CSC 373 Lecture 27

Announcements:

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

Term Test 2 question 3 regrading

Next assignment/test date

Hint for question 1b

Today

- Finish IP/LP rounding.
  - Review and finish makespan on unrelated machines.
  - Max Cut as an IP
  - Start randomized algorithms

IP/LP with a non naive rounding.

- The makespan problem for the unrelated machines model. The input consists of a given *m* (the number of machines) and *n* jobs *J_1,…,J_n* where each job *J_j* is represented by a vector <*p_1j,p_2j,…,p_mj*> where *p_ij* represents the processing time of job *J_j* on machine *i*. WLG *m* <= *n*.
- We will sketch a 2-approximation IP/LP rounding algorithm. This is the best known poly time approximation and it is known that it is NP hard to achieve better than 3/2 approximation even for the special case of the restrictive machines model for which every *p_ij* is either some *p_j* or infinity.
- Note: Unlike identical machines case, I do not know of any greedy or local search or DP O(1) approx alg.

In the IP formulation, the problem is:

minimize *t* subject to

sum_{1 <= I <=m} x_{i,j} = 1 for each job*J_j.*

sum_{1 <= j <=n} p_ij x_{i,j} <= t for each machine.

x_{i,j} in {0,1} The intended meaning is that *x_{i,j} = 1* iff job *J_j* is scheduled on machine *i*.

The LP relaxation is that *0 <= x_ij* ; (<=1 implied)

The integrality gap is unbounded! Consider one job with processing time *m*, which has *OPT = m* and *OPT_{LP} = 1*.

Getting around the integrality gap

- The IP must set x_{i,j} = 0 if p_{i,j} > t whereas

the fractional OPT does not have this constraint. We want to say for all (i,j): "if p_{i,j} > t then x_{i,j} = 0"

But this isn't a linear constraint!

Since we are only hoping for a good approx, we can assume all *p_ij* are integral. We can then use binary search to find the best LP bound *T* by solving the search problem *LP(T) for fixed T* eliminating the objective function and then removing any *x_{i,j}* having *p_{i,j} > T*. We clearly have that *IP-OPT >= T*.

Rounding of LP(T) solution.

- The *LP(T)* solution *x*_ij* is the solution of a system of *m+n* equations over the *mn* variables *x_ij*. LP Theory tells us that when there is a solution to this system there is a (so-called basic) solution *x*ij* with at most *m+n* positive values. This implies by counting that there are at most *m* fractional (not integral) values. If *x*ij = 1* then we assign job *j* to machine *i*. The remaining part of the proof (using more LP theory) is to show that there is a matching between the fractional *x*ij* and the machines.

A non obvious IP representation

- Consider the Max Cut problem. We can think of a solution as a choice about which vertices to (say) put into *A* in an *(A,B)* cut. We could have variables *y_i* {+1,-1} with the intended meaning *y_i* = 1 (resp -1) iff vertex *v_i* in *A* (resp. *B*).

- Then we would want to

maximize *sum_{1 <=i < j <=n} (1/2) w(i,j) (1-y_i y_j)*

subj to *y_i in* {+1,-1}; i.e. *y_i^2* = 1

Problem! While this leads to a very useful quadratic program (and a .878 approx) , it is NOT a linear program.

Max cut as an IP

Instead we will think of a cut as the edges crossing the *cut(A,B)* and have a {0,1} variable x_e for every edge *e* = *(u,v)* with the intended meaning that *x_e* = 1 iff *(u,v)* in the cut.

Now we need to find inequalities that will insure that the *{x_e | x_e = 1}* defines a cut.

This isn't at all obvious but here is what works.

Max *sum_{e in E} w_e x_e subj to x_e* in {0,1}

- *x_{ij} + x_{jk} >= x_{ik}* ; *x_{ik} + x_{kj} >= x_ij*, etc
  i.e. all permutations for every triangle *(v_i,v_j,v_k)*
- *x_{ij} + x_{ik} + x_{jk} <=2*

Why does this work?

You can think of these ``triangle inequalities'' as saying that the possible sizes of a cut for each triangle are 0 or 2.

Clearly every cut must satisfy these constraints and conversely we can show that every {0,1} solution of this IP defines a cut. This can be seen by the following argument:

- Define a relation $i \sim j$ if $x_{ij}=0$ or $i=j$

Show this is an equivalence relation: transitivity is the only thing to check, and by the triangle condition $x_{ij} = x_{ik} = 0$ implies $x_{jk} =0$.

Show that there are at most 2 equivalence classes. This follows from the second triangle condition; if i,j,k are in three different classes, then $x_{ij} + x_{ik} + x_{jk} = 3$. The equivalence classes are the cut.

Final topic of course: randomization

- We will show how to use randomization to either speed up computations and/or to improve an approximation and/or as a step towards a deterministic algorithm.
- There are computational settings (simulation, cryptography, sublinear time algorithms) where randomization is provably necessary.
- There are also problems where we do not know how to solve a problem efficiently without randomization.
- BUT as far as we know it could be that randomized polynomial time = polynomial time. In fact, this seems to be the current wisdom since if not the case then seemingly stranger things would result.
- We will recall probabilistic concepts as needed.

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*).
  $\mathbf{E}[W(F)] = (1/2)\,\mathbf{E}[W(F)|x = true] + (1/2)\,\mathbf{E}[W(F)|x = false]$
- Therefore at least one of these assignments must have 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 then 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.

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 *2n^2*)
- Claim: If *C* is not *A*B*, Probability[*C**x** = A*(B**x**)*] <= *1/|S|*