in F' \cup W' \) are weakly better off under \( \mu' \) than \( \mu \).
4. At least one \( c \in F' \cup W' \) is strictly better off under \( \mu' \) than \( \mu \).

We want to stipulate what it means for a matching to be stable, and what set the stable matchings are in.

**Definition 7.** A matching is a part of the **weak core** if there does not exist a blocking coalition and every agent is individually rational.

In [34], Fuhito and Kamada define a market failure in a matching problem as a system in which there exists a stable matching or a matching with fewer unmatched candidates but this better matching is not the
outcome; thus, the problem of identifying a market failure is reduced to that of finding one rogue pair, two
agents who would both rather be with each other than their current partner.

Market designers also worry about whether or not agents can use strategic behavior to defect from a
matching and improve their well-being. Given that economists typically assume agents will behave in a
self-serving way if possible, an optimal matching mechanism will not allow for profitable deviations, and so
we offer definitions of the following concepts.

Definition 8. A mechanism is manipulable if any agent can make itself better off by doing something other
than reporting the truth or accepting the outcome of the mechanism. This typically refers to manipulating
preferences lists, but there are other ways in which a mechanism can be manipulable, including pre-arranged
matches or lying about capacities. A mechanism is strategy-proof if it is not manipulable.

We now discuss several more specific types of matching problems and existing results about their solutions.
The problems that we discuss below are directly relevant to our model, and much the later analysis either
builds directly on these problems, or is inspired by them.

2.2. Many-to-Many. The many-to-many matching problem is a two-sided matching problem in which
each side of the market wants to be matched to many agents on the other side. For example, firms hiring
consultants want multiple consultants, and consultants are able to consult for a number of firms. In [29],
Hatfield, Kojima, and Natia study many-to-many matchings with preferences restricted by the max-min
criterion, such that agents try to maximize their minimum expected utility. They represent the problem
as a matrix with row and column players being the agents on each side of the market. The authors prove
that there does not exist a Pareto efficient, strategy-proof, or monotonic stable mechanism for agents on
one side of the market. In [13], Baiou and Balinski study the many-to-many problem, and show that if each
agent has a max-min preferences over the other set of agents, then the “man-optimal” and “woman-optimal”
assignments in the many-to-many problem can be characterized in terms of efficiency, monotonicity, and
strategy-proofness. They then frame the problem as a directed graph in which each row, column pair forms
a node, and arcs point towards more preferred nodes. They show that row players can compare their stable
assignments by looking at the least preferred column mate: either two assignments are the same, or the least
preferred mate in the better assignment is preferred to the least preferred mate in the worse assignment
(Corollary 1, [13]).

2.3. Preferences over Colleagues. The matchings with preferences over colleagues problem is a two-sided
matching problem in which agents on one side of the market not only have preferences over the other side
of the market, but over their side as well. This is where we take our definition of matching in Definition 1; the matching takes the set of firms and workers to the set of firms and all possible subsets of workers, and workers have preferences over each of these subsets. In [26], Echenique and Yenmez present an algorithm that finds stable matchings (if they exist) in the many-to-one matching model with preferences over colleagues. The authors show that the stable matchings coincide with the fixed points of an operator $T$. By composing the operator with itself, they develop an algorithm to determine the existence of stable matchings and find the matchings when they exist. The authors do not put restrictions on agents’ preferences to ensure the existence of stable matchings, but rather focus on finding the solutions if they exist, so their results hold for any structure on agent preferences. Their model extends the work in [25] to find all stable matchings in the Dutta-Masso model. In [25], Dutta and Masso also find a nonempty core for a class of models in which couples have colleague-lexicographic preferences, similar to the preference restriction that we will later impose on our agents. In [38], Kominers uses their solution to find all stable matchings in any many-to-one matching market.

2.4. Couples. The National Residency Matching Program (see [51]) encountered the problem of matching with couples when doctors wanted to defect from their stable match in order to be in the same location as their spouses. In a matching with couples problem, workers a priori define with whom they would like to be matched. In the new National Residency Matching Program, agents have lists of sets of hospitals (or firms) that they rank, so partners can choose to be close together. In [32], Klaus and Klijn define a couple a strongly unemployment averse if it prefers the full employment to the employment of only one member, and the employment to only one member to the employment of neither. A couple is strictly unemployment averse if either partner is worse off when the other loses their job. They develop several nice properties of matching couples when couples are strongly unemployment averse (which implies strictly unemployment averse). They find that restricted strict unemployment aversion, a weaker notion of strict unemployment aversion, is that maximal domain for the existence of stable matchings in a market with couples. In [37], Kojima, Pathak, and Roth discuss matching with couples in large markets. We generalize one of their algorithms, the Sequential Couples Algorithm, as an example of how to use existing models to match agents in groups. The generalization is very straightforward, and we leave the details to Appendix B. We do not try to generalize any of their other results but merely demonstrate how to alter an existing model.

2.5. Coalition Formation. The problem of coalition formation is one in which a set of agents seeks to find the optimal coalition. A pure hedonic game is one in which players’ utility depends only on the identity of the members of the group that player is in. Alcalde and Revilla define Tops Responsiveness, a preference
restriction relevant to coalition formation problems. The authors consider a coalition formation problem in a one-sided market, so agents’ preferences are only defined over the power set of agents. The say that an agent’s preferences are Tops Responsive if the agent ranks sets of agents by their choice set from that set, and if they prefer smaller coalitions. In [8], the same two authors produce the Choices Covering Algorithm, which defines a way to make coalitions that partition the set of agents. They then use the Tops Covering Algorithm to produce a stable outcome in the event of preferences with the Tops Responsiveness condition.

In [49], Revilla considers the many-to-one matching problem in the context of coalition formation. He defines coalitional substitutability, a property stronger than substitutability and similar to group substitutability defined in [25]. He finds a problem with a non-empty core when preferences satisfy the Common Best Colleague condition, which is similar to the Unanimous Ranking According to Desirability condition defined in [25]. As the names suggest, both conditions require that workers have a generally unified preferences ranking. In [46], Pycia considers coalition formation in problems with complementarities and peer effects. In [47], Pycia again considers coalition formation, and this time does not allow for substitutable preferences. For stability in this model, agents’ preferences must again be pairwise aligned, but whether or not they are pairwise aligned is a product of the sharing rule used by the coalition.

2.6. Housing. In problems in which groups of university students draw for housing, students would like to be matched in groups, and they define their groups ahead of time. In this problem, only one side of the market (the students) have preferences. In [56], Sönmez discusses applications of the Top Trading Cycles algorithm, another commonly used algorithm in the market design literature, to the housing match in the context of a collective ownership economy. He also discusses simple serial dictatorship and random serial dictatorship (RSD), both of which are mechanisms for a one-sided matching. One benefit of using serial dictatorship later in aggregating preferences is that it is strategy-proof, as well as group-strategy proof.

2.7. Roommates. The roommates problem is similar to the housing problem, except it is a one-sided market in which agents first want to form a coalition to all be matched together. As the name suggests, the roommates problem is about finding a stable set of roommates from a one-sided market. In [33], Klaus, Klijn, and Walzl discuss “stochastic” stability. They define a dynamic process in which at time $t$, we randomly select a pair, and match them at time $t+1$ if they form a blocking pair. The sequence of matchings at each is a blocking path. They define an absorbing set as a minimal set of matchings that once entered is never left by blocking dynamics, and a set of absorbing matchings $A(R)$ as the set of matchings such that there exists an absorbing set containing the matching. $A(R)$ is useful in measuring how close we are to a stable matching, or whether one exists for the given market. The authors introduce perturbation to the typical
dynamic model. In [1], Abraham, Biro, and Manlove describe a new notion of stability, almost stable, in search of a matching that admits the fewest number of blocking pairs possible. They show that the problem of finding this matching is NP-hard. We will extend the idea of almost stable to a matching that is only blocked by one particular type of blocking coalition. Morrill focuses on matchings that are Pareto-efficient and individually rational. He defines a new definition of stability, stable subject to initial assignment, in which both agents must want to leave a pair in order for the match to be unstable. Morrill notes that matching subject to initial assignments is particularly applicable in a dynamic environment.

2.8. Fractional Assignments. Not all matchings are integer-valued. This is more commonly an area of operations research and computer science, but it has applications to the matching markets studied in economics. For example, the classic job assignment problems seeks to assign a set of jobs to a set of machines, but each job can be completed on a number of different machines, so the goal is to optimally assign all jobs without over-exerting any machine. This type of problem relies heavily on linear programming, and the mathematics is more sophisticated because now the focus is not restricted only to the integers. Linear programming seeks to minimize or maximize some function subject to constraints; looking at the constraints can turn a minimization problem into a maximization problem, or vice versa. In [22], Dean and Swar study the generalized stable allocation problem, which looks at a two-sided set of ranked preference lists, so now machines have preferences over jobs. In [9], Alkan and Gale show that if preferences are size-monotonic, then the set of stable matchings is distributive lattice. The set of stable matchings in their case has other interesting properties. For example, the worker may work two different schedules in two different matchings, but he or she will always work the same number of hours. Sethuraman, Teo, and Dean, Goemans, and Immorlica [21] study the stable marriage problem with flexible capacities, so “machines” can increase their possible load marginally to keep the “jobs” integer-valued. Their model is highly relevant in the case of matching with couples, as a firm can extend their capacity by one if they want to hire a worker who requires that his or her spouse also be at the same firm. We can use existing results from [21] to match a group in the case of the group refusing to be split up, as we consider in Section 4.

2.9. Preference Aggregation. The existing algorithms for preference aggregation allow a set of agents to turn individual preference lists into a group preference list. In [17], Chae and Heidhues note that most existing literature on preference aggregation is biased in favor of one chosen representative, whichever is the best bargainer. They consider a bargaining problem in which at least one of the bargainers represents a coalition, and the coalition aggregates preferences before bargaining. They find that forming a coalition is unprofitable in pure-bargaining games, but if the bargaining game is not pure, forming a coalition can
lead to gains. The problem of aggregating preferences is a voting problem; in fact, voting schemes are just a type of aggregation rule. Social choice theory typically breaks down into two areas: choice rules that map individual preferences into outcomes (this includes voting theory and resource allocation [16]) and aggregation rules, which are social welfare functions. In [16], Bossert and Sprumont study strategy-proof social aggregation rules. An aggregation rule is strategy-proof if misreporting one’s preferences never produces a social preference ranking that is better strictly between the original preference list and the individual’s preference list. They also partition the set of aggregation rules into those that are manipulable and those that are strategy-proof.

3. Our Model

We will state our exact problem here, and then define the preference restrictions we impose on our firms and workers. We also describe which stability concepts will be most relevant in this paper.

3.1. Problem Statement. We here define how we are going to study matchings in groups. As mentioned above, this is a two-sided many-to-one matching problem. We will use the terminology of firms and workers throughout. There are two sets of agents, $W$ (workers) and $F$ (firms), with $|W| = n$, $|F| = m$.

**Definition 9.** A group is a set of agents who all have chosen to be in the set, so agents within the group have a reflexive friendship such that they prefer each other to agents outside the group.

Agents in $W$ are subdivided into groups $\{g_1, \ldots, g_l\} \in G$, where $G$ denotes the set of groups of elements of $W$, such that $\sum_{i=1}^{l} |g_i| = n$, so each agent in $W$ is in exactly one element of $G$. Let each firm $f_i$ have capacity $k_i$, and we write $\text{cap}(f_i)$ for the capacity of firm $f_i$. Let $|g_i| = p_i$, so $p_i$ is the size of group $g_i$.

Within a group, workers would rather stay with other agents in their group than be separated, and they are indifferent among workers from other groups who might be matched to the same firm.

3.2. Firm Preferences. Assume the firms each have strict, complete, and transitive preferences over the set of $n$ individual workers (rather than the set of groups), so each firm $f_i$ has preference ordering: $w_{i_1} \succ w_{i_2} \succ \cdots \succ w_{i_n}$. Each firm ranks all workers, so firms find all workers acceptable.

The most significant restriction on firm preferences is the way that firms derive group preferences from individual preferences. Assume the following property:

**Definition 10.** Let $g_1 = \{w_{1_1}, \ldots, w_{1_l}\}$ and $g_2 = \{w_{2_1}, \ldots, w_{1_k}\}$, and firm $f_i$’s preferences over individual workers be $w_{f_{i_1}} \succ w_{f_{i_2}} \succ \cdots \succ w_{f_{i_n}}$. If $f_i$’s preferences are favorite-lexicographic, then $f_i$’s preferences over groups are $g_1 \succ g_2$ if and only if for the first $i$ such that $w_{f_i} \notin g_1 \cap g_2$, then $w_{f_i} \in g_1$. This means that
firms rank groups based on their *most preferred* worker in each group. Note that this happens regardless of the size of the sets of agents, and this also implies that firms must think all workers are acceptable, or else they could at some point prefer a group with an unacceptable agent. However, we do specify that firms treat a group of workers of size larger than their capacity as unacceptable. The firms only rank sets of workers of size less than or equal to their capacity.

Favorite-lexicography allows firms preferences to extend naturally over the \(2^W\) possible subsets of workers, and it means firms will not regret their hiring decisions. At each stage of the algorithm, a firm either does or does not receive a proposal from a set of workers. This set might be a whole group that is unwilling to be split up, it might be a part of a group, or it might just be an individual worker. However, because firms' preferences are defined so that the firm can consistently rank a group of workers based on the firm's individual preferences, the firm will be able to evaluate any type of proposal it receives regardless of the number of workers proposing. Furthermore, because firms' preferences over individuals are strict, their preferences over groups are strict also, and they only rank two groups as equal if the groups contain exactly the same workers. Indeed, with favorite-lexicographic preferences, all properties true of firms’ preferences over individual workers extend to their preferences over groups.

Favorite-lexicography combined with the requirement that firms rank all workers implies a type size-monotonicity condition (this is not the exact definition of size-monotonicity, but does imply that firms prefer larger groups). If a firm \(f\) already employs workers \(W_f = \{w_{f_1}, \ldots, w_{f_r}\}\) and has a remaining capacity \(c_f\), then for any group \(W_p\), such that \(|W_p| \leq c_f\), that proposes to \(f\), \(f\) will prefer \(W_p \cup W_f\) to \(W_f\) because the first worker \(w_{f_i} \notin (W_p \cup W_f) \cap W_f\) (which is just \(W_f\)) is preferred to no worker, and thus \(f\) prefers the union even if the most preferred worker of \(f\) in \(W_p\) is less preferred than the most preferred worker \(f\) already employs. This is because \(f\) is not comparing \(W_p\) to \(W_f\), \(f\) is comparing \(W_p \cup W_f\) to \(W_f\). Thus, for the choice set in Definition 2, \(Ch_f(S) = S\) for all firms \(f\), which automatically implies substitutability and strong-substitutability given in Section 2.1.

3.3. *Worker Preferences.* Defining preferences of workers over firms is more challenging because workers are both individuals and members of a group. We have to define both types of preferences. In this section, we only discuss individual workers’ preferences, not the preferences of the whole group. We discuss group preferences in great detail in Section 3.4.

We have two cases:
Case 1. Workers’ preferences over other workers are based strictly on the group structure. The major departure that we make over existing matching with preferences or coalition formation problems is that workers never want to switch groups; they may be content with a matching that does not contain all their group members, but the groups are set from the start, and workers do not have any interest in joining another group. For example, if an entire group \( g \) is matched to firm \( f \), then all members of \( g \) are indifferent among all matchings that match the entire group to \( f \).

Case 2. For workers’ individual preferences over firms, we assume that each worker has strict preferences over all \( m \) firms. We assume that workers rank all firms.

The most important restriction on workers’ individual preferences is the following:

**Definition 11.** A worker \( w_i \) in a group has group-lexicographic preferences if \( w_i \) would always prefer to be matched with more agents in their group, regardless of the firm. So, if \( w_i \) has a choice to be matched with a subset of its group of size 2 at a \( f_1 \) or a subset of size 3 at \( f_2 \), even if \( f_1 \succ f_2 \) according to the individual firm rankings of \( w_i \), \( w_i \) would still prefer to be matched to \( f_2 \). Workers with group-lexicographic preferences only look at their individual preferences if they are considering firms that all employ the same number of workers from their group. Note that this refers to the number of group members that are not the worker, so the worker excludes him or herself from the count. We make an exception in the case of a whole group being unmatched. If we extended group-lexicography to also cover the option that the group be unmatched, then groups would necessarily remain unmatched if the whole group cannot be matched together. The workers in a group may prefer to split up than be unemployed.

We also assume that although workers are heterogeneous to firms, they are homogeneous within the group, so if a worker can only be matched with a subset of their group of size \( d \), they are indifferent among all subsets of size \( d \). Group-togetherness, in Definition 4, trivially applies to our model because for each agent, the acceptable coalitions are just that agent’s group, and if the whole group can be matched together, then the agent ranks firms based on individual preferences. Indeed, group-lexicography is a generalization of togetherness.

We also want groups to be able to decide whether or not they are willing to be split up, and, if so, how much a firm can split them.

**Definition 12.** Groups can stipulate an offer number, which is the minimum number of workers that a firm must accept from the group. If a group is unwilling to be split up, then their offer number is the size of
the whole group. If a group \( g \) proposes to firm \( f \) with offer number \( k \), then \( f \) can choose its \( k \) most preferred workers from \( g \) to employ, but if \( f \) only has space for \( k - 1 \) workers, or only wants \( k - 1 \) workers, then \( g \) retracts the proposal and moves on to their next most preferred firm. The offer number can change during the matching.

There is a precedent for restricting the domain of the types of preferences (see [15]). Note that embedded in the definition of an offer number, we have an example of a matching that would be unacceptable: one in which a group was matched in a subgroup of size smaller than their offer number. Thus, even though firms find all workers acceptable and vice versa, there still exist matchings from which individuals would choose to defect. As an example of how strong the assumption of group-lexicographic preferences is, consider the following market:

**Example 13.** Assume we have firms \( F = \{f_1, \ldots, f_m\} \) and workers \( W = \{w_1, \ldots, w_k\} = g_1 \), so all workers are in the same group, so they all want to be matched together. Assume the whole group chooses the preferences of worker \( w_k \). Let all firms have capacity \( k \), workers \( w_1, \ldots, w_{k-1} \) rank firm \( f_1 \) last (their preferences over the others is irrelevant in this example), and \( w_k \) rank \( f_1 \) first. The entire group first applies to \( f_1 \), because it is \( w_k \)’s top choice, and \( f_1 \) accepts them (because \( g_1 \) is the entire market). Under the assumptions of the model, this matching is stable even though all workers except \( w_k \) are matched to their least preferred firm, and their favorite firm would have the capacity to take all of them. This suggests that the way that we aggregate group preferences is important.

Our definitions of each individual worker’s preferences over firms and other workers extends naturally to their preferences over \( F \cup 2^W \), the range of all possible matchings. For a worker \( w \) in a group \( g \), and two matchings \( \alpha, \beta \in F \cup 2^W \) such that \( \alpha = f_\alpha \cup W_\alpha \) where \( W_\alpha = \{w_1, \ldots, w_\alpha\} \), and \( \beta = f_\beta \cup W_\beta \) where \( W_\beta = \{w_1, \ldots, w_\beta\} \), we say that \( \alpha \succ \beta \) for worker \( w \) if either:

- The number of members of \( g \) in \( W_\alpha \) is larger than the number of members of \( g \) in \( W_\beta \).
- The number of members of \( g \) in \( W_\alpha \) is equal to the number of members of \( g \) in \( W_\beta \) and \( f_\alpha \succ f_\beta \) in \( w \)’s preference ranking over firms.

We say that \( \alpha = \beta \) for worker \( w \) if \( f_\alpha = f_\beta \) and the number of members of \( g \) in \( W_\alpha \) is equal to the number of members of \( g \) in \( W_\beta \). Thus, for a fixed integer \( n \) and a fixed firm \( f \), an agent is indifferent over all matchings that match \( n \) members of their group to firm \( f \).

### 3.4. Group Preferences

We now need a way to define preferences of a group of workers over a set of firms. We want to consider several different aggregation rules that allow a group to derive group preferences
from individual preferences. We want our workers’ preferences to be defined such that we can measure the
ecfficacy of our aggregation rule. To have an equilibrium, we need an aggregation rule whereby no worker
wants to leave their group. Fortunately, group-lexicography takes care of this for the most part, but we will
see that a worker’s individual preferences can block a matching. We go through three examples of potential
aggregation rules, and then at the end specify how groups will aggregate preferences in our model.

(1) We can rank the firms based on a Borda count (see [16]) of workers in the group. Each worker’s most
preferred firm gets assigned $k$ points, the second most preferred firm $k - 1$ points, and so forth. The
firms are then ranked top down by the number of points. The problem with Borda voting is that it is
subject to tactical manipulations, like workers pushing popular firms they do not like further down
their list, even if they prefer that firm more than firms they rank higher. As such, if this aggregation
rule is used, then worker preferences are manipulable by preference lists. Nonetheless, this method
is appealing because we can aggregate all preferences in just one shot.

(2) We can use a popularity dictatorship mechanism. We assign all agents a “popularity” ranking, much
like that used in job-scheduling problems, and allow the most popular agent in a group to choose
the preferences of the whole group. We take this idea from [53], and say that a worker is the most
popular if the most firms rank that worker first; the ordering proceeds from there, so after all
workers ranked first by any firm have been ordered, it proceeds to the most number of second-
place rankings, and so on. We could also aggregate firm preferences by a Borda count and then
use this aggregation as the index for firm rankings of workers. Each group assumes the preferences
of their most popular agent. The benefit of using a rule like this is that the most popular agent
makes the group likelier to be accepted by firms because of favorite-lexicographic preferences, so
it makes sense to follow the lead of the most popular agent. The other benefit is that there is no
advantage in misreporting preferences, because the only determinant of the group’s preferences are
firms’ preferences over workers, something individual workers cannot control. Popularity dictatorship
is particularly effective when firms’ preferences are favorite-lexicographic.

(3) We can use a random dictatorship, in which we just choose one worker and assign the group the
preferences of that worker. The benefit is that this is the easiest aggregation rule to implement in
practice.

A problem that we face is that the groups may get split up in the middle of a matching algorithm but
workers still want to be matched together. If we have an aggregation rule that re-aggregates preferences
based strictly on individual preferences, then the preferences of two smaller subgroups of the group might
be completely different, which decreases the likelihood that they will be matched together. For example, an exponentially inefficient way of aggregating group preferences is to require a group of size $k$ to have each of its $2^k$ subsets to list their preferences over firms right at the start of the matching, and then no matter what subsets the group is divided into, the algorithm will know how they rank firms. This is only even mildly plausible if we have very small group sizes, and not all subgroups will aggregate preferences the same way. We thus want to stipulate from the beginning that two subsets of the same group aggregate preferences the same way at a fixed step of the matching algorithm.

**Definition 14.** We say an aggregation rule is **subgroup-consistent** if any two subgroups from the same original group have the same preference-rankings at any fixed stage of the algorithm.

We will here limit ourselves to only subgroup-consistent aggregation rules. One easy way to make any of the three aggregation rules above satisfy this property is to have groups stipulate their preferences at the start of the matching, and hold these preferences constant throughout. Despite the discussion of possible aggregation rules above, we want to give agency to the groups to aggregate preferences however they like. The problem is that it would be difficult, for practical purposes, to feed a matching algorithm input from agents while it is running. In this case, however, allowing each group to pick its own aggregation rule works perfectly well as long as we only allow groups to report their preferences once and use this ranking throughout the algorithm. We call this initial group preference ranking over all firms the group’s **master list**.

Sometimes, depending on the group’s offer number, some members of the group are already matched to a firm while others are still proposing. We define an aggregation rule so that groups can aggregate preferences when some members of their group are already matched to firms.

**Definition 15.** An **indicator aggregation rule** ranks firms on a point-based system by the number of agents in his or her group that the firm already employs. For each group member employed, that firm gets one point. The agent then ranks firms in descending order of total points. If two firms have the same number of points, we refer to the master list of the group’s preference ranking.

We call this an indicator aggregation rule because each worker employed by a firm either contributes 1 or 0 to a group’s ranking of the firm: 1 if the worker is in the group, 0 if not. The indicator aggregation rule is nothing more than the effect of group-lexicographic preferences. There is no better way to aggregate preferences when all members of the group most prefer being matched to the largest possible number of members of their group; the indicator aggregation rule simply counts group members. Consider the following
example that illustrates how workers with group-lexicographic aggregate rankings over firms according to an
indicator aggregation rule.

**Example 16.** Assume we have firms $f_1, f_2, f_3$ and one group of workers $g = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8\}$. We pick some way to aggregate the whole group’s preferences at the beginning of the algorithm. Assume at the first round of the algorithm, $w_1, w_2, w_3$ are matched to $f_1$, $w_4$ is matched to $f_2$, and $w_5$ is matched to $f_3$. Then, regardless of $w_6, w_7$ and $w_8$’s individual preferences over firms, each worker now ranks $f_1$ first. $f_2$ and $f_3$ are ranked the same, so we break the tie based on the group’s master list, which we assume has $f_2 \succ f_3$, so now $w_6, w_7$ and $w_8$’s preferences are each:

$$w_6 : f_1 \succ f_2 \succ f_3$$

$$w_7 : f_1 \succ f_2 \succ f_3$$

$$w_8 : f_1 \succ f_2 \succ f_3$$

Observe that any subset of the remaining agents in the group, so any of the sets (including the individual agents): $\{w_6, w_7\}, \{w_7, w_8\}, \{w_6, w_8\}, \{w_6, w_7, w_8\}, \{w_6\}, \{w_7\}, \{w_8\}$ now all define their preferences in the same way. This is the key feature of the indicator aggregation rule that makes it applicable to a wide variety of algorithms. For any group of size $k$, if $i$ agents are already matched to firms, then, regardless of how the remaining $k - i$ agents propose to firms (whether they split into individuals, split in two groups, stay as one group of size $k - i$ etc), every one of the $2^{k-i} - 1$ subsets of those workers (we subtract one because we exclude the empty set) rank all firms in the exact same way. Thus, the rule is subgroup-consistent.

Note that the indicator aggregation rule defines a preference ranking that changes dynamically at each step of the algorithm.

To conclude this section, we want to fully specify how groups aggregate preferences in our model. We allow each group to aggregate group preferences however they like at the start of the matching algorithm. This list is the master list of preferences, and it is static throughout the algorithm. Throughout the matching, we aggregate group preferences first using the indicator aggregation rule, then, to break ties, by using the master list of preferences. This is subgroup-consistent. One possible extension to our model is to study how the aggregation rules used above, Borda count, popular dictatorship, and random dictatorship, affect the outcome of matchings.
Remark 17. In the following sections, we often use the terms “group” and “subgroup” in ways that may be confusing. Our market is one in which workers initially come in groups, but may be proposing to firms in subgroups of that original group. The important feature to note is that workers may be split up into smaller subgroups of members of their group, but they still remember who was in their original group. A worker’s desire to be with other members of his or her group only ever refers to a worker’s original group \( g_i \). Workers do not have preferences over other members of their group, and this is true regardless of whether or not a group has been fragmented. Essentially, workers do not give preference to workers in their current subgroup over workers in their original group.

3.5. Solution Concepts. We want to use the exact definition of stability given in Definition 7: the weak core. Recall the definition of a blocking coalition defined earlier:

Definition. For \( F' \subseteq F, W' \subseteq W \), and a matching \( \mu' \), we say the triple \( (W', F', \mu') \) form a blocking coalition to a matching \( \mu \) if the following are true:

1. At least one agent is involved.
2. For all \( f \in F' \) and \( w \in W' \), \( \mu'(f) \in 2^{W'} \) and \( \mu'(w) \in F' \cup 2^W \); the agents in the blocking coalition can be matched under \( \mu' \) even if no other agents are involved.
3. All \( c \in F' \cup W' \) are weakly better off under \( \mu' \) than \( \mu \).
4. At least one \( c \in F' \cup W' \) is strictly better off under \( \mu' \) than \( \mu \).

Because we are dealing with agents who have preferences over both sides of the market, we want a stable outcome to be one in which there does not exist a blocking coalition. Note that this definition relies on agents being able to compare matchings, but, as we described above, our preference restrictions allow us to do that. From favorite-lexicography, we can derive firms’ preferences over matchings by simply ranking matchings by the most preferred worker in each. From group-lexicography, we can define workers’ preferences over matchings by order matching first by the number of group members that worker is matched to in each, then by the workers’ individual preferences over firms.

We want to explain intuitively what it would mean for there to exist a blocking coalition given our preference structures of the agents. Because workers always want to be with the most members of their group as possible, a group that gets split up will always prefer any matching that matches them all together to one firm to any matching that splits them up. Therefore, in order for a blocking pair not to exist, any group that gets split up cannot have a worker that any firm with the capacity for the whole group prefers
to their most preferred worker at the end of a matching, or the firm and the group would leave to form a blocking pair. This is very strong, so it is difficult to find a stable matching.

We should also consider to what extent our models are strategy-proof. Because of the application of matching in groups to one-sided problems, such as house allocation or drawing for dormitory rooms at a university, we care more about strategy-proofness for workers than firms (because our model can potentially be extended to one-sided problems, and we are primarily concerned with the groups). In this paper, we focus more on stability than strategy-proofness, but this also offers an idea for a possible extension to the work here.

We want to assess whether or not our algorithms terminate given any input of preferences. This is relevant in Section 5 when we propose a greedy algorithm.

**Definition 18.** We say an algorithm **terminates** in a finite number of steps if there does not exist any set of agents with any profile of preferences such that the algorithm will never output a solution.

We also want a solution concept that tells us whether or not our algorithm keeps groups together. This is not necessary because the agents’ preferences themselves take into account a desire to be together, so stability alone measures this. However, we do define a concept that we can use to assess whether or not agents are matched well in groups.

**Definition 19.** A **lonely** worker is a worker matched without any other members of their group. A matching algorithm satisfies the **loneliness-proof** condition if there are no lonely workers.

There are a couple ways any algorithm could adjust itself to satisfy the loneliness-proof condition. For example:

- We could start the algorithm by assigning two workers from each group to each firm. This is only possible if each firms’ capacity is larger than twice the number of groups, and if each group’s size is larger than twice the number of firms. We could confirm these matchings and then run the algorithm in which each firm has reduced capacity and each group has reduced numbers.
- We could allow workers to trade their spots after the matching if they are lonely. This opens up a major issue with strategy-proofness for the firms because firms can adjust their preference lists in order to get lonely workers who they can then trade for more favorable workers.
- We could allow firms to have flexible capacities, as in the unsplittable stable marriage problem in [12]. In this case, at the end of the algorithm, any worker who is alone can leave that firm to be matched to any firm employing other group members. However, this opens up the possibility that if
a worker leaves an open spot at a firm, there could be other workers from other groups who would rather move to that firm, and the firm might prefer them to existing employees.

In our analysis in Section 5.2, we define some numerical criteria to measure the extent to which groups are matched closely together, and we also measure how often a matching satisfies the loneliness-proof condition.

4. Simple Examples for Unsplittable Groups

In this section, we offer some very preliminary ideas about how to match groups that refuse to be separated. We already know that if our groups refuse to be separated, there exist markets with no stable outcomes, as shown in Example 3.9 of [32]. As Klaus and Klijn demonstrate, couples unwilling to be split up are cause for a stable matching not to exist, and the same is true for groups. However, the easiest way to match groups of agents is to assume that they refuse to be split up because then we can treat this problem as a standard many-to-one matching problem. We will discuss how to find a stable matching if it does exist. We here assume that groups would rather be unmatched than split, so the offer number of each group is the size of the group.

The first algorithm is very simple generalization of the deferred acceptance algorithm, and it applies only to a very specific class of preferences. Assume that we have defined the preferences of group using an aggregation rule. We run the group-proposing deferred acceptance algorithm in which each group applies as a unit, and the firms can either accept, hold, or reject the whole group. At each step of the algorithm, groups make an “aggregate” proposal to their desired firm. In order for a stable matching to exist, or indeed, for any matching to exist, firms must have the capacity to hold whole groups. If this holds and worker preferences are group-lexicographic, then the group-proposing deferred acceptance algorithm leads to a stable outcome. If firms cannot hold whole groups, then all whole groups remain unmatched. In [32], this outcome is what blocks the matching. However, if groups do refuse to be split, then being unmatched together is preferable to being matched apart, and thus this outcome is stable.

Another way to find a stable matching when workers have group-lexicographic preferences is to have a firm-proposing deferred acceptance algorithm in which firms propose to whole groups. If a firm has the capacity for multiple groups, then that firm can issue proposals to several groups at once. Firms can continue to propose until all groups are matched or all firms are full. If a firm is allowed to employ its favorite worker, and the firm has space for all the rest in the group, then the favorite worker can effectively “bring the group” with him or her. This is exactly the same as the college admissions problem discussed in [5] if we treat whole
groups as individual students. However, we do not focus on firm-proposing algorithms because we want our outcomes to be optimal for workers.

5. Greedy Group Algorithm

In this section, we frame our problem as a fractional assignment problem in which groups are each one unit, and we try to come as close to an integer matching as possible. The algorithm we define will not be stable, but a greedy algorithm has numerical benefits. As such, we will run simulations to show how well the algorithm here does. Furthermore, we hypothesize that a greedy algorithm works best if firms’ capacities are approximately equal to group sizes, and we run simulations to test this hypothesis.

Our work in this section is largely based off the row-greedy algorithm of Baiou and Balinski. Baiou and Balinski [12] describe the stable allocation, also known as the ordinal transportation problem, in which one set is allocated to the other set and allows for fractional matchings. They define a strongly polynomial algorithm that proves the existence of stable allocations. Their problem is a constrained problem in which each machine can only process a certain amount of each job.

We consider each $g_i \in G$ to be a unit that can be divided into $l$ parts, where each part is a worker in that group. We assume that groups have uniform size $l$ and that firms have uniform quota $q$. In our case, because the goal is to match groups in as close to integral assignments as possible, we consider an unconstrained problems in which the maximum number of workers from group $g_i$ that can be matched to firm $f_i$ is equal to $\min(q, l)$ (so the whole group can be assigned to the same firm, and a firm can have only members of one group). We assume that groups do not require being matched together, and the whole group has the same preferences.

The major departure that our model takes from the row-greedy algorithm in [12] is that even though the workers aggregate preferences consistently as a group, the firms do not have homogeneous preferences over workers in a group. In the stable allocation problem, machines have preferences over one whole job, not over individual units of a job. Therefore, it makes a difference whether or not all members of a group apply to a firm and the firm rejects the ones it does not have the capacity for, or only $q$ members of a group apply to a firm. In Baiou and Balinski’s row-greedy algorithm, they stipulate that if a machine cannot accept a whole job, then $q$ units (or the capacity of the machine) of a job apply to their most preferred machine, and the rest apply to their second most preferred machine. In their problem, it does not matter whether the job picks which $q$ units apply or the whole job applies and the machine takes only $q$ of them because all units
are homogeneous. In our problem, firms do not view the members of a group as homogeneous, so we need to define the following concepts.

**Definition 20.** We have a group-ordering algorithm when group members put themselves in order and apply to firms in that order. If a firm can only hold half the members of an unmatched group, then the group will choose which members apply to the firm. We have a firm-ordering algorithm when all unmatched group members apply to the same firm, and the firm picks which ones to keep. If a firm can only hold half the members of an unmatched group, then the whole group applies to the firm, and the firm chooses which members to take.

We are going to consider firm-ordering algorithms, as all group members rank firms the same way, so groups being able to order themselves does not benefit them overall (although it could benefit individual workers). On the other hand, firms rank group members differently, so they benefit by being able to see all available workers and choose their favorites.

5.1. **Greedy Group Algorithm.** We define an algorithm, the Greedy Group Algorithm. Groups issue aggregate proposals all at the same time, which means we do not proceed through the groups in any specified order.

- **Step 1:** For each group, greedily assign that group to their most preferred firm. At the end, if no firm is employing more than its capacity, then the resulting matching is stable because no worker would like to switch since every worker was matched to his or her most preferred firm. However, if a firm $f_i$ is over-capacitated by $c_i$, so they are currently assigned $q + c_i$ workers, then that firm keeps its $q$ most-preferred workers and rejects the rest.

- **Step k:** Now impose constraints on how many workers a group will have issue proposals to firm $f_i$. Assume that at step $k-1$, firm $f_j$ only accepted $\pi_{k-1}(i,j)$ workers from group $g_i$. At step $k$, only those $\pi_k(i,j)$ workers from $g_i$ will be allowed to apply to firm $f_j$, and the rest will apply to the next preferred firm. Workers who have been rejected from their least preferred firms are left unmatched. The algorithm terminates when all workers are either accepted or unmatched.

We give an example to explain how the algorithm works.

**Example 21.** Assume we have groups $g_1 = \{w_1, w_2, w_3\}$, $g_2 = \{w_4, w_5, w_6\}$, $g_3 = \{w_7, w_8\}$ and firms $f_1, f_2, f_3$ each with capacity 2. Assume the following preferences (having chosen some initial aggregation rule):
\[ g_1 : f_1 \succ f_2 \succ f_3 \]
\[ g_2 : f_2 \succ f_1 \succ f_3 \]
\[ g_3 : f_3 \succ f_2 \succ f_1 \]
\[ f_1 : w_3 \succ w_2 \succ w_1 \succ w_4 \succ w_6 \succ w_8 \succ w_7 \succ w_5 \]
\[ f_2 : w_4 \succ w_6 \succ w_7 \succ w_1 \succ w_5 \succ w_2 \succ w_3 \succ w_4 \]
\[ f_3 : w_7 \succ w_2 \succ w_8 \succ w_4 \succ w_6 \succ w_1 \succ w_2 \succ w_5 \]

**Step 1:**

- All of \( g_1 \) applies to \( f_1 \).
- All of \( g_2 \) applies to \( f_3 \).
- All of \( g_3 \) applies to \( f_3 \).
- \( f_1 \) takes \( w_2, w_3 \) and rejects \( w_1 \).
- \( f_2 \) takes \( w_4, w_6 \) and rejects \( w_5 \).
- \( f_3 \) takes \( w_7, w_8 \).

**Step 2:**

- \( w_2, w_3 \) apply to \( f_1 \).
- \( w_4, w_6 \) apply to \( f_2 \).
- \( w_7, w_8 \) apply to \( f_3 \).
- \( w_1 \) now applies to \( f_2 \), and \( w_5 \) now applies to \( f_1 \).
- \( f_1 \) takes \( w_3, w_2 \) and rejects \( w_5 \).
- \( f_2 \) takes \( w_4, w_6 \) and rejects \( w_5 \).
- \( f_3 \) takes \( w_7 \) and \( w_8 \).

**Step 3:**

- \( w_2, w_3 \) apply to \( f_1 \).
- \( w_4, w_6 \) apply to \( f_2 \).
- \( w_7, w_8 \) apply to \( f_3 \).
- \( w_1 \) applies to \( f_3 \) because it was rejected by its group’s second most-preferred firm.
- \( w_5 \) also applies to \( f_3 \).
• $f_1$ takes $w_3, w_2$.
• $f_2$ takes $w_4, w_6$.
• $f_3$ takes $w_7$ and $w_1$ and rejects $w_8$ and $w_5$.
• $w_5$ is left unmatched because he applied and was rejected by his least preferred firm.

Step 4:
• $w_2, w_3$ apply to $f_1$.
• $w_4, w_6$ apply to $f_2$.
• $w_7, w_1$ apply to $f_3, w_1$.
• $w_8$ applies to $f_2$. $f_1$ takes $w_3, w_2$.
• $f_2$ takes $w_4, w_6$ and rejects $w_8$.
• $w_8$ has now been rejected by his least preferred firm, so the algorithm is complete, and the final matching is:

$$
\begin{align*}
&f_1 : w_2, w_3 \\
&f_2 : w_4, w_6 \\
&f_3 : w_1, w_7 \\
\text{unmatched} : w_1, w_8
\end{align*}
$$

5.2. Analysis of the Greedy Group Algorithm. We have one theoretical result about the Greedy Group Algorithm, but we want to mostly focus on numerics here. The general idea is that applying in batches increases the odds of a group being matched together, rather than guarantees that the matching will be stable.

First, however, we discuss termination. Note a key distinction between this model and the row-greedy algorithm of Baiou and Balinski: in Baiou and Balinski, the amount of hours that each job submits to each machine decreases monotonically at every step of the algorithm because machines have preferences over jobs, not particular hours of jobs. However, in our case, at step 1 of the algorithm, if $|g_i| = \frac{3}{4}q$, then $q$ workers from $g_i$ are greedily allocated to their most preferred firm $f_p$ and $\frac{q}{2}$ are greedily allocated to the second most preferred firm $f_r$. If $f_p$ takes $\frac{3}{4}q$ of the workers rejects $\frac{q}{4}$ workers, and $f_r$ takes $\frac{q}{2}$ of them and rejects none, then at step 2, $\frac{q}{2}$ workers will apply to $f_p$ and $\frac{q}{4}$ workers will apply to $f_r$. Thus, it requires more work to show that this algorithm terminates because more workers can apply to a firm (in this case, $f_r$) at a later step. That is why we need Lemma 22.
Lemma 22. The Greedy Group Algorithm terminates in a finite number of steps.

Proof. Any firm over-capacitated at step $k - 1$ will not be over-capacitated by the same agents at step $k$ because the number of agents allowed to apply is fewer than the number who applied at the previous step if the firm had rejected any of those agents. Therefore, for each group $g_i$, at each step in which any member of $g_i$ is rejected from one firm $f_j$, at least one worker from $g_i$ will never again apply to $f_j$, so workers can not go on forever applying to firms from which they have been rejected. In [12], the algorithm terminates in at most $|I|d(J)$ steps, where $|I|$ is the total number of row players and $d(J)$ is the sum of the demands of all of the column players. In our algorithm, because groups can increase the number of workers who apply to some firm, but must decrease in at least 1 firm each step, convergence will instead be quadratic in the capacities of the firms, not linear. The important property for termination is that even though a firm can reject members of a group and still later receive proposals from members of that group, the individual workers will never apply to a firm from which they have been rejected. □

We will see numerically that the Greedy Group Algorithm is not stable. Rather than give up on this algorithm because it is unstable, we want to show that it does a better job of keeping agents in groups than if agents each had their own preference lists and proposed independently.

We wrote a program to simulate the firm-ordering version of the greedy group algorithm. The simulation takes user inputs on:

- $m$: the number of firms.
- $n$: the number of workers.
- $l$: the number of groups.
- $t$: the number of times to simulate the outcome.

The program randomly orders each worker and firm’s preference ranking over the other side, randomly places workers into groups, and gives each firm a random capacity based on the number of firms and workers. We hypothesized that this algorithm is particularly effective when the number of spots in firms is approximately equal to the number of workers, so if $n/m$ is the average number of workers/firm, then we want each firm’s capacity to be between $.8n/m$ and $1.2n/m$ (rounded to the nearest integer). The program will tell us what each firm’s capacity is.

We have several numerical criteria to test the percentage of groups that stay somewhat close together. We will compare how well a greedy algorithm does in keeping groups together relative to simply randomly sorting the workers into firms. To measure how well groups stay together, we want to see how often a large
number of group members are all matched together, and how few firms the group members are spread among. The specific criteria that we have to measure closeness within groups are the following:

- **L**: Lonely: the number of lonely workers, so workers matched with no one else in their group.

- **L/n**: the percentage of lonely workers. We want this percentage to be low because over group-lexicographic preferences.

- **M**: Majority: the number of groups with a majority of group members all in one firm. For each group that has over half of its members all matched to the same firm, we increase $M$ by 1.

- **M/l**: the percentage of groups with a majority of group members all in one firm. We want this percentage to be high because it means that more groups have a large number of their members close together.

- **S**: Spread: the number of groups that have their workers spread among fewer than half the firms. For each group, we count the total number of firms employing any member of the group, and if this number is less than half the total number of firms, we increase $S$ by 1.

- **S/l**: the percentage of groups that have their workers spread among fewer than half the firms. We want this percentage to be high because it means that more groups are gathered closely together.

- **FGP**: Firms-Group Pairings: the total number of firm-group pairings. For example, if one group is spread among two firms and a second group is spread among the same two firms, that contributes 4 to the $FGP$ count. Note that this metric studies the matching as a whole.

- **FGP/l**: the average number of firms that each group is spread among. We want this percentage to be low because it is higher if the groups are spread further apart, which is not something they want.

- **T**: Termination: the number of steps until the algorithm terminates. We want this number to be low. We count this only for the Greedy Group Algorithm, and will not be compared to random.

- **SO**: A boolean to indicate whether or not an outcome is stable. We use the stability concept of the weak core as defined in Definition 7.

We only report percentages in the output because these are most informative of how well the algorithm does. Note also that for each run of the algorithm, we divide each criterion by $t$, so the percentage across all runs of the simulation.

After running simulations, we obtained the following information. In all the tables, we set $t = 1000$, so we run the algorithm 1000 times. In each table, we vary the size and compare criteria $L/n$, $M/l$, $S/l$, and $FGP/l$ between the greedy algorithm and the random algorithm when all groups use a Borda Count aggregation rule.
Table I: $m = 5$, $n = 20$, $f = 4$, $A = BC$, so we have 5 groups, 20 workers, 4 firms.

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L/n$</td>
<td>40%</td>
<td>50%</td>
</tr>
<tr>
<td>$M/l$</td>
<td>40%</td>
<td>20%</td>
</tr>
<tr>
<td>$S/l$</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>$FGP/l$</td>
<td>2.4</td>
<td>2.8</td>
</tr>
</tbody>
</table>

Table II: $m = 10$, $n = 50$, $f = 12$, $A = BC$, so we have 10 groups, 50 workers, 12 firms.

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L/n$</td>
<td>33%</td>
<td>60%</td>
</tr>
<tr>
<td>$M/l$</td>
<td>10%</td>
<td>0%</td>
</tr>
<tr>
<td>$S/l$</td>
<td>70%</td>
<td>90%</td>
</tr>
<tr>
<td>$FGP/l$</td>
<td>3</td>
<td>3.7</td>
</tr>
</tbody>
</table>

Table III: $m = 25$, $n = 200$, $f = 22$, $A = BC$, so we have 25 groups, 200 workers, 22 firms

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L/n$</td>
<td>44.5%</td>
<td>74%</td>
</tr>
<tr>
<td>$M/l$</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>$S/l$</td>
<td>100%</td>
<td>95.4%</td>
</tr>
<tr>
<td>$FGP/l$</td>
<td>6.09</td>
<td>7.9</td>
</tr>
</tbody>
</table>

For additional information about the algorithms used to simulate the matching, please ask me for more details or see Appendix C.

From Tables I, II, and III, we see that the Greedy Group Algorithm does a much better job of keeping workers in the same group close together. Only in Table II does the random algorithm outperform the greedy algorithm in anyway. Here, we see that $S/l$, the percentage of groups with a low spread, is higher for the random algorithm. The only explanation for this is that, as we see in Table III, $S/l$ rapidly increases in the size of the matching, and our simulation may have caught the random algorithm’s $S/l$ percentage increasing faster than the greedy algorithm’s. We also see that larger matchings generally do a worse job of keeping groups intact by some of our metrics, but this is reasonable. In Table III, the percentage of groups who had more than half their workers matched together was 0% for both greedy and random, and this is probably because as the number of firms increases, it is unlikely that any one of them will employ more than half the members of a group. For the same reason, the number of firm-group pairings also increases in the size of
the matching. However, as the size of the matching increases, the number of firms with a low spread also increases rapidly, due to the fact that it is harder for a group to spread to all the firms when there are more firms.

6. Same Size Algorithm

We want to create an iterative deferred acceptance algorithm in which groups allow themselves to be split into smaller groups if they are unmatched at a certain step. We use the offer number defined in Definition 12 to give groups agency in specifying how divided they can be. We assume that all workers within a group have the same preferences. This is a strong restriction, and we try to lift it later, but we also show that even if we cannot find a matching in the core when workers within a group have unique preferences, there is only one type of blocking coalition possible (this borrows from the definition of almost stable given in [1]). We will ultimately show that there exists a stable outcome in the case of all groups of the same size, all firms of the same capacity, and all workers in a group having the same preference list. However, if we lift any one of these three restrictions, we can find examples of blocking coalitions.

Definition 23. In the following sections, we say a matching is almost stable if there exists only one particular type of blocking coalition, and, when we use this term, we will specify to which exact blocking coalition we refer. A matching is almost stable if all counter-examples to stability have the exact same form.

We simplify the analysis and assume that all groups have the same size as each other, so $|g_i| = p$ for all $i$, and all firms have the same capacity as each other, so $\text{cap}(f_i) = k$ for all $k$. For this reason, we call our algorithm the Same Size Algorithm. Recall from the problem statement that we have $n$ workers, $l$ groups, and $m$ firms. We let $ON_{ij}$ denote the offer number at Round $i$ Step $j$. In this case, the offer number will be the same for all proposing subgroups. We assume that $c_{ij}$ is the remaining capacity of firm $f$ at the start of step $i$.

Let’s first assume that firms can hold more than 1 whole group, so $(i - 1)p \leq k \leq ip$ for some $i$. This $i$ represents the number of slots for whole groups that each firm contains. We will see that after the first round of matching, this assumption does not matter. There are two cases to consider in defining the algorithm:

Case 1. $p \leq (i - 1)m$, so there are fewer groups than slots in firms (or the same number).

Case 2. $p > (i - 1)m$, so there are more groups than slots in firms.
In Case 1, we will run the deferred acceptance algorithm with whole groups proposing to firms. In this case, because all groups rank all firms and all firms rank all groups, everyone will be matched as a unit, and the matching will be stable because the deferred acceptance algorithm is stable. This only requires one step.

We now do the more interesting Case 2. Each round in the algorithm consists of two steps, and the second step has a variable number of sub-steps. All rounds take the same form, but the offer number changes.

**ROUND 1**

**STEP 1:**

In the first step, we run the deferred acceptance algorithm with whole groups proposing to firms. Here, the offer number of every group is $ON_1 = p$, the size of the group. Not all groups will be matched. We make several observations about the state of the matching at the end of the first step of Round 1:

1. All firms employ $i-1$ groups because the size of firms is fixed, and if a firm employs fewer than $i-1$ groups, then that firm has at least $l$ spots left, which means, because firms rank all groups, that it would have accepted an offer from a full group that proposed to it. Therefore, the remaining capacity of all firms $f$ is $c_{2f} < p$ (because the end of the first step is the start of the second step, from which we measure remaining capacity). This inequality is strict, or else firms would have space for one more group. Note that this is why the $i$ that we pick does not matter; we mod it out at the end of the first round. Note also that if we have $k < p$, so firms do not have space for a full group, we would just skip the Step 1 in the algorithm and move straight to Step 2.

2. All groups that are unmatched propose to every firm because groups rank all firms, so all unmatched groups cannot all be matched together at any later step of the algorithm.

3. All matched groups have achieved their best possible match because if they preferred a different firm, then they proposed to that firm, and the firm did not accept their offer because it filled its spots with more preferred workers.

4. No firm will ever want to later give up the workers that proposed to it this round because all unmatched groups did propose to it and were rejected, and all matched groups proposed to a firm they prefer, and these groups will continue to propose to this firm. Therefore, all matches are effectively fixed. This relies on a further restriction we articulate later in Remark 26.

**STEP 2:**

So now, at the start of Step 2, we have some completely unmatched groups of size $p$, and all $m$ firms have remaining capacity $c_{2f}$. Note that the remaining capacity of every firm is the same because they are all employing some multiple of $p$ workers, so each firm has remaining capacity $c_{2f} = k - ip$ for some $i$. This
\( i \) must be the same across all firms because \( k - ip \leq p \), equivalent to \( c_{2f} = k \mod p \), (or the firm could either employ another group or the firm is over-capacitated), and there is only one such \( i \) in this range. Thus, \( c_{2f} \) is constant across all firms. We will use this same logic again at later steps, and the fact that \( c_{2f} \) is constant is part of why a stable matching is easier to find in the case of same sizes. The number of remaining unmatched groups is \( l - (i - 1)m \) because each of \( m \) firms employ \( i - 1 \) groups, and we have \( l \) groups total.

We set the offer number for all unmatched groups to \( ON_{12} = c_{2f} \), and we keep it constant throughout Step 2.

**STEP 2\(_1\):**

We run the deferred acceptance algorithm again with whole groups proposing. Each of the whole groups that were matched in Step 1 propose directly to the firm to which they were matched, and this firm will accept them (see Observations 3 and 4 above for the optimality of this match). All unmatched group propose to firms in the same order as they did in Step 1 because their aggregation list has not changed since no members of their group are matched to a firm. Now, we run the deferred acceptance algorithm again between firms and unmatched groups as we did in Step 1. However, this time, at the end of that iteration, firms keep their favorite \( c_{2f} \) workers from the previously unmatched group and reject the rest. Note that because of favorite-lexicographic preferences, firms’ preferences over whole groups are equivalent to their preferences over the top \( c_{2f} \) workers. As we will see later, this protects firms from being matched to less preferred members of a group because firms are able to view all members of each group.

**STEP 2\(_2\):** We run the deferred acceptance algorithm again with only those workers not matched in Step 1 or Step 2\(_1\). Groups unmatched in Step 1 now have \( l - c_{2f} \) workers proposing (if none of their \( c_{2f} \) workers were accepted in Step 2\(_1\), then all firms are filled, and the match is over, so we can assume \( c_{2f} \) workers got matched).

**STEP 2\(_i\):**

We continue to run the deferred acceptance algorithm with firms still in the matching, and have all groups unmatched in Step 1 and at earlier iterations of Step 2 propose, and firms take their top \( c_{2f} \) workers. We continue this until either all firms are filled or there are fewer than \( c_{2f} \) unmatched workers in the groups that were unmatched in Step 1. If all firms are filled, then the matching is over. We remove capacitated firms from the algorithm.

This is the end of Step 2. Note that Step 2 consists of several rounds of the deferred acceptance algorithm. At each sub-step, only one subgroup from each group of unmatched workers could be matched to a new firm.
(this is why we divided it into sub-steps). At every sub-step, all workers matched in previous sub-steps apply straight to the firm that already accepted them.

**Remark 24.** It is important that we do not remove capacitated firms from the matching during Step 2. Consider, for example, $c_{2,j}$ workers from $g_1$ matched to their most preferred firm $f_1$ at Step 2, and $c_{2,j}$ from $g_2$ matched to $f_2$, the second most preferred firm of $g_1$. Assume $f_2$ prefers the remaining workers in $g_1$ to $g_2$. At Step 2, the remaining workers from $g_1$ will apply to $f_2$ with offer number $c_{2,j}$, and $f_2$ will prefer these $c_{2,j}$ to its $c_{2,j}$ workers from $g_2$. Thus, $f_2$ will not want its match with $g_2$ confirmed. However, we can remove firms at the end of the step because all firms will have seen all unmatched workers.

If not all firms are filled, then, for each group, either the whole group was matched to a firm in Step 1, or a subgroup of size $p - jc_{2,j}$ for the first $j$ such that $p - jc_{2,j} < c_{2,j}$ is still unmatched. Note that $j$ is just the number of multiples that fit into the group size. For example, if groups are of size $p = 17$, and all firms have a remaining capacity of $c_{2,j} = 3$, then groups in Step 1 will be matched in $j = 5$ subgroups of size 3, and there will remain 1 unmatched subgroup of size 2. We call this subgroup of a smaller size the **remainder subgroup.** We make several observations:

1. The unmatched workers are those in a remainder subgroup, and they cannot be matched to workers in the same group as them. This is true because the only firms that still have enough space did not match with any workers in Step 2, so they only matched with workers matched in Step 1. The workers matched in Step 1 do not have any group members in remainder subgroups because they were matched as a whole group, so none of their workers were matched in Step 2.

2. All firms now have capacity 0 or capacity $c_{2,j}$. We remove firms with capacity 0 from the matching, and now have $c_{3,j} = c_{2,j}$.

**ROUND 2**

At the start of Round 2 of the algorithm, we have $l - (i - 1)m$ unmatched subgroups of size $p \mod c_{3,j}$. This is only true if not all firms are filled, because a number of workers larger than $c_{2,j}$ could still be left unmatched if all firms are filled. However, if all firms are filled, then the matching is over. These subgroups all have the same preference lists as in Step 1 because none of the firms to which they are applying employ any members of their own group (because those firms were saturated and thus removed from the matching).

**STEP 1:**

We set each unmatched subgroup’s offer number to :
\[ (6.1) \quad ON_{2i} = p \mod c_{3j} \]

also equivalent to \( ON_{1i} \mod ON_{12} \), so the whole unmatched subgroup is proposing together. We run deferred acceptance as in Step 1 of Round 1.

**STEP 2:**
For all subgroups unmatched in Step 1, either all firms are full, or there are only \( ON_{1i} \mod ON_{2i} \) member of the group left. We set:

\[ (6.2) \quad ON_{22} = ON_{12} \mod ON_{21} \]

and repeat Step 2 from Round 1. We confirm all matchings at the end, and remove saturated firms.

**ROUND \( k \)**

**STEP 1:**
We set each unmatched subgroup’s offer number to:

\[ (6.3) \quad ON_{k1} = ON_{k-11} \mod ON_{k-12} \]

We run deferred acceptance as in Step 1 of Round 1.

**STEP 2:**
For all subgroups unmatched in Step 1, we set:

\[ (6.4) \quad ON_{k2} = ON_{k1} \mod ON_{k-12} \]

and repeat Step 2 from Round 1. We confirm all matchings at the end and remove saturated firms.

We continue until either all firms are full, all workers are matched, or we reach some \( k \) such that \( ON_{k_1} = 0 \) or \( ON_{k_2} = 0 \), at which point the algorithm is over. Note that we can stop the algorithm in the middle of a round in between Steps 1 and 2 because one of these three conditions might be met in the middle.
At each two-step iteration, subgroups are either matched as a whole subgroup to a firm that does not employ any other members of their group, or subgroups are unmatched but all firms that did employ members of their subgroup before are now filled and removed from the matching. Because workers do not care about other workers outside their group, all 2-Step rounds of this algorithm have the exact same form.

We want to give an example of how this algorithm runs.

**Example 25.** Assume we have groups \( g_1 = \{w_1, w_2, w_3, w_4\} \), \( g_2 = \{w_5, w_6, w_7, w_8\} \), \( g_3 = \{w_9, w_{10}, w_{11}, w_{12}\} \), \( g_4 = \{w_{13}, w_{14}, w_{15}, w_{16}\} \), and \( g_5 = \{w_{17}, w_{18}, w_{19}, w_{20}\} \) and firms \( f_1, f_2, f_3 \) each with capacity 6. Assume the following preferences:

\[
\begin{align*}
g_1 : & f_1 \succ f_2 \succ f_3 \\
g_2 : & f_2 \succ f_1 \succ f_3 \\
g_3 : & f_3 \succ f_2 \succ f_1 \\
g_4 : & f_3 \succ f_1 \succ f_2 \\
g_5 : & f_1 \succ f_3 \succ f_2 \\
f_1 : & w_3 \succ w_{15} \succ w_{17} \succ w_{13} \succ w_2 \succ w_{11} \succ w_{14} \succ w_{19} \succ w_{18} \succ w_9 \succ w_1 \succ w_4 \succ w_{10} \succ w_{16} \succ w_{20} \succ w_6 \succ w_8 \\
f_2 : & w_{14} \succ w_{20} \succ w_4 \succ w_{17} \succ w_{15} \succ w_6 \succ w_7 \succ w_{16} \succ w_{10} \succ w_{18} \succ w_1 \succ w_{19} \succ w_{11} \succ w_5 \succ w_2 \succ w_3 \succ w_4 \\
f_3 : & w_7 \succ w_{17} \succ w_{16} \succ w_2 \succ w_9 \succ w_{10} \succ w_{13} \succ w_8 \succ w_4 \succ w_{18} \succ w_6 \succ w_1 \succ w_{20} \succ w_2 \succ w_{19} \succ w_{11} \succ w_{12} \
\end{align*}
\]

**Round 1:**

**Step 1:**

- On the first round of deferred acceptance, \( g_1 \) and \( g_5 \) propose to \( f_1 \), \( g_2 \) proposes to \( f_2 \), \( g_3 \) and \( g_4 \) propose to \( f_3 \). \( f_1 \) prefers \( w_3 \) to any of the workers in \( g_5 \), so \( f_1 \) accepts \( g_1 \) and rejects \( g_5 \). \( f_3 \) prefers to any of the workers in \( g_3 \), so \( f_3 \) accepts \( g_4 \) and rejects \( g_3 \).
- On the second round of deferred acceptance, \( g_5 \) proposes to \( f_3 \) and \( g_3 \) proposes to \( f_2 \). \( f_2 \) prefers \( w_6 \) to any workers in \( g_3 \), so \( f_2 \) keeps \( g_2 \) and rejects \( g_3 \). \( f_3 \) prefers \( w_{17} \) to any of the workers in \( g_4 \), so \( f_3 \) accepts \( g_5 \) and rejects \( g_4 \).
- On the third round of deferred acceptance, \( g_4 \) and \( g_3 \) propose to \( f_1 \). \( f_1 \) prefers \( w_3 \) to any workers in \( g_3 \) and \( g_4 \), so \( f_1 \) keeps \( g_1 \) and rejects \( g_3 \) and \( g_4 \). \( g_3 \) has been rejected by all firms, so \( g_3 \) is unmatched.
On the fourth round of deferred acceptance, $g_4$ proposes to $f_2$. $f_2$ prefers $w_{14}$ to any worker in $g_2$, so $f_2$ keeps $g_4$ and rejects $g_2$.

On the fifth round of deferred acceptance, $g_2$ proposes to $f_1$. $f_1$ prefers $w_3$ to any worker in $g_2$, so $f_1$ keeps $g_1$ and rejects $g_2$. $g_2$ then proposes to $f_3$. $f_3$ prefers $w_7$ to any worker in $g_5$, so $f_3$ rejects $g_5$ and accepts $g_2$.

On the sixth round, $g_5$ proposes to $f_2$. $f_2$ prefers $g_4$ to $g_5$, so $f_2$ keeps $g_4$ and rejects $g_5$. $g_5$ has now been rejected from all firms, so the deferred acceptance algorithm is over.

At the end of Step 1, the matchings are:

\[
\begin{align*}
    f_1 & : g_1 \\
    f_2 & : g_4 \\
    f_3 & : g_2 
\end{align*}
\]

**Step 2:**

- **Step 2**: Now, $g_1$, $g_2$, and $g_4$ propose straight to the firms that accepted them. All firms have remaining capacity 2. We run deferred acceptance with $g_3$ and $g_5$. $g_3$ applies to $f_3$, and $g_5$ applies to $f_1$. $f_3$ takes its two favorites from $g_3$, which are $w_9$ and $w_{10}$. $f_1$ takes its two favorites from $g_5$, which is $w_{17}$ and $w_{19}$. Now $f_1$ and $f_3$ are full, so they are removed from the algorithm.

- **Step 2**: We now run the deferred acceptance algorithm again with only $f_2$ and the two sets of workers $\{w_{11}, w_{12}\}$ and $\{w_{18}, w_{20}\}$. They both propose to $f_2$, and $f_2$ accepts $\{w_{18}, w_{20}\}$ because $w_{20}$ is more preferred than $\{w_{11}, w_{12}\}$. Now all firms are full, so the matching is over.

**6.1. Analysis of the Same Size Algorithm.** We want to show that this algorithm is stable, and we start with several observations about the matching $\mu_S$ output by the algorithm. These are similar to the observations made in the previous section, but we restate them here for clarity:

1. Unless a group is matched at Step 1 of Round 1, then all groups propose to all firms at some point in the algorithm. In fact, all groups propose to all firms at every step until they are accepted. This follows from all groups ranking all firms.

2. Workers are always matched in the largest possible subgroup possible. The reason why workers are always matched in largest subgroup possible is that they either apply to fill the firm’s capacity, in
which case the firm is full after they apply, or they are applying in a whole group to firms that do not employ any members of their group, in which case getting matched in any larger-sized group is impossible.

(3) Because agents rank all other sides of the market, and we do not allow a firm to take less than the offer number of a group, all matchings will be individually rational.

These observations are about all we need to prove stability.

Remark 26. One possible problem is that on Step 1, firm \( f \) might take group \( g_1 \) over \( g_2 \) because the most preferred worker in \( g_1 \) not present in \( g_2 \) is more preferred to \( f \). However, at Step 2, \( f \) might prefer the top \( 2c_2j \) workers in \( g_2 \) to the bottom \( 2c_2j \) in \( g_1 \), and might want to swap these workers out. This would also make \( g_1 \) happier. In order to prevent this from being a blocking pair, we need to restrict the matching in some way to say that a group that proposes as a unit at Step 1 (or Step \( 2k + 1 \) until the match finishes because all 2-Step rounds are of the same form, and at the first step in each, firms have enough space for the whole group proposing) and is accepted as a unit in Step 1 requires that they always be matched in that unit. Therefore, maybe a firm prefers some mix of \( g_1 \) and \( g_2 \) to just \( g_1 \), but \( f \) does not have that option because \( g_1 \) does not allow the firm to split it. Therefore, we require that the offer numbers stipulated at any step are fixed at later steps. This is not much of a stretch from our definition of core stability, as workers would also need to consent to block a matching.

We need one proposition before our main result. The analysis later on depends on assuming that we have a blocking coalition, and this is much easier if the coalition contains only one group and one firm. However, we cannot make this reduction until we show that a matching that is not blocked by a one-firm-one-group blocking coalition cannot be blocked by any blocking coalition. This proposition is very dry and technical, but important for later.

**Proposition 27.** Assume firms have favorite-lexicographic preferences and workers have group-lexicographic preferences. If there does not exist a blocking coalition with only one firm and one group, then there does not exist a blocking coalition.

**Proof.** We will prove this by showing the contrapositive: if there is a blocking coalition, then there is a blocking coalition with only one firm and group. We will do this by building a one-firm-one-group blocking coalition from an arbitrary blocking coalition. Recall Definition 6: we say the triple \( (W', F', \mu') \) block a matching \( \mu \) if at least one agent is involved, the agents can be matched under \( \mu' \) even if no other agents are
involved, all agents are weakly better off, and at least one is strictly better off. Assume we have a blocking coalition \((W', F', \mu')\).

Divide the workers in \(W'\) into their groups (or subgroups, if the whole group is not represented), \(\{g_1, \ldots, g_h\}\). Choose some \(g_i\) such that \(g_i\) contains a worker \(w_h\) most preferred to one of the firms in \(F'\), and let this firm be \(f_h\). Now, let \(\{f_{i_1}, \ldots, f_{i_d}\}\) be the set of firms in \(F'\) that employ any workers in \(g_i\). Choose the firm \(f_{i_j}\) that employs \(w_h\), and let \(g_{i_j} = \{w_{i_1}, \ldots, w_{i_s}\}\) be the members of \(g_i\) employed by \(f_{i_j}\). If \(g_{i_j}\) prefers \(f_{i_j}\) to their current matching (call this set of firms \(F^c = \mu(g_{i_j})\)), then \(s\), the number of members of \(g_i\) in \(f_{i_j}\), is larger than the number of members of \(g_i\) employed by any individual member of \(F^c\). Thus, if we match all members of \(g_{i_j}\) to \(f_h\) instead, then \(f_h\) is more preferred by \(g_{i_j}\) because of group-lexicographic preferences.

Now, remember that we chose \(f_h\) such that \(w_h\) was its most preferred worker in the blocking coalition. If \(f_h\) is currently matched to \(W^h = \{w_{h_1}, \ldots, w_{h_r}\}\), then there exists some \(w_{h_k} \in W^h\) preferred by \(f_h\) to all workers in \(\mu(f_h)\). Therefore, if \(w_h\) is the most preferred worker of \(f_h\) in \(W'\), then by transitivity of preferences, \(w_h\) is also more preferred than any of the workers in \(\mu(f_h)\), and then \(f_h\) prefers \(g_{i_j} = \{w_{i_1}, \ldots, w_{i_s}\}\) to its current matching. Thus, the set \(\{f_h, g_{i_j}, u_{h}\}\) such that \(u_{h}(f_h) = g_{i_j}\) forms a one firm-one-group blocking coalition.

Note that it does not matter if \(f_h = f_{i_j}\); we can then just remove the existing firm-group set from the blocking coalition and have them form a blocking coalition on their own.

One slight problem that we glossed over is that it is possible for all agents to be indifferent. For example, \(f_h\) could already employ \(w_h\) in \(\mu\), and then \(f_h\) is indifferent (if all other workers are also the same). However, this follows with some minor tweaking by extracting an agent from \(u'\) who is strictly better off (because for \((W', F', \mu')\) to block in the first place, at least one agent must be better off). We could specifically choose \(w_h\) to be the worker most preferred to a firm \(f_h\) that is strictly better off.

We are now ready to prove our main result.

**Lemma 28.** If all members of a given group have the same preferences, workers’ preferences satisfy group-lexicography, and firms’ preferences satisfy favorite-lexicography, then the Same Size Algorithm is in the weak core as defined in Definition 7.

**Proof.** We prove this by contradiction. Assume \(\exists\) a blocking coalition \(\{\mu, f, W = \{w_1, \ldots, w_r\}\}\) to the matching \(\mu_S\) output by the Same Size Algorithm. From Proposition 27, we can assume all workers are in the same group \(g\) and only one firm is in the blocking coalition. Let \(W_S = \{w_{S_1}, \ldots, w_{S_m}\}\) be \(\mu_S(f)\), the
set of workers to whom \( f \) is matched under our algorithm. Assume \( f \)'s preference ranking over workers is \( w_{f_1} \succ \cdots \succ w_{f_n} \) for all \( n \) workers. Assume \( |f| = k \) for \( k \geq r \). This is necessary because of Condition 2 in Definition 6. The agents in a blocking coalition must be able to form a matching on their own, so \( \{\mu, f, W = \{w_1, \ldots, w_r\}\} \) cannot be a blocking coalition unless \( f \) can employ all \( r \) of the workers. To start, we have two cases depending on which agent is strictly better off:

**Case 1.** \( f \) is strictly better off, all workers in \( W \) are weakly better off.

**Case 2.** Some worker \( w_i \) is strictly better off, all other workers and firm \( f \) are weakly better off.

Note that because we assume all our workers' and firms' preferences are strict, the bulk of the work is done in just one of these cases. We do Case 1 first. Because \( f \) has favorite-lexicographic preferences, then for the first \( i \) such that worker \( w_{f_i} \notin W_S \cap W, w_{f_i} \in W \), so \( W \) contains the more preferred worker. In order for \( w_{f_i} \notin W_S \), then either:

**Case 1.** \( w_{f_i} \) proposed to \( f \) and \( f \) did not accept it.

**Case 2.** \( w_{f_i} \) proposed to \( f \) and \( f \) accepted it but later rejected it.

**Case 3.** \( w_{f_i} \) never proposed to \( f \).

We do Case 1 first. If \( w_{f_i} \) proposed to \( f \) and \( f \) did not accept it, then that means on Step \( j \), the Step in which \( w_{f_i} \) proposed to \( f \), \( f \) received a proposal from a group with a worker ranked higher by \( f \) than \( w_{f_i} \). By Observations 3 and 4 at the start of the section, this group will keep proposing to \( f \) because workers who have been accepted keep proposing to the same firm until the firm no longer holds their offers, which would only happen if \( f \) received a proposal from a group with an even more preferred top worker. In either case, \( f \) cannot prefer \( w_{f_i} \) to all the workers to whom it is matched in \( \mu_S \) if \( f \) rejected \( w_{f_i} \)'s proposal. The logic in Case 2 is exactly the same. If \( f \) rejected \( w_{f_i} \) at any step, then \( f \) received a proposal from a group with a more preferred worker, so again, this cannot happen.

Now we are on to Case 3, in which \( w_{f_i} \) never proposed to \( f \) but \( f \) would be strictly better off if \( w_{f_i} \) has proposed to \( f \), and all workers in \( W \) would be at least weakly better off if matched to \( f \). It is obvious that \( f \) could be strictly better off if \( w_{f_i} \) had proposed (\( w_{f_i} \) could be \( f \)'s favorite worker of all but got matched to a more preferred firm), so we must show that all other workers in \( W \) could not be weakly better off being matched to \( f \).

Note first that if \( w_{f_i} \) never proposed to \( f \), then \( w_{f_i} \) must be matched to some firm \( f_i \) because in the algorithm, workers are only unmatched if they have been rejected by all other firms, so \( w_{f_i} \) would have
proposed to \( f \) if it were unmatched. Let \( W_t = \{w_{t1}, \ldots, w_{tr}\} = \mu_S(f_t) \), the set of workers matched to firm \( f_t \) in \( \mu_S \). Furthermore, for each \( w_j \in W \), let \( f_j = \mu_S(w_j) \), the firm to which they are matched in \( \mu_S \), but assume that some of these firms can be the same one, and let \( F_W \) be the set of all these firms (including \( f_t \)).

The notation is very confusing, so we have included the following chart to help follow who is matched to whom in which algorithm.

<table>
<thead>
<tr>
<th>Agent ( a )</th>
<th>( \mu(a) ), Blocking Coalition Match</th>
<th>( \mu_S(a) ), Same Size Algorithm Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( W = {w_{1}, \ldots, w_{f_t}, \ldots, w_{r}} )</td>
<td>( W_S \not\supseteq w_{f_t} )</td>
</tr>
<tr>
<td>( f_t )</td>
<td>not defined</td>
<td>( W_t \subseteq w_{f_t} )</td>
</tr>
<tr>
<td>( W = {w_{1}, \ldots, w_{r}} )</td>
<td>( f )</td>
<td>( F_W = {f_{1}, \ldots, f_{r}} )</td>
</tr>
<tr>
<td>( w_{f_t} )</td>
<td>( f )</td>
<td>( f_t \in F_W )</td>
</tr>
</tbody>
</table>

To be precise, we defined a block in terms of the whole set \( W \) because that is what the definition of a blocking coalition specifies. However, it is completely possible that every single worker in \( W \) besides \( w_{f_i} \) is better off with \( f \) than their current match and \( f \) ranks them at the bottom of \( f \)'s preference list. Still, even in this case, due to favorite-lexicographic preferences, \( f \) only has to really like \( w_{f_t} \) for all of \( W \) to be a blocking coalition. Therefore, we are going to show that not every worker in \( W \) can be weakly better off by showing only that \( w_{f_t} \) is not weakly better off, which is stronger than what we need to show. We could show that any arbitrary worker is not better off, but \( w_{f_t} \) is the only worker we have any information about. If \( w_{f_t} \) would weakly prefer being matched to \( f \) than \( f_t \), then, because of group-lexicographic preferences, either:

**Case 1.** \( f \) employs more workers from \( g \) than \( f_t \) does (note that here and in the case below we mean workers from \( g \) that are NOT \( w_{f_i} \), because group-lexicography is measured from the perspective of one worker).

**Case 2.** \( f \) employs the same number of workers from \( g \) that \( f_t \) does and group \( g \) (because all workers in the same group have the same preferences, so \( w_{f_i} \)'s preferences are the same as \( g \)'s) ranks \( f \) above \( f_t \).

We can do both at once. Worker \( w_{f_t} \) never proposed to \( f \), so \( w_{f_t} \) must have been matched in the first step of the algorithm. Therefore, \( w_{f_t} \) is matched with all of \( g \), so Case 1 is impossible. Furthermore, \( w_{f_t} \) has the same preferences as all of \( g \), so if \( w_{f_t} \) never proposed to \( f \), than \( w_{f_t} \) was matched to a more preferred firm, so Case 2 is impossible.
Now we do the second half of Lemma 28, Case 2, in which some worker \( w_i \) is strictly better off, all other workers and firm \( f \) are weakly better off. We know already that \( f \) cannot be strictly better off, so in this case, \( f \) must be indifferent between \( W \) and \( \mu_S(f) = W_S \). Because of favorite-lexicographic preferences and the fact that \( f \) has strict preferences over workers, this can only be true if \( W = W_S \). Therefore, because the matching has to be the same, there is no way for any worker to have strict preferences. This completes the proof of Case 2 of Lemma 28, and thus we have proved the whole lemma. □

The lemma above holds only in the case of all groups having uniform preference lists. However, this is a very strong restriction, and we would not expect it to hold. Consider the following blocking coalition that we encounter when agents within a group have different preference lists:

**Example 29.** Assume we have a group \( g = \{w_1, w_2, w_3\} \), firms \( f_1 \) and \( f_2 \) each with capacity 2, and the following preference rankings:

\[
\begin{align*}
  w_1 : f_1 & \succ f_2 \\
  w_2 : f_1 & \succ f_2 \\
  w_3 : f_2 & \succ f_1 \\
  g : f_1 & \succ f_2 \\
  f_1 : w_3 & \succ w_2 \succ w_1 \\
  f_2 : w_1 & \succ w_2 \succ w_3
\end{align*}
\]

- On the first round, because the capacity is already below group size, we skip straight to Step 2. The offer number is 2, and \( g \) proposes to \( f_1 \). \( f_1 \) accepts \( w_3 \) and \( w_2 \), and \( w_1 \) forms the remainder subgroup.
- On the second round, \( w_1 \) proposes to \( f_2 \) because \( f_2 \) is the only firm with remaining capacity. The matching is over.

However, in this case \( w_3 \) prefers \( f_2 \) and \( f_2 \) has a remaining spot, so \( \{w_1, w_3, f_2\} \) forms a blocking coalition. We want to precisely define this type of blocking coalition.

**Definition 30.** We call a blocking coalition of the form in Example 29 an **off-by-one blocking coalition** to indicate that it occurred because the number of group members employed by different firms differed only
by 1. Specifically, an off-by-one blocking coalition is a coalition such that only a single worker \( w \in g \) defects, and this worker defects from a firm \( f \) to a firm \( f_c \) that employs one fewer member of \( g \) then \( f \) does, so if \( w \) defects, then \( f_c \) employs the same number \( f \) did. The reason that this is a block is that from the perspective of an individual group member, the two firms employ the same number of other group members, as each worker does not count himself or herself when comparing across firms.

We can now show that our matching is almost stable in the case of varying preference lists by showing there is only one possible type of blocking coalition. We state the following result.

**Lemma 31.** If workers within a group have different preference lists, workers’ preferences satisfy group-lexicography, and firms’ preferences satisfy favorite-lexicography, then the only possible blocking coalition to the Same Size Algorithm is an off-by-one blocking coalition.

The proof of this is very similar to the proof of Lemma 28, so we skip it here and include it in Appendix D.

### 6.2. Numerics

We will simulate the Same Size Algorithm without the restriction that all preferences be aligned and see how often there exist blocking pairs. We will also compare the Greedy Group Algorithm to the Same Size Algorithm. We use the exact same notation as in Section 5.2, and we aggregate preferences of each group at the start of each matching with a Borda count.

After running simulations, we obtained the following information. Note again that in all the tables, \( t = 1000 \), so we run the algorithm 1000 times. We first compare the stable solution to the greedy solution in terms of the metrics discussed in Section 5.2 to see which one does a better job of keeping groups intact. Interestingly, the non-stable greedy solution meets the criteria for closeness a larger percentage of the time. Table IV is the small case, V is the medium case, and VI is the large case. We aggregate preferences by Borda count.

Table IV: \( m = 5 \), \( n = 20 \), \( f = 4 \), so we have 5 groups, 20 workers, 4 firms.

<table>
<thead>
<tr>
<th>Table IV</th>
<th>Greedy</th>
<th>Same Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L/n )</td>
<td>40%</td>
<td>42%</td>
</tr>
<tr>
<td>( M/l )</td>
<td>40%</td>
<td>30%</td>
</tr>
<tr>
<td>( S/l )</td>
<td>20%</td>
<td>25%</td>
</tr>
<tr>
<td>( FGP/l )</td>
<td>2.4</td>
<td>2.6</td>
</tr>
</tbody>
</table>

Table V: \( m = 10 \), \( n = 50 \), \( f = 12 \), so we have 10 groups, 50 workers, 12 firms.
Table V: Greedy Same Size

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>Same Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L/n$</td>
<td>33%</td>
<td>40%</td>
</tr>
<tr>
<td>$M/l$</td>
<td>10%</td>
<td>5%</td>
</tr>
<tr>
<td>$S/l$</td>
<td>70%</td>
<td>60%</td>
</tr>
<tr>
<td>$FGP/l$</td>
<td>3</td>
<td>3.72</td>
</tr>
</tbody>
</table>

Table VI: $m = 25$, $n = 200$, $f = 22$, so we have 25 groups, 200 workers, 22 firms.

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>Same Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L/n$</td>
<td>44.5%</td>
<td>42%</td>
</tr>
<tr>
<td>$M/l$</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>$S/l$</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>$FGP/l$</td>
<td>6.09</td>
<td>7.6</td>
</tr>
</tbody>
</table>

We now want to simulate the Same Size Algorithm in the case of workers in the same group having distinct preferences. In this case, the algorithm is no longer stable, as shown in Example 29. However, after simulating the matching, we confirm that the only blocking pairs are those defined above in Definition 30. As we see below, the number of unstable outcomes as a percentage grows with the size of the matching. The rows tell us the size of the matching in terms of agents, and the column is the percentage of stable outcomes when using the Same Size Algorithm with varying preferences. Assume we use a Borda count aggregation rule.

Table VII: Percentage of Stable Outcomes

<table>
<thead>
<tr>
<th></th>
<th>Percentage of Stable Outcomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m = 5$, $n = 20$, $f = 5$</td>
<td>90%</td>
</tr>
<tr>
<td>$m = 10$, $n = 50$, $f = 12$</td>
<td>82%</td>
</tr>
<tr>
<td>$m = 15$, $n = 90$, $f = 14$</td>
<td>50%</td>
</tr>
<tr>
<td>$m = 25$, $n = 200$, $f = 22$</td>
<td>32%</td>
</tr>
<tr>
<td>$m = 40$, $n = 400$, $f = 40$</td>
<td>18%</td>
</tr>
</tbody>
</table>

As Table VII suggests, the impact of the off-by-one blocking coalitions is quite strong. However, we would expect the number of stable outcomes to decay rapidly as the size of the matching grows if there is any reason at all that the matching might be unstable. A matching is unstable if only one firm and one worker prefer each other (or prefer to be unmatched, but that is not an issue here since all agents rank the other
side), so as the number of agents grow, it becomes much easier to find two who would like to defect from the matching.

Table VI indicates that the price of stability is high. Indeed, the benefits of a greedy algorithm are that although we do not have theoretical stability properties, our groups stay more intact than they otherwise would. The fact that the unstable solution does a better job of meeting our intuitive numerical criteria of what “intact” means suggests that stability might not be the best property. However, the criteria we define to measure how close our groups stay together only measures utility in terms of what groups want, rather than what firms and groups want. Therefore, from a one-sided worker-only perspective, perhaps a greedy algorithm is preferred. Nonetheless, in order for an outcome to be stable, we also need to make sure firms will not defect from the outcome, and so in a two-sided matching problem, we do not focus solely on what the groups prefer.

7. VARYING SIZES ALGORITHM

We now want to let group sizes and/or firm sizes vary. It does not matter which we pick to vary because the basic problem is that different firms have different capacities on the same step of the algorithm, and this happens no matter which of the sizes vary. So, to construct a maximally general model, we let both vary and assume \(|g_i| = p_i, \text{cap}(f_i) = k_i\).

Assume that all groups have a master list ranking over firms but also assume that they start by applying first to firms that can hold the whole group then in decreasing order of the firms capacities. The general idea is that for each fixed size \(q \leq p_i\), the group \(g_i\) does not allow any firm to take \(q - 1\) workers until they have proposed to all firms requiring them to take at least \(q\) workers. Because of this, some groups might not finish the first full deferred acceptance algorithm until much later rounds.

We want to decrease the offer number by one for each unmatched group at every round. We do not want groups to decrease their offer numbers to the next largest capacity, or in any increment larger than 1, as this creates potential blocks. For example, if a group of size 9 is rejected from a firm \(f_1\) of capacity 10 and then matched to the next largest firm \(f_2\) of capacity 6, maybe \(f_1\) would have accepted 8 of them, and this creates a block. Therefore, we only want the offer number to decrease by 1 at every step.

Another problem is how many subgroups of one group can be proposing at one time, and we believe that the answer is 1. We do not ever want more than one subgroup of the same group issuing new proposals, but we allow one subgroup of a group to issue proposals to a firm that has accepted them, and the rest of the
group to issue new proposals if unmatched. We also need to ask what happens to matched subgroups that are rejected in the middle of the matching while the rest of their group is still proposing.

**Definition 32.** A matched group is **dislodged** if they are rejected by a firm at a later round of the deferred acceptance algorithm than the round in which they are matched. A group is dislodged only if they have been accepted and reapplied successfully to the same firm for multiple rounds, not if their offer is tentatively held and then rejected during a round of the deferred acceptance algorithm.

If a matched group is dislodged, then we remove it from the round and its proposing starts again the next round. Note that no matches are confirmed until the end, so the order that groups propose does not matter. Let \( ON_i \) denote the offer number of group \( i \). The algorithm is the following:

- Let \( k_{max} = \max_i k_i \), the capacity of the largest firm. Groups start applying with

\[
ON_i = \min(p_i, k_{max})
\]

so firms must take the whole group unless no firm is large enough to take the whole group. We run deferred acceptance.

- If a group has applied to all firms with a fixed offer number and none have accepted it, then the group decreases its offer number by 1 and starts applying again from the top of its preference list at the start of the next round.

- If a subgroup of a group is matched to the firm, the remainder subgroup continues to apply down the preference list with the same offer number (or the size of the subgroup, if it is smaller), and then decreases their offer number by one if no firm can accept all of them.

- If a subgroup of a group is matched to a firm but then that firm rejects them at later rounds, so they are dislodged, then that subgroup and all smaller subgroups join back together and continue applying down the list of firms. Note that all larger subgroups stay where they are.

- Continue until no changes are made.

One important point is that for any group \( g \), only one subgroup of \( g \) will be applying to new firms at any point in the algorithm (although multiple subgroups may be applying to firms that have previously accepted them).

One problem we face is what to do with subgroups once one part of a group has been accepted. The following example demonstrates this problem:
**Example 33.** Assume we have groups $g_1 = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7\}$ and $g_2 = \{w_8, w_9, w_{10}, w_{11}, w_{12}\}$, and $g_3$, and firms $f_1$ of capacity 7, $f_2$, and $f_3$. Assume that on some round, 3 members of $g_2$ are matched to $f_2$, five members of $g_1$ are matched to $f_1$, two members of $g_2$ are matched to $f_1$, and two members of $g_1$ are matched to $f_3$. If at some point, $g_3$ dislodges three members of $g_2$ from $f_2$, then the other two members of $g_2$ leave $f_1$ because they are a smaller subgroup. However, then the two members of $g_1$ matched to $f_3$ should try to get matched again to $f_1$, or the matching might be unstable, and we need a way to solve for this.

The reason why this is not a problem in the same size case is that we do not have dislodging as all subgroups matched as Step 1 fill up a firm, and thus the firm gets removed from the algorithm. The above example also demonstrates that matchings are not fixed at the end of a round, as they are in the same size case.

**Definition 34.** We call the type of blocking coalition in Example 33 an **abandonment blocking coalition**, which is defined as a blocking coalition that results from a group wanting to abandon its current firm when workers matched to members of their group are part of a smaller subgroup that is dislodged during the algorithm.

We illustrate the algorithm with the following example. It needs to be somewhat long, or we will not be able to capture what happens to remainder subgroups. We only go through the first five rounds. We will list each group’s offer number at the start of the round. We also let $g_{i,j}$ denote a subgroup of group $g_i$ of size $j$.

**Example 35.** Assume we have the following groups:

\[
g_1 = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7\}
g_2 = \{w_8, w_9, w_{10}, w_{11}, w_{12}, w_{13}, w_{14}, w_{15}, w_{16}\}
g_3 = \{w_{17}, w_{18}, w_{19}, w_{20}\}
g_4 = \{w_{21}, w_{22}, w_{23}\}
g_5 = \{w_{24}, w_{25}, w_{26}, w_{27}, w_{28}\}
\]

and firms $f_1$ with capacity 3, $f_2$ with capacity 4, $f_3$ with capacity 5, and $f_4$ with capacity 10. Assume the following preference lists (note that firms rank all workers, but we do not list the whole rankings here as they are not all necessary):
\[ g_1 : f_2 \succ f_3 \succ f_1 \succ f_4 \]
\[ g_2 : f_3 \succ f_2 \succ f_1 \succ f_4 \]
\[ g_3 : f_4 \succ f_1 \succ f_3 \succ f_2 \]
\[ g_4 : f_4 \succ f_2 \succ f_3 \succ f_1 \]
\[ g_5 : f_1 \succ f_3 \succ f_4 \succ f_2 \]

\[ f_1 : w_{16} \succ w_1 \succ w_{15} \succ w_{14} \succ w_2 \succ w_{12} \succ w_3 \succ w_{10} \succ w_9 \succ w_{28} \succ w_4 \succ w_{27} \succ w_5 \succ w_6 \succ w_{17} \ldots \]
\[ f_2 : w_{14} \succ w_8 \succ w_{15} \succ w_{24} \succ w_1 \succ w_5 \succ w_11 \succ w_{21} \succ w_{22} \succ w_{27} \succ w_{28} \succ w_3 \succ w_4 \succ w_7 \ldots \]
\[ f_3 : w_{15} \succ w_2 \succ w_{28} \succ w_{25} \succ w_4 \succ w_3 \succ w_{18} \succ w_{23} \succ w_{27} \succ w_5 \succ w_6 \succ w_8 \succ w_{11} \succ w_9 \succ w_{10} \ldots \]
\[ f_4 : w_{20} \succ w_{21} \succ w_6 \succ w_9 \succ w_{28} \succ w_7 \succ w_{16} \succ w_{17} \succ w_{19} \succ w_4 \succ w_{26} \succ w_{14} \ldots \]

**Round 1:** \( ON_1 = 7, ON_2 = 9, ON_3 = 4, ON_4 = 3, ON_5 = 5. \)

- \( g_1 \) starts by proposing to \( f_4 \) because \( f_4 \) is the only one large enough to take all of them.
- \( g_2 \) proposes first to \( f_4 \) as well because \( f_4 \) is the only one large enough to take all of them.
- \( g_3 \) proposes to \( f_4 \) first because \( f_4 \) is their favorite.
- \( g_4 \) proposes to \( f_4 \) first because \( f_4 \) is their favorite.
- \( g_5 \) proposes to \( f_3 \) because \( f_3 \) is their favorite of the firms big enough to hold all of them.
- \( f_4 \) tentatively accepts \( g_3 \) because \( w_{20} \) is their favorite worker and rejects \( g_1, g_2, \) and \( g_4 \). \( g_1 \) and \( g_2 \) need to decrease their offer number to propose again, so they wait until the next round.
- \( f_3 \) tentatively accepts \( g_5 \).
- \( g_4 \) applies to \( f_2 \). \( f_2 \) tentatively accepts them.
- Matches at the end of Round 1 are: \( f_2 : g_4, f_3 : g_5, f_4 : g_3 \). Unmatched groups decrease offer number by 1.

**Round 2:** \( ON_1 = 6, ON_2 = 8, ON_3 = 4, ON_4 = 3, ON_5 = 5. \)

- \( g_3, g_4, \) and \( g_5 \) apply to the firms that had already accepted them.
- \( g_1 \) applies to \( f_4 \), as \( f_4 \) is the only firm large enough to hold 6 members of \( g_1 \). \( f_4 \) rejects them and keeps \( g_3 \).
- \( g_2 \) applies to \( f_4 \) also, as \( f_4 \) is the only firm large enough to hold 6 members. \( f_4 \) rejects them and keeps \( g_3 \).
• Matches at the end of Round 2 are: \( f_2 : g_4, f_3 : g_5, f_4 : g_3 \). Unmatched groups decrease offer number by 1.

Round 3: \( ON_1 = 5, ON_2 = 7, ON_3 = 4, ON_4 = 3, ON_5 = 5 \).

• \( g_3, g_4, \) and \( g_5 \) apply to the firms that had already accepted them.
• \( g_1 \) applies to \( f_2 \), as now their offer number is below the capacity of \( f_2 \).
• \( f_2 \) prefers five members of \( g_1 \) to \( g_4 \), so \( f_2 \) takes \( g_1_5 = \{w_1, w_3, w_4, w_5, w_7\} \) and rejects \( g_4 \).
• \( g_4 \) has not yet decreased their offer number, so they can continue applying this round. \( g_4 \) applies to \( f_3 \). \( f_3 \) prefers \( g_5 \), so they reject \( g_4 \).
• \( g_4 \) then applies to \( f_4 \). \( f_4 \) accepts them, as they have capacity for \( g_3 \) and \( g_4 \).
• Matches at the end of Round 3 are: \( f_2 : g_1_5 = \{w_1, w_3, w_4, w_5, w_7\} \), \( f_3 : g_5 \), \( f_4 : g_3, g_4 \).

Round 4: \( ON_1 = 5, ON_2 = 6, ON_3 = 4, ON_4 = 3, ON_5 = 5 \).

• Note that the offer number of \( g_1 \) is still 5 rather than 2 because we index by the largest number in which the whole group can be matched.
• \( g_1_5, g_3, g_4, \) and \( g_5 \) apply to the firms that had already accepted them.
• \( g_1_2 = \{w_2, w_6\} \) now apply to \( f_3 \). \( f_3 \) prefers \( w_2 \) to \( w_{28} \), so they reject \( g_5 \) and accept \( g_1_2 = \{w_2, w_6\} \).
• \( g_5 \) has not yet decreased their offer number, so they can apply again this round. \( g_5 \) then applies to \( f_4 \). Because \( g_5 \) is applying with offer number 5, \( f_4 \) does not have the option to only accept some of them. Because \( f_4 \) prefers \( w_{20} \) and \( w_{21} \) to all workers in \( g_5 \), \( f_4 \) keeps \( g_3 \) and \( g_4 \) and rejects \( g_5 \).
• \( g_5 \) then applies to \( f_2 \). \( f_2 \) prefers \( w_{24} \) to all members of \( g_1_5 = \{w_1, w_3, w_4, w_5, w_7\} \), so \( f_2 \) rejects \( g_1_5 \) and accepts \( g_5 \).
• Now, because the largest subgroup of \( g_1 \) has been rejected, all smaller subgroups rejoin the largest subgroup with the same offer number as that largest subgroup had been previously applying, which, in this case, is 5. Because their offer number has not changed, \( g_1 \) continues applying down the list from the subgroup that was rejected. \( g_1 \) then applies to \( f_3 \) with offer number 5. \( f_3 \) is not matched to anyone because \( g_1_2 = \{w_2, w_6\} \) left to all be matched back together, so \( f_3 \) chooses their five most preferred out of \( g_1 \), which is \( g_1_5 = \{w_2, w_3, w_4, w_5, w_6\} \).
• Matches at the end of Round 4 are: \( f_3 : g_1_5 = \{w_2, w_3, w_4, w_5, w_6\} \), \( f_2 : g_5, f_4 : g_3, g_4 \).

Round 5: \( ON_1 = 5, ON_2 = 5, ON_3 = 4, ON_4 = 3, ON_5 = 5 \).

• \( g_1_5, g_3, g_4, \) and \( g_5 \) apply to the firms that had already accepted them.
• $g_2$ applies to $f_3$ because their offer number is no longer greater than the capacity of $f_3$. $f_3$ most prefers $w_{15}$, so they accept their five most preferred of $g_2$, which is $g_{25} = \{w_8, w_9, w_{10}, w_{11}, w_{15}\}$ and reject $g_{15} = \{w_2, w_3, w_4, w_5, w_6\}$.

• $g_1$ applies to $f_4$ with offer number 5. $f_4$ rejects them. $g_1$ then has to decrease their offer number, so they are done for this round.

• Matches at the end of Round 5 are:

\[
\begin{align*}
   f_3 &: g_{25} = \{w_8, w_9, w_{10}, w_{11}, w_{15}\} \\
   f_2 &: g_5 \\
   f_4 &: g_3, g_4
\end{align*}
\]

7.1. **Algorithmic Analysis.** We start with a few observations about the matching output by the Varying Sizes Algorithm, many of which are similar to those for the Same Size Algorithm.

(1) Workers are always matched in the largest subgroup possible up to swapping two members of the same group or decreasing the number of group members with which other members of their group are matched. However, not all workers are matched in the largest possible subgroup. Consider a group of size 7 which has 5 of its members matched to a firm $f_1$, and the remaining two members matched to another firm $f_2$. It is possible that a subgroup of size 4 including the two workers in $f_2$ could have been all matched together, which would improve the matching for those two workers, but we do not consider this type of block as it could be a worse matching for other members of the group.

(2) All groups that are unmatched propose to every firm because groups rank all firms, so all unmatched groups cannot all be matched together at any later step of the algorithm.

We have the following result about the Varying Sizes Algorithm being almost stable. Again, the proof of this lemma is very similar to that of Lemma 28, so we omit the proof here and refer the reader to Appendix E.

**Lemma 36.** If all members of a given group have the same preferences, workers have group-lexicographic preferences, and firms have favorite-lexicographic preferences, then the only blocking coalitions to the Varying Sizes Algorithm are abandonment blocks, as defined in Definition 34.

48
7.2. Numerics. We will now simulate how often there exists a blocking pair if the sizes vary. We have the algorithm randomly choose a capacity for firms in the range \([0.8(\text{number of workers/number of firms}), 1.2(\text{number of workers/number of firms})]\).

Table VIII below is the percentage of stable outcomes. The rows tell us the size of the matching in terms of agents, and the column is the percentage of stable outcomes when using the same size algorithm with varying preferences. Assume we use a Borda count aggregation rule.

<table>
<thead>
<tr>
<th>Table VIII</th>
<th>Percentage of Stable Outcomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m = 5, n = 20, f = 5)</td>
<td>81%</td>
</tr>
<tr>
<td>(m = 10, n = 50, f = 12)</td>
<td>62%</td>
</tr>
<tr>
<td>(m = 15, n = 90, f = 14)</td>
<td>53%</td>
</tr>
<tr>
<td>(m = 25, n = 200, f = 22)</td>
<td>20%</td>
</tr>
<tr>
<td>(m = 40, n = 400, f = 40)</td>
<td>8%</td>
</tr>
</tbody>
</table>

As we can see, the percentage of stable outcomes decreases rapidly in the size of the problem, demonstrating the relevance of the abandonment block. Despite the fact that we were able to isolate one particular type of block that prevents the algorithm from producing a stable matching, this block occurs with great frequency as the size of the problem grows.

8. Sororities

We now move into an application of the theoretical work done earlier. The inspiration for this paper initially came from the problem of matching groups of friends to sororities. We here discuss how our model could be applied to a sorority matching problem, and show that it improves upon the current sorority matching algorithm.

As mentioned in Example 1 of Section 1, a number of women who try to join a sorority end up ultimately defecting from the matching process and opting out of joining a sorority or only listing one sorority on their preference list. We want to ask why so many women are suiciding from sorority recruitment, and if perhaps a different type of matching system can decrease this number.

The timing of sorority rush suggests that rushees may already have made close friends by the time of the matching. At Stanford University, for example, rush takes place in April of the first year, which means the rushees have had many months to form groups of friends. Close ties may influence which women would like to join which sororities. In fact, Princeton University recently banned rush during freshman year at all (see
which only increases the likelihood that closer friendships will have formed, as women are not allowed to rush until they have already been at the school for a year. Therefore, we can safely assume that girls rushing sororities have preferences over being matched with their group of friends.

Sorority rush happens over the course of several days. It consists of several rounds of rush parties during which rushees meet current sisters. The sorority sisters present about their sororities, and the potential new members introduce themselves. At the end of each day, sisters and rushees will list their preferences over the other side of the market. In a process of mutual selection, desired rushees continue to attend the rush parties of preferred sororities. The algorithm used to match women is the Preferential Bidding System (PBS). PBS formal rush works as follows, as summarized from [41]:

All rushees list acceptable sororities in order, and sororities list acceptable rushees. Each sorority has a uniform quota \( q \) during formal rush; \( q \) is the number of rushees accepting at least one invitation to the first round of invitational parties divided by \( n \), the number of sororities. In alphabetical order, each rushee’s name is read. If she is in the top \( q \) listed of her preferred sorority, she is matched to them. If not, she is laid aside. Each time a girl is matched, she is removed from all other sororities’ preference lists, and their top \( q \) girls changes if she was in the top \( q \). Each time a sorority is matched to a girl, they remove one from the remaining quota, so they would then have space for only \( q - 1 \). The cards laid aside in round one are read again according to the first choice of the rushee, and this continues until a rushee cannot possibly be matched to their most preferred sorority. The same process is repeated for each unmatched rushee’s second choice, and so on. At the end, if a girl is unmatched, the Panhellenic Executive may ask her if she will accept a bid from a previously unlisted sorority. This is the end of formal rush, but afterwards, sororities with available capacity can recruit members informally.

We would like to consider a a set of rushees \( R = \{r_1, \ldots, r_m\} \) and sororities \( S = \{S_1, \ldots, S_n\} \), and divide the set of rushees into \( l \) sets of friend groups, \( G = \{\{r_{1,1}, \ldots, r_{1,k_1}\}, \ldots, \{r_{l,1}, \ldots, r_{l,k_l}\}\} = \{g_1, \ldots, g_l\} \) where \( g_i \) is simply the set \( \{r_{i,1}, \ldots, r_{i,k_i}\} \). We will first look only at formal rush, in which this \( q \) is the same for all participating sororities. Still, groups of friends might be different sizes, and so we apply the Varying Size Algorithm in Section 7. Assume that groups of friends here have all the same properties of groups as defined in 9. We will compare the output of the current system used to match girls to sororities, the Preferential Bidding System with the output of our Varying Size Algorithm.

We assume each rushee in \( g_i \) has the same preference lists (so, effectively, each friend group comes up with a list). Furthermore, sororities do not have information about the friend groups and have preferences only
over individual girls; they do not benefit from having an entire set of friends. One particularly interesting and applicable feature of sorority rush is that in practice, sororities’ preferences actually are favorite-lexicographic. We know from several sources about sorority matching, including [45, 3, 48, 4], that women in a sorority will often choose friends of a preferred rushee to be in the sorority as well. Girls rushing a sorority have reported being approached by existing members of the sorority near the end of rush, and being told by the member that the sorority was very eager to recruit the girl, so they were willing to accept any other girls with whom the desired rushee wanted to be matched. Thus, sororities’ preferences are favorite-lexicographic, as they are willing to choose girls they might not rank highly in order to get their top candidates.

We consider the following example to illustrate what effect the matching has on groups of friends, and to explain why it is important. We first want to analyze the current Preferential Bidding System, and see what effect it has on keeping matchings intact. We already know from [41] that the PBS algorithm is unstable, but here we want to show that it also fails to meet the numerical criteria described in Section 5.2 to keep groups together.

**Example 37.** Consider 8 friend groups

\[ g_1 = \{r_{11}\}, g_2 = \{r_1, r_5, r_{10}\}, g_3 = \{r_9\}, g_4 = \{r_2, r_{12}\}, g_5 = \{r_6, r_7, r_{13}\}, g_6 = \{r_4, r_{14}\}, g_7 = \{r_8, r_{15}\}, g_8 = \{r_3\} \]

with preference rankings over 3 sororities \( S_1, S_2, S_3 \). Note that we have 15 rushees and 3 sororities, so \( q = 5 \).

The preference lists are as follows:

\[
\begin{align*}
g_1 & : S_1 \succ S_2 \succ S_3 \\
g_2 & : S_1 \succ S_3 \succ S_2 \\
g_3 & : S_2 \succ S_3 \succ S_2 \\
g_4 & : S_2 \succ S_1 \succ S_3 \\
g_5 & : S_3 \succ S_2 \succ S_1 \\
g_6 & : S_1 \succ S_3 \succ S_2 \\
g_7 & : S_2 \succ S_1 \succ S_3 \\
g_8 & : S_3 \succ S_1 \succ S_2 \\
S_1 & : r_2 \succ r_4 \succ r_9 \succ r_{11} \succ r_{13} \succ r_1 \succ r_3 \succ r_{10} \succ r_8 \succ r_7 \succ r_5 \succ r_{14} \succ r_{12} \succ r_6 \succ r_{15} \\
S_2 & : r_8 \succ r_9 \succ r_1 \succ r_{10} \succ r_{15} \succ r_{13} \succ r_{12} \succ r_7 \succ r_2 \succ r_3 \succ r_5 \succ r_{11} \succ r_6 \succ r_4 \succ r_{14} \\
S_3 & : r_4 \succ r_9 \succ r_{15} \succ r_{13} \succ r_2 \succ r_{12} \succ r_5 \succ r_1 \succ r_{11} \succ r_8 \succ r_{14} \succ r_3 \succ r_6 \succ r_7 \succ r_{10}
\end{align*}
\]

Using the PBS algorithm, we get the following output:
• Round 1: $r_1$ and $r_{11}$ get matched to $S_1$, $r_8, r_9, r_{12}$, and $r_{15}$ get matched to $S_2$, and $r_{13}$ gets matched to $S_3$.

• Round 2: $r_1$ and $r_{10}$ get matched to $S_1$, and $r_3$ gets matched to $S_3$.

• Round 3: none get matched to their first choice, so the algorithm proceeds to match all rushes to their second choice.

• Round 4: $r_2$ gets matched to $S_1$, $r_7$ gets matched to $S_2$, and $r_5$ gets matched to $S_3$.

• Round 5: $r_6$ gets matched to $S_3$.

Thus, the final sets of matches are:

- $S_1 : r_1, r_2, r_4, r_{10}, r_{11}$
- $S_2 : r_7, r_8, r_9, r_{12}, r_{15}$
- $S_3 : r_3, r_5, r_6, r_{13}, r_{14}$

As we can see, $g_1, g_3,$ and $g_8$ are singleton groups so obviously each one stays together. In $g_2$, $r_1$ and $r_{10}$ stay together but $r_5$ is separated. In $g_4$, both are split up. In $g_5$, $r_6$ and $r_{13}$ are together but $r_7$ is not. In $g_6$, both are split up. In $g_7$, both are together, and this is the only example of a non-singleton group of friends that remains intact. Therefore, even when friends rank preferences the same way and when all parties want to make a match, it is very important to distinguish how important staying together is for a group of friends.

We now simulate the matching using the Varying Size Algorithm from Section 7:

Round 1: $ON_1 = 1, ON_2 = 3, ON_3 = 1, ON_4 = 2, ON_5 = 3, ON_6 = 2, ON_7 = 2, ON_8 = 1$.

  • $g_1, g_2,$ and $g_6$ propose to $S_1$. $S_1$ takes $g_1$ and $g_6$ and rejects $g_2$.
  • $g_3, g_4,$ and $g_7$ propose to $S_2$. $S_2$ can hold all of them.
  • $g_2, g_5,$ and $g_8$ propose to $S_3$. $S_3$ takes $g_5$ and $g_8$ and rejects $g_2$.
  • $g_2$ proposes to $S_2$. $S_2$ prefers its current offers, so it rejects $g_2$. $g_2$ must now lower its offer number.
  • Matches at the end of Round 1 are: $S_1 : g_1, g_6$, $S_2 : g_3, g_4, g_5, g_7$, $S_3 : g_5, g_8$. Unmatched groups decrease offer number by 1.

Round 2: $ON_1 = 1, ON_2 = 2, ON_3 = 1, ON_4 = 2, ON_5 = 3, ON_6 = 2, ON_7 = 2, ON_8 = 1$.

  • All groups matched in Round 1 apply to the same firms.
  • $g_2$ applies to $S_1$ with offer number 2. $S_1$ has space for two more, so it accepts $r_1$ and $r_{10}$.
  • Matches at the end of Round 2 are: $S_1 : g_1, g_6, r_1, r_{10}$, $S_2 : g_3, g_4, g_7$, $S_3 : g_5, g_8$. Now, $g_2$ decreases its offer number to 1, the size of the remainder subgroup.
Round 3: $ON_1 = 1$, $ON_2 = 1$, $ON_3 = 1$, $ON_4 = 2$, $ON_5 = 3$, $ON_6 = 2$, $ON_7 = 2$, $ON_8 = 1$.

- All matched groups apply to firm that has accepted them.
- $r_5$, the only unmatched worker, applies first to $S_1$. $S_1$ does not have space, so the worker then applies to $S_3$. $S_3$ has space, so accepts $r_5$. The matching is complete.

The final output of the algorithm is:

$$S_1 : r_1, r_{10}, r_{11}, r_4, r_{14}$$
$$S_2 : r_9, r_2, r_{12}, r_{18}, r_{15}$$
$$S_3 : r_5, r_6, r_7, r_3, r_{13}$$

Note here, unlike in the matching produced by PBS, only one group, $g_2$ is split up. There is only one lonely worker, as opposed to 2, all groups are matched with majority, as opposed to 6 in the matching made by PBS, and no groups are spread among more than half the sororities. Thus, we can see by example that our algorithm does a better job than existing algorithms to keep groups of sorority girls together. This demonstrates one way to put our model into practice, and it can be further applied to the other examples listed in the introduction.

9. Conclusion

There exists a rich market design literature to answer questions about how to optimize matching and create stable markets. Much work has been done on two-sided matching problems, and many types of variations have been considered. Matching in groups presents a market design challenge to traditional matching mechanisms. Depending on how much the agents care about staying in groups, existing models might be sufficient, but, if not, our work in this paper hopefully helps find an answer to this problem. We attempted to tackle the problem of matching agents who prefer to stay in groups. We presented several simple algorithms to match groups that refuse to be split up. We then described a greedy algorithm, the Greedy Group Algorithm, and demonstrated how well our greedy algorithm performed numerically against a random algorithm. This algorithm is unstable but had appealing numerical properties, particularly from the perspective of only the workers. We produced a stable algorithm, the Same Size Algorithm, in the case of very narrowly defined preferences. We extended this stable algorithm to two different cases in which it was more general yet unstable, but we proved that our algorithms were almost stable because they only had
blocking coalitions of a specific type. We then explored the practical applications of our model by using one of our algorithms, the Varying Size Algorithm, to match women rushing sororities.

Matching agents becomes more and more difficult as agents become more particular about to whom they hope to be matched. Current market design literature provides answers to all sorts of problems when agents define their preferences in different ways. The question that we tried to tackle here is new, but the reason for answering it is the same. Market designers want to build models that can be applied to real-world matching problems, and the goal is to satisfy the preferences of those being matched. We hoped to satisfy the preferences of agents who desire to be matched in groups, but our model has a long way to go before it can meet the needs of a wide variety of agents. Hopefully, we can extend our work to a broader class of preferences and use this model to continue to better match agents in groups.

References


[31] Immorlica, Nicole, Li Li, Vahab Mirrokni, and Andres Schulz. “Coordination Mechanisms for Selfish Scheduling.”


[34] Kojima, Fuhito and Yuichi Kamada. “Stability and Strategy-Proofness for Matching with Constraints: A Problem in the Japanese Medical Match and Its Solution.” The American Economic Review. 2011. https://60816cba-a-62cb3a1a-s-sites.googlegroups.com/site/fuhitokojimaeconomics/Fuhito-Kojima/pandp5pages15-fk.pdf?attachauth=ANoY7crkvBfKpqQ9dQ1gs5XwEzLh9XHXsrbsh2S9fIzh7xy7KeP...20mACHTeQsTA4AS2PaFtcfln4O5JT1nNH60Ym-zh0cPHzjWARP4PilhcPerg7bh0OpudngEDcNmmUpL0y7U22kYxtmHJ11PFOpwoUVPdMvKooy8Kx5q9tCieahBtCID09oFNEND9vz98ED7UOLbQpBxda1teSLzhb5D7DgJtbAU7TSxLbKGfcw83_BJoq&attredirects=1.


Appendix A. Glossary of New Terms

We include a list of all the terms that we defined for the first time in the paper.

- Abandonment Blocking Coalition: a blocking coalition that results from a group wanting to abandon their current firm when workers matched to members of their group are part of a smaller subgroup that is dislodged.
- Dislodged: a matched group is dislodged if they are rejected by a firm at a later round of the deferred acceptance algorithm than the round in which they are matched.
- Favorite-Lexicographic: If $f$’s preferences over workers are $w_{f_1} \succ w_{f_2} \succ \ldots \succ w_{f_n}$, then $f$’s preferences over groups are $g_1 \succ g_2$ if and only if for the first $i$ such that $w_{f_i} \notin g_1 \cap g_2$, $w_{f_i} \in g_1$. 

57
• Firm-Ordering: algorithm in which all unmatched group members apply to the same firm, and the firm picks which ones to keep.
• Group: a set of agents who all have chosen to be in the set.
• Group-Lexicographic: worker $w_i$ in a group has group-lexicographic preferences if $w_i$ would always prefer to be matched with more agents in their group, regardless of the firm.
• Group-Ordering: algorithm in which group members put themselves in order and apply to firms in that order.
• Indicator Aggregation Rule: ranks firms on a point-based system by the number of agents in his or her group that the firm already employs. For each group member employed, that firm gets one point. The agent then ranks firms in descending order of total points.
• Loneliness-Proof: a lonely worker is a worker matched without any other members of their group. A matching algorithm satisfies the loneliness-proof condition if there are no lonely workers.
• Off-by-One Blocking Coalition: a blocking coalition such that only a single worker $w \in g$ defects, and this worker defects from a firm $f$ to a firm $f_c$ that employs one fewer member of $g$ then $f$ does, so if $w$ defects, then $f_c$ employs the same number $f$ did.
• Offer Number: the minimum number of workers that a firm must accept from the group.
• Remainder Subgroup: subgroup of smaller size left over after part of a group has been matched.
• Subgroup-Consistent: any two subgroups from the same original group have the same preference-rankings at any fixed stage of the algorithm.

APPENDIX B. SEQUENTIAL GROUP ALGORITHM

The following is a very straightforward generalization of the Sequential Couples Algorithm given in [35] to match couples together. We use the same language as in the original paper, with hospitals and doctors. The algorithm in [35] is used to prove that the probability that there exists a stable matching in a regular sequence of random markets converges to one as the number of hospitals approaches infinity. Our analysis does not relate to regular markets or probabilistic outcomes, but we want to demonstrate that it is possible to generalize existing results for matching with couples to matching in groups. This is a generalization of Step 2 in Appendix A.3 in [35]. We let our set of friends be $F$ and our set of singletons be $B$, and our hospitals be $H$.

(1) Initialization:
Let $\mu$ b the output of the deferred acceptance algorithm in the sub-market without groups.
(2) Iterate through groups: Set $F^0 = F, i = 0$ and $B = \emptyset$.

(a) If $F^i$ is empty, then go to Step 3. Otherwise, pick some group $f = (f_1, \ldots, f_n) \in F^i$. Let $F^{i+1} = F^i \setminus f$ and increment $i$ by one.

(b) Let group $f$ apply to their most preferred set of hospitals $(h_1, \ldots, h_n)$ that has not yet rejected them.

i. If such a hospital set does not exist, modify matching $\mu$ such that group $f$ is unassigned and then go to Step 2a.

ii. If such a hospital set exists, then if any of the hospitals have been previously applied to by a member of a group different from $f$, then terminate the algorithm.

iii. Otherwise, let

A. If $h_1 = h_2 = \cdots = h_n \neq \emptyset$ and $\{f_1, \ldots, f_n\} \subseteq Ch_{h_1}(\mu(h_1) \cup f)$, then modify $\mu$ by assigning $\{f_1, \ldots, f_n\}$ to hospital $h_1$ and having $h_1$ reject $(\mu(h_1) \cup f_1 \cup \cdots \cup f_n) \setminus Ch_{h_1}(\mu(h_1) \cup f)$. Add the rejected single doctors to $B$ and go to Step 2a.

B. (In contrast to the original algorithm, this step involves $2^{n-1}$ possible cases, one for each way to place $\neq$ between $n$ options.) If $h_i \neq h_j \neq \emptyset$ for some $i, j$ and all members of the group are still in the choice set of their desired hospital, then modify $\mu$ by assigning $\{f_1, \ldots, f_n\}$ to their desired hospitals, having the hospitals reject the excess doctors. Add the rejected single doctors to $B$ and go to Step 2a.

C. Otherwise, let hospitals $(h_1, \ldots, h_n)$ reject the applications by $f$ and go to Step 2b.

(3) Iterate through rejected singletons: set $B^1 = B$ and $j = 1$.

Round $j$:

(a) If $B^j$ is empty, then terminate the algorithm.

(b) Otherwise, pick some single doctor $s \in B^j$. Let $B^{j+1} = B^j \setminus s$ and increment $j$ by one.

*Iterate through the rank order lists of singletons.*

i. If singleton $s$ has applied to every acceptable hospital, then modify matching $\mu$ such that $s$ is unassigned and go to Step 3a.

ii. If not, then let $h'$ be hospital more preferred to singleton $s$ among those to which $s$ has not yet applied.

iii. If there is no group member who has ever applied to $h'$, then there are three cases.

A. If hospital $h'$ has a vacancy and $s$ is acceptable to $h'$, then matching them and go to Step 3a.
B. If either hospital $h'$ prefers all of its current groups to $s$ and there is no vacant position or $s$ is unacceptable to $h'$, then $h'$ rejects $s$ and go to Step 3(b)i.

C. If hospital $h'$ prefers $s$ to one of its current group members and there is no vacant position, then match $s$ to $h'$, and have $h'$ reject the least preferred doctor currently assigned there. Put this doctor into the set of rejected singletons and go to Step 3(b)i.

iv. If there is a group member who has ever applied to $h'$, then terminate the algorithm.

We again say that the algorithm “succeeds” if it terminates at Step 3a. In [35], this algorithm is used to prove Lemma 1 by showing that if the algorithm succeeds, then the resulting matching is stable. We do not extend these results here.

Appendix C. Additional Numerical Results

We here show the output of our computer programs in Sections 5.2, 6, and 7. We programmed the computer to tell us what was happening over the course of the matching, so these images offer additional insight into how the matching algorithms run, as well as demonstrate how the user is allowed to change the parameters of the match. We use small input sizes to keep the images of reasonable size.

The output of the greedy group algorithm, allowing the user to also enter the aggregation rule:
The following is the output of the Same Size Algorithm from Section 6.2, assuming we use a Borda count aggregation rule and all workers within a group have the same preference list:
The following image shows the output from one round Varying Sizes Algorithm in Section 7.2, including algorithmic analysis:
We also ran simulations using different aggregation rules that are not relevant to the rest of our paper but are interesting from a numerical perspective, especially if we consider expanding our research to study stability under various aggregation rules. In Section 5.2, after running the Greedy Group Algorithm using only a Borda count aggregation rule, we allow the user to give input on what type of aggregation rule to use. This can be effective in measuring which aggregation rules lead to more stable outcomes. The user has a choice between the aggregation rules discussed in Section 3.4:

(1) RD: Random Dictator—Randomly choosing a dictator, so setting each group’s preferences equal to the preferences of the worker with the lowest index.
(2) PD: Popular Dictator– Choosing as a dictator the most popular member of a group. To assess who is the most popular, we aggregate firms’ preferences via a Borda count, and select the most popular worker in each group from that aggregation.

(3) BC: Borda Count– Using a Borda count to aggregate preferences.

Note that the idea of simulating different aggregation rules is not to obtain any theoretical results, but simply as an example of how the aggregation method effects the outcome.

Table A.I: \( m = 5 \), \( n = 20 \), \( f = 4 \), so we have 5 groups, 20 workers, 4 firms.

<table>
<thead>
<tr>
<th></th>
<th>Random Dictator</th>
<th>Popular Dictator</th>
<th>Borda Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( SO )</td>
<td>48%</td>
<td>45%</td>
<td>52%</td>
</tr>
<tr>
<td>( T )</td>
<td>4</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>

Table A.II: \( m = 10 \), \( n = 50 \), \( f = 12 \), so we have 10 groups, 50 workers, 12 firms.

<table>
<thead>
<tr>
<th></th>
<th>Random Dictator</th>
<th>Popular Dictator</th>
<th>Borda Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( SO )</td>
<td>25%</td>
<td>26%</td>
<td>32%</td>
</tr>
<tr>
<td>( T )</td>
<td>24</td>
<td>32</td>
<td>30</td>
</tr>
</tbody>
</table>

Table A.III: \( m = 25 \), \( n = 200 \), \( f = 22 \), so we have 25 groups, 200 workers, 22 firms.

<table>
<thead>
<tr>
<th></th>
<th>Random Dictator</th>
<th>Popular Dictator</th>
<th>Borda Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( SO )</td>
<td>9%</td>
<td>11%</td>
<td>11%</td>
</tr>
<tr>
<td>( T )</td>
<td>53</td>
<td>65</td>
<td>117</td>
</tr>
</tbody>
</table>

From Tables A.I, A.II, and A.III, we can compare the different aggregation rules. In general, a Borda count seems like a more democratic aggregation rule than a dictatorship, so we expect more workers to be happy with a Borda count aggregation rule, and this is what our simulations suggest. We see that the percentage of stable outcomes with a Borda count is higher for every size. The popular dictatorship and random dictatorship aggregation rules yield roughly the same percentage of stable outcomes. Interestingly, with a random dictatorship, the algorithm terminates much faster. One potential reason for this is that the groups’ preferences are more likely to be completely distinct from one another (rather than if the preferences represented all the workers’ preferences), and thus groups are more likely to apply to available firms at earlier steps. Another reason that a random dictatorship could terminate faster than a popular dictatorship is that workers might apply through their whole list of firms and be rejected by all of them faster, because their preferences are not following that of the more desirable worker. On the other hand, in a popular dictatorship,
the group is more likely to be accepted at earlier stages and not be rejected until later, which increases the number of rounds. However, this explanation does not account for why a Borda count is slower than a popular dictatorship, and these simulations leave open several interesting questions for further research.

**Appendix D. Proof of Lemma 31**

We here prove the result cited at the end of Section 6 regarding the Same Size Algorithm when workers within a group have varying preference lists. Recall Lemma 31:

**Lemma 38.** If workers within a group have different preference lists, workers’ preferences satisfy group-lexicography, and firms’ preferences satisfy favorite-lexicography, then the only possible blocking coalition to the Same Size Algorithm is an off-by-one coalition from Definition 30.

Note again that we can use results in Section 6.1 to conclude that the matching is already individually rational, and this is not something we need to show.

*Proof.* We show this by assuming we have a blocking coalition, and then showing that it must be of this form. This will be very similar to the proof of Lemma 28.

Assume \( \exists \) a blocking coalition \( \{ \mu, f, W = \{w_1, \ldots, w_r\} \} \) to the matching \( \mu_s \) output by the algorithm. From Proposition 27, we can assume all workers are in the same group \( g \) and only one firm is in the blocking coalition. Let \( W_S = \{w_{S_1}, \ldots, w_{S_m}\} \) be \( \mu_S(f) \), the set of workers to whom \( f \) is matched under our algorithm. Assume \( f \)'s preference ranking over workers is \( w_{f_1} \succ \cdots \succ w_{f_n} \) for all \( n \) workers. Again, we have two cases based on which agent is strictly better off:

*Case 1.* \( f \) is strictly better off, all workers in \( W \) are weakly better off.

*Case 2.* Some worker \( w_i \) is strictly better off, all other workers and firm \( f \) are weakly better off.

Note that because we assume all our workers’ and firms’ preferences are strict, the bulk of the work is done in just one of these cases. We do Case 1 first. Because \( f \) has favorite-lexicographic preferences, then for the first \( i \) such that worker \( w_{f_i} \notin W_S \cap W, w_{f_i} \in W \), so \( W \) contains the more preferred worker. In order for \( w_{f_i} \notin W_S \), then either:

*Case 1.* \( w_{f_i} \) proposed to \( f \) and \( f \) did not accept it.

*Case 2.* \( w_{f_i} \) proposed to \( f \) and \( f \) accepted it but later rejected it.

*Case 3.* \( w_{f_i} \) never proposed to \( f \).
Case 1 and 2 are exactly the same as in Lemma 28 because they deal only with firm preferences.

Now we are on to Case 3, in which $w_f_i$ never proposed to $f$ but $f$ would be strictly better off if $w_f_i$ has proposed to $f$, and all workers in $W$ would be at least weakly better off if matched to $f$. Note that this is what happens in Example 29. Again, it is obvious that $f$ could be strictly better off if $w_f_i$ had proposed, so we must show that all other workers in $W$ could not be weakly better off being matched to $f$ or that we have an off-by-one block.

We will show that if all other workers in $W$ are weakly better off matched to $f$, then the block is an off-by-one block. Assume all workers in $W$ are weakly better off. If $w_f_i$ never proposed to $f$, then $w_f_i$ must be matched to some firm $f_t$. We will show that if $w_f_i$ is weakly better off matched to $f$ even though $w_f_i$ never proposed to $f$, then $w_f_i$ is the single worker who defects to form an off-by-one blocking coalition (as in Lemma 28, it’s easy to see that all other workers in $W$ could be better off with $f$). If $w_f_i$ would weakly prefer being matched to $f$ than $f_t$, then, because of group-lexicographic preferences, either:

**Case 1.** $f$ employs more workers from $g$ (excluding $w_f_i$) than $f_t$ does.

**Case 2.** $f$ employs the same number of workers from $g$ (excluding $w_f_i$) that $f_t$ does and $w_f_i$ ranks $f$ above $f_t$.

In Case 1, we have the same analysis as Lemma 28: $w_f_i$ never proposed to $f$, so $w_f_i$ must have been matched in the first step of the algorithm. Therefore, $w_f_i$ is matched with all of $g$, so the case is impossible. Case 2 is where this proof departs from Lemma 28 and we make use of the idea of almost stability. If $w_f_i$ individually prefers $f$ to $f_t$ and $f$ currently employs one fewer member of $g$ than $f_t$ does, then from the perspective of $w_f_i$, switching to $f$ means that $w_f_i$ will have the same number of group members. If $w_f_i$ prefers $f$ individually, then this blocking coalition is an off-by-one blocking coalition. This completes the first half of the lemma.

The second half of this lemma, Case 2 is exactly the same as Lemma 28 because we already know the firm cannot be strictly better off. This completes the proof of Lemma 31. □

**Appendix E. Proof of Lemma 36**

Recall the statement of Lemma 36:

**Lemma 39.** If all members of a given group have the same preferences, workers’ preferences are group-lexicographic, and firms’ preferences are favorite-lexicographic, then the only blocking coalitions to the Varying Sizes Algorithm are abandonment blocks, as defined in Definition 34.
Proof. We prove this by contradiction. Assume \( \exists \) a blocking coalition \( \{ \mu, f, W = \{ w_1, \ldots, w_r \} \} \) to the matching \( \mu_V \) output by the algorithm. From Proposition 27, we can assume all workers are in the same group \( g \) and only one firm is in the blocking coalition. Let \( W_V = \{ w_{V_1}, \ldots, w_{V_n} \} \) be \( \mu_V(f) \), the set of workers to whom \( f \) is matched under our algorithm. Assume \( f \)'s preference ranking over workers is \( w_{f_1} \succ \cdots \succ w_{f_n} \) for all \( n \) workers. Assume \( |f| = k \) for \( k \geq r \). To start, we have two cases based on which agent is strictly better off:

Case 1. \( f \) is strictly better off, all workers in \( W \) are weakly better off.

Case 2. Some worker \( w_i \) is strictly better off, all other workers and firm \( f \) are weakly better off.

We do Case 1 first. Because \( f \) has favorite-lexicographic preferences, then for the first \( i \) such that worker \( w_{f_i} \notin W_V \cap W \), so \( W \) contains the more preferred worker. In order for \( w_{f_i} \notin W_S \), then either:

Case 1. \( w_{f_i} \) proposed to \( f \) and \( f \) did not accept it.

Case 2. \( w_{f_i} \) proposed to \( f \) and \( f \) accepted it but later rejected it.

Case 3. \( w_{f_i} \) never proposed to \( f \).

We do Cases 1 and 2 first. These are the two cases that result in the blocking coalition discussed above. If \( w_{f_i} \) proposed to \( f \) and \( f \) did not accept it, then that means on Step \( j \), the Step in which \( w_{f_i} \) proposed to \( f \), \( f \) received a proposal from a group with a worker ranked higher by \( f \) than \( w_{f_i} \). This is the block demonstrated in Example 33 because \( f \) did not initially want \( w_{f_i} \), but then, when other workers employed by \( f \) left, \( f \) wanted \( w_{f_i} \). Therefore, the blocking coalition is an abandonment block.

Now we are on to Case 3, in which \( w_{f_i} \) never proposed to \( f \) but \( f \) would be strictly better off if \( w_{f_i} \) has proposed to \( f \), and all workers in \( W \) would be at least weakly better off if matched to \( f \). It is obvious that \( f \) could be strictly better off if \( w_{f_i} \) had proposed \( (w_{f_i} \) could be \( f \)'s favorite worker of all but got matched to a more preferred firm), so we must show that all other workers in \( W \) could not be weakly better off being matched to \( f \). Note first that if \( w_{f_i} \) never proposed to \( f \), then \( w_{f_i} \) must be matched to some firm \( f_t \) because in the algorithm, workers are only unmatched if they have been rejected by all other firms, so \( w_{f_i} \) would have proposed to \( f \) if it were unmatched. Let \( W_t = \{ w_{t_1}, \ldots, w_{t_n} \} = \mu_V(f_t) \), the set of workers matched to firm \( f_t \) in \( \mu_V \).

We are going to show that not every worker in \( W \) can be weakly better off by showing only that \( w_{f_i} \) is not weakly better off. If \( w_{f_i} \) would weakly prefer being matched to \( f \) than \( f_t \), then, because of group-lexicographic preferences, either:

67
Case 1. \( f \) employs more workers from \( g \) than \( f_t \) does.

Case 2. \( f \) employs the same number of workers from \( g \) that \( f_t \) does and group \( g \) (because all workers in the same group have the same preferences, so \( w_{f_i} \)’s preferences are the same as \( g \)’s) ranks \( f \) above \( f_t \).

We can do both at once. Worker \( w_{f_i} \) never proposed to \( f \), so \( w_{f_i} \) must have been matched in the first step of the algorithm. Therefore, \( w_{f_i} \) is matched with all of \( g \), so Case 1 is impossible. Furthermore, \( w_{f_i} \) has the same preferences as all of \( g \), so if \( w_{f_i} \) never proposed to \( f \), than \( w_{f_i} \) was matched to a more preferred firm, so Case 2 is impossible.

Now we do the second half of Lemma 36, Case 2, in which some worker \( w_i \) is strictly better off, all other workers and firm \( f \) are weakly better off. We know already that \( f \) cannot be strictly better off, so in this case, \( f \) must be indifferent between \( W \) and \( \mu_V(f) = W_V \). Because of favorite-lexicographic preferences and the fact that \( f \) has strict preferences over workers, this can only be true if \( W = W_V \). Therefore, because the matching has to be the same, there is no way for any worker to have strict preferences. This completes the proof of Case 2 of Lemma 36, and thus we have proved the whole lemma. \( \Box \)