CSC2420: Lecture 7

- Todays agenda : Introduce randomization

1. The naïve random algorithm for MaxSat (revisionist history of Johnson's algorithm)
2. Randomized LP rounding for MaxSat
3. Randomized SDP rounding for Max2Sat
4. Modifying Johnson's algorithm
    1. The random order (Johnson) model
    2. The slack algorithm
    3. Limitations to the random order model and the slack algorithm

Randomized algorithms

- Our emphasis for the next couple of classes will be on randomized algorithms for problems where it is not a-priori clear to what extent randomization is necessary or helpful.
- This is in contrast to other applications where randomization is necessary; namely,
  - Simulation of stochastic processes
  - Cryptography
  - Sublinear time algorithms

MaxSat

- The (weighted) MaxSat problem: given a propositional CNF formula $F = C_1 \vee C_2 \vee \ldots \vee C_m$ (with weights w_i for clause C_i), find a truth assignment so as to maximize the (weight of) satisfied clauses.
- When there are at most (resp. exactly) k literals in each clause, this is called (exact) Max-k-Sat.
- Since 3-SAT is NP-complete, it is clear that MaxSat and Max-3-Sat are NP hard. Surprisingly, even though 2-SAT is poly time, Max-2-Sat is NP-hard and hard to approximate better than ~ .9401 under UGC (Austrin)

Naïve randomized algorithm for MaxSat: Revisionist view of Johnson's algorithm

- Lets consider the most naïve randomized algorithm for MaxSat. Namely, randomly set each variable to true (false) with prob ½.
- Suppose that every clause has at least k literals (as in exact Max-*k*-Sat). Let *W = sum_i w_i*, the sum of all clause weights, and let RW be the random variable defined as the weight of the solution of the naïve algorithm. The **E**[RW] >= *W * (2^k-1)/2^k* with equality for exact Max-*k*-Sat.
- What is not so clear is that for any CNF *F*, **E**[RW] >= (2/3) OPT.

Derandomizing the naïve algorithm

- We can use the method of conditional expectations to derandomize the naïve algorithm.
- $\mathbf{E}[\mathrm{RW}(F)] = \frac{1}{2}\,\mathrm{E}[\mathrm{RW}(F \mid x\_1 = t)] + \frac{1}{2}\,\mathbf{E}[\mathrm{RW}(F \mid x\_1 = f)]$
- At least one of the terms must be at least $\mathbf{E}[\mathrm{RW}(F)]$
- If we can determine which term is at least $\mathbf{E}[\mathrm{RW}(F)]$ then we can set x_1 accordingly.

The naïve algorithm derandomized to an *online* greedy algorithm

- Lets assume that we view the input as a set of input items where each item is a propositional variable. We represent each variable x by the clauses in which it appears. More specifically, here are three input models:
  - We specify the length of each clause *C_i* in which *x* appears positively, and for each clause *C_j* in which it appears negatively.
  - We list the names of variables in each such *C_i* and *C_j*
  - We give a complete specification of each clause in which x occurs

The resulting greedy online algorithm

- It is easy to see that the first (and weakest input model) is sufficient to derandomize the naïve algorithm. For each unset variable we keep track of the current length of each "positive" clause *C_i* and and each "negative" clause *C_j*; this allows us to determine the change in the expected values so as to determine $\mathrm{E}[\mathrm{RW}(F|x = t)]$ and $\mathbf{E}[\mathrm{RW}(F|x = f)]$.
- The resulting algorithm is called Johnson's algorithm since in fact it was stated by David Johnson before it was seen to be the derandomization of the naïve algorithm.

History of Johnson's algorithm

- David Johnson stated the algorithm in 1974 showing that it obtains at least weight W/2.
- Yannakakis (1994) observed that Johnson's algorithm was the derandomization of the naïve alg and that w(J) <=(2/3) OPT . A simple bad example is a 2 CNF with clauses *(x v y'), (x' v y), y'*
- Chen, Friesen, Zheng (1999) showed that w(J) >= (2/3) OPT.

Johnson's algorithm

For all *C_i, M_i = w_i/(2^|C_i|)*

Let *L* be the set of *clauses in F and X the variables*

For x in X (or until L empty)

Let P = {C_i in L such that x occurs positively}

Let N = {C_j in L such that x occurs negatively}

IF sum_{C_i in P} M_i} >= sum{C_j in N} M_j

x := t; L := F – P

For all *C_r* in *P*, *M_r := 2 M_r* End For

Else *x:= f, L:= L-N*

For all *C_r* in *N*, *M_r := 2M_r* End For

End For

The status of MaxSat

- For *k >= 3*, the worst case complexity status of exact Max-k-Sat is known since there is an NP hardness result (Hastad) matching the bound obtained by Johnson's algorithm.
- For MaxSat, Yanakakis gave a ¾ approximation using the naïve algorithm and a IP/LP algorithm with randomized rounding .
- For MaxSat (resp.Max-2-Sat), the best approximation .797 (resp. .931) use semi-definite programming (SDP) and randomized rounding. Conjecture UGC hardness is correct.

The LP with randomized rounding

- The IP/LP formulation: We have 0-1 IP variables *y_i* (corresponding to prop var *x_i*) and variables *z_j* (corresponding to clause *C_j*)
- We want to maximize sum_j *w_j z_j* subj to:
  sum_{*x_i* occurs positively in *C_j*} *y_i*
  + sum_{*x_i* occurs negatively in *C_j*} (*1-y_i*) *>= z_j*
  for all clauses *C_j*
- The LP relaxation is: *0 <= y_i <=1; 0 <=z_j <=1*
- We view the optimal LP variables {*y*_i*} as probabilities and round to 1 with that probability.
- It remains to compute the expected weight for the resulting truth assignment.

Need to analyze the Prob that C_j is satisfied in rounded solution.

- Let *C_j* be a clause with *k* literals and by renaming we can assume that *C_j = (x_1 v x_2 …v x_k)*.
- Let *b_k = 1- (1-1/k)^k*. We will show the *Prob[C_j satisfied]* is at least *b_k z*_j*. By linearity of expectations, the contribution (in expectation to the rounded solution) of a clause *C_j* having *k* literals is then at least *b_k w_j*. Since *(1-1/k)^k < 1/e* (and converges to *1/*e with *k*), the approx ratio is *>= 1-1/e > .632*. (We will need one further idea to obtain a (3/4) ratio.)

Prob[C_j satisfied]

- *C_j* is satisfied if not all of the *y_i* are set to 0 (when we set *y_i = 1* with probability *y*_i*).
- The probability that *C_j* is satisfied is then *[1 – product_i (1-y*_i)]*.
- By the arithmetic-geometric mean inequality this probability is then at least
  *1 – [(1/k) (1-y*_1) +…+ (1-y*_k)]^k*
  *= 1 – [(1-(1/k) (y*_1 + …y*_k)]^k >= 1 – [1-(z*_j/ k)]^k* where the inequality is by LP constraint:
  *sum_{y*_i in C_j} y*_i >= z*_j* (keeping in mind the renaming making literals positive).

End of analysis

- We define *g(z) = 1 - (1 – z/k)^k* then *g(z)* is a concave function with *g(0) = 0* and *g(1) = b_k*.
- By concavity, *g(z)>= (b_k)*z* for all *0 <=z<=1*.
- Hence if *C_j* is a clause with *k* literals, then the *Prob[C_j satisfied] >= (b_k) z*_j*
- Like the naive randomized alg, this algorithm can also be de-randomized to obtain a *(1-1/e)* approximation.
- The IP/LP development gives a (1-1/e) approximation. Here it is the big clauses that are more problematic whereas for the naïve method it is the small clauses. Taking the best of the two algorithms yields a ¾ approximation. (See text by Vazirani)

The SDP/vector program approach

- We define an integer quadratic program for Max-2-Sat having {-1,1} variables *y_i* (corresponding to the variables *x_i*) and one additional {-1,1} variable *y_0* (corresponding to the constant true). When *y_i = y_0* then we are setting *x_i* to true, else false; that is, in terms of {0,1} truth values, val(*x_i*) = (*1+y_i y_0*)/2 and val(*x'_i*) = (*1-y_i y_0*)/2. This takes care of the unit clauses. Consider a 2-clause say *C* = (*x_i v x_j*); val(*C*) = 1-val(*x'_i* and *x'_j*) = (*3+y_i y_0 + y_j y_0 - y_i y_j*)/4. Similar values can be *given for other 2 clauses; if x'_i occurs then replace y_i by –y_i* in val(*C*).

Vector program for Max-2-Sat

- The Max-2-Sat problem is then to maximize sum *w_k* val(*C_k*) subj to (*y_i*)^2 = 1 for all *i*
- Like an IP this integer quadratic program cannot be solved efficiently. In this case we now relax the quadratic program to a vector program where each *y_i* is now a vector *v_i* in Re^{*n+1}* and scalar multiplication is replaced by vector dot product. This vector program can be (approx) solved efficiently.
- The randomized rounding (to y_i) proceeds by choosing a random hyperplane in Re^{*n+1}* and then setting *y_i* to 1 iff it is on the same side of the hyperplane as *y_0*.

Outline of analysis (Vazirani text)

- The vector program can be rewritten as maximizing sum_{*0 <=i<j<=n*} *a_{ij} (1+v_i* v_j) + *b_{ij}* (*1-v_iv_j*). The expected value of the rounded solution is then 2 sum_{*0 <=i<j<=n*} *a_{ij}* Prob[*y_i =y_j] +* *b_{ij}* Prob[y_i neq y_j]
- A random hyperplane is determined by a random vector and then we let *t_{ij}* be the angle between *v_i* and v_j.
- Prob[*y_i neq y_j*] = *t_{ij}/pi >= (R/2) (1-* cos*(t_{ij}))* *where R = (2/pi) min_{0 <= t <=pi} (t*/(1-cos(*t*))>.87856
- Prob[yi=y_j] = 1-t_{ij}/pi >= *(1+* cos*(t_{ij}))*
- Hence expected value *>= R vector-OPT*

Returning to Johnson's algorithm

For all *C_i, M(C_i) = M_i = w_i*/(2^|*C_i*|)

Let *L* be set of clauses in *F* and *X* the variables*; S current* solution

For x in X (or until L empty)

Let P = {C_i in L such that x occurs positively}

Let N = {C_j in L such that x occurs negatively}

IF sum_{C_i in P} M_i} >= sum{C_j in N} M_j

x := t; L := F – P; S:= S+P

For all *C_r* in *P*, *M_r := 2 M_r* End For

Else *x:= f, L:= L-N; S:= S + N*

For all *C_r* in *N*, *M_r := 2M_r* End For

End For

Idea of 2/3 approximation ratio

- By changing variables, we can think of an optimal truth assignment pi as setting all variables to true. Then in the algorithm we can think of *N* at every iteration as the set of clauses that will be false at termination should the remaining unset variables be assigned according to *pi*. Extend the definition of *M* to a set by *M*(*N*) = sum_{*C* in *N*} *M*(*C*).
- The crux of the approx bound is to show that at every iteration, *D*(*S*) >= *2 D*(*M*(*N*)) where *D* is the change after setting *the variable in* that iteration.
- Thus w(S_e) – w(S_0) >= 2 [*D*(*M*(*N_e*) – *D*(*M*(*N_0*))]

Approximation ration continued

- $w(S\_0) = 0$; $w(S\_e)$ = *w(S) is the weight of satisfied formulas at termination; W = sum_{C in F} w(C*)
- $M(N\_0) <= (1/2)\, M(N\_0)$ (from ½ approx)
  $= (1/2)\,(\, W - w(\mathrm{OPT}))$
- $M(N\_e) = W - w(S\_e)$
- $w(S) - w(S\_0) >= 2\,[D(M(N\_e) - D(M(N\_0))]$
  $>= 2[W\text{-}w(S) - (1/2)\,(W\text{-}w(\mathrm{OPT}))]$
  $= W - 2w(S) + w(\mathrm{OPT})$
- Thus $w(S) >= [W + w(\mathrm{OPT})]/3 >= (2/3)\, w(\mathrm{OPT})$

Modifying Johnson's algorithm: What is best online/greedy algorithm?

- In proving the (2/3) approximation, Chen et al asked whether or not the ratio could be improved by using a random ordering of the propositional variables (i.e. the input items). This is an example of the random order model, a randomized variant of online algorithms and hence a special case of a randomized greedy or priority algorithm.
- This question was recently answered by Costello, Shapira and Tetali (SODA 2011) who show that in the ROM model, Johnson's algorithm achieves approximation (2/3 + eps) for eps ~ .003653

Can we do better?

- Poloczek and Schnitger (same SODA 2011) show that the approx ratio for Johnson's algorithm in the ROM model is at most 2 sqrt{15} – 7 ~ .746 < 3/4 , the ratio obtained by the IP/LP approach.
- By considering a modification of a canonical randomization of Johnson's rule (chosing to set x to true with prob $M(P)/(M(P)+M(N))$, they construct a new algorithm SLACK that does achieve expected ratio ¾. The idea is that since it is the singletons that are causing problems, one idea is let the prob[x set true] is now $M'_x(P)/M'_x(P) + M'_{x'}(N)$) where $M'_x(P)$ = M(x) + M(P) and M(x) is the contribution of unit clauses in which x occurs. This doesn't work in the problematic situation that M(P) and M(N) are close and an additional skewing of the probabilities is needed.

What are the limits to such combinatorial algorithms for MaxSat?

- Apparent contradictions: The Slack algorithm is a randomized online algorithm achieving approx ¾.
- Azar, Gamzu, Roth (ESA 11) show that no randomized algorithm can obtain an approx ratio better than 2/3.
- As observed in Azar et al, the negative result is for the weakest (priority) input model and the Slack algorithm is implemented in the middle model.
- Poloczek (ESA 11) shows that no deterministic priority algrorithm (in the middle model) can achieve the randomized ¾ ratio. In a strong sense this show that (unlike the naïve randomized algorithm), the Slack algorithm cannot be derandomized.

Submodular MaxSat

- The main result in Azar et al is that the monotone submodular MaxSat problem can be approximated with ratio 2/3 (in the weakest model).
- Submodular MaxSat is an instance of maximizing a submodular function subject to a binary partition matroid. Namely, we think of the universe U as being {*a_1,b_1*, …,*a_n,b_n*} where each pair {*a_i,b_i*} is a block of the partition and *a_i* (resp. *b_i*) corresponds to setting *x_i* true (resp. false). We then think of the submodular function *g* (on sets of clauses) as a function *f* on the subset of *U* chosen (subject to choosing at most one element from each pair). Namely, any such subset determines a set of satisfied clauses.
- The algorithm is a canonical randomized version of the natural (local) greedy algorithm for maximizing a submodular function subject to a partition matroid.