# Dueling algorithms [*]


Nicole Immorlica
Northwestern University

Adam Tauman Kalai
MSR New England

Brendan Lucier
University of Toronto

Ankur Moitra [†]
MIT

Andrew Postlewaite
University of Pennsylvania

Moshe Tennenholtz
Microsoft R&D Israel and the Technion, Israel



## ABSTRACT

We revisit classic algorithmic search and optimization problems from the perspective of competition. Rather than a single optimizer minimizing expected cost, we consider a zero-sum game in which a search problem is presented to two players, whose only goal is to *outperform the opponent*. Such games are typically exponentially large zero-sum games, but they often have a rich structure. We provide general techniques by which such structure can be leveraged to find minmax-optimal and approximate minmax-optimal strategies. We give examples of ranking, hiring, compression, and binary search duels, among others. We give bounds on how often one can beat the classic optimization algorithms in such duels.



## Categories and Subject Descriptors

J.4 [**Social and Behavioral Sciences**]: [economics]

## General Terms

Theory

## Keywords

algorithms, competition, equilibrium


## 1. INTRODUCTION

Many natural optimization problems have two-player competitive analogs. For example, consider the optimization problem facing a search engine: for a given query, there are $n$ potential search results, or webpages, which must be presented in a list. The cost of searching for a particular webpage is its rank in the list. Given a fixed probability distribution over which webpage is the desired target for the query, this problem can be construed as a one-player game, or optimization problem, in which the search engine wishes to minimize the expected search cost. The trivial greedy algorithm, which orders webpages in decreasing probability, is an optimal strategy for this one-player game.

The two-player variant of this problem models a user choosing between two search engines. The user thinks of a desired webpage and a query and executes the query on both search engines. The engine that ranks the desired page higher is chosen by the user as the "winner." If the greedy algorithm has the ranking of pages $\omega_1, \omega_2, \ldots, \omega_n$, then the ranking $\omega_2, \omega_3, \ldots, \omega_n, \omega_1$ beats the greedy ranking on every item except $\omega_1$. We say the greedy algorithm is $1 - 1/n$ *beatable* because there is a probability distribution over pages for which the greedy algorithm loses $1 - 1/n$ of the time. Thus, in a competitive setting, an "optimal" search engine can perform poorly against a clever opponent.

This *ranking duel* can be modeled as a symmetric constant-sum game, with $n!$ strategies, in which the player with the higher ranking of the target page receives a payoff of 1 and the other receives a payoff of 0 (in the case of a tie, say they both receive a payoff of 1/2). As in all symmetric one-sum games, there must be (mixed) strategies that guarantee expected payoff of at least 1/2 against any opponent. Put another way, there must be a (randomized) algorithm that takes as input the probability distribution and outputs a ranking, which is guaranteed to achieve a payoff of at least 1/2 in expectation against any opposing algorithm.

This conversion can be applied to any optimization problem with an element of uncertainty. Such problems are of the form $\min_{x \in X} \mathrm{E}_{\omega \sim p}[c(x, \omega)]$, where $p$ is a probability distribution over the *state of nature* $\omega \in \Omega$, $X$ is a feasible set, and $c : X \times \Omega \rightarrow \mathbf{R}$ is an objective function. The dueling analog has two players simultaneously choose $x, x'$; player 1 receives payoff 1 if $c(x, \omega) < c(x', \omega)$, payoff 0 if $c(x, \omega) > c(x', \omega)$, payoff 1/2 otherwise, and similarly for player 2.[1]

There are many natural examples of this setting beyond the ranking duel mentioned above. For example, for the shortest-path routing under a distribution over edge times,

---

[*]Part of this work was performed while the authors were visiting Microsoft Research, New England.


Nicole Immorlica (nickle@eecs.northwestern.edu)
Adam Tauman Kalai (adum@microsoft.com)
Brendan Lucier (blucier@cs.toronto.edu)
Ankur Moitra (moitra@mit.edu)
Andrew Postlewaite (apostlew@econ.upenn.edu)
Moshe Tennenholtz (moshet@microsoft.com)



[†]This work was supported in part by a Fannie and John Hertz Foundation Fellowship




*STOC'11,* June 6–8, 2011, San Jose, California, USA.



[1]Our techniques will also apply to asymmetric payoff functions; see Section 7.

the corresponding *racing duel* is simply a race, and the state of nature encodes uncertain edge delays.[2] For the classic secretary problem, in the corresponding *hiring duel* two employers must each select a candidate from a pool of $n$ candidates (though, as standard, they must decide whether or not to choose a candidate before interviewing the next one), and the winner is the one that hires the better candidate. This could model, for example, two competing companies attempting to hire CEOs or two opposing political parties selecting politicians to run in an election; the absolute quality of the candidate may be less important than being better than the other's selection. In a *compression duel*, a user with a (randomly chosen) sample string $\omega$ chooses between two compression schemes based on which one compresses that string better. This setting can also model a user searching for a file in two competing, hierarchical storage systems and choosing the system that finds the file first. In a *binary search duel*, a user searches for a random element in a list using two different search trees, and chooses whichever tree finds the element faster.

### *Our contribution.*

For each of these problems, we consider a number of questions related to how vulnerable a classic algorithm is to competition, what algorithms will be selected at equilibrium, and how well these strategies at equilibrium solve the original optimization problem.

QUESTION 1. *Will players use the classic optimization solution in the dueling setting?*

| Optimization Problem | Upper Bound | Lower Bound |
|---|---|---|
| Ranking | $1-1/n$ | $1-1/n$ |
| Racing | 1 | 1 |
| Hiring | 0.82 | 0.51 |
| Compression | 3/4 | 2/3 |
| Search | 5/8 | 5/8 |

Intuitively, the answer to this question should depend on how much an opponent can *game* the classic optimization solution. For example, in the *ranking duel* an opponent can beat the greedy algorithm on almost all pages – and even the most oblivious player would quickly realize the need to change strategies. In contrast, we demonstrate that many classic optimization solutions – such as the secretary algorithm for hiring, Huffman coding for compression, and standard binary search – are substantially less vulnerable. We say an algorithm is $\beta$-beatable (over distribution $p$) if there exists a response which achieves payoff $\beta$ against that algorithm (over distribution $p$). We summarize our results on the beatability of the standard optimization algorithm in each of our example optimization problems in the table above.

QUESTION 2. *What strategies do players play at equilibrium?*

We say an algorithm efficiently *solves* the duel if it takes as input a representation of the game and probability distribution $p$, and outputs an action $x \in X$ distributed according to some minmax optimal (i.e., Nash equilibrium) strategy. As our main result, we give a general method for solving duels that can be represented in a certain bilinear form. We also show how to convert an approximate best-response oracle for a dueling game into an approximate minmax optimal algorithm, using techniques from low-regret learning. We demonstrate the generality of these methods by showing how to apply them to the numerous examples described above.

A principle challenge in computing a Nash equilibrium is that the number of strategies can be exponential in the natural representation of the problem. For example, in the *ranking duel*, the number of pure strategies is the number of orderings of the $n$ webpages. Our approach is based on concisely summarizing a strategy based on statistical information about the assosciated randomized algorithm. This statistical information is enough to compute head-to-head payoffs (of one randomized algorithm against another), and from this representation we can compute equilibrium strategies in the original (exponentially-sized) game.

We leave as an open question:

QUESTION 3. *Are these equilibrium strategies still good at solving the optimization problem?*

As an example, consider the ranking duel. How much more time does a web surfer need to spend browsing to find the page he is interested in, because more than one search engine is competing for his attention? In fact, the surfer may be *better* off due to competition, depending on the model of comparison. For example, the cost to the web surfer may be the minimum of the ranks assigned by each search engine. And we leave open the tantalizing possibility that this quantity could in general be smaller at equilibrium for two competing search engines than for just one search engine playing the greedy algorithm.

### *Related work.*

The work most relevant to ours is the study of ranking games [4], and more generally the study of social context games [1]. In these settings, players' payoffs are translated into utilities based on social contexts, defined by a graph and an aggregation function. For example, a player's utility can be the sum/max/min of his neighbors' payoffs. This work studies the effect of social contexts on the existence and computation of game-theoretic solution concepts, but does not re-visit optimization algorithms in competitive settings.

For the hiring problem, several competitive variants and their algorithmic implications have been considered (see, e.g., [9] and the references therein). A typical competitive setting is a (general sum) game where a player achieves payoff of 1 if she hires the very best applicant and zero otherwise. But, to the best of our knowledge, no one has considered the natural model of a duel where the objective is simply to hire a better candidate than the opponent. Also related to our algorithmic results are succinct zero-sum games, where a game has exponentially many strategies but the payoff function can be computed by a succinct circuit. This general class has been showed to be EXP-hard to solve [6], and also difficult to approximate [7].

Finally, we note the line of research on competition among mechanisms, such as the study of competing auctions (see e.g. [5, 14, 15, 16]) or schedulers [2]. In such settings, each player selects a mechanism and then bidders select the auction to participate in and how much to bid there, where both

[2] We also refer to this as the *primal duel* because any other duel can be represented as a race with an appropriate graph and probability distribution $p$, though there may be an exponential blowup in representation size.

designers and bidders are strategic. This work is largely concerned with the existence of sub-game perfect equilibrium.

### *Outline.*

In Section 2 we define our model formally and provide a general framework for solving dueling problems as well as the warmup example of the ranking duel. We then use these tools to analyze the more intricate settings of the hiring duel (Section 3), the compression duel (Section 4), the search duel (Section 5), We conclude by describing avenues of future research in Section 8.

## 2. PRELIMINARIES

A problem of optimization under uncertainty, denoted $(X, \Omega, c, p)$, is specified by a feasible set $X$, a commonly-known distribution $p$ over the state of nature, $\omega$, chosen from set $\Omega$, and an objective function $c : X \times \Omega \to \mathbf{R}$. For simplicity we assume all these sets are finite. When $p$ is clear from context, we write the expected cost of $x \in X$ as $c(x) = \mathrm{E}_{\omega\sim p}[c(x, \omega)]$. The one-player optimum is $\mathrm{opt} = \min_{x\in X} c(x)$. Algorithm $A$ takes as input $p$ and randomness $r \in [0, 1]$, and outputs $x \in X$. We define $c(A) = \mathrm{E}_r[c(A(p, r))]$ and an algorithm $A$ is *one-player optimal* if $c(A) = \mathrm{opt}$.

In the two-person constant-sum duel game, denoted $D(X, \Omega, c, p)$, players simultaneously choose $x, x' \in X$, and player 1's payoff $v(x, x', p)$ is

$$\Pr_{\omega\sim p}[c(x, \omega) < c(x', \omega)] + \frac{1}{2}\Pr_{\omega\sim p}[c(x, \omega) = c(x', \omega)].$$

When $p$ is understood from context we write $v(x, x')$. Player 2's payoff is $v(x', x) = 1 - v(x, x')$. This models a tie, $c(x, \omega) = c(x', \omega)$, as a half point for each. We define the value of a strategy, $v(x, p)$, to be how much that strategy guarantees, $v(x, p) = \min_{x'\in X} v(x, x', p)$. Again, when $p$ is understood from context we write simply $v(x)$.

The set of probability distributions over set $S$ is denoted $\Delta(S)$. A *mixed strategy* is $\sigma \in \Delta(X)$. As is standard, we extend the domain of $v$ to mixed strategies bilinearly by expectation. A *best response* to mixed strategy $\sigma$ is a strategy which yields maximal payoff against $\sigma$, i.e., $\sigma'$ is a best response to $\sigma$ if it maximizes $v(\sigma', \sigma)$. A *minmax* strategy is a (possibly mixed) strategy that guarantees the safety value, in this case 1/2, against any opponent play. The best response to such a strategy yields payoffs of 1/2. The set of minmax strategies is denoted $MM(D(X, \Omega, c, p)) = \{\sigma \in \Delta(X) \mid v(\sigma) = 1/2\}$. A basic fact about constant-sum games is that the set of Nash equilibria is the cross product of the minmax strategies for player 1 and those of player 2.

### 2.1 Bilinear duels

In a bilinear duel, the feasible set of strategies are points in $n$-dimensional Euclidean space, i.e., $X \subseteq \mathbf{R}^n$, $X' \subseteq \mathbf{R}^{n'}$, and the payoff to player 1 is $v(x, x') = x^t M x'$ for some matrix $M \in \mathbf{R}^{n\times n'}$. Let $K$ be the convex hull of $X$. Any point in $K$ is achievable (in expectation) as a mixed strategy. Similarly define $K'$. In $n \times n$ bimatrix games, $K$ and $K'$ are simplices $\{x \in \mathbf{R}^n_{\geq 0} \mid \sum x_i = 1\}$. As we will point out in this section, solving a bilinear duel reduces to linear programming with a number of constraints proportional to the number of constraints necessary to define the feasible sets, $K$ and $K'$. (In typical applications, $K$ and $K'$ have a polynomial number of facets but an exponential number of vertices.)

Let $K$ be a polytope defined by the intersection of $m$ halfspaces, $K = \{x \in \mathbf{R}^n \mid w_i \cdot x \geq b_i \text{ for } i = 1, 2, \ldots, m\}$. Similarly, let $K'$ be the intersection of $m'$ halfspaces $w'_i \cdot x \geq b'_i$. The typical way to reduce to an LP for constant-sum games is:

$$\max_{v\in\mathbf{R}, x\in\mathbf{R}^n} v \text{ s.t. } x \in K \text{ and } x^T M x' \geq v \text{ for all } x' \in X'.$$

The above program has $m + |X'|$ constraints ($m$ constraints guaranteeing that $x \in K$). However, $|X'|$ (the number of pure strategies for player 2) is typically exponential. Alternatively, the following linear program has $O(n' + m + m')$ constraints, and hence can be found in time polynomial in $n', m, m'$ and the bit-size representation of $M$ and the constraints in $K$ and $K'$.

$$\max_{x\in\mathbf{R}^n, \lambda\in\mathbf{R}^{m'}} \sum_{i=1}^{m'} \lambda_i b'_i \text{ s.t. } x \in K \text{ and } x^t M = \sum_{i=1}^{m'} \lambda_i w'_i. \quad (1)$$

LEMMA 1. *For any constant-sum game with strategies $x \in K, x' \in K$ and payoffs $x^t M x'$, the maximum of the above linear program is the value of the game to player 1, and any maximizing $x$ is a minmax optimal strategy.*

PROOF. First we argue that the value of the above LP is at least as large as the value of the game to player 1. Let $x, \lambda$ maximize the above LP and let the maximum be $\alpha$. For any $x' \in K'$,

$$x^t M x' = \sum_{1}^{m'} \lambda_i w'_i \cdot x' \geq \sum_{1}^{m'} \lambda_i b'_i = \alpha.$$

Hence, this means that strategy $x$ guarantees player $x$ at least $\alpha$ against any opponent response, $x' \in K$. Hence $\alpha \leq v$ with equality iff $x$ is minmax optimal. Next, let $x$ be any minmax optimal strategy, and let $v$ be the value of the constant-sum game. This means that $x^t M x' \geq v$ for all $x' \in K'$ with equality for some point. In particular, the minmax theorem (equivalently, duality) means that the LP $\min_{x'\in K'} x^t M x'$ has a minimum value of $v$ and that there is a vector of $\lambda \geq 0$ such that $\sum_1^{m'} \lambda_i w'_i = x^t M$ and $\sum_1^{m'} \lambda_i b'_i = v$. Hence $\alpha \geq v$. □

### 2.2 Reduction to bilinear duels

The sets $X$ in a duel are typically objects such as paths, trees, rankings, etc., which are not themselves points in Euclidean space. In order to use the above approach to reduce a given duel $D(X, \Omega, c, p)$ to a bilinear duel in a *computationally efficient manner*, one needs the following:

1. An efficiently computable function $\phi : X \to K$ that maps each strategy $x \in X$ to a feasible point in $K \subseteq \mathbf{R}^n$.
2. A matrix $M$ such that $v(x, x') = \phi(x)^t M \phi(x')$, demonstrating that the problem is indeed bilinear.
3. A set of polynomially many feasible constraints that defines $K$.
4. A "randomized rounding algorithm" which takes as input a point in $K$ outputs an object in $X$.

Parts (1) and (2) are used to construct the bilinear duel corresponding to the original duel. Part (3) guarantees that this bilinear duel can be solved efficiently, and part (4) allows us to map a solution for the bilinear duel to a solution for the original duel.

In many cases, parts (1) and (2) are straightforward. Parts (3) and (4) may be more challenging. For example, for the binary trees used in the compression duel, it is easy to map a tree to a vector of node depths. However, we do not know how to efficiently determine whether a given vector of node depths is indeed a mixture over trees (except for certain types of trees which are in sorted order, like the binary search trees in the binary search duel). In the next subsection, we show how computing approximate best responses suffices.

## 2.3 Approximate minmax

In some cases, the polytope $K$ may have exponentially or infinitely many facets, in which case the above linear program is not very useful. In this section, we show that if one can compute *approximate* best responses for a bilinear duel, then one can *approximate* minmax strategies.

For any $\epsilon > 0$, an $\epsilon$-best response to a player 2 strategy $x' \in K'$ is any $x \in K$ such that

$$x^t M x' \geq \min_{y \in K} y^T M x' - \epsilon.$$

Similarly for player 1. An $\epsilon$-minmax strategy $x \in K$ for player 1 is one that guarantees player 1 an expected payoff not worse than $\epsilon$ minus the value, i.e.,

$$\min_{x' \in K} v(x, x') \geq \max_{y \in K} \min_{x' \in K} v(y, x') - \epsilon.$$

Best response oracles are functions from $K$ to $K'$ and vice versa. However, for many applications (and in particular the ones in this paper) where all feasible points are nonnegative, one can define a best response oracle for all nonnegative points in the positive orthant. (With additional effort, one can remove this assumption using Kleinberg and Awerbuch's elegant notion of a Barycentric spanner [3]). For scaling purposes, we assume that for some $B > 0$, the convex sets are $K \subseteq [0, B]^n$ and $K' \subseteq [0, B]^{n'}$ and the matrix $M \in [-B, B]^{n \times n'}$ is bounded as well.

Fix any $\epsilon > 0$. We suppose that we are given an $\epsilon$-approximate best response oracle in the following sense. For player 1, this is an oracle $\mathcal{O} : [0, B]^{n'} \to K$ which has the property that $\mathcal{O}(x')^t M x' \geq \max_{x \in K} x^t M x' - \epsilon$ for any $x' \in [0, B]^{n'}$. Similarly for $\mathcal{O}'$ for player 2. Hence, one is able to potentially respond to things which are not feasible strategies of the opponent. As can be seen in a number of applications, this does not impose a significant additional burden.

LEMMA 2. *For any $\epsilon > 0$, $n, n' \geq 1$, $B > 0$, and any bilinear duel with convex $K \subseteq [0, B]^n$ and $K' \subseteq [0, B]^{n'}$ and $M \in [-B, B]^{n \times n'}$, and any $\epsilon$-best response oracles, there is an algorithm for finding*

$$(24(\epsilon \max(m, m'))^{1/3} B^2 (nn')^{2/3})\text{-minmax}$$

*strategies $x \in K, x' \in K'$. The algorithm runs in time $\mathrm{poly}(\beta, m, m', 1/\epsilon)$ and makes $\mathrm{poly}(\beta, m, m', 1/\epsilon)$ oracle calls.*

The reduction and proof appear in the full version of the paper. Our approach is to use Hannan-type algorithms, i.e. "follow the expected leader" [10].

For an example, in Section 4 we reduce the compression duel, where the base objects are trees, to a bilinear duel and use the approximate best response oracle. To perform such a reduction, one needs the following.

1. An efficiently computable function $\phi : X \to K$ which maps any strategy $x \in X$ to a feasible point in $K \subseteq \mathbf{R}^n$.
2. A bounded payoff matrix $M$ demonstrating such that $v(x, x') = \phi(x)^t M \phi(x')$, demonstrating that the problem is indeed bilinear.
3. $\epsilon$-best response oracles for players 1 and 2. Here, the input to an $\epsilon$ best response oracle for player 1 is $x' \in [0, B]^{n'}$.

## 2.4 Beatability

One interesting quantity to examine is how well a one-player optimization algorithm performs in the two-player game. In other words, if a single player was a monopolist solving the one-player optimization problem, how badly could they be beaten if a second player suddenly entered. For a particular one-player-optimal algorithm $A$, we define its *beatability over distribution* $p$ to be $\mathrm{E}_r[v(A(p, r), p)]$, and we define its *beatability* to be $\inf_p \mathrm{E}_r[v(A(p, r), p)]$.

## 2.5 A warmup: the ranking duel

In the ranking duel, $\Omega = [n] = \{1, 2, \ldots, n\}$, $X$ is the set of permutations over $n$ items, and $c(\pi, \omega) \in [n]$ is the position of $\omega$ in $\pi$ (rank 1 is the "best" rank). The greedy algorithm, which outputs permutation $(\omega_1, \omega_2, \ldots, \omega_n)$ such that $p(\omega_1) \geq p(\omega_2) \geq \cdots \geq p(\omega_n)$, is optimal in the one-player version of the problem.[3]

This game can be represented as a bilinear duel as follows. Let $K$ and $K'$ be the set of doubly stochastic matrices, $K = K' = \{x \in \mathbf{R}^{n^2}_{\geq 0} \mid \forall j \sum_i x_{ij} = 1, \forall i \sum_j x_{ij} = 1\}$. Here $x_{ij}$ indicates the *probability* that item $i$ is placed in position $j$, in some distribution over rankings. The Birkhoff-von Neumann Theorem states that the set $K$ is precisely the set of probability distributions over rankings (where each ranking is represented as a permutation matrix $x \in \{0, 1\}^{n^2}$), and moreover any such $x \in K$ can be implemented efficiently via a form of randomized rounding. See, for example, Corollary 1.4.15 of [13]. Note $K$ is a polytope in $n^2$ dimensions with $O(n)$ facets. In this representation, the expected payoff of $x$ versus $x'$ is

$$\sum_i p(i) \left( \frac{1}{2} \Pr[\text{Equally rank } i] + \Pr[\text{P1 ranks } i \text{ higher}] \right)$$

$$= \quad \sum_i p(i) \sum_j x_{ij} \left( \frac{1}{2} x'_{ij} + \sum_{k > j} x'_{ik} \right).$$

The above is clearly bilinear in $x$ and $x'$ and can be written as $x^t M x'$ for some matrix $M$ with bounded coefficients. Hence, given $p$, we can solve the bilinear duel by the linear program (1) and round it to a (randomized) minmax optimal algorithm for ranking.

[3]In some cases, such as a model of competing search engines, one could have the agents rank only $k$ items, but the algorithmic results would be similar.

We next examine the beatability of the greedy algorithm. Note that for the uniform probability distribution $p(1) = p(2) = \ldots = p(n) = 1/n$, the greedy algorithm outputting, say, $(1, 2, \ldots, n)$ can be beaten with probability $1 - 1/n$ by the strategy $(2, 3, \ldots, n, 1)$. One can make greedy's selection unique by setting $p(i) = 1/n + (i - n/2)\epsilon$, and for sufficient small $\epsilon$ greedy can be beaten a fraction of time arbitrarily close to $1 - 1/n$.

## 3. HIRING DUEL

In a hiring duel, there are two employers $A$ and $B$ and two corresponding sets of workers $U_A = \{a_1, \ldots, a_n\}$ and $U_B = \{b_1, \ldots, b_n\}$ with $n$ workers each. The $i$'th worker of each set has a common value $v(i)$ where $v(i) > v(j)$ for all $i$ and $j > i$. Thus there is a total ranking of workers $a_i \in U_A$ (similarly $b_i \in U_B$) where a rank of 1 indicates the best worker, and workers are labeled according to rank. The goal of the employers is to hire a worker whose value (equivalently rank) beats that of his competitor's worker. Workers are interviewed by employers one-by-one in a random order. The relative ranks of workers are revealed to employers only at the time of the interview. That is, at time $i$, each employer has seen a prefix of the interview order consisting of $i$ of workers and knows only the projection of the total ranking on this prefix.[4] Hiring decisions must be made at the time of the interview, and only one worker may be hired. Thus the employers' pure strategies are mappings from any prefix and permutation of workers' ranks in that prefix to a binary hiring decision. We note that the permutation of ranks in a prefix does not effect the distribution of the rank of the just-interviewed worker, and hence without loss of generality we may assume the strategies are mappings from the round number and current rank to a hiring decision.

In dueling notation, our game is $(X, \Omega, c, p)$ where the elements of $X$ are functions $h : \{1, \ldots, n\}^2 \to \{0, 1\}$ indicating for any round $i$ and projected rank of current interviewee $j \leq i$ the hiring decision $h(i, j)$; $\Omega$ is the set $(\sigma_A, \sigma_B)$ of all pairs of permutations of $U_A$ and $U_B$; $c(h, \sigma)$ is the value $v(\sigma^{-1}(i^*))$ of the first candidate $i^* = \text{argmin}_i\{i : h(i, [\sigma^{-1}(i)]_i) = 1\}$ (where $[\sigma^{-1}(i)]_j$ indicates the projected rank of the $i$'th candidate among the first $j$ candidates according to $\sigma$) that received an offer; and $p$ (as is typical in the secretary problem) is the uniform distribution over $\Omega$. The mixed strategies $\pi \in \Delta(X)$ are simply mappings $\pi : \{0, \ldots, n\}^2 \to [0, 1]$ from rounds and projected ranks to a probability $\pi(i, j)$ of a hiring decision.

The values $v(\cdot)$ may be chosen adversarially, and hence in the one-player setting the optimal algorithm against a worst-case $v(\cdot)$ is the one that maximizes the probability of hiring the best worker (the worst-case values set $v(1) = 1$ and $v(i) << 1$ for $i > 1$). In the literature on secretary problems, the following *classical algorithm* is known to hire the best worker with probability approaching $\frac{1}{e}$: Interview n/e workers and hire next one that beats all the previous. Furthermore, there is no other algorithm that hires the best worker with higher probability.

[4]In some cases, an employer also knows when and whom his opponent hired, and may condition his strategy on this information as well. Only one of the settings described below needs this knowledge set; hence we defer our discussion of this point for now and explicitly mention the necessary assumptions where appropriate.

### 3.1 Common pools of workers

In this section, we study the *common hiring duel* in which employers see the *same* candidates in the *same* order so that $\sigma_A = \sigma_B$ and each employer observes when the other hires. In this case, the following strategy $\pi$ is a symmetric equilibrium: If the opponent has already hired, then hire anyone who beats his employee; otherwise hire as soon as the current candidate has at least a 50% chance of being the best of the remaining candidates.

LEMMA 3. *Strategy $\pi$ is efficiently computable and constitutes a symmetric equilibrium of the common hiring duel.*

The computability follows from a derivation of probabilities in terms of binomials, and the equilibrium claim follows by observing that there can be no profitable deviation. This strategy also beats the classical algorithm, enabling us to provide non-trivial lower and upper bounds for its beatability.

PROOF. For a round $i$, we compute a threshold $t_i$ such that $\pi$ hires if and only if the projected rank of the current candidate $j$ is at most $t_i$. Note that if $i$ candidates are observed, the probability that the $t_i$'th best among them is better than all remaining candidates is precisely $\binom{i}{t_i}/\binom{n}{t_i}$. The numerator is the number of ways to place the 1 through $t_i$'th best candidates overall among the first $i$ and the denominator is the number of ways to place the 1 through $t_i$'th best among the whole order. Hence to efficiently compute $\pi$ we just need to compute $t_i$ or, equivalently, estimate these ratios of binomials and hire whenever on round $i$ and observing the $j$'th best so far, $\binom{i}{j}/\binom{n}{j} \geq 1/2$.

We further note $\pi$ is a symmetric equilibrium since if an employer deviates and hires early then by definition the opponent has a better than 50% chance of getting a better candidate. Similarly, if an employer deviates and hires late then by definition his candidate has at most a 50% chance of being a better candidate than that of his opponent. □

LEMMA 4. *The beatability of the classical algorithm is at least* 0.51 *and at most* 0.82.

The lower bound follows from the fact that $\pi$ beats the classical algorithm with probability bounded above $1/2$ when the classical algorithm hires early (i.e., before round $n/2$), and the upper bound follows from the fact that the classical algorithm guarantees a probability of $1/e$ of hiring the best candidate, in which case no algorithm can beat it.

PROOF. For the lower bound, note that in any event, $\pi$ guarantees a payoff of at least $1/2$ against the classical algorithm. We next argue that for a constant fraction of the probability space, $\pi$ guarantees a payoff of strictly better than $1/2$. In particular, for some $q, 1/e < q < 1/2$, consider the event that the classical algorithm hires in the interval $\{n/e, qn\}$. This event happens whenever the best among the first $qn$ candidates is not among the first $n/e$ candidates, and hence has a probability of $(1-1/qe)$. Conditioned on this event, $\pi$ beats the classical algorithm whenever the best candidate overall is in the last $n(1-q)$ candidates,[5]

[5]This is a loose lower bound; there are many other instances where $\pi$ also wins, e.g., if the second-best candidate is in the last $n(1-q)$ candidates and the best occurs after the third best in the first $qn$ candidates.

which happens with probability $(1-q)$ (the conditioning does not change this probability since it is only a property of the permutation projected onto the first $qn$ elements). Hence the overall payoff of $\pi$ against the classical algorithm is $(1-q)(1-1/qe)+(1/2)(1/qe)$. Optimizing for $q$ yields the result.

For the upper bound, note as mentioned above that the classical algorithm has a probability approaching $1/e$ of hiring the *best* candidate. From here, we see $((1/2e)+(1-1/e)) = 1-1/2e < 0.82$ is an upper bound on the beatability of the classical algorithm since the best an opponent can do is always hire the best worker when the classical algorithm hires the best worker and always hire a better worker when the classical algorithm does not hire the best worker. □

## 3.2 Independent pools of workers

In this section, we study the *independent hiring duel* in which the employers see *different* candidates. Thus $\sigma_A \neq \sigma_B$ and the employers do not see when the opponent hires. We use the bilinear duel framework introduced in Section 2.1 to compute an equilibrium for this setting, yielding the following theorem.

THEOREM 1. *The equilibrium strategies of the independent hiring duel are efficiently computable.*

The main idea is to represent strategies $\pi$ by vectors $\{p_{ij}\}$ where $p_{ij}$ is the (total) probability of hiring the $j$'th best candidate seen so far on round $i$. Let $q_i$ be the probability of reaching round $i$, and note it can be computed from the $\{p_{ij}\}$. Recall $\pi(i,j)$ is the probability of hiring the $j$'th best so far at round $i$ conditional on seeing the $j$'th best so far at round $i$. Thus using Bayes' Rule we can derive an efficiently-computable bijective mapping (with an efficiently computable inverse) $\phi(\pi)$ between $\pi$ and $\{p_{ij}\}$ which simply sets $\pi(i,j) = p_{ij}/(q_i/i)$. It only remains to show that one can find a matrix $M$ such that the payoff of a strategy $\pi$ versus a strategy $\pi'$ is $\phi(\pi)^t M \phi(\pi')$. This is done by calculating the appropriate binomials.

We show how to apply the bilinear duel framework to compute the equilibrium of the independent hiring duel. This requires the following steps: define a subset $K$ of Euclidean space to represent strategies, define a bijective mapping between $K$ and feasible (mixed) strategies $\Delta(X)$, and show how to represent the payoff matrix of strategies in the bilinear duel space. We discuss each step in order.

### *Defining $K$.*

For each $1 \le i \le n$ and $j \le i$ we define $p_{ij}$ to be the (total) probability of seeing and hiring the $j$'th best candidate seen so far at round $i$. Our subspace $K = [0,1]^{n(n+1)/2}$ consists of the collection of probabilities $\{p_{ij}\}$. To derive constraints on this space, we introduce a new variable $q_i$ representing the probability of reaching round $i$. We note that the probability of reaching round $(i+1)$ must equal the probability of reaching round $i$ and *not* hiring, so that $q_{i+1} = q_i - \sum_{j=1}^{n} p_{ij}$. Furthermore, the probability $p_{ij}$ can not exceed the probability of reaching round $i$ and interviewing the $j$'th best candidate seen so far. The probability of reaching round $i$ is $q_i$ by definition, and the probability that the projected rank of the $i$'th candidate is $j$ is $1/i$ by our choice of a uniformly random permutation. Thus $p_{ij} \le q_i/i$. Together with the initial condition that $q_i = 1$, these constraints completely characterize $K$.

### *Mapping.*

Recall a strategy $\pi$ indicates for each $i$ and $j \le i$ the *conditional* probability of making an offer given that the employer is interviewing the $i$'th candidate and his projected rank is $j$ whereas $p_{ij}$ is the *total* probability of interviewing the $i$'th candidate with a projected rank of $j$ and making an offer. Thus $\pi(i,j) = p_{ij}/(q_i/i)$ and so $p_{ij} = q_i\pi(i,j)/i$. Together with the equailities derived above that $q_1 = 1$ and $q_{i+1} = q_i - \sum_{j=1}^{n} p_{ij}$, we can recursively map any strategy $\pi$ to $K$ efficiently. To map back we just take the inverse of this bijection: given a point $\{p_{ij}\}$ in $K$, we compute the (unique) $q_i$ satisfying the constraints $q_1 = 1$ and $q_{i+1} = q_i - \sum_{j=1}^{n} p_{ij}$, and define $\pi(i,j) = p_{ij}/(q_i/i)$.

### *Payoff Matrix.*

By the above definitions, for any strategy $\pi$ and corresponding mapping $\{p_{ij}\}$, the probability that the strategy hires the $j$'th best so far on round $i$ is $p_{ij}$. Given that employer $A$ hires the $j$'th best so far on round $i$ and employer $B$ hires the $j'$'th best so far on round $i'$, we define $M_{iji'j'}$ to be the probability that the overall rank of employer $A$'s hire beats that of employer $B$'s hire plus one-half times the probability that their ranks are equal. We can derive the entries of the this matrix as follows: Let $E_r^X$ be the event that with respect to permutation $\sigma_X$ the overall rank of a fixed candidate is $r$, and $F_{ij}^X$ be the event that the projected rank of the last candidate in a random prefix of size $i$ is $j$. Then

$$M_{iji'j'} = \sum_{r,r':1\le r<r'\le n} \Pr[E_r^A|F_{ij}^A]\Pr[E_{r'}^B|F_{i'j'}^B] + \frac{1}{2}\sum_{1\le r\le n} \Pr[E_r^A|F_{ij}^A]\Pr[E_r^B|F_{i'j'}^B].$$

Furthermore, by Bayes rule,

$$\Pr[E_r^X|F_{ij}^X] = \Pr[F_{ij}^X|E_r^X]\Pr[E_r^X]/\Pr[F_{ij}^X]$$

where $\Pr[E_r^X] = 1/n$ and $\Pr[F_{ij}^X] = 1/i$. To compute $\Pr[F_{ij}^X|E_r^X]$, we select the ranks of the other candidates in the prefix of size $i$. There are $\binom{r-1}{j-1}$ ways to pick the ranks of the better candidates and $\binom{n-r+1}{i-j}$ ways to pick the ranks of the worse candidates. As there are $\binom{n-1}{i-1}$ ways overall to pick the ranks of the other candidates, we see:

$$\Pr[F_{ij}^X|E_r^X] = \frac{\binom{r-1}{j-1}\binom{n-r+1}{i-j}}{\binom{n-1}{i-1}}.$$

Letting $\{p_{ij}\}$ be the mapping $\phi(\pi)$ of employer $A$'s strategy $\pi$ and $\{p'_{ij}\}$ be the mapping $\phi(\pi)$ of employer $B$'s strategy $\pi'$, we see that $c(\pi,\pi') = \phi(\pi)^t M \phi(\pi')$, as required.

By the above arguments, and the machinery from Section 2.1, we have proven Theorem 1 which claims that the equilibrium of the independent hiring duel is computable.

# 4. COMPRESSION DUEL

In a compression duel, two competitors are given a distribution $p$ an each chooses a binary tree with leaf set $\Omega$. An element $\omega \in \Omega$ is then chosen according to distribution $p$, and whichever player's tree has $\omega$ closest to the root is the winner. This game can be thought of as a competition between prefix-free compression schemes for a base set of words. The Huffman algorithm, which repeatedly pairs nodes with lowest probability, is known to be optimal for single-player compression.

The compression duel is $D(X, \Omega, c, p)$, where $\Omega = [n]$ and $X$ is the set of binary trees with leaf set $\Omega$. For $T \in X$ and $\omega \in \Omega$, $c(T, \omega)$ is the depth of $\omega$ in $T$. In Section 4.3 we consider a variant in which not every element of $\Omega$ must appear in the tree.

## 4.1 Computing an equilibrium

The compression duel can be represented as a bilinear game. In this case, $K$ and $K'$ will be sets of stochastic matrices, where a matrix entry $\{x_{ij}\}$ indicates the probability that item $\omega_i$ is placed at depth $j$. The set $K$ is precisely the set of probability distributions over node depths that are consistent with probability distributions over binary trees. We would like to compute minmax optimal algorithms as in Section 2.2, but we do not have a randomized rounding scheme that maps elements of $K$ to binary trees. Instead, following Section 2.3, we will find approximate minmax strategies by constructing an $\epsilon$-best response oracle.

The mapping $\phi : X \to K$ is straightforward: it maps a binary tree to its depth profile. Also, the expected payoff of $x \in K$ versus $x' \in K'$ is

$$\sum_i p(i) \sum_j x_{ij} \left( \frac{1}{2} x'_{ij} + \sum_{k>j} x'_{ij} \right)$$

which can be written as $x^t M x'$ where matrix $M$ has bounded entries. To apply Lemma 2, we must now provide an $\epsilon$ best response oracle, which we implement by reducing to a knapsack problem.

Fix $p$ and $x' \in K'$. We will reduce the problem of finding a best response for $x'$ to the multiple-choice knapsack problem (MCKP), for which there is an FPTAS [12]. In the MCKP, there are $n$ lists of items, say $\{(\alpha_{i1}, \ldots, \alpha_{ik_i}) \mid 1 \le i \le n\}$, with each item $\alpha_{ij}$ having a value $v_{ij} \ge 0$ and weight $w_{ij} \ge 0$. The problem is to choose exactly one item from each list with total weight at most 1, with the goal of maximizing total value. Our reduction is as follows. For each $\omega_i \in \Omega$ and $0 \le j \le n$, define $w_{ij} = 2^{-j}$ and $v_{ij} = p(\omega_i) \left( \frac{1}{2} x'_{ij} + \sum_{d>j} x'_{id} \right)$. This defines a MCKP input instance. For any given $t \in X$, $v(\phi(t), x') = \sum_{\omega_i \in \Omega} v_{i d_t(i)}$ and $\sum_{\omega_i \in \Omega} w_{i, d_t(i)} \le 1$ by the Kraft inequality. Thus, any strategy for the compression duel can be mapped to a solution to the MCKP. Likewise, a solution to the MCKP can be mapped in a value-preserving way to a binary tree $t$ with leaf set $\Omega$, again by the Kraft inequality. This completes the reduction.

## 4.2 Beatability

We will obtain a bound of 3/4 on the beatability of the Huffman algorithm. The high-level idea is to choose an arbitrary tree $T$ and consider the leaves for which $T$ beats $H$ and vice-versa. We then apply structural properties of trees to limit the relative sizes of these sets of leaves, then use properties of Huffman trees to bound the relative probability that a sampled leaf falls in one set or the other. The details appear in the full version of the paper.

Proposition 2. *The beatability of the Huffman algorithm is at most* $\frac{3}{4}$.

We now give an example to demonstrate that the Huffman algorithm is at least $(2/3 - \epsilon)$-beatable for every $\epsilon > 0$. For any $n \ge 3$, consider the probability distribution given by $p(\omega_1) = \frac{1}{3}$, $p(\omega_i) = \frac{1}{3 \cdot 2^{i-2}}$ for all $1 < i < n$, and $p(\omega_n) = \frac{1}{3 \cdot 2^{n-3}}$. For this distribution, the Huffman tree $t$ satisfies $d_t(\omega_i) = i$ for each $i < n$ and $d_t(\omega_n) = n - 1$. Consider the alternative tree $t'$ in which $d(\omega_1) = n - 1$ and $d(\omega_i) = i - 1$ for all $i > 1$. Then $t'$ will win if any of $\omega_2, \omega_3, \ldots, \omega_{n-1}$ are chosen, and will tie on $\omega_n$. Thus

$$v(t', t) = \sum_{i>1} \frac{1}{3 \cdot 2^{i-2}} + \frac{1}{2} \cdot \frac{1}{3 \cdot 2^{n-3}} = \frac{2}{3} - \frac{1}{3 \cdot 2^{n-2}},$$

and hence the Huffman algorithm is $(\frac{2}{3} - \frac{1}{3 \cdot 2^{n-2}})$-beatable.

We note that if all probabilities are inverse powers of 2, the Huffman algorithm is minmax optimal.

Proposition 3. *Suppose that* $p(\omega_i) = 2^{-a_i}$ *for some integer* $a_i$*, for each* $i \le n$*. Then the value of the Huffman tree* $H$ *is* $v(H) = 1/2$.

## 4.3 Variant: allowed failures

We consider a variant of the compression duel in which an algorithm can fail to encode certain elements. If we write $L(T)$ to be the set of leaves of binary tree $T$, then in the (original) model of compression we require that $L(T) = \Omega$ for all $T \in X$, whereas in the "Fail" model we require only that $L(T) \subseteq \Omega$. If $\omega \notin L(T)$, we will take $c(T, \omega) = \infty$. The Huffman algorithm is optimal for single-player compression in the Fail model.

We note that our method of computing approximate minmax algorithms carries over to this variant; we need only change our best-response reduction to use a Multiple-Choice Knapsack Problem in which *at most* one element is chosen from each list. What is different, however, is that the Huffman algorithm is completely beatable in the Fail model. If we take $\Omega = \{\omega_1, \omega_2\}$ with $p(\omega_1) = 1$ and $p(\omega_2) = 0$, the Huffman tree $H$ places each of the elements of $\Omega$ at depth 2. If $T$ is the singleton tree that consists of $\omega_1$ as the root, then $v(T, H) = 1$.

# 5. BINARY SEARCH DUEL

In a binary search duel, $\Omega = [n]$ and $X$ is the set of binary search trees on $\Omega$ (i.e. binary trees in which nodes are labeled with elements of $\Omega$ in such a way that an in-order traversal visits the elements of $\Omega$ in sorted order). Let $p$ be a distribution on $\Omega$. Then for $T \in X$ and $\omega \in \Omega$, $c(T, \omega)$ is the depth of the node labeled by "$\omega$" in the tree $T$. In single-player binary search and uniform $p$, selecting the median $m$ element in $\Omega$ as the root node and recursing on the left $\{\omega | \omega < m\}$ and right $\{\omega | \omega > m\}$ subsets to construct sub-trees is known to be optimal.

The binary search game can be represented as a bilinear duel. In this case, $K$ and $K'$ will be sets of stochastic matrices (as in the case of the compression game) and the entry $\{x_{i,j}\}$ will represent the probability that item $\omega_j$ is placed at depth $i$. Of course, not every stochastic matrix is realizable as a distribution on binary search trees (i.e. such that the probability $\omega_j$ is placed at depth $i$ is $\{x_{i,j}\}$). In order to define linear constraints on $K$ so that any matrix in $K$ is realizable, we will introduce an auxiliary data structure in Section 5.1 called the State-Action Structure that captures the decisions made by a binary search tree. Using these ideas, we will be able to fit the binary search game into the bilinear duel framework introduced in Section 2.2

and hence be able to efficiently compute a Nash equilibrium strategy for each player.

Given a binary search tree $T \in X$, we will write $c_T(\omega)$ for the depth of $\omega$ in $T$. We will also refer to $c_T(\omega)$ as the time that $T$ finds $\omega$.

## 5.1 Computing an equilibrium

In this subsection, we give an algorithm for computing a Nash equilibrium for the binary search game, based on the bilinear duel framework introduced in Section 2.2. We will do this by defining a structure called the STATE-ACTION STRUCTURE that we can use to represent the decisions made by a binary search tree using only polynomially many variables. The set of valid variable assignments in a STATE-ACTION STRUCTURE will also be defined by only polynomially many linear constraints and so these structures will naturally be closed under taking convex combinations. We will demonstrate that the value of playing $\sigma \in \Delta(X)$ against any value matrix $V$ is a linear function of the variables in the STATE-ACTION STRUCTURE corresponding to $\sigma$. Furthermore, all valid STATE-ACTION STRUCTURES can be efficiently realized as a distribution on binary search trees which achieves the same expected value.

To apply the bilinear duel framework, we must give a mapping $\phi$ from the space of binary search trees to a convex set $K$ defined explicitly by a polynomial number of linear constraints (on a polynomial number of variables). We now give an informal description of $K$: The idea is to represent a binary search tree $T \in X$ as a layered graph. The nodes (at each depth) alternate in type. One layer represents the current knowledge state of the binary search tree. After making some number of queries (and not yet finding the token), all the information that the binary search tree knows is an interval of values to which the token is confined - we refer to this as the *live interval.* The next layer of nodes represents an action - i.e. a query to some item in the live interval. Correspondingly, there will be three outgoing edges from an action node representing the possible replies that either the item is to the left, to the right, or at the query location (in which case the outgoing edge will exit to a terminal state).

We will define a flow on this layered graph based on $T$ and the distribution $p$ on $\Omega$. Flow will represent total probability - i.e. the total flow into a state node will represent the probability (under a random choice of $\omega \in \Omega$ according to $p$) that $T$ reaches this state of knowledge (in exactly the corresponding number of queries). Then the flow out of a state node represents a decision of which item to query next. And lastly, the flow out of an action node splits according to Bayes' Rule - if all the information revealed so far is that the token is confined to some interval, we can express the probability that (say) our next query to a particular item finds the token as a conditional probability. We can then take convex combinations of these "basic" flows in order to form flows corresponding to distributions on binary search trees.

We give a randomized rounding algorithm to select a random binary search tree based on a flow - in such a way that the marginal probabilities of finding a token $\omega_i$ at time $r$ are exactly what the flow specifies they should be. The idea is that if we choose an outgoing edge for each state node (with probability proportional to the flow), then we have fixed a binary search tree because we have specified a decision rule for each possible internal state of knowledge. Suppose we were to now select an edge out of each action node (again with probability proportional to the flow) and we were to follow the unique path from the start node to a terminal node. This procedure would be equivalent to searching for a randomly chosen token $\omega_i$ chosen according to $p$ and using this token to choose outgoing edges from action nodes. This procedure generates a random path from the start node to a terminal node, and is in fact equivalent to sampling a random path in the path decomposition of the flow proportionally to the flow along the path. Because these two rounding procedures are equivalent, the marginal distribution that results from generating a binary search tree (and choosing a random element to look for) will exactly match the corresponding values of the flow.

## 5.2 Beatability

We next consider the beatability of the classical algorithm when $p$ is the uniform distribution on $\Omega$. For lack of a better term, let us call this single-player optimum the median binary search - or median search.

Here we give matching upper and lower bounds on the beatability of median search. The idea is that an adversary attempting to do well against median search can only place one item at depth 1, two items at depth 2, four items at depth 3 and so on. We can regard these as budget restrictions - the adversary cannot choose too many items to map to a particular depth. There are additional combinatorial restrictions, as well. For example, an adversary cannot place two labels of depth 2 both to the right of the label of depth 1 - because even though the root node in a binary search tree can have two children, it cannot have more than one right child.

But suppose we relax this restriction, and only consider budget restrictions on the adversary. Then the resulting best response question becomes a bipartite maximum weight matching problem. Nodes on the left (in this bipartite graph) represent items, and nodes on the right represent depths (there is one node of depth 1, two nodes of depth 2, ...). And for any choice of a depth to assign to a node, we can evaluate the value of this decision - if this decision beats median search when searching for that element, we give the corresponding edge weight 1. If it ties median search, we give the edge weight $\frac{1}{2}$ and otherwise we give the edge zero weight.

We give an upper bound on the value of a maximum weight matching in this graph, hence giving an upper bound on how well an adversary can do if he is subject to only budget restrictions. If we now add the combinatorial restrictions too, this only makes the best response problem harder. So in this way, we are able to bound how much an adversary can beat median search. In fact, we give a lower bound that matches this upper bound - so our relaxation did not make the problem strictly easier (to beat median search).

We focus on the scenario in which $|\Omega| = 2^\ell - 1$ for some $\ell \geq 1$ and $p$ is the uniform distribution. Throughout this section we denote $n = |\Omega|$. The reason we fix $n$ to be of the form $2^\ell - 1$ is because the optimal single-player strategy is well-defined in the sense that the first query will be at precisely the median element, and if the element $\omega$ is not found on this query, then the problem will break down into one of two possible $2^{\ell-1} - 1$ sized sub-problems. For this case, we give asymptotically matching upper and lower bounds on the beatability of median search. The details appear in the full version of the paper.

LEMMA 5. *The beatability of median search is at least* $\frac{2^{\ell-1}-1+2^{\ell-3}}{2^{\ell}-1} \approx \frac{5}{8}$.

LEMMA 6. *The beatability of median search is at most* $\frac{2^{\ell-1}-1+2^{\ell-3}}{2^{\ell}-1} \approx \frac{5}{8}$.

# 6. A RACING DUEL

The racing duel illustrates a simple example in which the beatability is unbounded, the optimization problem is "easy," but finding polynomial-time minmax algorithms remains a challenging open problem. The optimization problem behind the racing duel is routing under uncertainty. There is an underlying directed multigraph $(V, E)$ containing designated start and terminal nodes $s, t \in V$, along with a distribution over bounded weight vectors $\Omega \subset \mathbf{R}^{E}_{\geq 0}$, where $\omega_e$ represents the delay in traversing edge $e$. The feasible set $X$ is the set of paths from $s$ to $t$. The probability distribution $p \in \Delta(\Omega)$ is an arbitrary measure over $\Omega$. Finally, $c(x, \omega) = \sum_{e \in x} \omega_e$.

For general graphs, solving the racing duel seems quite challenging. This is true even when routing between two nodes with parallel edges, i.e., $V = \{s, t\}$ and all edges $E = \{e_1, e_2, \ldots, e_n\}$ are from $s$ to $t$. As mentioned in the introduction, this problem is in some sense a "primal" duel in that it can encode any duel and finite strategy set. In particular, given any optimization problem with $|X| = n$, we can create a race where each edge $e_i \in E$ corresponds to a strategy $x_i \in X$, and the delays on the edges match the costs of the associated strategies.

## 6.1 Shortest path routing is 1-beatable

The single-player racing problem is easy: take the shortest path on the graph with weights $w_e = \mathrm{E}_{\omega \sim p}[\omega_e]$. However, this algorithm can be beaten almost always. Consider a graph with two parallel edges, $a$ and $b$, both from $s$ to $t$. Say the cost of $a$ is $\epsilon/2 > 0$ with probability 1, and the cost of $b$ is 0 with probability $1 - \epsilon$ and 1 with probability $\epsilon$. The optimization algorithm will choose $a$, but $b$ beats $a$ with probability $1 - \epsilon$, which is arbitrarily close to 1.

## 6.2 Price of anarchy

Take social welfare to be the average performance, $W(x, x') = (c(x) + c(x'))/2$. Then the price of anarchy for racing is unbounded. Consider a graph with two parallel edges, $a$ and $b$, both from $s$ to $t$. The cost of $a$ is $\epsilon > 0$ with probability 1, and the cost of $b$ is 0 with probability 3/4 and 1 with probability 1/4. Then $b$ a dominant strategy for both players, but its expected cost is 1/4, so the price of anarchy is $1/(4\epsilon)$, which can be arbitrarily large.

# 7. ASYMMETRIC GAMES

We note that all of the examples we considered have been symmetric with respect to the players, but our results can be extended to asymmetric games. Our analysis of bilinear duels in Section 2.1 does not assume symmetry when discussing bilinear games. For instance, we could consider a game where player 1 wins in the case of ties, so player 1's payoff is $\Pr[c(x, \omega) \leq c(x', \omega)]$. One natural example would be a ranking duel in which there is an "incumbent" search engine that appeared first, so a user prefers to continue using it rather than switching to a new one. This game can be represented in the same bilinear form as in Section 2.5, the only change being a small modification of the payoff matrix $M$. Other types of asymmetry, such as players having different objective functions, can be handled in the same way. For example, in a hiring duel, our analysis techniques apply even if the two players may have different pools of candidates, of possibly different sizes and qualities.

# 8. CONCLUSION

The dueling framework is a way to look at classic optimization problems through the lens of competition. As we have demonstrated, standard algorithms for many optimization problems do not, in general, perform well in these competitive settings. This leads us to suspect that alternative algorithms, tailored to competition, may find use in practice. We adapted linear programming and learning techniques into methods for constructing such algorithms.

There are many open questions yet to consider. For instance, one avenue of future work is to compare the computational difficulty of solving an optimization problem with that of solving the associated duel. We know that one is not consistently more difficult than the other: in Section 6 we provide an example in which the optimization problem is computationally easy but the competitive variant appears difficult; an example of the opposite situation is given in the full version of the paper, where a computationally hard optimization problem has a duel which can be solved easily. Is there some structure underlying the relationship between the computational hardness of an optimization problem and its competitive analog?

Perhaps more importantly, one could ask about performance loss inherent when players choose their algorithms competitively instead of using the (single-player) optimal algorithm. In other words, what is the *price of anarchy* [11] of a given duel? Such a question requires a suitable definition of the social welfare for multiple algorithms, and in particular it may be that two competing algorithms perform better than a single optimal algorithm. Our main open question is:

OPEN QUESTION 1. *Does competition between algorithms improve or degrade expected performance?*

# 9. REFERENCES


[1] Itai Ashlagi, Piotr Krysta, and Moshe Tennenholtz. Social context games. In *WINE*, pages 675–683, 2008.

[2] Itai Ashlagi, Moshe Tennenholtz, and Aviv Zohar. Competing schedulers. In *AAAI*, 2010.

[3] Baruch Awerbuch and Robert D. Kleinberg. Adaptive routing with end-to-end feedback: distributed learning and geometric approaches. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 45–53, New York, NY, USA, 2004. ACM.

[4] Felix Brandt, Felix A. Fischer, Paul Harrenstein, and Yoav Shoham. Ranking games. *Artif. Intell.*, 173(2):221–239, 2009.

[5] R. Burguet and J. Sakovics. Imperfect Competition in Auction Designs. *International Economic Review*, 40(1):231–247, 1999.

[6] J. Feigenbaum, D. Koller, and P. Shor. A game-theoretic classification of interactive complexity classes. In *SCT '95: Proceedings of the 10th Annual Structure in Complexity Theory Conference (SCT'95)*, page 227, Washington, DC, USA, 1995. IEEE Computer Society.

[7] Lance Fortnow, Russell Impagliazzo, Valentine Kabanets, and Christopher Umans. On the complexity of succinct zero-sum games. In *CCC '05: Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pages 323–332, Washington, DC, USA, 2005. IEEE Computer Society.

[8] Yoav Freund and Robert E. Schapire. Game theory, on-line prediction and boosting. In *COLT '96: Proceedings of the ninth annual conference on Computational learning theory*, pages 325–332, New York, NY, USA, 1996. ACM.

[9] Nicole Immorlica, Robert Kleinberg, and Mohammad Mahdian. Secretary problems with competing employers. In Paul Spirakis, Marios Mavronicolas, and Spyros Kontogiannis, editors, *Internet and Network Economics*, volume 4286 of *Lecture Notes in Computer Science*, pages 389–400. Springer Berlin / Heidelberg, 2006.

[10] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291 – 307, 2005. Learning Theory 2003.

[11] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th annual conference on Theoretical aspects of computer science*, STACS'99, pages 404–413, Berlin, Heidelberg, 1999. Springer-Verlag.

[12] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

[13] L. Lovász and M. D. Plummer. Matching theory. *Ann. Discrete Math.*, 29, 1986.

[14] P. McAfee. Mechanism Design by Competing Sellers. *Econometrica*, 61:1281–ï£¡1312, 1993.

[15] D. Monderer and M. Tennenholtz. K-price auctions: Revenue Inequalities, Utility Equivalence, and Competition in Auction Design. *Economic Theory*, 24(2):255–270, 2004.

[16] M. Peters and S. Severinov. Competition Among Sellers Who Offer Auctions Instead of Prices. *Journal of Economic Theory*, 75:141ï£¡–179, 1997.