Theoretical Foundations of Team Matchmaking

Josh Alman[*]
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts
jalman@mit.edu

Dylan McKay[†]
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts
dmmckay@mit.edu

ABSTRACT

Online team games need matchmaking systems which can handle a high throughput of players and form fair teams to play matches together. We study this problem from a theoretical perspective, by defining and analyzing a broad model which captures many applications. In the most basic formulation, we desire a data structure which supports adding and removing players, and extracting the best possible games. We design an efficient solution, where with n players in the structure, operations can be performed in $O(\log n)$ time. We then consider a natural extension one might want in a team game setting, where each team has different roles, and each player is only willing to play in some of these roles. We show that this extension is computationally intractable, conditioned on the popular 3SUM conjecture; nevertheless, we are able to design an efficient constant-factor approximate solution. Our results help explain recent practical issues with the matchmaking systems in some of the most popular online games. We also prove similar results in an offline setting, which has many applications to matching and partitioning beyond online games.

Keywords

Cooperative games: theory & analysis; Game theory for practical applications

1. INTRODUCTION

Matchmaking is one of the most important aspects of online games. No matter how well-designed a game is, being matched against an opponent with significantly more or less skill and familiarity with the game can ruin the experience. Meanwhile, the experience may never even begin if matchmaking takes too long and players get bored waiting in queue to play. Hence, matchmaking systems need to be computationally efficient, and rapidly create balanced matches.

The matchmaking problem for two-player games like Chess, Hearthstone, or Street Fighter, has been studied recently [14, 5]. However, many of today's most popular games are multiplayer games in which two teams face each other. League of Legends has over 100 million players who play for over one billion hours each month, Dota 2 has over 10 million players per month, and Counter Strike: Global Offensive has 3 million players who play for over 250 million hours per month. These are three of the five most played online games today, and a match in one of these games is played between two teams which typically consist of five players each [25].

In this paper, we undertake the first theoretical study of *team* matchmaking, to the best of our knowledge. While we focus on applications to online games, our model and results are very general and apply to other natural settings where individual agents are divided into teams, such as designing clinical trials and other experiments, dorm room assignment, and recreational sports.

1.1 Role-restricted queues

We were inspired to study team matchmaking from a theoretical perspective by recent practical issues with the matchmaking systems in some of the most popular online games. In most games, players enter a 'queue' to be put into a match, and the matchmaking system creates pairs of teams to play against each other. However, many of these games have different 'roles' that players might play on their team. For instance, one may be a tank character, while another is a healer, and another is a sniper. It can be problematic if a team is formed where everyone wants to play the same role.

To solve this issue, many games recently moved to a 'role-restricted queue', in which players pick some desired roles when they enter the queue, and teams are formed where each player can play in a desired role. However, these role-restricted queues have been widely unsuccessful in practice. In late 2015, League of Legends made such a switch, which widely angered players, as queue times went up, and game balance went down [22, 23]. Another popular game, Heroes of the Storm, also had much longer queue times when they restricted which characters could be together in a team [2].

Our results help explain this issue. Our main results show that the basic formulation of team matchmaking has a simple and efficient solution, but that the role-restricted queueing problem is computationally intractable (given a popular assumption from complexity theory). Hence, it is *impossible* to implement a role-restricted queue with short queue times which makes fair matches.

1.2 Model

An important ingredient of a matchmaking system is a well-defined measure to determine the best matches to form. Most online games only release vague information about

[*]Supported by NSF DGE-114747.
[†]Supported by NSF CCF-1552651 (CAREER).

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

what measures they optimize. We therefore consider a broad class of *imbalance functions*, which capture the known priorities of the three most popular games mentioned above by an appropriate choice of parameters. In these games [4, 28, 29], a game is informally said to be balanced if the following two properties are satisfied:

- *Fairness:* the two teams have roughly equal chance of winning, and
- *Uniformity:* there is not much deviation between the best and worst players in the game.

Each player has a skill rating which is maintained similar to the ELO rating system [13, 17, 19], and these skill ratings can be used to determine how well a game satisfies these two requirements.

Formally, the matchmaking problem is as follows. Given a set R of n players, a *game* is a pair of disjoint sets X and Y of k players each. The parameter k should be thought of as a constant, for instance $k = 5$.

Each player $r \in R$ has an associated skill level s_r, which is a nonnegative real number. An imbalance function f maps games to nonnegative real numbers, where games mapping closer to zero are more desirable. Given an imbalance function f, a game (X, Y) minimizing f among all those which can be formed with the n players is called a *best game*. As described above, imbalance functions will have a fairness component and a uniformity component.

1.2.1 Fairness

In order to determine whether a game is fair, we need a measure of the skill of a team. A simple choice would be the sum of the skills of the team members,

$$s(X) = \sum_{x \in X} s_x.$$

However, this choice might not be sufficient to capture the team dynamics in many applications. Depending on how much of an impact a highly-skilled player can have on a team, a team with one skilled player and many unskilled players might have a good chance or almost no chance of winning against a team of moderately-skilled players. This same idea is captured by the standard notion of the p-norm of a vector[1], where we can select higher values of p to mean that large entries have a bigger impact on the total norm. Hence, for real parameter $p \geq 1$, we define the p-skill of a team, denoted s_p, to be the p-norm of the team's vector of player skills,

$$s_p(X) := \left(\sum_{x \in X} s_x^p \right)^{1/p},$$

and the p-fairness of a game, denoted d_p, to be the difference of p-skills,

$$\begin{aligned} d_p(X, Y) : &= |s_p(X) - s_p(Y)| \\ &= \left| \left(\sum_{x \in X} s_x^p \right)^{1/p} - \left(\sum_{y \in Y} s_y^p \right)^{1/p} \right|. \end{aligned}$$

[1]Recall that for a vector $v = (v_1, v_2, \ldots, v_n)$, its p-norm, parameterized by a real value $p \geq 1$, is defined as $||v||_p := \left(\sum_{i=1}^n |v_i|^p\right)^{1/p}$. It is sometimes also called the ℓ_p norm. For $p = 1$ it is sometimes called the Manhattan distance, and for $p = 2$ it is also called the Euclidean distance.

The p-fairness of a game is a nonnegative real value, with 0 being the most fair.

When we pick $p = 1$, we get the simple choice from above that the skill of a team is the sum of the skills of the team members. In the limit as $p \to \infty$, we get that s_∞ is simply equal to the skill of the most skilled player on the team,

$$s_\infty(X) = \max_{x \in X} s_x.$$

Choices of p in-between interpolate between these two options.

1.2.2 Uniformity

Even though a game is fair, it might still not be fun for all the players involved. If a game involves players with a wide variety of skill levels, the less skilled players might feel that they are just being strung along by the more skilled players rather than controlling the flow of the game and enjoying themselves. Similarly, an unskilled player in a game of mostly skilled players might feel demoralized, or anger their teammates. As mentioned above, the most popular games have identified this issue, and their matchmaking systems aim to avoid games with a large deviation between the best and worst players in the game.

The natural choice to measure the deviation of a set $Z = X \cup Y$ of player skills is the standard deviation,

$$v(Z) := \sqrt{\frac{1}{|Z|} \sum_{z \in Z} (s_z - \mu_Z)^2},$$

where $\mu_Z := \sum_{z \in Z} s_z / |Z|$ is the mean skill of Z. But again, this might not be tuned well to all applications. In some settings, having some players with much more or less skill can be fine, as long as most players' skill levels are around the mean. In others, a single player whose skill level deviates substantially from the mean can be a big issue. (How important it is to minimize deviation in the first place also varies between applications. This is a separate issue which we discuss soon.)

Just as with fairness, choosing a different norm or mean to take of the vector of deviations from μ_Z captures this distinction well, and it is a standard technique in statistics. We introduce a real parameter $q \geq 1$ to measure this, and define the q-uniformity, denoted v_q, to be the q-norm of the deviations from the mean,

$$v_q(Z) := \left(\frac{1}{|Z|} \sum_{z \in Z} |s_z - \mu|^q \right)^{1/q}.$$

The q-uniformity of a game is, again, a nonnegative real value, with 0 being the most uniform.

When $q = 2$, we recover the standard deviation. But, for other choices of q, this is the qth root of the qth *central moment*[2] from statistics of the set of skill levels. When $q = 1$, v_q is the average deviation from the mean, and so many skill levels must be far from the mean to decrease the uniformity substantially. As $q \to \infty$, we get that v_∞ is just the farthest value from the mean,

$$v_\infty(Z) = \max_{z \in Z} |s_z - \mu|,$$

[2]Actually our definition only agrees with central moments when q is an even integer, since the typical definition uses $s_z - \mu_Z$ rather than $|s_z - \mu_Z|$. Taking absolute values makes more sense in our application, where having skill less than the mean should lead to less uniformity, not more.

and so a single player whose skill level is far from the mean makes the uniformity low on their own. Again, choices of q in-between interpolate between these options.

1.2.3 Imbalance Function

We consider imbalance functions, parameterized by three positive real constants α, p, q with $p, q \geq 1$, of the form

$$f(X, Y) := \alpha \cdot d_p(X, Y) + v_q(X \cup Y),$$

where d_p is the p-fairness, and v_q is the q-uniformity, as defined above. The imbalance of a game is a nonnegative real number, with 0 being the most balanced. The parameter α can be picked depending on how important fairness is relative to uniformity in the particular application. A large α means that fairness is more important, while a small α means that more weight is put on uniformity. Hence, for instance, if uniformity has little importance in one application, a larger choice of α can reflect this.

In the data structures we design, we also allow for the choices $p = \infty$ or $q = \infty$ as discussed earlier. Throughout our results, we use the convention that if $p = \infty$ then $1/p = 0$, and similarly for q.

1.2.4 Queueing Problem

Now that we have defined imbalance functions, we need a model of how a matchmaking system will process tasks. The matchmaking systems in popular online games have large numbers of players constantly entering the matchmaking queue, and so they also need to be making new games at a consistently high rate. We model this problem as a data structure problem, where players can be added to and removed from the data structure as they come and go, and the game host can query for the best new game to make as frequently as they can support it.

A *game queue* with imbalance function f is a data structure which stores a collection of players and supports the following operations:

- Insertion: insert a new player into the queue
- Deletion: remove a player from the queue
- Query: find a best $2k$-player game (X, Y) amongst the players in the queue

Our results apply to reasonable variants on this model as well.

1.2.5 Time-sensitive Queueing Problem

In most applications, we are also interested in making sure no player waits for too long in the queue. A *time-sensitive* game queue additionally stores, for each player r, the time t_r which that player was inserted into the queue. The goal now is not necessarily to pick the best game, but also to ensure no player is waiting for too long.

There are various mechanisms for this which all of our results support. One example is to augment the imbalance function f by defining a new *time-sensitive priority function* g defined by

$$g(X, Y) := f(X, Y) + \beta \cdot \min_{r \in X \cup Y} t_r,$$

where β is an appropriate positive real parameter which dictates how important it is that players do not wait for too long in the matchmaking queue. By finding a game minimizing g rather than minimizing f, we are allowing more leniency in how imbalanced a game is when it includes a player who has been waiting for a long time. It is worth noting that with this definition of g, if a player r has been waiting a long time in the queue, and we make a game involving r as a result of this, we are still making the game of least imbalance that includes r.

1.2.6 Role-Restricted Queueing Problem

An important aspect of many team games, including League of Legends and Dota 2, is that a team will consist of players taking on a variety of different roles. For instance, one may be a tank character, while another is a healer, and another is a sniper. It can be problematic if a team is formed where everyone wants to play the same role. This can be solved by including an option for players to select which roles they are willing to play when they enter the queue, and then only creating games where each player can play a desirable role. We model this via the *role-restricted queueing problem.*

A role-restricted player p has a skill score s_p and also a subset $r_p \subseteq \{1, \ldots, k\}$ of roles they are willing to play. A game $g = (X, Y, m)$ in a role-restricted queue consists of two sets, X and Y, of k players, along with a map $m : X \cup Y \to \{1, \ldots, k\}$ from players to roles such that each team has one player in each role, and each player p is mapped to a role from r_p. All the other notions are the same, including the definition of the *time-sensitive role-restricted queueing problem.*

1.2.7 Offline Partitioning Problem

In applications like dorm room assignment or recreational sports, it is more natural to consider the *offline partitioning problem* where, given n players, we would like to partition them into $n/(2k)$ games. There are many sensible options for what objective to optimize for in this partitioning. For illustration in our results, we choose to minimize the imbalance of the most imbalanced game we create. However, we later discuss other alternative options.

Using data structures for the queueing problem can give suboptimal results for the partitioning problem, since removing the best game might leave the remaining players in a very imbalanced game. We will nonetheless see that we can get similar results for the partitioning problem as for the queueing problem, using almost the same techniques.

1.2.8 Further Extensions

Although we define many variants on the queueing problem, a unifying property that we will see is that the same main ideas work, both to design data structures and to prove lower bounds, in each regime. The extensions we focus on model the matchmaking systems for the applications we discuss, but other extensions to imbalance functions of the queueing problem should also be amenable to our methods; we discuss this further in the future work section.

1.3 Other Applications

Our model and results are very general and apply to many natural settings outside of online games where individual agents are divided into teams, such as:

Designing clinical trials: human participants volunteer and are put in a queue, from which we wish to extract control and experiment groups for trials which are 'balanced' in terms of parameters relevant to the trial (age, health, etc)

Other marketing or scientific experiments: the queueing and partitioning problem can similarly be used to design unbiased experiments in many areas, including partitioning subjects either online or offline in A/B testing

Dorm room assignment: many schools divide students into dorms in 'fair' or evenly distributed ways which could be modeled by our imbalance functions

Recreational sports: players need to be partitioned into balanced teams for games other than online games, like in recreational sports leagues

2. OUR RESULTS

First, we give an efficient data structure for the queueing problem.

THEOREM 1. *For any constant k, there is a data structure for the queueing problem which takes $O(\log n)$ time per insertion, deletion, or query.*

In contrast, we give strong evidence that there is no efficient data structure for the role-restricted queueing problem. We give a lower bound conditioned on the popular 3SUM conjecture from theoretical computer science. Given a multiset S of n integers, the KSUM problem asks to find K elements of S which sum to zero. The 3SUM conjecture states that any algorithm solving this problem with $K = 3$ requires time $n^{2-o(1)}$. This conjecture is widely believed and has been used to prove lower bounds in many areas like computational geometry, graph algorithms, and pattern matching [16, 24, 20]. We will use it to prove lower bounds for team matchmaking problems.

THEOREM 2. *Any data structure for the role-restricted queueing problem must require $n^{1-o(1)}$ time per insertion or $n^{2-o(1)}$ time per query when $k \geq 3$, assuming the 3SUM conjecture.*

Assuming the more general KSUM conjecture, which asserts that the current best known algorithms for KSUM are essentially optimal, we can improve the lower bounds to $n^{k-2-o(1)}$ time per insertion or $n^{k-1-o(1)}$ time per query. On the scale of the most popular online games, even linear time per query is computationally intractable.

With Theorem 2 in mind, we look towards approximations. Since skill ratings are only an approximation of a player's skill in the first place, an approximation algorithm is sufficient for many applications. We are able to construct an efficient data structure which achieves a constant factor approximation for the role-restricted queueing problem. For notational convenience, we define $\rho := 2k^{1/q}(1+\alpha)$. Notice that $\rho > 1$ is a constant defined in terms of the parameters of the imbalance function.

THEOREM 3. *For any constant k, there is a data structure which achieves a ρ-approximation for the role-restricted queueing problem and takes $O(\log n)$ time per operation.*

All of the above results, including the upper bounds, hold for time-sensitive queues as well. We also emphasize that these results hold for *any* choice of the parameters defining the imbalance function.

We are able to prove similar types of results for the offline partitioning problem. We find that the problem is already hard to solve exactly, even without any role-restriction.

THEOREM 4. *The offline partitioning problem requires time $n^{2-o(1)}$ when $k \geq 3$ assuming the 3SUM conjecture.*

We then design a constant-factor approximation algorithm which runs in near-linear time.

THEOREM 5. *For any constant k, there is a $O(n \log n)$ time ρ-approximation algorithm for the offline partitioning problem.*

3. RELATED WORK

Matching problems have been among the most important problems in combinatorial optimization since the seminal work of Edmonds [11, 12]. Matchmaking for two-player games in an online setting was recently studied in [14]. The offline setting for two-player games might be most reasonably modeled as a maximum-weight bipartite matching problem, which can be computed exactly in subcubic time [27] and approximated in linear time [10]. The problem of dynamically maintaining a maximum weight bipartite matching also has lower bounds conditioned on the 3SUM conjecture and other popular conjectures [1].

Much work has focused on devising systems for rating how skilled players are, in order to predict whether a two-player or team game is fair, such as [17, 19]. We largely abstract this problem away in this paper. There has also been substantial research on whether the parameters of the matchmaking systems in particular games are tuned correctly; a sample includes [8, 18, 30].

Our data structures have a fixed-parameter tractability (FPT) flavor to them. The big-O notation in the times per operation hides large, exponential factors in the team size parameter k. In most applications, k is such a small constant that these factors do not preclude our data structures from being practical. FPT algorithms have been designed for numerous other matching, partitioning, and optimization problems; see [6] for a survey.

4. EFFICIENT DATA STRUCTURE FOR MATCHMAKING

In this section, we prove Theorem 1 by designing an efficient game queue data structure. We then discuss the practicality of our data structure.

Restatement of Theorem 1. *There exists a game queue which performs insertions and deletions in time $((1+\alpha)k)^{O(k)} \cdot \log(n)$ and queries in time $O(k)$.*

Our data structure relies on the relationship between the fairness component d_p and the uniformity component v_q which compose the imbalance function. While games with high values of v_q might have high or low values of d_p, games with low values of v_q must also have low values of d_p. Intuitively, if there is not much deviation in the skill levels of a set of players, then it shouldn't be hard to partition them into two teams in a fair way. Lemma 1 formalizes this idea. (The proofs of Lemmas 1 and 2 can be found in Appendix A).

LEMMA 1. *Let S and T be collections of $2k$ players such that*

$$(1+\alpha)(max_S - min_S) < k^{-\frac{1}{q}} \left(\frac{max_T - min_T}{2} \right),$$

where $max_S := \max_{r \in S} s_r$, and similarly for min_S, max_T, and min_T. Then S can be partitioned into two teams to form a game whose imbalance is less than the imbalance of any game formed by a partition of T.

Hence, our data structure can limit its search space to games which are fairly uniform and guarantee that it is looking at some of the best games. Lemma 2 shows exactly how to limit the search space without eliminating the best games from consideration.

LEMMA 2. *Let $P = \{p_1, ..., p_n\}$ be a collection of n players such that for all $i \in [1, n-1]$, $s_{p_i} \leq s_{p_{i+1}}$. Then, there exists a best game (X, Y) over the imbalance function with parameters α, p, and q such that*

$$\max_{p_i \in X \cup Y}(i) - \min_{p_i \in X \cup Y}(i) \leq 4(1+\alpha)k^{1+\frac{1}{q}}.$$

Lemma 2 shows that, for each player p in our queue, if p is in the best game, then this game can be found by searching a small set of games containing p, and any insertion or deletion of a player changes this set of games for only a small number of other players.

Proof of Theorem 1. The main idea is to keep a list $l = [p_1, ..., p_n]$ of all players sorted by skill, and a min-heap containing one game $g_{p_i} = (X_{p_i}, Y_{p_i})$ per player p_i, prioritized by $f(X_{p_i}, Y_{p_i})$. The game g_{p_i} will be the most balanced possible game containing p_i such that if p_j is in g_{p_i}, then $i \leq j \leq i + 4(1+\alpha)k^{1+\frac{1}{q}}$. Hence, by Lemma 2, g_{p_i} is the best game in which p_i is the least skilled player, and since we do this for each player, our min-heap is guaranteed to contain the overall best game. To retrieve a best game, we will simply query the top of the heap.

To insert a player p into the game queue, simply insert p into l and recompute the new optimal game g_{p_i} for the $4(1+\alpha)k^{1+\frac{1}{q}}$ players p_i whose game could have been affected by this insertion. Each of these updates can be done in time

$$O\left(\binom{4(1+\alpha)k^{1+\frac{1}{q}}}{2k} \cdot \binom{2k}{k} \cdot 2k\right) = k^{O(k)}$$

by simply testing all possible games which can be formed by each p_i and the $4(1+\alpha)k^{1+\frac{1}{q}}$ players after p_i in l. This yields a total runtime of $4(1+\alpha)k^{1+\frac{1}{q}}(\log(n) + k^{O(k)}) = ((1+\alpha)k)^{O(k)}\log(n)$.

Player deletions are performed nearly identically and can be carried out in the same time $((1+\alpha)k)^{O(k)}\log(n)$. $\square$

Our data structure as given also works in the time-sensitive setting with the same runtime guarantees.

Notice that while the time bounds given in Theorem 1 grow quickly as a function of k, in practice k tends to be a small constant like 5, and so even the exponential factor in k is fairly insignificant. Furthermore, parts of the game optimization step of insertions and deletions to our data structure have a simple reduction to the PARTITION problem[3], which is NP-hard but known to be efficiently solvable in practice [21], so an implementation without great worst-case guarantees but with good practical results is possible.

[3]The PARTITION problem asks whether a collection of n real numbers can be partitioned into two collections of size $n/2$ with the same sum.

5. HARDNESS OF ROLE-RESTRICTED MATCHMAKING

In this section, we give the main ideas behind Theorem 2, a lower bound against data structures for role-restricted queueing. A key ingredient of our lower bound is a standard assumption about the hardness of 3SUM. However, this assumption is actually stronger than we need to prove intractability; we will prove the following generalization instead.

Restatement of Theorem 2. *Any data structure for the role-restricted queueing problem must require $n^{c-1-o(1)}$ time per insertion or $n^{c-o(1)}$ time per query, assuming the $(2k-2)$SUM problem requires $n^{c-o(1)}$ time to solve.*

The standard assumption is that 3SUM requires $n^{2-o(1)}$ time, but weaker assumptions still prove intractability results. For instance, even the substantially more modest assumption that 8SUM requires $n^{1.5-o(1)}$ time is sufficient to prove that the problem is intractable in our application with $k = 5$. The fastest known algorithm for $(2k-2)$SUM uses a folklore meet-in-the-middle approach and runs in $O(n^{k-1}\log n)$ time. Hence, for example, a data structure for the role-restricted queueing problem for $k = 5$ where each operation takes $O(n^{2.99})$ time would already imply a faster algorithm for 8SUM than is known.

Proof of Theorem 2. $(2k-2)$SUM is known to be equivalent to the following problem: given $2k-2$ multisets $\ell_1, \ldots, \ell_{2k-2}$ of integers, determine whether there is a choice of one from each list which sums to zero [15]. We give a reduction from this to the offline role-restricted queueing problem: given n players, find the best game among them. The reduction will be such that if we can solve the offline role-restricted queueing problem in T time, then we can solve $(2k-2)$SUM in $O(T)$ time. The result then follows from noting that if we can solve the role-restricted queueing problem in P time per insertion and Q time per query, then we can solve the offline role-restricted queueing problem in $O(n \cdot P + Q)$ time.

We first further transform the $(2k-2)$SUM problem. By appropriately translating each list, we can assume that ℓ_i only consists of positive integers when i is even, and only consists of negative integers when i is odd, and that the target sum is still zero. Let $m_1, \ldots, m_{k-1}$ be the first $k-1$ odd primes. For each $1 \leq i \leq k-1$, let b_i be the integer between 0 and $m_1 \cdot m_2 \cdots m_{k-1}$ which is 1 (mod m_i) and 0 (mod m_j) for $j \neq i$. Multiply every element of all of the multisets by $m_1 \cdot m_2 \cdots m_{k-1}$, and then add b_i to every element of ℓ_{2i} for each $1 \leq i \leq k-1$. Finally, create two more lists ℓ_{2k-1} and ℓ_{2k}, where ℓ_{2k-1} consists of only the number $-M$, and ℓ_{2k-1} consists of only the number $M - \sum_{i=1}^{k-1} b_i$, where M is a large value to be determined.

We now create merged lists for each $1 \leq i \leq k$ defined as

$$\ell'_i := \{(a)^{1/p} \mid a \in \ell_{2i}\} \cup \{(-a)^{1/p} \mid a \in \ell_{2i-1}\}.$$

Now we can interpret these lists as an offline role-restricted queueing instance: Each player is only willing to play one role, and list ℓ'_i consists of the skill levels of players who are only willing to play role i. Note that there is a solution (X, Y) with $d_p(X, Y) = 0$ if and only if the original $(2k-2)$SUM was an accept instance. Because of our choice of the b_is, if $d_p(X, Y) = 0$, then one team must consist only of players from ℓ_i for i even, and the other must consist only of players from ℓ_i for i odd.

Finally, we need to pick M large enough so that $v_q(X, Y)$ must be very large for any valid choice of X, Y, and different choices of which players from $\ell'_1, \ldots, \ell'_{k-1}$ to pick for the teams hardly change $v_q(X, Y)$, so that the best team choice must have $d_p(X, Y) = 0$ if possible. It is sufficient to pick M to be polynomially large in the maximum element of $\ell_1, \ldots, \ell_{k-1}$. □

In the hard instance we constructed, each player is only willing to play in one role. We note that a modification of this proof also gives hard instances when, for instance, each player is willing to play in two roles.

The hard instances we design involve situations where the set of players willing to play in one role looks very different from the set of players willing to play in the other roles. We emphasize that situations like this are *not* necessarily unrealistic, and actually occur in many popular online games like League of Legends, where some roles are more popular than others among the players at certain skill levels [26].

6. EFFICIENT APPROXIMATE ROLE-RESTRICTED MATCHMAKING

In the previous section we gave strong evidence that there is no efficient data structure for the role-restricted queueing problem. We now present an efficient data structure for the approximate problem instead.

Restatement of Theorem 3. *There is a data structure which achieves a ρ-approximation for the role-restricted queueing problem and takes time $k^{O(k)} \log n$ for insertions and deletions and $O(k)$ time for queries.*

At a high level, our data structure is similar to that of Theorem 1. We will still maintain a heap of candidate games, with the guarantee that a game within a ρ factor of optimal must appear in our heap. Whereas before we kept a candidate game for each small window players, now we will keep a small collection of candidate games for each player minimizing the difference in skill between the most and least skilled players while ensuring every role can be filled. In order to do this, we give an efficient datastructure solving the problem we call the *dynamic shortest interval with distinct representatives problem*, and then show how to modify this structure in order to solve the approximate role-restricted queueing problem.

6.1 The Shortest Interval Problem

The *shortest interval problem* is as follows: given k lists $l_1, ..., l_k$ of n integers, find a pair (a, b) such that for every list l_i, there exists $x_i \in l_i$ such that $a \leq x_i \leq b$ and $b - a$ is minimal.

As a slight generalization, the *shortest interval containing distinct representatives problem* is as follows: given k lists $l_1, ..., l_k$ of n objects p with associated rank s_p, find a pair (a, b) such that there exists distinct objects $q_1, ..., q_k$ where each object's rank is contained in $[a, b]$, for each i, $q_i \in l_i$, and $b-a$ is minimal. We call such an $[a, b]$ a *shortest interval containing distinct representatives*.

We consider the dynamic version of this second problem. That is, we consider data structures maintaining k lists $l_1, ..., l_k$ with the following operations:

Insertion. The structure supports inserting an object p with rank s into a list l_i.

Deletion. The structure supports deleting an object p from a list l_i containing p.

Query. The structure supports a query for a shortest interval containing distinct representatives.

In order for these structures to be useful in tracking candidate games, we are specifically interested in structures augmented to have the following property which we call the *candidate property.* Let $\prec$ be some ordering of objects by rank with ties broken consistently. Then, the candidate property says that, for each permutation of the lists $l'_1, ..., l'_k$, each object $q_1 \in l'_1$ contains pointers to objects $q_2, ..., q_k$ such that $q_1 \prec q_2 \prec ... \prec q_k$, each $q_i \in l'_i$, and $s_{q_k} - s_{q_1}$ is minimal, unless there exists a $q'_1, ..., q'_{k-1}$ where $q'_i \in l'_i$ such that $q_1 \prec q'_1 \prec q'_2 \prec ... \prec q'_{k-1} \prec q_k$, in which case q_1 will, in place of a pointer to q_k, have a pointer to *void.*

Intuitively, the candidate property says that the structure maintains a collection of k-tuples $(q_1, ..., q_k)$ where each q_i in the tuple is distinct and comes from a distinct list. Each of these tuples represents some interval with distinct representatives, and the specific set of tuples required by the candidate property will dictate that one of these intervals is the smallest interval with distinct representatives. While it is worth noting that there are other properties one might consider which offer this guarantee, the candidate property has the benefit of being simple to maintain in our inductive construction.

Lemma 3. *There is data structure solving the dynamic shortest interval containing distinct representatives problem performing insertions and deletion in time $O((k+1)! \log(n))$ and queries in time $O(1)$ and has the candidate property.*

Our strategy will be to maintain the collections of pointers required by the candidate property and store these collections in a min-heap which prioritizes by $s_{q_k} - s_{q_1}$ where for some permutation of lists $l'_1, ..., l'_k$, q_1 points to $q_2, ..., q_k$ and $q_1 \prec ... \prec q_k$.

Given this, queries are simple. We need only query the top of the min-heap to retrieve a shortest interval containing distinct representatives, which can be done in $O(1)$ time. We need only describe how to maintain the collection of intervals admitting the candidate property upon insertions and deletions in time $O((k + 1)! \log(n))$.

To maintain insertions and deletions in our structure S, we will recursively keep k interval structures $S_1, ..., S_k$ each on a different set of $k - 1$ of our k lists (assume S_i does not contain l_i). Then for each permutation of lists $l'_1, ..., l'_{k-1}$ in S_i and each object $q_1 \in l'_1$ which points to $q_2, ..., q_{k-1}$ for the given permutation, none of which are *void*, we keep a pointer to least $q_k \in l_i$ satisfying $q_1 \prec ... \prec q_k$ or we point to *void* if some q'_1 such that $q_1 \prec q'_1$ could instead point to q_k in l_i for the given permutation of lists.

It is straightforward case-work and induction to show that these new pointers can be maintained in time $k^{O(k)} \log(n)$ upon insertions and deletions. First, observe that by induction, for each permutation of the lists $l'_1, ..., l'_k$ and each $q_1 \in l'_1$, the $q_2, ..., q_k$ pointer to by q_1 are each the least possible choice which q_1 could point to and therefore, for each permutation and list l'_i, each $q_i \in l_i$ can only be pointed to once. From this and simple case work, we can conclude that for each permutation of lists $l'_1, ..., l'_k$, at most one $q_1 \in l'_1$ will change its collection of pointers to other players and at most one $q_1 \in l'_1$ will change any of its pointers to *void.*

We can then conclude that during insertions and deletions, the number of pointers updated is $O(k!k)$, each requiring time at most $O(\log(n))$ to update the heap. This yields the desired runtime of $O((k+1)!\log(n))$.

6.2 Applying Shortest Intervals

We will use a simple modification of the data structure from Lemma 3 to construct our approximation data structure for Theorem 3.

Proof of Theorem 3. To approximate the role-restricted game queue problem for team size k, we keep a dynamic shortest interval containing distinct representatives structure on $2k$ lists $l_1, l'_1, ..., l_k, l'_k$ where the objects in l_i are the players willing to play role r_i and their ranks are their skills. l'_i will be a copy of l_i. Observe now that the collections of pointers admitted by the candidate property each represent a collection of players which could form a role-restricted game, as there must be a distinct representative from l_i and l'_i for each i, and further more one of these collections is a collection M minimizing $c = \max_{p\in M}(s_p) - \min_{p\in M}(s_p)$. Lemma 6 (found in Appendix A) shows that for any game (X, Y), we have $f(X,Y) \geq k^{-\frac{1}{q}}(\frac{c}{2})$ and Lemma 5 (found in Appendix A) shows that there is some role-restricted game (X, Y, m) in M such that $f(X, Y) \leq (1+\alpha)c$. From this, we see there is some game (X, Y, m) which can be formed from M such that $f(X,Y) \leq 2k^{1/q}(1+\alpha)OPT = \rho \cdot OPT$, where OPT is the imbalance of a best role-restricted game. To take advantage of this, we simply store in a min-heap the best game for each collection of pointers. We just showed that at least one of the games has imbalance at most $\rho\cdot OPT$, so the game at the top of the heap will have imbalance at most $\rho\cdot OPT$, showing that queries can be performed in time $O(k)$.

To perform insertions and deletions, we simply perform insertions and deletions into the interval structure and recompute the optimal game for any collection of pointers that changes. This yields a runtime of $O((2k+1)!\log(n) + O((2k+1)!4^k/k)$ which is $k^{O(k)}\log(n)$. □

Like in the unrestricted case, the dependence on k should not be considered overwhelming, since k is a small constant, and the game optimization steps are known to be efficiently solvable, in practice.

7. OFFLINE PARTITIONING PROBLEM

Most applications outside of online games are best modeled as an offline partitioning problem rather than a high-throughput data structure problem. It seems difficult to apply results about the queueing problems to the offline partitioning problem in a black-box way, since finding the absolute best games might not be the best way to partition all of the players. Interestingly, our results in this setting nevertheless follow from almost the same ideas as the results about the queueing problems. Indeed, Theorems 4 and 5 follow very quickly from the ideas in the proofs of Theorems 2 and 3, respectively, so we just sketch the main ideas.

Restatement of Theorem 4. *The offline partitioning problem requires time $n^{c-o(1)}$ assuming the $(2k-2)SUM$ problem requires $n^{c-o(1)}$ time to solve.*

Proof. (Sketch) The property that we can take advantage of in proving a lower bound for the partitioning problem, which did not exist for the queueing problem, is that we actually need to include all players in a game. Suppose we take a $(2k-2)$SUM instance and then turn it into an offline partitionining problem instance by converting each number into a player with that skill. Then, we add in two players with skill M, where M is a very large value we pick later. These two players must be put in the same game, one on each team, or else the games they are in will have too high a value of d_p. Then, since the presence of these players makes v_q so high, this game will be the game of maximum imbalance. Hence, we need to assign the remaining players to minimize d_p. A $(2k-2)$SUM solution would achieve the minimum value $d_p = 0$ if possible. □

Theorem 4 has implications from a parameterized complexity theory perspective which further support that the offline partitioning problem is a difficult problem. Parameterized complexity theory studies the difficulty of problems with respect to parameters which are part of the problem definition or input; see for instance [3] for the relevant notions.

Our proof of Theorem 4 gives a reduction from the $(2k-2)$SUM problem to the offline partitioning problem. But, the KSUM problem is known to be hard for the complexity class $W[1]$ from parameterized complexity theory [9]. Our reduction therefore implies that the offline partitioning problem is also hard for $W[1]$. It is widely believed that if a problem is $W[1]$-hard, then it does not have a fixed-parameter tractable algorithm [7]. To illustrate, this means there is no constant a such that we could find an algorithm running in time $n^a \cdot 2^{k^{O(1)}}$ for the offline partitioning problem; the exponent of n must be growing with k. By comparison, all of the efficient data structure and algorithm running times in this paper are of this form.

An interesting contrast between the offline partitioning problem and the queueing problems is that, in the offline partitioning problem, we already get a hardness result without including role-restrictions, whereas the queueing problem without role-restrictions has an efficient data structure solution. Pesky outlier players in the offline partitioning problem can make the problem much harder, since we need to partition every player into a game. Nonetheless, similar to the online setting, we can design an efficient approximation algorithm for the offline problem.

Restatement of Theorem 5. *For any constant k, there is a $O(n\log n)$ time ρ-approximation algorithm for the offline partitioning problem.*

Proof. (Sketch) We will find a partition which minimizes the maximum value of $v_q(X \cup Y)$ over all formed games (X, Y). The same argument as in the proof of Theorem 1 shows that such a partition will also give a ρ-approximation for the offline partitioning problem. We simply order all the players by skill, then partition the players into intervals of length $2k$ in this order, and divide each interval into two teams X, Y in order to minimize $d_p(X, Y)$. It follows from a convexity argument that this minimizes the maximum value of v_q. The total runtime is only $O(n\log n)$. □

8. FUTURE WORK

8.1 Dependences on k

The runtimes of all our data structures have exponential dependences on the team size parameter k. These factors seem inherent to our approach, which involves solving the NP-hard PARTITION problem on many sets of $2k$ player skill levels. Perhaps we could achieve a better dependence on k using a different approach instead. It would be a big breakthrough to achieve a subexponential dependence on k while maintaining a small dependence on n, since when $k = n/2$, each query to a game queue is solving the PARTITION problem. As discussed earlier, the offline partitioning problem is hard for the class $W[1]$ from parameterized complexity theory, which gives further evidence that a much better dependence on k is unlikely for these team matchmaking problems. Nonetheless, a smaller exponential factor might be possible.

A big area of current research concerns fixed-parameter tractable (FPT) algorithms, in which one seeks to reduce the dependence of different parameters in the runtimes of algorithms. See, for instance, [6] for a survey. Perhaps the techniques from FPT algorithms could be useful here. It is also worth noting, as we discussed earlier, that the PARTITION problem can be solved much more quickly in practice than its exponential runtime suggests.

8.2 Objectives in Offline Partitioning

In the offline partitioning problem, we choose to minimize the imbalance of the most imbalanced game we form. However, there are other natural objectives we could choose to minimize instead. One choice would be the sum of the imbalances of all the games we form. Another would be the maximum *unhappiness* of any player, where unhappiness is defined as the ratio of the imbalance of the game they are assigned, to the imbalance of the best game one could form which includes them. Analogues of Theorems 4 and 5 should hold for these and other similar objectives as well.

8.3 Further Extensions of our Model

Our model was designed to be very general and capture a wide variety of applications. Nonetheless, it would be interesting to see what other additional components of imbalance functions, or additional constraints on valid games, could be included such that the queueing problem can still be efficiently solved. For instance, one might consider a model where a player's skill level changes depending on how many of their teammates they have played with before, or even a model where players are not allowed to be matched with other players they have played with recently.

Our results still hold for many possible extensions with only minor modifications to the proofs, like extending our measure of a player's skill to a multidimensional measure. For other extensions, like adding in constraints about players not being matched together on a team too many times in a row, it may require more work to design efficient data structures. We note that *our lower bounds still hold* for any extension of imbalance functions or our model, as long as the most basic formulation of team matchmaking, against which we proved lower bounds, is a special case of the extension.

One component of matchmaking systems for two-player games that we intentionally do not consider here is network latency between players. In most team-games, unlike in many two-player games, a central server hosts all matches, and players communicate with this server rather than with each other. This means that latency between players is not an issue. High latency to the server might impact a player, but if it does, this impact is typically consistent between games and is already reflected in the player's skill level.

Acknowledgments

This research was conducted while the authors were at Stanford University. The authors thank Amir Abboud, Tim Roughgarden, Greg Valiant, Virginia Vassilevska Williams, and Ryan Williams for heplful conversations.

A. IMBALANCE FUNCTION PROPERTIES

We present several Lemmas about imbalance functions, with the goal of proving Lemmas 1 and 2. We only provide sketches of some more straightforward proofs; full proofs will appear in the full version.

LEMMA 4. *Let S be a multiset of integers of size $2k$, $max = \max(S)$, and $min = \min(S)$. For all p, there exists a partition $X \cup Y = S$ such that $d_p(X, Y) \le max - min$.*

Actually, we prove a generalization of Lemma 4 which we will also be able to use in the role-restricted queue setting.

LEMMA 5. *Let $S_1, \ldots, S_k$ be k multisets of two integers each, and let $max = \max\{S_1 \cup \cdots \cup S_k\}$ and $min = \min\{S_1 \cup \cdots \cup S_k\}$. Then there is a partition $X \cup Y = S_1 \cup \cdots \cup S_k$ such that for each i, one element of S_i goes to X and the other goes to Y, and such that $d_p(X, Y) \le max - min$.*

PROOF. We design our partition greedily. Initially X and Y are empty. For i from 1 up to k, if currently

$$\sum_{x \in X} x^p \le \sum_{y \in Y} y^p, \tag{1}$$

then we add the bigger of the two elements of S_i to X and the smaller to Y, and otherwise we add the bigger to Y and smaller to X. Throughout this process, we will always have

$$\left| \left(\sum_{x \in X} s_x^p \right)^{1/p} - \left(\sum_{y \in Y} s_y^p \right)^{1/p} \right| \le max - min$$

by a straightforward inductive argument, as desired. □

Given a collection of role-restricted players which form a game (X, Y), Lemma 5 gives us an upper bound on $d(X, Y)$ as a function of the highest and lowest skills among the players. A convexity argument can be used to prove the following lemma, which gives us similar upper and lower bounds on $v(X \cup Y)$.

LEMMA 6. *Let S be a set of players of size $2k$, $max = \max_{p \in S}(s_p)$, and $min = \min_{p \in S}(s_p)$. Then for all q,*

$$k^{-\frac{1}{q}} \left(\frac{max - min}{2} \right) \le v_q(S) \le max - min.$$

Combining Lemmas 5 and 6 yields almost immediately Lemma 1 from Section 4. It shows that, when k, α, and q are fixed constants, there exists a constant c such that finding a best game for the queueing problem only requires considering games formed from players contained in windows of c consecutive players sorted by skill. This is formalized by Lemma 2 in Section 4.

Proof of Lemma 2. For any game G consisting of players who come from a window of size greater than $4(1+\alpha)k^{1+\frac{1}{q}}$, we can create at least $2(1+\alpha)k^{\frac{1}{q}}$ different games consisting of disjoint players in intervals of the window. Lemma 1 implies that one of these must be better than G. □

REFERENCES

[1] A. Abboud and V. Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS 2014*, pages 434–443, 2014.

[2] Blizzard Entertainment. The current state of matchmaking. In *Heroes of the Storm Blog, http://us.battle.net/heroes/en/blog/20034041/the-current-state-of-matchmaking-2-17-2016*, 2016.

[3] J. F. Buss and T. Islam. Simplifying the weft hierarchy. *Theoretical Computer Science*, 351(3):303–313, 2006.

[4] T. Cadwell. Lol matchmaking explained. In *League of Legends Forum, http://forums.na.leagueoflegends.com/board/showthread.php?t=12029*, 2009.

[5] S. Cooper, C. S. Deterding, and T. Tsapakos. Player rating systems for balancing human computation games: Testing the effect of bipartiteness. In *International Joint Conference of DiGRA and FDG 2016*, 2016.

[6] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

[7] S. Dantchev, B. Martin, and S. Szeider. Parameterized proof complexity. *Computational Complexity*, 20(1):51–85, 2011.

[8] O. Delalleau, E. Contal, E. Thibodeau-Laufer, R. C. Ferrari, Y. Bengio, and F. Zhang. Beyond skill rating: Advanced matchmaking in ghost recon online. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):167–177, 2012.

[9] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness ii: On completeness for w [1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995.

[10] R. Duan and S. Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1, 2014.

[11] J. Edmonds. Maximum matching and a polyhedron with 0, l-vertices. *J. Res. Nat. Bur. Standards B*, 69(1965):125–130, 1965.

[12] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[13] A. E. Elo. *The rating of chessplayers, past and present.* Arco Pub., 1978.

[14] Y. Emek, S. Kutten, and R. Wattenhofer. Online matching: haste makes waste! In *STOC 2016*, pages 333–344, 2016.

[15] A. Gajentaan and M. H. Overmars. On a class of o (n2) problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

[16] A. Gajentaan and M. H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

[17] M. E. Glickman. The glicko system. *Boston University*, 1995.

[18] Y. Guo, S. Shen, O. Visser, and A. Iosup. An analysis of online match-based games. In *IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE) 2012*, pages 134–139, 2012.

[19] R. Herbrich, T. Minka, and T. Graepel. Trueskill(tm): A bayesian skill rating system. In *NIPS 2006*, pages 569–576, 2007.

[20] T. Kopelowitz, S. Pettie, and E. Porat. Higher lower bounds from the 3sum conjecture. In *SODA 2016*, pages 1272–1287, 2016.

[21] R. E. Korf. Multi-way number partitioning. In *IJCAI 2009*, pages 538–543, 2009.

[22] Y. LeJacq. League of legends is fixing one key matchmaking tool. In *Kotaku, http://kotaku.com/league-of-legends-is-fixing-one-key-matchmaking-tool-1707296218*, 2015.

[23] C. Marshall. The dynamic queue conundrum: why riot is wrestling with their community. In *PC Gamer, http://www.pcgamer.com/the-dynamic-queue-conundrum-why-riot-is-wrestling-with-their-community/*, 2016.

[24] M. Patrascu. Towards polynomial lower bounds for dynamic problems. In *STOC 2010*, pages 603–610, 2010.

[25] J. Paul. By the numbers: Most popular online games right now. In *Now Loading, https://nowloading.co/posts/3916216*, 2016.

[26] Riot Games. Dynamic queue and the future of league. In *League of Legends Game Updates, http://oce.leagueoflegends.com/en/news/game-updates/features/dynamic-queue-and-future-league*, 2016.

[27] P. Sankowski. Maximum weight bipartite matching in matrix multiplication time. *Theoretical Computer Science*, 410(44):4480–4488, 2009.

[28] Valve. Competitive skill groups faq. In *Counter Strike Blog, http://blog.counter-strike.net/index.php/2012/10/5565/*, 2012.

[29] Valve. Matchmaking. In *Dota 2 Blog, http://blog.dota2.com/2013/12/matchmaking/*, 2013.

[30] M. Véron, O. Marin, and S. Monnet. Matchmaking in multi-player on-line games: studying user traces to improve the user experience. In *Network and Operating System Support on Digital Audio and Video Workshop 2014*, page 7, 2014.