UNM Physics 452/581: Introduction to Quantum Information, Problem Set 4, Fall 2007

Instructor: Dr. Landahl
Issued: Wednesday, September 26, 2007
Due: Thursday, October 4, 2007

Do all of the problems listed below. Hand in your problem set at the beginning of class on the desk at the front of the classroom or after class in the box in the Physics and Astronomy main office by 5 p.m. **Please put your name and/or IQI number number on your assignment**, as well as the course number (Physics 452/581). Please show all your work and write clearly. Credit will be awarded for clear explanations as much, if not more so, than numerical answers. Avoid the temptation to simply write down an equation and move symbols around or plug in numbers. Explain what you are doing, draw pictures, and check your results using common sense, limits, and/or dimensional analysis.

4.1. ΘΩO frat initiation

The co-ed fraternity ΘΩO is renowned for its great parties and successful alums. After college, most members go on to be leaders in business and industry—alums tend to have a knack for seeing where ideas will lead in the long run. Perhaps it has something to do with the unusual ΘΩO initiation process, which some say borders on hazing.

After going to some ΘΩO parties and having a good time, you decide to rush the frat. Eventually, the time comes for initiation. A dark hood is placed over your head, you're thrown into a van, driven to a seemingly remote location, dragged into a room and your hood is removed. The room is spartan, containing only a desk, a chair, a pencil, and a piece of paper. On the piece of paper, you read the following set of instructions.

To join ΘΩO, you must demonstrate your knowledge of the secrets of Θ, Ω, and $\mathcal{O}$ notations. For functions f and g, these notations are defined as:

$$f = \mathcal{O}(g) \Leftrightarrow (\exists N_0 > 0)(\exists k > 0)(\forall N > N_0)[f(N) \leq kg(N)]$$

$$f = \Omega(g) \Leftrightarrow (\exists N_0 > 0)(\exists k > 0)(\forall N > N_0)[f(N) \geq kg(N)]$$

$$f = \Theta(g) \Leftrightarrow [(f = \mathcal{O}(g)) \wedge (f = \Omega(g))]$$

Use these notations to rank the following functions from slowest-growing to fastest-growing. Put a $\leq$ symbol between functions f and g if $f = \mathcal{O}(g)$ (or equivalently if $g = \Omega(f)$) and put an $=$ symbol between functions f and g if $f = \Theta(g)$.

$$n, \quad n\log_2 n, \quad (\log_2 n)^n, \quad (\ln n)^n, \quad n^{\ln n}, \quad 2^n, \quad n!, \quad \binom{2n}{n}, \quad \ln(n!), \quad \ln\binom{2n}{n}.$$

Your first thought is "Do I really want to join this fraternity?" Nevertheless, you remember how great those parties were and decide to answer the initiation question. Fortunately for you, you've seen these notations before in your IQI class, and you remember Stirling's approximation, which says that $n! \approx \sqrt{2\pi n}(n/e)^n$. With this preparation, you feel confident you'll answer the question correctly and pass this initiation test.

(a) Answer the initiation question.

After you finish answering the question, the hood is placed back over your head and you are once again whisked away and sat down in a chair in what seems to be a new room. You hear the sounds of people rustling and murmuring; you guess that these are frat members. Finally, you recognize the fraternity president's voice as he speaks to you, "You have passed the initiation test. But to be a true member, you must also learn the darkest secret of our fraternity. Not all functions can be ranked using Θ, Ω and $\mathcal{O}$ notation. There are some functions such that neither $f = \mathcal{O}(g)$ nor $g = \mathcal{O}(f)$. Tell me an example of such a pair of functions and your initiation will be complete."

(b) Answer the frat president's question.

You answer the question successfully and, after a round of applause, your hood is removed and you recognize the other frat members—you are in the $\Theta\Omega$O main party room. The president gets everyone to settle down, congratulates you, and offers you a chance to join the fraternity leadership committee. He tells you, "To be on the committee, you have to show not just understanding but true mastery of the Θ, Ω and $\mathcal{O}$ notations. To prove you are worthy to be on this committee, answer the following question about one of our most secret functions. It is the function f defined so that $f(f(N)) = 2^N$. Where does f fit in the ranking you did for the initiation exam?"

(c) (Extra credit.) Answer the frat president's question to join the leadership committee.

4.2 Dr. Falken, Would you like to play a game?

In the hacker-movie classic, *War Games*, a computer known as "Joshua" or "The WOPR" is programmed to learn from playing games. In the film's climax, the computer learns that the only way to win the "game" of Global Thermonuclear War is "not to play." In this problem, we examine just how hard it is for a computer to evaluate whether the first or second player of a two-player game has a winning strategy.

Consider a two-player game in which the players, whom we shall call Alice and Bob (or A and B), alternate public moves until a final game configuration is reached in which one player wins while the other loses. Such a game is called a *finite perfect-information zero-sum game*. These kinds of games can be described by a *game tree*: a rooted tree in which each child node corresponds to a game configuration reachable by a possible move from the parent node. Each leaf of a game tree is given the value 0 or 1 depending on whether or not the correpsonding configuration indicates a win for player A or B respectively. The *value* of a game tree is a bit, 0 or 1, indicating whether or not the first player, whom we take to be Alice without loss of generality, wins or loses the game when she plays optimally.

In the scenario just described, the computational question we are interested in is how difficult it is to evaluate the game tree's value, which is manifestly a function of the game

tree leaf values, *i.e.*, $g = g(x_0, \ldots, x_N)$ where N is the number of leaves in the tree. In particular, we are interested in the *query complexity* of this problem: how many leaf values need to be queried in the worst case to evaluate the game tree value? By focusing on the query complexity of this problem, we abstract away the details of how the terminal game configurations are evaluated. To simplify the analysis of this problem, in this exercise we will confine our attention to games which always terminate after exactly the nth move and where exactly k moves are possible from any nonterminal game configuration.

It is straightforward to see that the value of a game tree is 0 if and only if there exists a first move for Alice such that for every first move for Bob, there exists a second move for Alice such that for every second move for Bob, ..., there exists a last move by Alice that is a win for Alice. (Or if Bob moves last, there exists a last move by Alice such that for every last move by Bob, Alice wins.) This alternation of "for all"/"exists" reasoning is captured by evaluating the function that is the k-bit OR of the k-bit AND of the k-bit OR of the k-bit AND, etc. For this reason, game trees are sometimes called AND-OR trees.

(a) Game trees are also sometimes also called NAND trees, because, up to a possible negation of inputs and outputs, the alternating application of ANDs and ORs yields the same result as if NAND had been applied at each level. Prove this for the case when the fanout of the tree is $k = 2$. That is, prove that (i)

$$(a \wedge b) \vee (c \wedge d) = (a \mathbin{\bar{\wedge}} b) \mathbin{\bar{\wedge}} (c \mathbin{\bar{\wedge}} d),$$

and (ii)

$$\neg \left((\bar{a} \vee \bar{b}) \wedge (\bar{c} \vee \bar{d}) \right) = (a \mathbin{\bar{\wedge}} b) \mathbin{\bar{\wedge}} (c \mathbin{\bar{\wedge}} d),$$

where $\bar{\wedge}$ denotes the NAND gate and both $\neg\, x$ and $\bar{x}$ denote NOT(x).

(*Hint*: You can prove this by exhausting all possible bit values (the symmetry of the functions makes casing this out simpler than you might think), but it is also possible to prove this algebraically using *de Morgan's laws*, which say that $\neg(x \vee y) = \bar{x} \wedge \bar{y}$ and $\neg(x \wedge y) = \bar{x} \vee \bar{y}$.)

(b) An optimal Las Vegas classical algorithm for evaluating a NAND tree value is called *depth-first pruning*. Let's call this algorithm $A(n, k)$ when it is applied to a depth-n, fanout-k NAND tree. The way $A(n, k)$ works is as follows. Starting at the root of the tree, one of the k children is selected uniformly at random. The selected child is itself the root of a NAND subtree that is evaluated recursively using $A(n-1, k)$. If the value returned is sufficient to determine $A(n, k)$, then the algorithm stops and returns that value. Otherwise, one of the remaining children is selected uniformly at random and evaluated recursively using $A(n-1, k)$. This process repeats until the value of $A(n, k)$ is determined.

Recall that the way that query complexity is measured for a Las Vegas algorithm is by the expected number of queries it must make for the worst-case input. Let $C_b(n, k)$ denote the query complexity of $A(n, k)$ when the NAND tree evaluates to b, where b is either 0 or 1.

Given the description of the depth-first pruning (DFP) algorithm, asserted to be optimal in that it uses the smallest possible query complexity for this problem, show that the following

relations hold

$$C_0(n,k) = kC_1(n-1,k)$$
$$C_1(n,k) = C_0(n-1,k) + \frac{k-1}{2}C_1(n-1,k)$$

(*Hint*: You may find it easier to study the case for $k = 2$ first, then $k = 3$, etc., until you see a pattern emerge.)

(*Further hint 10/3/07*: Let $k = 3$, $n = 1$, and suppose that the NAND of three bits evaluates to 1. Then the **worst-case** input for the DFP (which does *not* know that the NAND of the three bits evaluates to 1) is any of the instances 011, 101, or 110. For any of these instances, the DFP will make one query 1/3 of the time and stop. The other 2/3 of the time it made the one query and will need to make more queries. How many? I leave that for you to figure out, but the reasoning is similar and considers how the DFP acts for the **worst-case** of the remaining unqueried bits. When deriving the complexity relation formulas, it is important to look at the **worst-case** instance for the DFP and not at a distribution over instances to the DFP.)

(*c*) Solve the recurrence relation in part (*b*) to show that

$$\begin{aligned} C(n,k) &:= \max\{C_0(n,k), C_1(n,k)\} \\ &= \mathcal{O}(N^{\log_k\left(\frac{k-1+\sqrt{k^2+14k+1}}{4}\right)}), \end{aligned}$$

where $N := k^n$ is the number of leaves in the tree.

(*Hint*: Substitute one recurrence relation into the other to obtain a recurrence relation entirely in terms of C_1. Then use the *Ansatz* solution $C_1(n,k) = Ar^n$ for some unknown constants A and r. This will lead to a quadratic equation in r whose positive root is the argument to the logarithm in the solution.)

(*d*) The deterministic query complexity for this problem is N—given any deterministic algorithm for this problem, there is some input which will require the algorithm to examine every leaf to determine the NAND tree value. Proving this is harder than it first seems, but we can get a glimmer of why this is true by imagining what happens to the DFP algorithm when the choice about which child to evaluate is made not randomly but deterministically. Let $A'(n,k)$ be a variant of the DFP algorithm in which child nodes are always evaluated from left to right until the parent node's value is determined. Give an example of an input string $x_0 \ldots x_N$ for which all N bits must be evaluated for $A'(n,2)$. (*Warning*: Be careful—coming up with a consistent answer requires a bit of thought.)

* * *

This problem in context:

Your analysis in this problem shows that for any binary ($k = 2$) NAND tree f on N leaves,

$$R_0(f) = \mathcal{O}(N^\alpha), \qquad D(f) = \mathcal{O}(N),$$

where $\alpha := (1 + \sqrt{33})/4 \approx 0.753$. It turns out that these equalities are tight [1], even when two-sided error randomized algorithms are considered [2], although we didn't prove the corresponding lower bounds. Namely, it is known that binary NAND tree evaluation has query complexity

$$R_0(f) = R_2(f) = \Theta(N^\alpha), \qquad D(f) = N.$$

What's the significance of this? Well in class it was noted that for an arbitrary total Boolean function it is known that [3]

$$R_2(f) = \Omega((D(f)^{1/3}), \qquad Q_2(f) = \Omega(D(f)^{1/6}).$$

These bounds suggest that there is hope to get third-root query algorithm speedups by using randomized, rather than deterministic algorithms and sixth-root speedups by using quantum, rather than deterministic algorithms for total Boolean functions. However, to date, the largest-known query complexity speedup for a total Boolean function is a square-root for quantum algorithms, *e.g.*, by Grover's algorithm, and a $1/\alpha \approx 1.327$th-root speedup for randomized algorithms, precisely for this NAND tree problem. It is widely conjectured that this $1/\alpha$ randomizing speedup is the best possible for total Boolean functions, but a proof is still lacking. One reason for believing this conjecture is that any read-once total Boolean function, namely a total Boolean function composed of AND, OR, and NOT gates such that its inputs appear at most once, evaluates to the value of some binary (and possibly unbalanced) NAND tree.

A natural question is, "What is the quantum query complexity of evaluating a NAND tree?" It turns out that this question was (mostly) settled only very recently. In 2003, Barnum *et al.* [4] proved that $Q_2(f) = \Omega(D(f)^{1/2})$, and earlier this year (2007) a series of papers by various authors presented quantum query algorithms whose complexity is very close to this. In particular, the latest in this string of papers is a result by Ambainis [5] demonstrating that evaluating binary NAND trees has query complexity $Q_2(f) = \mathcal{O}(D(f)^{1/2})$ for balanced NAND trees and query complexity $Q_2(f) = \mathcal{O}(D(f)^{1/2+\mathcal{O}(\frac{1}{\sqrt{\log N}})})$ for evaluating unbalanced NAND trees.

4.3. Clifford, but not the big red dog

We've discussed a number of quantum gate bases in this course, such as the following four:

$$\begin{aligned}
&\text{Standard} : \{H, S, T, \Lambda(X)\} \\
&\text{Shor} : \{H, S, \Lambda^2(X)\} \\
&\text{Kitaev} : \{H, \Lambda(S)\} \\
&\text{Gottesman-Knill} : \{H, S, \Lambda(X)\}
\end{aligned}$$

As noted in class, the first three are universal for quantum computation but the last one is not—circuits in the Gottesman-Knill basis can be efficiently simulated by classical

circuits. So what do the first three circuits have that the last one doesn't? They contain gates that are not in what is called the *Clifford group*. A gate G is said to be in the (n-qubit) Clifford group if and only if for every (n-fold tensor product) Pauli operator P, $GPG^{-1} = \{\pm 1, \pm i\}P'$ for some (n-fold tensor product) Pauli operator P'. The content of the Gottesman-Knill Theorem is that circuits composed solely of gates in the Clifford group are easy to simulate classically.

(a) Show that H and S are in the Clifford group by expressing HPH and SPS^{-1} as Pauli operators, up to an overall ± 1 or $\pm i$ phase, for each of the Pauli operators $P \in \{I, X, Y, Z\}$. (I.e., evaluate HIH, HXH, HZH, etc.)

(b) Show that $\Lambda(X)$ is in the Clifford group by expressing $\Lambda(X) \cdot P \cdot \Lambda(X)$ as a two-qubit Pauli operator, up to an overall ± 1 or $\pm i$ phase, for each two-qubit Pauli operator P. (*Hint*: Although there are sixteen two-qubit Pauli operators II, IX, IY, IZ, XI, etc., to consider, evaluating $\Lambda(X) \cdot P \cdot \Lambda(X)$ for all these combinations is not as tedious as it seems if you make clever use of the following two factoids:

$$\text{If } \Lambda(X) \cdot P \cdot \Lambda(X) = P' \text{ then } \Lambda(X) \cdot P' \cdot \Lambda(X) = P.$$
$$\text{If } \Lambda(X) \cdot P_1 \cdot \Lambda(X) = P_1' \text{ and } \Lambda(X) \cdot P_2 \cdot \Lambda(X) = P_2', \text{ then } \Lambda(X) \cdot P_1 P_2 \cdot \Lambda(X) = P_1' P_2'.)$$

(c) Show that T is not in the Clifford group by expressing TXT^{-1} as a sum of one-qubit Pauli operators.

(d) Show that $\Lambda(S)$ is not in the Clifford group by expressing $\Lambda(S)(XI)\Lambda(S)^{-1}$ as a sum of two-qubit Pauli operators.

(e) Show that $\Lambda^2(X)$ is not in the Clifford group by expressing $\Lambda^2(X)(IIZ)\Lambda^2(X)^{-1}$ as a sum of the three-qubit Pauli operators.

(*Hint*: For parts (c) through (e), you may find it helpful to use the fact that any n-qubit matrix M can be written as

$$M = \sum_{i_1,\ldots,i_n=0}^{3} \alpha_{i_1 \ldots i_n} \sigma_{i_1} \otimes \cdots \otimes \sigma_{i_n}$$

where each σ_i is a one-qubit Pauli matrix and each $\alpha_{i_1 \ldots i_n}$ is a complex number. Given a matrix M from which one wants to extract a particular coefficient $\alpha_{j_1 \ldots j_n}$ in this expansion, one can use the readily-verified fact that

$$\mathrm{tr}(M(\sigma_{j_1} \otimes \cdots \otimes \sigma_{j_n})) = 2^n \alpha_{j_1 \ldots j_n}$$

On the other hand, you may find it just as easy for these small examples to pick out the resultant sum of Pauli operators "by eye.")

References

[1] M. Saks and A. Wigderson, *Probabilistic Boolean decision trees and the complexity of evaluating game trees*, in *Proceedings of the 27th Annual Symposium on Foundations*

of Computer Science, IEEE (IEEE Press, New York, 27–29 Oct. 1986, Toronto, ONT, Canada, 1986), pp. 29–38.

[2] M. Santha, *On the Monte Carlo Boolean decision tree complexity of read-once formulae*, Random Structures and Algorithms **6**, 75 (1995).

[3] H. Buhrman and R. de Wolf, *Complexity measures and decision tree complexity: A survey*, Theo. Comp. Sci. **287**, 21 (2002), doi:10.1016/S0304-3975(02)00377-8, URL http://homepages.cwi.nl/~rdewolf/publ/qc/dectree.pdf.

[4] H. Barnum, M. Saks, and M. Szegedy, *Quantum query complexity and semi-definite programming*, in *Proceedings of the 18th IEEE International Conference on Computational Complexity*, IEEE (IEEE Press, 7–10 Jul. 2003, Aarhus, Denmark, 2003), pp. 179–193.

[5] A. Ambainis, *A nearly optimal discrete query quantum algorithm for evaluating* NAND *formulas* (2007), arXiv:0704.3628.