STUDY GROUP
RANDOMIZED ALGORITHMS

June 28, 2003
Topics to be Covered

- Recap of Game Tree Evaluation Problem
- Yao’s Technique
- Lower Bound for Game Tree Evaluation
Recap of Game Tree Evaluation Problem

The expected cost of evaluating any instance of $T_{2,k}$ is at most $3^k$, in other words, the expected running time of the randomized algorithm is $n \log_4 3 = n^{0.793}$. (note: $n = 4^k$ is the number of leaves in $T_{2,k}$)
Yao’s Technique

• How good is your randomized algorithm?
  – *Deterministic Algorithm*: Your opponent can always choose an input which force you to evaluate all the $4^k$ leaves.
  – *Randomized Algorithm*: No matter how your opponent chooses the input, you can always guarantee the *expected* cost is evaluating no more than $3^k$ leaves.
  – Is there any *better* algorithms?

Yao’s Technique is a general technique for proving the lower bound on the running time of randomized algorithms.
Apply the game theory to analyze the complexity:

<table>
<thead>
<tr>
<th>Player</th>
<th>Role</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>“input provider”</td>
<td>max runtime of given algorithm</td>
</tr>
<tr>
<td>column</td>
<td>“algorithm designer”</td>
<td>min runtime of given input</td>
</tr>
</tbody>
</table>

Example of the payoff matrix:

<table>
<thead>
<tr>
<th></th>
<th>$Alg_1$</th>
<th>$Alg_2$</th>
<th>$\cdots$</th>
<th>$Alg_{100}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Input_1$</td>
<td>20</td>
<td>20</td>
<td>$\cdots$</td>
<td>0</td>
</tr>
<tr>
<td>$Input_2$</td>
<td>5</td>
<td>53</td>
<td>$\cdots$</td>
<td>-24</td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$Input_{1000}$</td>
<td>95</td>
<td>-89</td>
<td>$\cdots$</td>
<td>-30</td>
</tr>
</tbody>
</table>
von Neumann’s Minimax Theorem:

\[
\max_p \min_q p^T M q = \min_q \max_p p^T M q
\]  

(1)

Loomis’ Theorem:

\[
\max_p \min_j p^T M e_j = \min_q \max_i e_i^T M q
\]  

(2)

\(\mathcal{I}\): set of input instances, \(\mathcal{A}\): set of deterministic algorithms  
\(C(I, A)\): running time of algorithm \(A \in \mathcal{A}\) on input \(I \in \mathcal{I}\)  
\(I_p\): random input chosen according to prob distribution \(p\) over \(\mathcal{I}\)  
\(A_q\): random algorithm chosen according to \(q\) over \(\mathcal{A}\)

\[
\max_p \min_q \mathbb{E}[C(I_p, A_q)] = \min_q \max_p \mathbb{E}[C(I_p, A_q)]
\]

\[
\max_p \min_{A \in \mathcal{A}} \mathbb{E}[C(I_p, A)] = \min_q \max_{I \in \mathcal{I}} \mathbb{E}[C(I, A_q)]
\]
**Yao’s Minimax Principle**

For all distributions $p$ over $\mathcal{I}$ and $q$ over $\mathcal{A}$,

$$\min_{A \in \mathcal{A}} E[C(I_p, A)] \leq \max_{I \in \mathcal{I}} E[C(I, A_q)] \quad (3)$$

LHS: The expected running time of the best deterministic algorithm for an arbitrarily chosen input distribution $p$.

RHS: The expected running time of an arbitrarily chosen randomized algorithm for the worst-case input.
Lower Bound for Game Tree Evaluation

1. Pick a distribution \( p \) of inputs over \( \mathcal{I} \)
2. Find the best deterministic algorithm \( A \) for this distribution
3. Calculate the runtime of \( A \)

For simplicity, we first transform the AND-OR tree to a NOR tree. By DeMorgan’s Law, we then have:

\[
(x_1 \lor x_2) \land (x_3 \lor x_4) = \neg((\neg(x_1 \lor x_2) \lor \neg(x_3 \lor x_4)))
\]  

Apply this to the AND-OR tree recursively, we can transform it into a tree with all the internal nodes being NOR gates.
\begin{center}
\begin{tikzpicture}
  \node (root) [circle, draw] {0} child {node (and) [circle, draw] {AND} child {node (or1) [circle, draw] {OR} child {node (or11) [circle, draw] {0} child {node (or111) [circle, draw] {0}} child {node (or112) [circle, draw] {1}}} child {node (or12) [circle, draw] {1}}}} child {node (or2) [circle, draw] {OR} child {node (or21) [circle, draw] {0} child {node (or211) [circle, draw] {0}} child {node (or212) [circle, draw] {0}}} child {node (or22) [circle, draw] {0}}};
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tikzpicture}
  \node (norr) [circle, draw] {0} child {node (nor1) [circle, draw] {NOR} child {node (nor11) [circle, draw] {0} child {node (nor111) [circle, draw] {0}} child {node (nor112) [circle, draw] {1}}} child {node (nor12) [circle, draw] {1}}};
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tikzpicture}
  \node (nor) [circle, draw] {0} child {node (nor1) [circle, draw] {NOR} child {node (nor11) [circle, draw] {0} child {node (nor111) [circle, draw] {0}} child {node (nor112) [circle, draw] {1}}} child {node (nor12) [circle, draw] {1}}};
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tikzpicture}
  \node (root) [circle, draw] {0} child {node (and) [circle, draw] {AND} child {node (or1) [circle, draw] {OR} child {node (or11) [circle, draw] {0} child {node (or111) [circle, draw] {0}} child {node (or112) [circle, draw] {1}}} child {node (or12) [circle, draw] {1}}}} child {node (or2) [circle, draw] {OR} child {node (or21) [circle, draw] {0} child {node (or211) [circle, draw] {0}} child {node (or212) [circle, draw] {0}}} child {node (or22) [circle, draw] {0}}};
\end{tikzpicture}
\end{center}
1. **How to pick the distribution $p$?**

   As to ease the analyze, we choose a uniform distribution for each leaf node to have a probability of $p = (3 - \sqrt{5})/2$ to be the value 1.

2. **How to find the best deterministic algorithm $A$?**

   DFS (depth-first search) is optimal! The intuition is that when we evaluate the tree in depth-first fashion, we can prune it as early as possible. Tarsi [393] gives a formal proof.

3. **How to calculate its expected ruining time?**

   Let $W(h)$ denote the expected number of leaves the algorithm need to inspect so as to determining the value of a node at distance $h$ from the leaves.

   
   $W(h) = W(h - 1) + (1 - p)W(h - 1)$

   
   $= (2 - p)W(h - 1)$
Therefore

\[ W(h) = (2 - p)^h \]  
(since \( W(0) = 1 \))

\[ = (2 - p)^{2k} \]

\[ = (2 - p)^{2\log_4 n} \]  
(since \( n = 4^k \))

\[ = \left( 2 - \frac{3 - \sqrt{5}}{2} \right)^{2\log_4 n} \]

\[ = n^{0.694} \]

But \( n^{0.694} < n^{0.793}! \)

Actually it is because of the probability distribution \( p \) we chooses. Saks and Wigderson [362] gives a better lower bound and shows \( n^{0.793} \) is optimal.
Conclusions

- Yao’s technique applies game theoretic approach to complexity analysis of randomized algorithms.
- It is the only known general technique to prove the lower bound on the running time of randomized algorithms.
- The choice of input distribution is subtle.