CS 201: Adversary arguments

This handout presents two lower bounds for selection problems using adversary arguments [Knu73, HS78, FG76]. In these proofs an imaginary adversary called an *oracle* is created to force an algorithm to use the maximum number of comparisons to solve the problem. The adversary's task is to determine the outcomes of comparisons in a manner which impedes the progress of the algorithm, and yet is still *consistent*. The outcomes of comparisons are consistent as long as there are never three keys A, B and C with $A < B$, $B < C$ and $C < A$. That is, at anytime the comparisons form a partial order on the keys.

A selection algorithm can be viewed as a tournament in which n players participate in a sequence of pairwise matches. Assuming that each player has a distinct skill level and always plays at this level, the outcomes of matches are determined solely by these skill levels. These skill levels are not known. Only by playing a match can the skill levels of two players be compared. The better player always wins the match and the outcomes are always consistent since they are based on a (hidden) total ordering of the players.

The three quantities of interest in selection problems are:

1. $V_k(n)$, the number of comparisons necessary to determine the k^{th} player out of n players,
2. $U_k(n)$, the number of comparisons necessary to determine the set consisting of the top k players out of n players, and
3. $W_k(n)$, the number of comparisons necessary to determine the $1^{st}, 2^{nd}, \ldots, k^{th}$ players out of n players.

These three quantities are related as follows:

$$U_k(n) \leq V_k(n) \leq W_k(n).$$

The first inequality follows from the observation that if you have determined through comparisons that x is the k^{th} player then you also must have identified the $k-1$ players better than x as well as the $n-k$ players worse than x.

In the next section it is shown that $W_2(n) = V_2(n) = n - 2 + \lceil \log_2 n \rceil$, and in the following section it is shown that $\lceil \frac{3n}{2} \rceil - 2$ matches is necessary and sufficient for finding the top and bottom players. More recent results are presented in [BJ85].

Finding the top two players

Result 1: The top two players can be found in $n - 2 + \lceil \log_2 n \rceil$ matches.

Proof: Consider the following algorithm (schedule of matches) to find the first and second players:

1. First hold a single elimination tournament for the n players. That is, pair up the players in $\lfloor \frac{n}{2} \rfloor$ matches, and repeat the process with the undefeated players until one undefeated player, x, remains.
2. Play a second single elimination tournament among the m players who were defeated by x in the first tournament.

Since each player loses at most once in a single elimination tournament and at the end only one player is undefeated, there are exactly $n-1$ losses and hence $n-1$ matches. So the first tournament has exactly $n-1$ matches. Likewise, the number of matches in the second tournament is $m-1$ where m is the number of players who participated, lost to x in the first tournament. To determine m, consider the number of rounds played in the first tournament. In each round of a single elimination tournmanent the players are paired and $\lceil \frac{n}{2} \rceil$ players remain for the next round. Hence there will be at most $\lceil \log_2 n \rceil$ rounds. In each round x will defeat one player, so $m = \lceil \log_2 n \rceil$. Summing the number of matches played in the two tournaments we obtain,

$$n - 2 + \lceil \log_2 n \rceil.$$

□

Result 2: Any algorithm for finding the first and second players will require at least $n - 2 + \lceil \log_2 n \rceil$ matches.

Proof: Suppose we have an arbitrary algorithm $\mathcal{A}$ for finding the first and second player. Since $\mathcal{A}$ is completely arbitrary we can assume nothing about the matches that $\mathcal{A}$ arranges. We invent a fictitious adversary for $\mathcal{A}$ called an *oracle* as follows. As the matches organized by $\mathcal{A}$ are played our oracle keeps track of the following information:

1. let **TOP** = {currently undefeated players},
2. for any player $x \in$ **TOP**, let $Dom(x, 0) = \{x\}$
3. for any player $x \in$ **TOP** and $i > 0$,
   let $Dom(x, i) =$ {players whose first defeat was to a player in $Dom(x, i-1)$ },
4. let $Dom(x) = \bigcup_{i=0}^{\infty} Dom(x, i)$, and
5. let $Win(x) = |Dom(x, 1)|$.

Our oracle uses the following rules to determine the outcome of matches:

1. In a match between two players currently in **TOP**, the player with the larger $Win()$ value will win. If the two players have the same $Win()$ value then the winner is selected arbitrarily.
2. In a match between a player currently in **TOP** and a player not in **TOP**, the player in **TOP** wins.
3. If neither player is currently in **TOP**, then the winner of the match is selected arbitrarily in a manner consistent with the outcome of previous matches.

The outcomes of the matches determined by our oracle are always consistent since the winner is always an undefeated player in the first two rules and the outcome in rule 3 is selected to ensure consistency.

Initially each player y belongs to $Dom(y)$ and **TOP**. When y suffers its first loss, the oracle's second rule ensures that y loses to player z which is also in **TOP**. So y along with all of the other players in $Dom(y)$ join $Dom(z)$. Hence at anytime during the matches each player belongs to $Dom(z)$ for some player z currently in **TOP**. When the algorithm terminates there is only one player in **TOP**, say x, and all of the players are in $Dom(x)$.

Claim: After k matches any player x still in **TOP** satisfies,

$$|Dom(x)| \leq 2^{Win(x)}. \tag{1}$$

Proof of Claim: By induction on the number of matches, k.

Base: $k = 0$. No matches have been played so $|Dom(x)| = 1 = 2^0 = 2^{Win(x)}$ for all players.

Step: Assume the claim is true after the first $k-1$ matches and consider the k^{th} match and a player x still in TOP after this match. Note that since x is still undefeated after the k^{th} match, x either won or did not play in the k^{th} match.

Case 1 x did not play in the k^{th} match.
In this case, we had $|Dom(x)| \leq 2^{Win(x)}$ before the match. $Win(x)$ doesn't change since x doesn't play. Is it possible for $Dom(x)$ to change? The answer is no since a player would join $Dom(x)$ only if it is currently in **TOP** and loses to a player in $Dom(x)$ (other than x in this case). However, such a loss is not possible according to the oracle's second rule. Since neither $Dom(x)$ nor $Win(x)$ changes, Equation (1) still holds after the match.

Case 2 x played y who was not in **TOP** before the k^{th} match.
Since y was not in **TOP**, again neither $Dom(x)$ nor $Win(x)$ changes since this is not y's first defeat and Equation (1) still holds after the match.

Case 3 x played y who was in **TOP** before the k^{th} match.
According to the oracle's first rule, we have $Win(x) \geq Win(y)$ since x won. After the match $Dom(x)$ will become $Dom(x) \cup Dom(y)$ and we have,

$$|Dom(x) \cup Dom(y)| \leq |Dom(x)| + |Dom(y)| \leq 2^{Win(x)} + 2^{Win(y)} \leq 2 \cdot 2^{Win(x)} = 2^{Win(x)+1}.$$

Since after the match $Win(x)$ will go up by 1, the Equation (1) will still hold.

□

When algorithm $\mathcal{A}$ terminates there will be only one player in **TOP**, say x. Since all players will be in $Dom(x)$, we have

$$2^{Win(x)} \geq |Dom(x)| = n,$$

or

$$Win(x) \geq \lceil \log_2 n \rceil.$$

Consider the players in $Dom(x, 1)$. There are $Win(x)$ of them. Each of them suffered their first defeat to x. Anyone of these players that does not lose a second match, could be the second best player. Algorithm $\mathcal{A}$ cannot rule them out as second best players, since they have only been shown to be worse than x. In order to determine the second best player, algorithm $\mathcal{A}$ must have arranged matches in which all but one of the players in $Dom(x, 1)$ lost a second time. The number of matches played corresponds exactly to the number of losses. There are $n-1$ first losses and at least $Win(x) - 1$ second losses, so the number of matches used by $\mathcal{A}$ is at least

$$n - 2 + \lceil \log_2 n \rceil.$$

□

Finding the top and bottom players

Result 3: The top and bottom players can be found in $\lceil \frac{3n}{2} \rceil - 2$ matches.

Proof: Consider the following algorithm (schedule of matches) to find the first and last players:

1. First hold a single elimination tournament for the n players. That is, pair up the players in $\lfloor \frac{n}{2} \rfloor$ matches, and repeat the process with the undefeated players until one undefeated player, x, remains.

2. Collect the $\lceil \frac{n}{2} \rceil$ players who did not win in the first round of the previous tournament. (If n is odd this will include the player who did not play in the first round.) Play a tournament with these players in which the loser advances to the next round rather than the winner. That is, pair up the players in matches, and repeat the process with the winless players until one winless player, z, remains.

After the first tournament all but one player is undefeated so the top player has been determined in $n-1$ matches. The only players who have not won any matches are those that lost or possibly did not play in the first round. All the other players have won at least one match and therefore cannot possibly be the worst player. So to determine the worst player, matches are played among the at most $\lceil \frac{n}{2} \rceil$ players who are still winless. After the second tournament all but one player, z, will have won a match. Only z will be winless and so z is the worst player. The second tournament requires $\lceil \frac{n}{2} \rceil - 1$ matches so altogether, a total of $\lceil \frac{3n}{2} \rceil - 2$ matches are played. □

Result 4: Finding the top and bottom players requires $\lceil \frac{3n}{2} \rceil - 2$ matches.

Proof: As before we assume an arbitrary algorithm $\mathcal{A}$ for finding the first and last players and we invent a fictitious adversary for $\mathcal{A}$ called an *oracle*. During the sequence of matches arranged by $\mathcal{A}$ we keep track of the following types of players.

U is the group of players that have never played; they are untested.

W is the group of players that have played and never lost; they are winners.

L is the group of players that have played and never won; they are losers.

B is the group of players that have both won and lost matches; they are both losers and winners.

After m matches we will represent the *state* of the tournament by (u, w, l, b) where u, w, l, and b are the numbers of players in **U**, **W**, **L**, and **B** respectively. The initial state before any matches are played is $(n, 0, 0, 0)$ and at the end, when the top and bottom players are known, the state must be $(0, 1, 1, n-2)$. How the state changes after a match depends on the group memberships of the winner and loser; the state transistions are given in Table 1.

If winner is in	If loser is in **U**	**W**	**L**	**B**
U	$(u-2, w+1, l+1, b)$	$(u-1, w, l, b+1)$	$(u-1, w+1, l, b)$	$(u-1, w+1, l, b)$
W	$(u-1, w, l+1, b)$	$(u, w-1, l, b+1)$	(u, w, l, b)	(u, w, l, b)
L	$(u-1, w, l, b+1)$	$(u, w-1, l-1, b+2)$	$(u, w, l-1, b+1)$	$(u, w, l-1, b+1)$
B	$(u-1, w, l+1, b)$	$(u, w-1, l, b+1)$	(u, w, l, b)	(u, w, l, b)

Table 1: If the current state is (u, w, l, b) then after a match the next state can be found in the row and column corresponding to the groups (before the match) containing the match's winner and loser.

Our oracle tries to impede the progress of the algorithm by minimizing the change in state. It decides outcomes of matches as follows:

1. A player in **W** always wins against a player not in **W**.

2. A player in **L** always loses against a player not in **L**.

3. Otherwise the winner is selected arbitrarily in a manner consistent with the outcome of previous matches.

The outcome is consistent since players in **L** can always lose without violating consistency and players in **W** can always win without violating consistency. In Table 2 the outcomes prohibited by the oracle have been noted. All n players start in **U**, $n-2$ must move to **B** while one player moves to **W** and

If winner is in group	If loser is in group **U**	**W**	**L**	**B**
U	$(u-2, w+1, l+1, b)$	not possible	$(u-1, w+1, l, b)$	$(u-1, w+1, l, b)$
W	$(u-1, w, l+1, b)$	$(u, w-1, l, b+1)$	(u, w, l, b)	(u, w, l, b)
L	not possible	not possible	$(u, w, l-1, b+1)$	not possible
B	$(u-1, w, l+1, b)$	not possible	(u, w, l, b)	(u, w, l, b)

Table 2: Table 1 modified to reflect state changes which the oracle will not allow.

one to **L**. We can view each match as transferring units from u to b, leaving one unit behind in both w and l. In the remaining possible state changes, there is no match which transfers units from u to b, so transferring a unit from u to b must be accomplished by transferring it first to w or l and then to b. Hence a total $2(n-2)+2 = 2n-2$ unit transfers must occur. All matches transfer at most one unit with the exception of matches between two players in **U**, but at most $\lfloor \frac{n}{2} \rfloor$ matches of this type can be played. The remaining $2n-2-2\lfloor \frac{n}{2} \rfloor$ units would have to be transferred one at a time. Thus the minimum number of matches necessary to accomplish all the transfers is

$$\left\lfloor \frac{n}{2} \right\rfloor + 2n - 2 - 2\left\lfloor \frac{n}{2} \right\rfloor = 2n - \left\lfloor \frac{n}{2} \right\rfloor - 2 = \left\lceil \frac{3n}{2} \right\rceil - 2.$$

□

References

[BJ85] Samuel W. Bent and John W. John. Finding the median requires $2n$ comparisons. In *Proceedings of the* 17^{th} *ACM Symposium on Theory of Computing*, pages 213–216, 1985.

[FG76] Frank Fussenegger and Harold N. Gabow. Using comparison trees to derive lower bounds for selection problems. In *Proceedings of the* 17^{th} *IEEE Symposium on Foundations of Computer Science*, pages 178–182, 1976.

[HS78] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Rockville, Maryland, 1978.

[Knu73] Donald E. Knuth. *The Art of Computer Programming: Sorting and Searching*. Addison Wesley, Reading, Massachusetts, 1973.