Recent Developments in Algorithm Design

Lecture 4: Stochastic Boolean Function Evaluation Continued (Hellerstein)

NYU TANDON SCHOOL OF ENGINEERING

k-of-n functions

- $f(x_1, \ldots, x_n) = 1$ if $x_1 + x_2 + \ldots + x_n \geq k$

  $= 0$ otherwise

- Need to perform tests until either get k 1's or (n – k + 1) 0's
- Optimal testing order? Consider unit cost case.

Fact about Optimal Adaptive Strategies

- Consider an optimal adaptive strategy for evaluating a Boolean function f
  - Suppose first test it performs is x_i
- Let f_1 be the function induced from f by setting $x_i = 1$
- Let f_0 be the function induced from f by setting $x_i = 0$
- Let S be an adaptive strategy for evaluating f that begins by testing x_i
  - Let S_0 be the substrategy peformed if $x_i = 0$ and S_1 be the substrategy performed if $x_i = 1$
- Strategy S is an optimal strategy for evaluating f iff S_0 is an optimal strategy for evaluating f_0 and S_1 is an optimal strategy for evaluating f_1

Algorithm for unit-cost k-of-n evaluation

- Not clear whether to favor tests with high p_i or low p_i
- Consider verification problem
  - Before testing, little birdie tells you value of $f(x)$
  - Need to perform tests to verify that value
- Can use different strategy depending on value of $f(x)$
  - say π_1 is strategy for $f(x) = 1$ and π_0 is strategy for $f(x) = 0$
  - Expected cost using these two strategies is:

    $$P[f(x) = 1]\ E[\text{cost}(\pi_1, x) \mid f(x) = 1] + P[f(x) = 0]\ E[\text{cost}(\pi_0, x) \mid f(x) = 0]$$

  - opt expected # tests for verification ≤ opt expected # tests for evaluation

- For k-of-n verification problem:
  - Optimal strategy to verify $f(x) = 1$: test in decreasing p_i order
  - Optimal strategy to verify $f(x) = 0$: test in increasing p_i order

- Number variables so $p_1 \geq \ldots \geq p_n$
- To verify $f(x) = 1$
  - Optimal to test in order $x_1, \ldots, x_n$
  - Need to perform at least first k tests
  - Still optimal if you change order of first k tests.
- To verify $f(x) = 0$
  - Optimal to test in order $x_n, x_{n-1}, \ldots, x_1$
  - Need to perform at least first $n - k + 1$ of those tests
  - Still optimal if you change order of first $n - k + 1$ tests.
- Both orderings still optimal if move kth test to first position.
- Testing x_k first optimal in both cases

- Recursive strategy for k-of-n evaluation
    - Test x_k (variable with kth largest pi)
    - If $x_k = 1$
        - If k = 1, know $f(x) = 1$. Exit.
        - Else have induced (k – 1)-of-(n – 1) evaluation problem. Recurse.
    - If $x_k = 0$
        - If k = n, know $f(x) = 0$. Exit.
        - Else have induced k-of-(n – 1) evaluation problem. Recurse.
- Single strategy, optimal for verifying both $f(x) = 1$ and $f(x) = 0$
- ⇒ It is also an optimal evaluation strategy (originally shown using inductive proof by Halpern in the 1970’s)

[Salloum, Breuer, Ben-Dov '79, '81, '84]

- What about non-unit costs?
- Can show optimal strategy for verifying $f(x) = 1$ is to test in increasing order of $\frac{c_i}{p_i}$.
  (Needs new proof: k can be greater than 1, need to condition on $f(x) = 1$, doesn't follow from proof we did for evaluating OR function)
- Similarly, can show optimal strategy for verifying $f(x) = 0$ is to test in increasing order of $\frac{c_i}{1 - p_i}$
- Second ordering is not necessarily the reverse of the first one!
- But still know:
  - First order is optimal if permute first k tests
  - Second order is optimal if permute first $(n - k + 1)$ tests
  - By pigeonhole, there is a test that is within the first k tests of first order, and within first $n - k + 1$ tests of second order
  - Test that one first and recurse!

Verification vs. Evaluation

- For evaluation of k-of-n functions
    - opt expected cost of verification
      (i.e., little birdie, two different strategies)
      = opt expected cost of evaluation

Notation and terminology

- Represent an assignment to the Boolean variables $x_1, \ldots, x_n$ as vector $\alpha \in \{0,1\}^n$
- Use x=$(x_1, \ldots, x_n)$ to denote a random assignment where $p_i = P[x_i = 1]$, the x_i are independent
- A partial assignment to the variables $x_1, \ldots, x_n$ is a vector $\beta \in \{0,1,*\}^n$
    - Can think of * as meaning unknown value
    - Say α is an extension of β if for all i such that $\beta_i \neq *$, $\alpha_i = \beta_i$
    - Also say that β is contained in α
- Let $O = \{0,1\}$
- Say $\beta \in (O \cup \{ * \})^n$ is a 1-certificate of f if
  $\forall \alpha \in \{0,1\}*$ such that α is an extension of β, $f(\alpha) = 1$
- Say $\beta \in (O \cup \{ * \})^n$ is a 0-certificate of f if
  $\forall \alpha \in \{0,1\}*$ such that α is an extension of β, $f(\alpha) = 0$
- In evaluation problem, if β represents the outcomes of tests performed so far, need to continue testing until β is a 1-certificate or 0-certificate of f

- Consider a strategy S for evaluating a Boolean function $f(x_1, \ldots, x_n)$
- For $\alpha \in \{0,1\}^n$, executing S on α means execution of S when each test x_i has outcome α_i
- Given any strategy S for evaluating a Boolean function $f(x_1, \ldots, x_n)$, and assignment $\alpha \in \{0,1\}^n$, let

  cost(S,α) = cost of all tests performed when executing S on α

- E[cost(S,x)] is expected cost of strategy S on random assignment x
  - where x is generated by setting each x_i to 1 with independent probability p_i
  - SBFE problem asks for strategy S minimizing this expected cost

A trivial approximation algorithm for SBFE problems

Naive strategy for SBFE problems

- **Increasing Cost Strategy** S_c:
  Perform the tests in increasing order of costs until can determine value of the function
- **Claim:** $E[\text{cost}(S_c, x)] \leq n \times E[\text{cost}(S^*, x)]$
  i.e., expected cost of increasing cost strategy is at most n * OPT
- **Pf:**
  - Let S^* be optimal strategy for evaluating f, expected cost of S^* is OPT
  - Consider execution of S^* on an assignment α. Let i^* be index of highest cost test performed. Thus $c_{i^*} \leq \text{cost}(S^*, \alpha)$
  - Consider execution of S_c on α. It must stop at or before executing all tests of cost $\leq c_{i^*}$. Why?
  - Therefore, S_c performs $\leq n$ tests each of cost $\leq c_{i^*}$, so
    $$\text{cost}(S_c,\alpha) \leq n \times c_{i^*} \leq n \times \text{cost}(S^*, \alpha)$$
  - $\Rightarrow E[\text{cost}(S_c, x)] \leq n \times E[\text{cost}(S^*, x)]$

Evaluation of Symmetric Boolean Functions

Symmetric Boolean function

- A Boolean function is symmetric if its value depends only on how many of its inputs are 1, and not which inputs are 1
    - equivalently, it is symmetric if its value is a function of $x_1 + x_2 + \ldots + x_n$
- k-of-n functions are symmetric Boolean functions
- so is the parity function

Stochastic Score Classification

- Imagine doctor who wants to determine how much risk patient has for a certain disease
  - Can perform 10 tests on patient with binary outcomes
  - Patient's score is number of positive tests (out of 10)
  - Score determines the patients risk classification:
    Low: 0-3
    Medium: 4-8
    High: 9-10
  - Perform tests sequentially, may be able to determine classification before performing all tests. (e.g., perform 6 tests, get 4 positives and 2 negatives)
- More generally, have n tests, divide range from 0 to n into risk classes
- Suppose each test has cost c_i, $P[\text{test } i \text{ is positive}] = p_i$, independent tests
- Stochastic test ordering problem: Given probability that each tests is positive, determine order to perform tests so as to minimize expected costs of tests

Equivalent to SBFE problem for Symmetric Boolean Functions

Why?

- Equivalence of Score Classifiction Problem and SBFE problem for Symmetric Boolean Functions
  - Can represent a symmetric Boolean function by a "value vector" v of length $n+1$, indexed from 0 to n

    $v_i =$ value of f on inputs x with exactly i 1's

    e.g., for Boolean OR function, $v = [0,0,\ldots,0,1]$

  - In value vector, blocks of contiguous 0's alternate with blocks of contiguous 1's
  - To determine value of f on input x, need to determine the block to which x belongs
  - Consider each block to be a risk class

- Thm [Das et al. 2012]: In the unit-cost case, for symmetric Boolean functions,

  opt expected cost of verification
  (i.e., little birdie, two different strategies)
  = opt expected cost of evaluation

- (This result does not hold for arbitrary costs.)

- Pf Sketch for Thm. of Das et al.:
  - To determine value of f on input x, need to determine the block in value vector to which x belongs

- Same basic idea as in k-of-n evaluation (which is special case where value vector has two-blocks)
    - Showed that x_k is an optimal first test for verifying membership of x in 1st block, or membership in 2nd block. Symmetric function can have more than 2 blocks.
- Can show that there is a test x_i that is an optimal first test for verifying membership in any one of the blocks.
- Proof is by induction and uses case analysis
- Tells you such an x_i exists, but doesn't tell you exactly which variable is x_i (?!?)

- Open Problem :
  - Is SBFE problem for symmetric Boolean functions NP-hard? With arbitrary costs? Unit costs?

Approximation algorithm for evaluating symmetric Boolean functions

- We'll show a simple 6-approximation algoithm from [Liu 2022].
  - There's a more complicated 5.8 approximation [Planck and Schewior 24]
- Uses cost-sensitive round-robin from [Allen et al. 17] for performing modified round robin between k different testing strategies, $S_1, \ldots, S_k$, when tests can have different costs

- Cost-Sensitive Round-Robin (RR):
    - Keep track of cost C_i incurred so far by each strategy S_i.
      Initially $C_i = 0$ for all i
    - Repeat until some stopping criterion is reached or some strategy has no next test:
        - For all i, let d_i be cost of next test to be performed by S_i
        - Perform next test in strategy i having minimum value of $C_i + d_i$
          // if test already performed, can actually just use answer obtained before
          // but we're assuming here that you pay again
        - update $C_i = C_i + d_i$

- Let $C_i(t)$ and $d_i(t)$ be the values of C_i and d_i right before the t th test, $C_i(t+1)$ and $d_i(t+1)$ be values right after the t-th test
- Call $C_i(t)$ the cumulative cost of S_i at time t and $C_i(t) + d_i(t)$ the prospective cost of S_i at time t
- RR chooses test from strategy with minimum prospective cost
- RR Fact:
  - If t th test is from C_i , then for all $j \neq i$
    $C_j(t+1) \leq C_i(t+1) \leq C_j(t+1) + d_j(t+1)$
    cumulative cost of C_j at time $t+1 \leq$ cumulative cost of C_i at time $t+1$ $\leq$ prospective cost of C_j at time $t+1$
- Pf: Suppose t th test chosen by RR is from S_i
  - Then $C_i(t) + d_i(t) = C_i(t+1)$, and for all $j \neq i$, $C_j(t) = C_j(t+i)$
  - Also, $d_j(t+1) = d_j(t)$ for all $j \neq i$
  - $C_i(t+1) \leq C_j(t+1) + d_j(t+1)$ because chose t th test from C_i and not C_j
  - Now show $C_j(t+1) \leq C_i(t+1)$

- Suppose for contradiction that $C_j(t+1) > C_i(t+1)$

- Consider the step prior to t at which a test of S_j was last chosen.
  Say that was step τ .

- Then
  $$C_j(\tau) + d_j(\tau) = C_j(\tau+1) = C_j(t+1)$$
  $$> C_i(t+1) \quad \text{by assumption}$$
  $$= C_i(t) + d_i(t)$$
  $$\geq C_i(\tau) + d_i(\tau) \quad \text{since } \tau < t$$

- But then RR wouldn't have chosen a test from
  S_j at step τ, because S_i had lower propective cost

- Contradiction

Suppose for contradiction, choosing test t

Algorithm for Stochastic Score Classification

- Run cost-sensitive round-robin between 3 strategies, until have enough 0's and 1's to determine which block contains x (i.e., until have at least z_j 0's and at least o_j 1's)
  - S_0 : Increasing $\frac{c_i}{1 - p_i}$ order
  - S_1 : Increasing $\frac{c_i}{p_i}$ order
  - S_c : Increasing cost order

- Thm: Proposed cost-sensitive round robin algorithm between S_c, S_1, S_0 is a 6-approximation algorithm for verifying membership in Block j

- Pf:

  - Consider verification problem for verifying membership in block j (risk class j)

  - As perform tests and get results, value vector "shrinks":

    - e.g., Suppose $n = 4$ and $v =$ [0,1,1,0,0]

      - if perform test with outcome 1, know total number of 1's in input is between 1 and 3. "Shrinks" value vector to [0,1,1,0,0]

      - in this case, to verify membership in the middle block, consisting of 1's, would need to do enough tests to get at least one 1 and at least two 0's

- So to verify membership in a block j , need to get at least o_j 1's and z_j 0's (for some values of o_j and z_j)
- Consider following verification strategy V_j for block j
  - Phase 1: Perform the cheapest $o_j + z_j$ tests
    - If found o_j 1's and z_j 0's, done
  - Phase 2:
  - If found fewer than z_j 0's in Phase 1
    - follow increasing $\frac{c_i}{1-p_i}$ order for remaining tests until found z_j 0's total
  - if found fewer than o_j 1's in Phase 2
    - follow increasing $\frac{c_i}{p_i}$ order for remaining tests until found o_j 1's total

- **Claim**: $E[cost(V_j, x) \mid x$ in block $j] \le 2E[cost(V_j^*, x) \mid x$ in block $j]$,
  where V_j^* is optimal verification strategy for block j

- **Pf:**

  - Let $cost(V_j^1, \alpha)$ and $cost(V_j^2, \alpha)$ denote the cost incurred by V_j on assignment α in Phases 1 and 2

  - To verify that an assignment α is in block j, V_j^* must find $\ge o_j$ 1's and $\ge z_j$ 0's, so it must do at least $o_j + z_j$ tests. So

    $E[cost(V_j^1, x) \mid x$ in block j $] = o_j + z_j \le$ E[cost($V_j^*, x) \mid x$ in block $j]$ (Statememt 1)

  - In Phase 1, since V_j does $o_j + z_j$ tests, it must either find o_j 1's or z_j 0's or both. Thus if not done at end of Phase 1, it still needs either more 1's, or more 0's

  - Suppose the cheapest $o_j + z_j$ tests were free. Then it would be optimal to perform these tests first, and then to use either increasing $\frac{c_i}{p_i}$ order, or increasing $\frac{c_i}{1 - p_i}$ order, depending on whether still needed to find 1's or 0's. That is, V_j would be optimal and its cost on any α would be its Phase 2 cost.

  - Since making tests free can only decrease the expected cost of verification, we have

    $E[cost(V_j^2, x) \mid x$ in block j $] \le E[cost(V_j^*, x) \mid x$ in block $j]$ (Statement 2)

  - Claim follows from Statements 1 and 2, by linearity of expectation since cost of V_j is the sum of costs in Phases 1 and 2.

- Now consider cost of proposed RR on a fixed assignment α in block j
    - Consider tests performed in S_0, S_1, S_c by RR, on assignment α
      Divide analysis into cases depending on what happens when V_j run on assignment α
    - **Case 1:** V_j done at end of first phase
      Then V_j performs precisely the first $o_j + z_j$ tests of S_c, $cost(V_j, \alpha) =$ total cost of first $o_j + z_j$ tests of S_c

      ⇒ RR chooses at most $o_j + z_j$ tests from S_c before stopping

      if last test of RR is chosen from S_c, by Fact, at end of RR
      cumulative cost of $S_0 \leq$ cumulative cost of $S_c \leq$ total cost of first $o_j + z_j$ tests of S_c
      cumulative cost of $S_1 \leq$ cumulative cost of $S_c \leq$ total cost of first $o_j + z_j$ tests of S_c
      ⇒ Total cost of RR ≤ 3*(total cost of first $o_j + z_j$ tests of S_c) = 3*cost(V_j, α)

      if last test of RR is from S_0 or S_1 , suppose wlog it is S_0. then RR didn't choose all of the first $o_j + z_j$ tests of S_c and at end of RR
      prospective cost of $S_c \leq$ total cost of the first $o_j + z_j$ tests of S_c
      ⇒ (by fact) cumulative cost of $S_0 \leq$ total cost of the first $o_j + z_j$ tests of S_c
      cumulative cost of $S_1 \leq$cumulative cost of S_0
      ⇒ Total cost of RR ≤ 3*(total cost of first $o_j + z_j$ tests of S_c) = 3*cost(V_j, α)

- **Case 2**: At end of Phase 1, V_j hasn't found o_j 1's and z_j 0's. wlog suppose it hasn't found enough 1's.
  Can again show Cost(RR,α) $\leq$ 3Cost(V_j, α).
  - Consider last test performed by V_j in Phase 2. Say it is the m th test of S_1
    - When V_j terminates at the end of Phase 2, it has performed the union of the first m tests of S_1 and the first $o_j + z_j$ tests of S_c
    - So, RR can't choose all of the first m tests in S_1 AND all of the first $o_j + z_j$ tests of S_c without terminating.
      - Right before last test is chosen by RR, at least one of the following must hold
        - prospective cost of $S_1 \leq$ total cost of its first m tests (which is $\leq cost(V_j, \alpha)$)
        - prospective cost of $S_c \leq$ total cost of its first $z_j + o_j$ tests (which is $\leq cost(V_j, \alpha)$)
      - $\Rightarrow$ last test of RR is chosen from strategy with prospective cost $\leq cost(V_j, \alpha)$
      - $\Rightarrow$ by Fact, cumulative costs of S_0, S_1, S_c at end of RR are $\leq cost(V_j, \alpha)$
      - $\Rightarrow$ Cost(RR,α) $\leq$ 3Cost(V_j, α).

- So E[cost(RR,x)] $\leq 3 * E[cost(V_{j(x)}, x)]$ where $j(x)$ is block containing x

  i.e., if use V_j for all assignments in block j

  $\leq$ 3*(2*Exp cost of optimal verification strategy)

  $\leq$ 6 * OPT

Evaluation of Linear Threshold Functions

SBFE problem for Boolean Linear Threshold Functions

- Linear threshold function
  - Boolean function $f(x_1, \ldots, x_n)$ such that for some $a_1, \ldots, a_n, \theta \in \mathbb{Z}$

$$f(x_1, \ldots, x_n) = 1 \text{ iff } a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq \theta$$
$$= 0 \text{ otherwise}$$

- k-of-n functions are the special case of linear threshold functions where $a_1 = a_2 = \ldots = a_n = 1$
  (aka unweighted linear threshold functions)

NP-hardness

- Easy to show that the SBFE problem for Boolean Linear Threshold Functions is NP-hard
  - To evaluate f, need to determine whether $a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq \theta$
    - cost of testing x_i is c_i
    - $p_i = P[x_i = 1]$
  - Consider the case where all $a_i \geq 0$ and $f(1,\ldots,1) = 1$, i.e., $\sum_{i=1}^{n} a_i \geq \theta$
  - Suppose the p_i values are all equal to 1 (or arbitrarily close to 1)
  - Need to find minimum cost subset of tests S certifying that $a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq \theta$
    - equivalently, find $S \subseteq \{1,\ldots,n\}$ minimizing $\sum_{i \in S} c_i$ such that $\sum_{i \in S} a_i \geq \theta$
    - This is the Min-Knapsack problem

Min-Knapsack Problem

- Min-Knapsack is minimization problem (closely related to the classical NP-hard Knapsack problem)
  - Given
    - $S = \{o_1, \ldots, o_n\}$ of n objects
      - with weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$
    - and a target value V
    - Find subset $S' \subseteq S$ minimizing $\sum_{o_i \in S'} w_i$

      such that $\sum_{o_i \in S'} v_i \geq V$
- NP-hard
- Pseudopolynomial time dynamic programming algorithm
  - runtime depends polynomially on the v_i values

NP-hardness

- Easy to show that the SBFE problem for Boolean Linear Threshold Functions is NP-hard
  - To evaluate f, need to determine whether $a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq \theta$
    - cost of testing x_i is c_i
    - $p_i = P[x_i = 1]$
  - Consider the case where the a_i values are all positive
  - Suppose the p_i values are extremely close to 1
  - If $a_1 * 1 + a_2 * 1 + \ldots + a_n * 1 < \theta$ then know $f(x_1, \ldots, x_n) = 0$
  - Otherwise, assuming all tests will have outcome 1, need to find minimum cost subset of tests S certifying that $a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq \theta$
    - equivalently, find $S \subseteq \{1,\ldots,n\}$ minimizing $\sum_{i \in S} c_i$ such that $\sum_{i \in S} a_i \geq \theta$
    - This is the Min-Knapsack problem where weight of object i is c_i, value of object i is a_i, target value is θ
  - So can easily show reduction from Min-Knapsack to SBFE problem for Boolean Linear Threshold functions

Appoximation algorithm for evaluating Linear Threshold Functions

- [Deshpande et al. 2016]
  - Approximation algorithm with expected cost 3*OPT
  - Reduces the evaluation problem to the *Stochastic* Submodular Cover Problem, by construction of an associated utility function
  - Solves resulting Stochastic Submodular Cover instance using a generalization of primal-dual approximation algorithm for Submodular Cover (in HW1) to *Stochastic* Submodular Cover

Stochastic Submodular Cover Problem

Stochastic Submodular Cover

- Items $I = \{1, \ldots, n\}$
- Finite set of states O containing d states
  - e.g., $O = \{working, broken\}$, $O = \{0,1\}$, $O = \{low, medium, high\}$
  - Each item in I is in one of the d states
  - x_i is a random variable whose value is the state of item i
  - $p_i^o = Pr[x_i = o]$
- Can only determine the state of i by performing test, which costs c_i
  - Can represent the states of the n items by vector $\alpha \in O^n$
  - Can represent knowledge of states of some of the n items by vector $\beta \in (O \cup *)^n$ where $*$ means unknown

State-dependent utility function

- Utility of set of items depends not only on which items are in the set, but also on the states of those items
- Utility function $u : (O \cup *)^n \rightarrow \mathbb{Z}^{\geq 0}$
- In Stochastic Submodular Cover problem:
  - u is monotone, submodular, and $u([*, *, \ldots, *]) = 0$
    [see next slide]
  - there exists ``goal value" $Q \in \mathbb{Z}^{>0}$ such that for all $\alpha \in O^n$, $u(\alpha) = Q$

Monotonicity and Subodularity for state-dependent utility functions

- Say u is monotone if for $\alpha, \beta \in \{0,1,*\}^n$
  if α is an extension of β then $u(\beta) \leq u(\alpha)$
  i.e., more information can only increase utility

- Say u is submodular if for all $\alpha, \beta \in \{0,1,*\}^n$
  if α is an extension to β and $\alpha_i = \beta_i = *$ then
  $u(\alpha_{i\leftarrow 0}) - u(\alpha) \leq u(\beta_{i\leftarrow 0}) - u(\beta)$ and
  $u(\alpha_{i\leftarrow 1}) - u(\alpha) \leq u(\beta_{i\leftarrow 1}) - u(\beta)$

  where e.g., $\alpha_{i\leftarrow 1}$ is the partial assignment derived from α by setting $\alpha_i = 1$

Stochastic Submodular Cover Problem (continued)

- Testing
  - Perform tests on the items, sequentially and adaptively
  - Can only test each item once (at most)
  - Represent outcomes of test so far by vector $\beta \in (O \cup \{ * \})^n$
  - Need to continue testing until $u(\beta) = Q$
- Stochastic Submodular Cover Problem
  - Find an order in which to perform the tests that minimizes the expected testing cost
- If outcomes of tests were known in advance (deterministic or offline version of the problem), but still need to test until $u(\beta) = Q$, this would be a Submodular Cover problem

Algorithms for Stochastic Submodular Cover

- Adaptive Greedy [GolovinKrause 11]
- Adaptive Dual Greedy [Deshpande et al. 16]

(we'll discuss later)

Solving SBFE problems by reduction to Stochastic Submodular Cover

Reducing SBFE problem to Stochastic Submodular Cover

- Construct utility function $u : (O \cup \{ * \})^n \to \mathbb{Z}^{\geq 0}$ from f such that
  - $O = \{0,1\}$
  - $u(\emptyset) = 0$
  - There exists $Q \in \mathbb{Z}^{>0}$ such that for all $a \in \{0,1\}^n$, $u(a) = Q$
  - For all $b \in \{O,1,*\}^n$, $u(b) = Q$ iff b is a 0-certificate of 1-certificate of f
- Testing until $u(b) = Q$ is equivalent to testing until b is a certificate of f
- Run algorithm for Stochastic Submodular Cover on utility function u, use resulting testing strategy to evaluate f

When is this approach useful?

- Need to construct utility function u with given properties
  - For any Boolean function f, can always construct such a function u, but approximation bounds of Adaptive Greedy and Adaptive Dual Greedy may be bad with this u
  - When f is a Boolean linear threshold function, can construct u so that Adaptive Dual Greedy achieves a constant-factor approximation bound.

Construction of u for Linear Threshold Functions

- Boolean linear threshold function f defined by the inequality

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq \theta$$

  - All a_i and θ are integers. Assume they're all positive integers (else can easily reduce to this case)
- Let $A = \sum_{i=1}^{n} a_i$
- $\beta \in \{0,1*\}^n$ is a 1-certificate for f iff $\sum_{i:\beta_i=1} a_i \geq \theta$
- $\beta \in \{O,1,*\}^n$ is a 0-certificate for f iff $\sum_{i:\beta_i=0} a_i \geq A - \theta + 1$
- Define $u_1 : \{0,1,*\}^n \to \mathbb{Z}^{\geq 0}$ such that $u_1(\beta) = \min \{ \sum_{i:\beta_i=1} a_i , \theta \}$
- Define $u_0 : \{0,1,*\}^n \to \mathbb{Z}^{\geq 0}$ such that $u_0(\beta) = \min \{ \sum_{i:\beta_i=0} a_i, \ A - \theta + 1 \}$
- Both u_0 and u_1 are monotone and submodular
- β is a certificate for f iff $u_1(\beta) = \theta$ or $u_0(\beta) = A - \theta + 1$

Use OR construction

- β is a certificate for f iff $u_1(\beta) = \theta$ or $u_0(\beta) = A - \theta + 1$
- Let $Q_1 = \theta$ and $Q_2 = A - \theta + 1$
- Use OR construction (cf. [Golovin et al. 10]) to produce a new function

  $u : \{0,1,*\}^n \Rightarrow \mathbb{Z}^{\geq 0}$ such that for all $\beta \in \{0,1,*\}^n$

  $$Q_1 Q_0 - (Q_1 - u_1(\beta))(Q_0 - u_0(\beta))$$

- Since u_0 and u_1 are submodular and monotone, so is u
- Since $u_1(*, \ldots, *) = u_0(*, \ldots, *) = 0$, also have $u(*, *, \ldots, *) = 0$
- $u(\beta) = Q_1 Q_0$ iff $u_1(\beta) = Q_1$ or $u_0(\beta) = Q_0$
  - and $u(\alpha) = Q_1 Q_0$ for all $\alpha \in \{0,1\}^n$ (why?)

- Summary: Approximation algorithm solving the SBFE problem for linear threshold functions
  - Given representation of linear threshold function f
    $$a_1x_1 + a_2x_2 + \ldots + a_nx_m \geq \theta$$
    - Construct u as described
    - Run Adaptive Dual Greedy to solve the Stochastic Submodular Cover problem for utility function u
      - for any β , easy to compute $u(\beta)$ from above representation of f , so easy to simulate oracle for utility function u
- Can show with variant of bound on Adaptive Dual Greedy that this algorithm achieves an approximation factor of 3
- Open: Polytime 2-approximation algorithm? (achievable for Min-Knapsack)

- Technique of reducing problems to (Stochastic) Submodular Cover is useful for other problems

Adaptive Greedy and Adaptive Dual Greedy

Algorithms for the Stochastic Submodular Cover Problem

- Adaptive Greedy
    - Essentially same algorithm as the greedy algorithm for Submodular Cover
    - but in rule for choosing next item to pick, use *expected* increase in utility
        - i.e., choose item that would give largest *expected* increase in utility per unit cost (i.e., *expected* bang for the buck)
- Adaptive Dual Greedy
    - Essentially same algorithm as the primal-dual algorithm for Submodular Cover
    - but in rule for choosing next item to pick, use *expected* increase in utility
        - (modifying rule you were asked to describe in HW 1)

Bounds on Greedy algorithms for Stochastic Submodular Cover

- Adaptive Greedy
  - $O(\log Q)$ approximation bound
    - algorithm introduced by Golovin and Krause, but their proof of the bound had error [GolovinKrause 11]
    - correct proof by “latency based argument”, large constant [Im et al. 12, 16]
    - proof of more general result, improved latency based argument, $4(1 + \ln Q)$ approximation bound [CuiNagarajan 23]
    - $(1 + \ln Q)$ approximation bound using amortization arguments [Parthasarathy et al. 21]
      essentially best possible bound if P $\neq$ NP, by hardness of approximating set cover problem
  - Other bounds have dependence on parameters like the p_i, and/or compare expected cost to expected minimum certificate cost (optimal offline cost)

- Adaptive Dual Greedy [Deshpande et al. 16]
  - Approximation bound

$$\max_{\alpha,S} \frac{\sum_{i\in I} u_{S,\alpha}(i)}{Q - u(S,\alpha)}$$ where max is over pairs α, S where $\alpha \in \{0,1\}^n,$ and $S \subseteq I$

$u_{S,\alpha}(i)$ and $u(S,\alpha)$ analogous to $u_S(i)$ and $u(S)$ for α

(assumes for all i that state of x_i is α_i)

  - A variant of this bound restricts S and sums over only some $i \in I$
  - Proof of bound based on an IP (and then LP) relaxation of the problem of finding an optimal decision tree
    - Related to the LP from Homework 1 and its dual, but with significant differences

Weighted Stochastic Score Classification

Weighted Stochastic Score Classification

- Again, determine risk class of patient using binary-valued tests
- Some tests more important than others, so each test i has a weight w_i
- Patient's score is total weight of tests that have positive results

$$\sum_{i=1}^{n} w_i x_i$$

- Maximum score is $W := \sum_{i=1}^{n} w_i$
- Range from 0 to W is divided into risk categories
- Must continue testing until determine patient's risk score
- SBFE problem for Boolean Linear Threshold functions equivalent to Weighted Stochastic Score classification with 2 risk classes
- So generalizes Stochastic Score Classification and SBFE problem for Linear Threshold Functions and Symmetric Functions

- Thm [Ghuge et al. 2022]: There is a poly-time approximation algorithm for the Weighted Stochastic Score Classification problem that produces a non-adaptive strategy with expected cost $O(OPT)$, where OPT is the expected cost of the optimal adaptive strategy.
  - Constant factor approximation bound, but large-ish constant (much more than 6)
- Algorithm sketch:
  - Algorithm runs in phases to construct non-adaptive strategy (test sequence) we'll call NA. In each phase, chooses tests to add to end of currrent test sequence
    - cost of tests added in phase ℓ is at most $C \times 2^{\ell}$, where C a constant (budget increases exponentially with each phase)
  - Chooses tests to add in phase by running approximation algorithm for carefully chosen set of instances of the (standard deterministic) knapsack problem
    - Get deterministic instances of knapsack problem by replacing each remaining test x_i with a truncated version of its expected value.

- Sketch of proof of approximation bound
  - Consider cost of test to be time to peform the test.
  - Let S^* be optimal adaptive strategy.
  - Consider execution of S^* on random α as time increases. Divide time into phases of length 2^ℓ.
  - Consider execution of NA on random α as time increases. Phases of strategy NA are of length $C \times 2^\ell$, for a chosen constant C
  - Key Lemma: $r_\ell \leq 0.3 r_{\ell-1} + r^*_\ell$ where r_ℓ is probability that NA has not finished at end of its phase ℓ, and r^*_ℓ is probability that S^* has not finished by end of its phase ℓ
  - Key Lemma implies E[cost(NA)] = 10C × E[cost(S^*)]
- "Latency based" argument, looks at probability that algorithms haven't finished (probability mass of all α that are still waiting to be finished) at various times

Summary

- Stochastic Boolean Function Evaluation
  - Showed poly-time exact algorithms for OR, k-of-n functions
  - Described or sketched constant-factor approximation algorithms for
    - symmetric Boolean functions (score classification functions)
    - linear threshold functions
    - weighted score classification functions

Techniques

- Algorithmic techniques
  - greedy
  - round robin
  - reduce to stochastic submodular cover
  - relate to verification problem
  - replacing variables x_i by (function of) their expectation to make problem determimistic
- Techniques for proving approximation bounds
  - LP-based
  - comparing to optimal verification strategies
  - latency based arguments
  - amortized arguments
  - and more…

Open Questions

- Does SBFE problem for Symmetric Boolean Function Evaluation have a poly-time exact algorithm?
- Does SBFE problem for Read-Once Formulas have a poly-time exact algorithm? (see HW2)
- Other classes of functions? Improved approximation factors?

Questions?