A Survey of Yao's Minimax Principle

Brian Shimanuki [*]
Mayuri Sridhar [†]
Chelsea Voss [‡]

May 11, 2015

Abstract

We introduce Yao's Minimax Principle [?] and present a series of online and non-local games for which we illustrate the use of Yao's Principle in bounding the effectiveness of an optimal solution. In particular, we look at the removable online knapsack problem [?], the online coloring co-interval graph problem [?], and a multiplayer extension of the CHSH quantum game [?] to demonstrate how combining an algorithm with an input distribution can tighten the interval on which we know the optimal solution lies.

1 Introduction

Yao's minimax principle, proposed by Yao in 1977, was first used to study lower bounds on the optimal running times of randomized algorithms for a particular problem. In this paper, we survey a series of games for which Yao's principle can be used to bound the effectiveness of all randomized algorithms against an adversary. In this section, we present Yao's Principle and give a proof of it.

Yao's Minimax Principle *The best deterministic algorithm, when run on any distribution over random inputs, is at least as good as any randomized algorithm when run on the worst-case input.*

More formally,

$$\min_{r \in \mathcal{R}} \mathop{\mathbb{E}}_{x \in X} \texttt{cost}(r, x) \leq \max_{x \in \mathcal{X}} \mathop{\mathbb{E}}_{r \in R} \texttt{cost}(r, x)$$

[*]bshimanuki@mit.edu
[†]mayuri@mit.edu
[‡]csvoss@mit.edu

where

$$\begin{aligned}
\mathcal{X} &= \text{the set of all possible inputs to the algorithm.}\\
\mathcal{R} &= \text{the set of all possible strings defining deterministic algorithms.}\\
X &= \text{any probability distribution over inputs to the algorithm.}\\
R &= \text{any probability distribution over strings defining deterministic algorithms.}\\
\texttt{cost}(r,x) &= \text{the cost (often running time but in this paper will be an optimization quantity)}\\
& \text{of the algorithm running on input } x\text{, if the specific algorithm to use is defined}\\
& \text{by string } r.
\end{aligned}$$

In other words, if you can figure out how well the best deterministic algorithm runs against any distribution X of random inputs, you know that any randomized algorithm you formulate can do no better against its worst-case input – giving you a lower bound on the randomized algorithm's running time. Although X need not be the worst possible distribution over random inputs for this principle to be useful, note that the worse X is as an input distribution, the tighter the resulting lower bound will be.

To further develop the intuition behind Yao's minimax principle, we will briefly cover its proof.

1.1 Proof of Yao's Principle

Consider a two-player game, as defined in game theory. Each player i has a set of moves M_i that they may choose from. The game is defined by a *payoff matrix* of outcomes for each player, a function mapping $M_1 \times M_2 \to \mathbb{R} \times \mathbb{R}$.

In a *pure strategy*, player 1 chooses exactly one move x from M_1 to play. In a *mixed strategy*, player 1 instead chooses a probability distribution $\pi_1(x)$ of moves from M_1.

Let $\texttt{payoff}_1(x, y)$ denote the payoff to player 1 when the players choose moves x and y, respectively.

This leads to von Neumann's minimax theorem,

$$\max_{A\in\mathcal{A}} \min_{B\in\mathcal{B}} \mathop{\mathbb{E}}_{a\in A, b\in B} \texttt{payoff}_1(a,b) = \min_{B\in\mathcal{B}} \max_{A\in\mathcal{A}} \mathop{\mathbb{E}}_{a\in A, b\in B} \texttt{payoff}_1(a,b)$$

where

$$\begin{aligned}
\mathcal{A} &= \text{set of moves available to player 1.}\\
\mathcal{B} &= \text{set of moves available to player 2.}\\
A &= \text{any probability distribution over moves to player 1}\\
& \quad \text{– that is, any mixed strategy for player 1.}\\
B &= \text{any probability distribution over moves to player 2}\\
& \quad \text{– that is, any mixed strategy for player 2.}\\
\texttt{payoff}_1(a,b) &= \text{the payoff to player 1}
\end{aligned}$$

This in turn can be rewritten to yield the following:

$$\max_{a \in \mathcal{A}} \mathop{\mathbb{E}}_{b \in B} \texttt{payoff}_1(a,b) \geq \max_{b \in \mathcal{B}} \mathop{\mathbb{E}}_{a \in A} \texttt{payoff}_1(a,b)$$

for any mixed strategies A and B.

Finally, given this inequality, we need only observe the following correspondence between algorithms and strategies and between inputs and strategies in order to yield Yao's principle, as follows:

$$\min_{r \in \mathcal{R}} \mathop{\mathbb{E}}_{x \in X} \texttt{cost}(r,x) \leq \max_{x \in \mathcal{X}} \mathop{\mathbb{E}}_{r \in R} \texttt{cost}(r,x)$$

for any distribution X of inputs and any distribution R of randomized algorithms.

- The set of moves for player 1 ($\mathcal{A}$) is the same as the set of deterministic algorithms ($\mathcal{R}$).
- The set of moves for player 2 ($\mathcal{B}$) is the same as the set of possible inputs ($\mathcal{X}$).
- A mixed strategy for player 1 (A) is the same as a randomized algorithm – that is, a probability distribution over deterministic algorithms (R).
- A mixed strategy for player 2 (B) is the same as a probability distribution over inputs to the algorithm (X).
- The payoff to player 1 (`payoff`) is the negated cost to player 1 ($-\texttt{cost}$).

2 Removable Online Knapsack Problem

In this section, we present the 2-competitive solution of Han et al. [?] to the removable online knapsack problem as well as use Yao's Principle show $1 + 1/e$ is a lower bound on the competitiveness of any algorithm against an oblivious adversary.

The knapsack problem asks: Given a set of items e_i with weights w_i and values v_i, select a subset with the maximum sum of values such that the sum of the weights is at most the capacity of the knapsack. We call this the value of the knapsack. The offline version is a classic NP-hard optimization problem, though it has a simple dynamic programming pseudo-polynomial solution. Here, we consider the *online* version, meaning each item is given to the algorithm after the algorithm makes a decision on the previous item. *Removable* means the knapsack is a working set, from which we can remove items and add new items, but not add previously removed items. At all times, the knapsack is limited by its capacity. The objective is to maximize the value of the final knapsack. We say an algorithm is α-competitive if the expected value of the final knapsack is at least $\frac{1}{\alpha}$ times the expected value of the optimal knapsack.

We assume the knapsack has capacity 1. This is valid because only weights relative to the knapsack's capacity matters for the capacity constraint, so we can always scale down to a unit capacity.

2.1 Upper Bound

To prove an upper bound on the competitiveness of algorithms which yield a solution to the problem, we present a 2-competitive randomized algorithm.

Since the online nature of the problem is what makes it impossible to always achieve the optimal solution, it is insightful to first consider algorithms which behave similarly in both online and offline situations. We look at MAX and $GREEDY$.

MAX simply selects the item with the largest value whose weight does not exceed 1. Since in the offline case, we can process each item in an arbitrary order and keep track of the best item so far, in the online case, we can do exactly the same with the order given to us, storing the best item so far in our knapsack.

$GREEDY$ selects the k items with the highest ratios of value to weight $\frac{v_i}{w_i}$, maximizing k while keeping the sum of the weights of the selected items at most 1. In the offline case, this can be done trivially by sorting the items by ratio and then selecting the highest k items. An online algorithm can perform at least as well by keeping a working set of the items with the highest ratio while processing each input.

Algorithm 1: online $GREEDY$

```
$S \leftarrow \emptyset$;
**foreach** *item $e_i$, in order of arrival* **do**
    **if** $w_i \leq 1$ **then**
        $S \leftarrow S \cup \{e_i\}$;
        **while** $\sum_{e_j \in S} w_j > 1$ **do**
            Remove the item from $S$ with the smallest ratio
```

Since when the sum of the weights in S is greater than 1, an item with ratio less than that of the k items with the largest ratios must be in S, the k items will never be removed. Thus S is a superset of the items the offline $GREEDY$ algorithm selects.

Neither of these algorithms alone are competitive. Consider an input sequence $\{(w_i, v_i)\}$ of $\{(\epsilon, 2\epsilon),\ (\epsilon, \epsilon),\ (\epsilon, \epsilon),\ \ldots\}$ on the MAX algorithm and $\{(\epsilon, 2\epsilon),\ (1, 1)\}$ on the $GREEDY$ algorithm. In fact, no deterministic online algorithm is competitive with a constant factor [?]. However, using both with randomness can yield a competitive algorithm.

Theorem 1. [?] *Running MAX and $GREEDY$ uniformly at random is a 2-competitive algorithm.*

Proof. For a given input sequence T, let $GREEDY(T)$, $MAX(T)$, and $OPT(T)$ be the values of the sets returned by $GREEDY$, MAX, and an optimal solution, respectively. Consider the fractional knapsack solution to the offline version, that is, when we are allowed to take fractional parts of items. Then a greedy solution is optimal: Sort by ratio and take the highest ratio items until the knapsack is full, potentially taking only part of the last item in the knapsack. In the fractional knapsack problem, we have relaxed the integrality constraint, so the fractional solution is not worse than the integral solution.

Now the offline $GREEDY$ algorithm by definition selects all of these except the fractional item (if one exists). Thus the online $GREEDY$ algorithm also selects all except the fractional item. Finally the MAX algorithm selects an item with value at least that of the fractional item, so $GREEDY(T) + MAX(T) \geq OPT(T)$. Thus the competitive ratio is $\frac{2 \cdot OPT(T)}{GREEDY(T)+MAX(T)} \leq 2$. □

2.2 Lower Bound

We use Yao's Principle to prove a lower bound of $1 + 1/e$ for the competitive ratio of any removable online knapsack algorithm.

To do so, we must find a distribution of inputs for which no deterministic algorithm performs "well", which means the expected value of the output from any deterministic algorithm is at most $\frac{1}{1+1/e}$ times the expected optimum. As the power of an offline algorithm is in the additional information it has over an online algorithm, it is reasonable that in constructing our "bad" input, we would want to limit the information the algorithm has about future items. Thus it is not surprising that we will construct a family of inputs where each input is a prefix of the same base sequence.

Let the probability of $k + 1$ items be p_k. For simplicity in analysis, we will let the first item $e_0 = (w_0, v_0) = (1, 1)$ with future items having much smaller weights and values but a higher ratio so that the problem is essentially reduced to determining when to remove the first item from the knapsack.

This is reminiscent of the *ski rental problem*, one of the most well-understood online problems [?], where a client must choose whether and when to buy skis for a fixed cost versus paying a smaller cost each day. Our knapsack problem is the reverse of this: instead of deciding when to switch from renting skis to buying skis to minimize total cost, we are deciding when to switch from having a single large item in our knapsack to getting a stream of smaller items with a better ratio to maximize total value.

Finally, we want the expected value of removing e_0 and selecting all following items to be close to 1 at any given point. A simple way to achieve this is to let the remaining items have a constant value and have the p_k form an exponential distribution.

This leads us towards the input sequence given by Han et al. [?]:

$$(1,1),\ (1/n^2, 1/n),\ (1/n^2, 1/n), \ldots\ (1/n^2, 1/n)$$

for a given value of n and where there are $k+1$ items with probability $p_k = \frac{1-e^{-1/n}}{1-e^{-n}} \cdot e^{-(k-1)/n}$ for $k = 1, 2, \ldots, n^2$.

Theorem 2. [?] *No online algorithm has competitive ratio less than* $1+1/e$ *for the removable online knapsack problem.*

Proof. We apply Yao's Principle on the input distribution above. The optimal strategy with full information is to keep e_0 for $k \leq n$ and remove e_0 immediately otherwise. This leads us to the expected optimal value

$$\sum_{i=1}^{n} 1 \cdot p_i + \sum_{i=n+1}^{n^2} \frac{i}{n} \cdot p_i = \frac{1-e^{-1/n}}{1-e^{-n}} \left(\sum_{i=1}^{n} e^{-(i-1)/n} + \frac{1}{n} \sum_{i=n+1}^{n^2} i \cdot e^{-(i-1)/n} \right)$$

which has value $1 + 1/e$ as $n \to \infty$.

Now an online deterministic algorithm can only decide on which item e_l to remove e_0. Then the expected value of this algorithm is

$$\sum_{i=1}^{l} 1 \cdot p_i + \sum_{i=l+1}^{n^2} \frac{i-l}{n} \cdot p_i = \frac{1-e^{-1/n}}{1-e^{-n}} \left(\sum_{i=1}^{l} e^{-(i-1)/n} + \frac{1}{n} \sum_{i=l+1}^{n^2} (i-1) \cdot e^{-(i-1)/n} \right)$$

which has value 1 as $n \to \infty$.

Thus any online algorithm has competitive ratio at least $1 + 1/e$. $\square$

3 Online Coloring Co-Interval Graphs

This section studies the online problem of coloring co-interval graphs as described by Zarrabi-Zadeh [**?**]. This problem is set up as follows:

Input A set of intervals on the real line in arbitrary order

Output A color matched to each interval satisfying the condition that any interval I_k must be assigned a color that's different from the color assigned to any I_j such that $j < k$ and intervals j and k don't intersect.

Since this is an online problem, the algorithm must assign colors as it goes, without knowing the complete set of intervals. That is, the algorithm will only receive information about I_{j+1} after it has assigned colors to all intervals I_k where $k \leq j$.

First, we present the First-Fit approximation algorithm to solve this problem, which has a competitive ratio of at most 2. Then, using Yao's Principle, we can prove that there exist no randomized algorithm that achieves a competitive ratio that's better than $\frac{3}{2}$.

3.1 Upper Bound

To solve the problem of assigning intervals different colors, the set of intervals is first constructed as a graph. That is, the input to our problem becomes:

$V :=$ Set of intervals where v_i represents the i'th interval

$E :=$ Set of edges where $(u, v) \in E$ iff interval u and interval v are disjoint

Note that any vertices that aren't connected by an edge can be colored in the same color since they overlap. Thus, in such a graph, our problem is reduced to finding the minimum number of colors such that no connected vertices are the same color.

The First-Fit algorithm is the simplest solution to this problem and does the following:

```
L = sorted list of colors
for a new vertex v:
    assign v the smallest color that doesn't cause a collision
```

Essentially, the algorithm maintains a list of colors that can be used. These colors are sorted in an arbitrary order. The first vertex is simply assigned the smallest color. For any future vertex v, the algorithm scans the list. We can define a collision between vertices u and v

at color k if u is colored k and v is connected to u by an edge in E. If a collision occurs where $L[i] = k$, then we increment i. We assign v the color at $L[j]$ where j doesn't cause any collisions and try to minimize j.

It has been shown by Gyarfas and Lehel [?] that First-Fit uses at most $2\omega + 1$ colors on any co-chordal graph, which is a class that includes co-interval graphs. It has also been shown by Kierstead and Qin [?] that no online deterministic algorithm can beat the 2-competitive ratio on co-interval graphs.

In the next section, we look at using randomization in co-interval graphs and lower bound the best competitive ratio that can be obtained.

3.2 Lower Bound

Yao's minimax principle proves that the expected competitive ratio on an optimal deterministic algorithm on the worst-case input is a lower bound on the expected competitive ratio of all randomized algorithms. That is, if there exists a set of inputs such that all optimal deterministic algorithms are at least α-competitive, then α is the best ratio that can be obtained by using randomization.

To define such a set of inputs, essentially, we need to find a set of intervals that satisfy the condition that any deterministic algorithm is at most $\frac{3}{2}$-competitive. The idea behind constructing such a set of intervals is that any deterministic algorithm will make the same decision on any specific interval given the same information, regardless of what the future intervals are.

3.2.1 Problem Construction

The intervals defined by Zarrabi-Zadeh are:

```
For $k \geq 1$:
    For all 1 $\leq i \leq k$:
        $a_i$ = [3$i$-3, 3$i$-2]
        $b_i$ = [3$i$+1, 3$i$+2]
        $c_i$ = [3$i$+2, $\infty$]
```

We can then define a block as: $B_i = \begin{cases} b_1, c_1 & \text{if } i \equiv 1 \\ a_i, b_i, c_i & \text{if } i \in [2,k] \\ a_k, b_k & \text{if } i \equiv k \end{cases}$

From this set of blocks, we can choose a set of k input sequences, simply by defining our input sequence X_i as the concatenation of blocks B_1, B_2... B_i, in order. Our set of all possible inputs then becomes the set:

$I = \{X_i \text{ where } i \in [1,k]\}$

It can be shown that the minimum number of colors required to color X_i without any edges sharing a color is i.

To use Yao's Principle on this input distribution, we need to show that if we set up a probability distribution over all possible X_i that the expected competitive ratio will be at least $\frac{3}{2}$. The optimal deterministic algorithm that we will examine per each X_i input is

First-Fit - as previously shown, this algorithm uses $2\omega - 1$ colors on each X_i. In this case, the chromatic number $\omega = i$, so, the expected number of colors for a given X_i is exactly $2i - 1$.

For a given X_i, a new color is chosen (assuming First-Fit) on each a_m and each b_m, since the blocks are given in order. A probability distribution P can be defined over all X_i, where p_i is the probability that X_i is chosen as input, as follows:

$$p_i = \begin{cases} \frac{2^k}{2^k-1} * \frac{i}{2^i} & \text{if } i < k \\ \frac{k}{2^k-1} & \text{if } i \equiv k \end{cases}$$

The expected competitive ratio of First-Fit on this distribution then becomes $\sum_{i=1}^{k} p_i \frac{2i-1}{i}$ since FF uses $2i-1$ colors, while the optimal number of colors that can be used is exactly i.

This sum becomes:

$$\rho = \frac{3}{2} - \frac{1}{2(2^k - 1)}$$

where ρ is the best competitive ratio that can be obtained by using First-Fit on this probability distribution.

If no deterministic algorithm can obtain a better competitive ratio on this set of inputs, then this is the best that a randomized algorithm could achieve.

3.2.2 Deterministic Algorithm Bounds

Theorem 3. *No deterministic algorithm can obtain a better competitive ratio on the set of inputs I*

Proof. To show this, we choose an arbitrary deterministic algorithm A. We can then define d_i as the decision that A makes on receiving interval c_i where $d_i = 1$ iff a new color is chosen for c_i. In an optimal solution, all c_i can be colored the same as b_i since they don't intersect, since all b_i don't intersect any preceding a_j or b_j, but intersect all previous c_j.

Given a sequence of decisions, $<d_1, d_2 ... d_{k-1}>$, then we can define a deterministic algorithm FF as follows:

If we receive an interval c_j, we check what d_j is and assign a new color iff $d_j = 1$. Else, we assign it the same color as b_j.

If we receive an interval of type a or b, we behave as the online First-Fit algorithm normally does.

Lemma 4. *Among all deterministic online algorithms, FF behaves optimally and chooses the minimum number of colors for any input I.*

Proof. This can be shown by induction, over the minimum number of colors on an input of size m. □

Lemma 5. *The competitive ratio of FF is ρ*

Proof. This proof follows by induction:

Define $P(m)$ as an upper bound for the competitive ratio of FF where there are m 1's in our sequence of decisions D. $P(0)$ holds trivially - in a sequence of all zero's, FF behaves exactly the same as First-Fit and our competitive ratio is still 2.

Using strong induction, we assume that P(k) is true for all $k \in [0, m-1]$.

By looking at the bits where FF_m would differ from FF_{m-1}, we can derive the formula that:

$$\rho_m = \rho_{m-1} + \rho_i(\frac{1}{i}) - \sum_{j=i+1}^{k} \rho_i(\frac{1}{i})$$

Substituting $\rho_i = \frac{2^k}{2^k-1} * \frac{i}{2^i}$ gives us that the last term sums to exactly $\rho_i(\frac{1}{i})$. This gives us that $\rho_m = \rho_{m-1}$ and the proof follows by induction. □

In turn, this implies that the expected competitive ratio of any deterministic algorithm is at most ρ, which shows that no deterministic algorithm has a better competitive ratio on I. □

By choosing k arbitrarily large, ρ tends to $\frac{3}{2}$.

Yao's Principle then gives us a lower bound of $\frac{3}{2}$ as the competitive ratio on any randomized algorithm.

4 Yao's Principle in Quantum Information

Yao's principle isn't just used for proving worst-case bounds on the competitiveness of algorithms. Another area where Yao's principle has found an application is in the study of certain games in quantum information, permitting proof of worst-case bounds for strategies in these games. In this section, we will examine how Yao's minimax principle may be used for this purpose, as demonstrated in *Worst Case Analysis of Non-local Games* by Ambainis et al. [?].

4.1 Quantum Preliminaries

Consider setting up a game between two players and an adversary, where the two players receive random input from the adversary and must each return specific outputs based on those inputs without communicating with each other. Analyzing the best strategies for these games, under the assumptions of both classical physics and quantum physics, has been used to demonstrate that better strategies exist if the players are permitted access to quantum information.

An example of such a game is the CHSH (Clauser-Horne-Shimonyi-Holt) game:

- The two players may collude to set up a strategy beforehand, but may not communicate afterwards.
- The adversary chooses a random $\vec{x} = (x_1, x_2) \in X = \{(0,0), (0,1), (1,0), (1,1)\}$. The random choice is uniform, with $\frac{1}{4}$ probability each.
- The players 1 and 2 receive the values x_1 and x_2, respectively.

- In the quantum setting, the players 1 and 2 may also possess an entangled 2-part quantum state $|\psi\rangle$, which they may measure.
- Using their strategy, the players 1 and 2 output the values $a_1(x_1)$ and $a_2(x_2)$, respectively.
- The players win if $a_1 \oplus a_2 = x_1 \wedge x_2$.

Both under classical physics and under any "local hidden variable theory" – explanation of quantum phenomena as being due to extra local hidden variables – the best that the players can achieve is a win probability of 0.75. If the players are allowed to share entangled quantum bits, however, then they can win more often: $\frac{1}{2} + \frac{1}{2\sqrt{2}} = 0.8535...$ becomes the best win probability that the players can achieve [?, ?].

The CHSH game was designed so as to be an actual realizable experiment, and experiments have since confirmed that the way the universe works is consistent with quantum entanglement, and not consistent with local hidden variable theories [?]

In *Worst Case Analysis of Non-local Games* by Ambainis et al., the authors examine generalized games beyond the simple CHSH game, and analyze the worst-case performance of strategies in both classical and quantum physics [?].

4.2 Generalized CHSH

The CHSH game is played between two players and an adversary; one simple generalization that can be made is to extend the game from two players to n players. Ambainis et al. call this the *n-party AND game*.

An additional generalization is possible: the n-party AND game is similar to the CHSH game because of its win condition, $\bigoplus \vec{a} = \bigwedge \vec{x}$. We can generalize this game further still by permitting arbitrary win conditions.

- The n players may collude to set up a strategy beforehand, but may not communicate afterwards.
- The adversary chooses a random $\vec{x} = (x_1, ..., x_n) \in X_1 \times ... \times X_n$, for some $X_1, ..., X_n$. The random choice is governed by a probability distribution π; each $\vec{x}$ is chosen with probability $\pi(\vec{x})$.
- The players $1, ..., n$ receive the values $x_1, ..., x_n$, respectively.
- In the quantum setting, the players may also possess an entangled n-part quantum state $|\psi\rangle$, which they may measure.
- Using their strategy, the players $1, ..., n$ output the values $\vec{a} = a_1, ..., a_n$, respectively.
- The players win if some proposition $\texttt{win}(\vec{a}|\vec{x})$ is true. Let the indicator variable $\texttt{win}(\vec{a}|\vec{x}) = 1$ if they win, -1 otherwise.

For any game G, let $\omega^\pi(G) \equiv Pr(Win) - Pr(Lose)$ when π is a probability distribution according to which the adversary chooses $\vec{x}$. Let $\omega(G)$ be the worst case of $\omega^\pi(G)$: the minimum value of $\omega^\pi(G)$ that the adversary can achieve by selectively choosing π. Let ω_q and ω_c be ω in the quantum and classical cases, respectively. We defined the indicator variable $\mathtt{win}(\vec{a}|\vec{x})$ such that $Pr(Win) - Pr(Lose) = \mathbb{E}\,\mathtt{win}(\vec{a}|\vec{x})$. Therefore, we can more formally define $\omega(G)$ and $\omega^\pi(G)$ as follows:

$$\omega^\pi(G) \equiv \max_{\vec{a}} \mathop{\mathbb{E}}_{\vec{x}\in\pi} \mathtt{win}(\vec{a}|\vec{x})$$

$$\omega(G) \equiv \max_{\vec{a}} \min_{\pi} \mathop{\mathbb{E}}_{\vec{x}\in\pi} \mathtt{win}(\vec{a}|\vec{x})$$

One research pursuit in quantum information has been finding games that show the greatest possible separation between the quantum strategy and the classical strategy: seeking to maximize the ratio $\frac{\omega_q^\pi(G)}{\omega_c^\pi(G)}$ [**?**]. Interestingly, no matter what the proposition `win` is, the separation ratio $\frac{\omega_q(G)}{\omega_c(G)}$, in the case where the adversary chooses the worst possible distribution π *separately* for *each* of the classical case and the quantum case, is never bigger than the maximum value of $\frac{\omega_q^\pi(G)}{\omega_c^\pi(G)}$ for a fixed probability distribution π. This proof uses Yao's minimax principle.

Theorem: $\frac{\omega_q(G)}{\omega_c(G)} \leq \max_\pi \frac{\omega_q^\pi(G)}{\omega_c^\pi(G)}$.

Proof. As with more familiar applications of Yao's minimax principle, here the adversary is still choosing a strategy over a distribution of inputs $\vec{x}$ to pass to the players, and the players may still use randomized algorithms as strategies. However, instead of using *running time* as the cost which the adversary attempts to maximize and the algorithm attempts to minimize, we have *probability of winning* as the cost which the adversary attempts to minimize and the two players' algorithms attempt to maximize.

Recall Yao's minimax principle: Instead of *the best deterministic algorithm, when run on any (or the worst) distribution over random inputs, is at least as good as any (or the best) randomized algorithm when run on the worst-case input*, we will have that *the players' best deterministic algorithms, when run the worst distribution π over random inputs from the adversary, is at least as good as the best randomized algorithm when run on the worst-case input.*

The players' best deterministic algorithm against any the worst distribution π over random inputs from the adversary is $\max_{\vec{a}} \min_\pi \mathbb{E}_{\vec{x}\in pi}\, \mathtt{win}(\vec{a}|\vec{x})$, which is defined to be $\omega_c(G)$. The players' best randomized algorithm against the worst input is $\min_\pi \max_{\vec{a}} \mathbb{E}_{\vec{x}\in pi}\, \mathtt{win}(\vec{a}|\vec{x})$, which is $\min_\pi \omega_c^\pi(G)$.

This gives us a bound on the worst-case win probability under classical conditions: $\omega_c(G) \geq \min_\pi \omega_c^\pi(G)$. Suppose π is the probability distribution that achieves this minimum; then $\omega_q^\pi(G) \geq \omega_q(G)$, because fixing π can only allow the players to achieve a better win probability than their worst-case $\omega_q(G)$.

□

5 Conclusion

We have demonstrated Yao's Principle to give lower bounds on the competitiveness of algorithms for a couple of online games, which is an upper bound on their effectiveness. When these bounds match the competitiveness of a particular algorithm, can be used to show the tightness of the algorithm.

Also, there is a dual between using an algorithm to demonstrate a lower bound on the solvability of a problem (alternatively, an upper bound on the hardness of a problem), and using a series of inputs to demonstrate an upper bound on the solvability of a problem (alternatively, an lower bound on the hardness). As more work is done on a particular problem, the gap between the two shrink until a provably optimal algorithm is found.

Yao's Principle is applicable to many types of costs when analyzing problems, including running time, and, as we saw here, competitiveness of algorithms in partial information scenarios.