CSC 373 Lecture 28

Announcements:

As posted, weekly TA office hour Fridays 1-2 in Pratt 378.

Term Test 2 question 3 regrading

Next assignment/test date; P3 finalized

Why 2W_0 <= W_1 in exact max-2-sat local search

Today

- Finish up discussion of exact-k-sat random alg
- Polynomial identities

First application: Exact Max-k-Sat

- We are going to use randomization to show that a very naive use of randomization computes a truth assignment that (in expectation) satisfies a fraction *(2^k-1)/2^k* of all clauses (or in the weighted case, this fraction of the total weight of all clauses). Let F be an exact *k*-CNF formula with say *m* clauses.
- The algorithm simply sets each variable randomly (and independently) so that Prob[x_j=true] = Prob[x_j = false] = ½.
- Claim: The random variable C_i = 1 (resp 0) if tau(C_i) true (resp. false) has expectation *(2^k-1)/2^k. Why?*
- By linearity of expectation
  E*_tau[W(F)] = W*(2^k-1)/(2^k)* where W = sum of all clause weights. (Compare this with oblivious local search.)

De-randomization of the algorithm

- This naive randomized algorithm is an *online algorithm* in the sense that the order in which we set the variables does not matter.
- We can de-randomize this algorithm by the *method of conditional expectations* to yield a deterministic (still online) greedy algorithm.
- Namely, let us think of the input items being the propositional variables where we represent each variable by the clauses in which it occurs. For each variable we want to set its truth value so as to maximize the expectation of the formula F given whatever assignments have already been made.

De-randomization continued

- Consider the first variable assignment (say to *x*). $\mathbf{E}[W(F)] = (1/2)\, \mathbf{E}[W(F) | x = \text{true}] + (1/2)\, \mathbf{E}[W(F) | x = \text{false}]$
- Therefore at least one of these assignments must result in at least the desired expectation and we can decide this by computing the expectation knowing the sign of *x* in each clause to which it belongs.
- Having set *x* appropriately, we then can consider the next variable always maintaining the weight of satisfied clauses and the number of literals in each unsatisfied clause.

From good expectation to good probability for almost the expectation

- In some sense the Max-*k*-Sat randomized algorithm is an example of random sampling. For any exact *k* CNF formula, if we simply try a random truth assignment *tau*, it is "likely" to satisfy a "good" number of clauses. Lets consider the # of unsatisfied clauses.
- Given the expectation ***E***, we can use *Markov's inequality* to show that the Prob[# unsatisfied >= *c* ***E***] is <= *1/c* . For example, say *c* = (8/7) then the probability is at most 7/8.
- To drive this probability down, independently repeat the "trial" *t* times so that the probability of always finding a *tau* with more than 8/7 ***E*** unsatisfied clauses is at most *(7/8)^t*. For example, for k = 3, we expect a 1/8 fraction of unsatisfied clauses, and the probability of always getting more than a 1/7 fraction unsatisfied is at most (7/8)^t.
- This idea of repeated indep.trials is a key aspect of randomized algorithms. Useful fact: *(1-1/t)^t <= 1/e* for all *t* and limits to *1/e* as *t* goes to infinity.

Final comments on randomized and deterministic online max-sat

- It is interesting that the deterministic online algorithm (Johnson's algorithm) that results from the de-randomization was known before it was seen to follow from the randomized algorithm.
- It was proven 25 years later that Johnson's 1974 algorithm is a 2/3 approximation for general Max-Sat (any number of literals per clause) Last year it was shown that Johnson's algorithm with a random ordering of the variables is better than 2/3 but not as good as a 3/4 approx for Max-Sat which can be done with IP/LP and randomized rounding. . Current best ratio .797 by semi definite programming algorithm.
- It was shown at the same time that a different randomized online algorithm achieves a ¾ approx for Max-Sat. There is evidence that this ¾ approx alg cannot be de-randomized.

Polynomial identities; more random sampling

- We want to exploit the fact “low degree” non zero polynomials have “few” zeros. In probabilistic terms when evaluated on a random point, a low degree non zero polynomial will likely not evaluate to zero. More precisely, we have the Schwartz-Zipple Lemma: Let *f* be a non zero m-variate polynomial (say over a ring R) of degree d >=0. Let each *r_i* be randomly chosen from a subset *S* of *R*. Then *Prob_[f(r_1, …, r_m) = 0] <= d/|S|*.
- We will consider two applications relating to polynomial identities, namely testing a matrix multiplication algorithm, and determining if a symbolic determinant is identically zero.

Testing if C = A*B

- We might have a fast but not proven matrix multiplication algorithm. We want to use it but would like to be confidant that when using it for a given input *A,B*, it is unlikely to have made a mistake. (Debugging vs testing vs proving correctness) Suppose these are *n*x*n* matrices with elements in a ring *R* (eg integers). We want to be able to test that the result *C = A*B* and do so much faster than say using a standard well proven algorithm (say with time *n^3*).
- Let *S* be an arbitrary subset of *R* and choose a random vector ***x*** in *S^n*. Now test if *C**x** = A*(B**x**)*. (Time *3n^2*)
- Claim: If *C* is not *A*B*, Probability[*C**x** = A*(B**x**)*] *<= 1/|S|*

Symbolic Determinant

- Recall the definition of a matrix determinant *det A*: *sum_{permutations pi} (-1)^sgn(pi) prod_{i} a_{i,pi(i)}*
- The definition makes sense when the matrix elements are in any ring R; in particular, we can have R can be ring of polynomials in variables $\boldsymbol{x}$ and say integer or rational coefficients.
- Let A be an n x n matrix and say each matrix entry *a_ij* is a linear polynomial, then *det(A)* is a degree n polynomial in the variables $\boldsymbol{x}$.
- The symbolic determinant problem is to determinant whether or not det(A) is the zero polynomial.

Motivation for symbolic determinant

- Suppose we consider the adjacency matrix for a (*n,n*) bipartite graph *G*. Suppose we wish to determine if *G* has a perfect matching. As we have seen, this problem can be solved in poly time by a transformation to max flow. But the max flow algorithm seems to be inherently sequential.
- We can solve the perfect matching problem by a transformation to the symbolic det problem. Namely, let *A_G* be defined by *a_ij = x_ij* if (i,j) is an edge; else 0. That is, the entries are linear in the variables ***x***.
- It is easy to observe that *G* has a perfect matching iff the *det(A_G)* is not the zero polynomial.

The complexity of symbolic det

- As a polynomial, *det(A)* could have *n!* terms and hence just writing out *det(A)* is not feasible for large *n*.
- But since *det(A*) is a degree *n* poly in the *x_ij*, we can invoke the Schwartz-Zipple lemma using say a set *S* of scalars with say *|S| >= 2n*. Then assuming *A* is not the zero polynomial, *prob_{s* uniform random in *S}* *[det(A|a = s) = 0]* is at most ½. Note that *det(A|a=s)* can be computed as fast as matrix product and can be efficiently computed in parallel.
- The symbolic det problem is one of the main examples of a decision problem that can be computed efficiently with randomization but (currently) not known to be deterministically computed in poly time.