CSC 373: Algorithm Design and Analysis
Lecture 23

Allan Borodin

March 15, 2013

Announcements and Outline

Announcements

- I have now posted two questions on local search in addition to the previously posted three questions.

Today's outline

- Continue discussion of Exact Max-2-Sat
- Local search for makespan
- Some concluding remarks on local search
- Begin randomization

Review Exact Max-2-Sat

- **Given:** An exact 2-CNF formula

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m,$$

where $C_i = (\ell_i^1 \vee \ell_i^2)$ and $\ell_i^j \in \{x_k, \bar{x}_k \mid 1 \leq k \leq n\}$.

- In the weighted version, each C_i has a weight w_i.

- **Goal:** Find a truth assignment τ so as to maximize

$$W(\tau) = w(F \mid \tau),$$

the weighted sum of satisfied clauses w.r.t the truth assignment τ.

The natural oblivious local search

- A natural oblivious local search algorithm uses a Hamming distance d neighbourhood

$$N_d(\tau) = \{\tau' \mid \ \tau \text{ and } \tau' \text{ differ on at most } d \text{ variables}\}$$

Oblivious local search for Exact Max-2-Sat

1: Choose any initial truth assignment τ
2: **while** there exists $\hat{\tau} \in N_d(\tau)$ such that $W(\hat{\tau}) > W(\tau)$ **do**
3: $\quad \tau := \hat{\tau}$
4: **end while**

How good is this algorithm?

- Note: in what follows I will use approximation ratios < 1.
- It can be shown that for $d = 1$, the approximation ratio is $\frac{2}{3}$.
- In fact, for every formula, the algorithm finds an assignment τ such that $W(\tau) \geq \frac{2}{3} \sum_{i=1}^{m} w_i$, the weight of all clauses, and we say that the "totality ratio" is at least $\frac{2}{3}$.
- (More generally for Exact Max-k-Sat the ratio is $\frac{k}{k+1}$).
- This locality ratio is essentially a tight ratio for any $d = o(n)$.
- This is in contrast to a naive greedy algorithm derived from a randomized algorithm that achieves totality ratio $(2^k - 1)/2^k$.
- "In practice", the local search algorithm often performs better than the naive greedy and one could always start with the greedy algorithm and then apply local search.

Analysis of the oblivious local search for Exact Max-2-Sat

- Let τ be a local optimum and let
  - S_0 be those clauses that are not satisfied by τ
  - S_1 be those clauses that are satisfied by exactly one literal by τ
  - S_2 be those clauses that are satisfied by two literals by τ

  Let $W(S_i)$ be the corresponding weight.

- We will say that a clause involves a variable x_j if either $x_j$or $\bar{x}_j$ occurs in the clause. Then for each j, let
  - A_j be those clauses in S_0 involving the variable x_j.
  - B_j be those clauses C in S_1 involving the variable x_j such that it is the literal x_j or $\bar{x}_j$ that is satisfied in C by τ.

  Let $W(A_j), W(B_j)$ be the corresponding weights.

Analysis of the oblivious local search (continued)

- Summing over all variables x_j, we get
  - $2W(S_0) = \sum_j W(A_j)$ noting that each clause in S_0 gets counted twice.
  - $W(S_1) = \sum_j W(B_j)$

- Given that τ is a local optimum, for every j, we have

$$W(A_j) \leq W(B_j)$$

  or else flipping the truth value of x_j would improve the weight of the clauses being satisfied.

- Hence (by summing over all j),

$$2W_0 \leq W_1.$$

Finishing the analysis

- It follows then that the ratio of clause weights not satisfied to the sum of all clause weights is

$$\frac{W(S_0)}{W(S_0)+W(S_1)+W(S_2)} \leq \frac{W(S_0)}{3W(S_0)+W(S_2)} \leq \frac{W(S_0)}{3W(S_0)}$$

- It is not easy to verify but there are examples showing that this $\frac{2}{3}$ bound is essentially tight for any N_d neighbourhood for $d = o(n)$.

- It is also claimed that the bound is at best $\frac{4}{5}$ whenever $d < n/2$. For $d = n/2$, the algorithm would be optimal.

- In the weighted case, as in the max-cut problem, we have to worry about the number of iterations. And here again we can speed up the termination by insisting that any improvement has to be sufficiently better.

Using the proof to improve the algorithm

- We can learn something from this proof to improve the performance.
- Note that we are not using anything about $W(S_2)$.
- If we could guarantee that $W(S_0)$ was at most $W(S_2)$ then the ratio of clause weights not satisfied to all clause weights would be $\frac{1}{4}$.
- Claim: We can do this by enlarging the neighbourhood to include $\tau' =$ the complement of τ.

The non oblivious local search

- We consider the idea that satisfied clauses in S_2 are more valuable than satisfied clauses in S_1 (because they are able to withstand any single variable change).
- The idea then is to weight S_2 clauses more heavily.
- Specifically, in each iteration we attempt to find a $\tau' \in N_1(\tau)$ that improves the potential function

$$\frac{3}{2}W(S_1) + 2W(S_2)$$

instead of the oblivious $W(S_1) + W(S_2)$.

Sketch of $\frac{3}{4}$ totality bound for this non oblivious local search

- Without loss of generality (by renaming variables), assume we have a local optimum $\tau(x_j) = \text{true}$ for all variables x_j.
- Let $P_{i,j}$ be the weight of all clauses in S_i containing x_j.
- Let $N_{i,j}$ be the weight of all clauses in S_i containing $\bar{x}_j$.
- Here is the key observation for a local optimum τ wrt the stated potential:

$$-\frac{1}{2}P_{2,j} - \frac{3}{2}P_{1,j} + \frac{1}{2}N_{1,j} + \frac{3}{2}N_{0,j} \leq 0$$

- Summing over variables $P_1 = N_1 = W(S_1)$, $P_2 = 2W(S_2)$ and $N_0 = 2W(S_0)$ and using the above inequality we obtain

$$3W(S_0) \leq W(S_1) + W(S_2)$$

The jump local search algorithm for makespan on identical machines

- Start with any initial solution
- It doesn't matter how jobs are arranged on a machine so the algorithm can move any job (on a "critical machine" defining the current makespan value) if that move will "improve things".
  - That is, a (successful) jump move is one that moves any job to another machine so that either the makespan is decreased or the number of machines determining the current makespan is decreased.
- **Note:** this is a non-oblivious local search as we may not be decreasing the current makepsan in moving to a better solution.
- Finn and Horowitz [1979] prove: that the "locality gap" for this local search algorithm is $2 - \frac{2}{m+1}$. That is, this is the worst case ratio for some local optimum compared to the global optimum.
- To bound the number of iterations, in moving J_k, it should be moved to a machine having the current minimum load.

A more complicated local search for makespan

- The following local search algorithm is only being presented as an example of a more complicated neighbourghood.

- The jump local search does not provide as good an approximation as the LPT greedy algorithm and doesn't provide a constant (independent of m) approximation for the makespan problem in the uniformly related machines model.

- There is a more involved neighbourhood called the **push neighbourhood**, inspired by the Kernighan and Lin variable depth local search algorithms for graph partitioning and TSP.

Push operation

A push operation is a sequence of jumps defined as follows:

- A push is initiated by a jump of a job J_k on a critical machine to a machine M_i on which it "fits in the sense that

$$p_k + \sum_{J_j \text{ on } M_i \text{ and } p_j \geq p_k} p_j$$

  is less than the current makespan.

- If smaller (i.e. with $p_j < p_k$) jobs on M_i cause the makespan on M_i to equal or exceed the current makespan then in order of smallest jobs first, we keep moving small jobs to a priority queue.

- We then try to move jobs (in order of the largest job first) on the queue to a machine on which it fits and continue the process until either there is no machine on which it fits or the priority queue is empty.

Locality gaps for push local search

- Since a push optimal solution is also a jump optimal solution, it follows that the push local search has locality gap at most $2 - \frac{2}{m+1}$.
- The current lower bound on the locality gap is $\frac{4m}{3m+1}$
- The bound $\frac{8}{7}$ is tight for $m = 2$ and hence beats LPT for $m = 2$ machines.
- For uniformly related machines, the jump locality gap is at most $2 - \frac{2}{m+1}$ and the lower bound is arbitrarily close to $3/2$.
- Push does not give a constant (independant of input values) approximation for the restricted or unrelated machines models.

Some concluding comments (for now) on local search

- If time permits, we will return later to local search and in particular non-oblivious local search.
- But suffice it to say now that local search is the basis for many practical algorithms, especially when the idea is extended by allowing some well motivated ways to escape local optima (e.g. simulated annealing, tabu search).
- Although local search with all its variants is viewed as a great "practical" approach for many problems, local search is not often analyzed. It is not surprising then that there hasn't been much interest in formalizing the method and establishing limits.
- LP is itself often solved by some variant of the simplex method, which can also be thought of as a local search algorithm, moving fron one vertex of the LP polytope to an adjacent vertex.
  - No such method is known to run in polynomial time in the worst case.

Ford Fulkerson max flow algorithms

As mentioned before, Ford Fulkerson can be viewed as a local search algorithm.

Ford Fulkerson

1: Initialize $f := 0$ and $G_f := G$
2: **while** there is an augmenting path π in G_f **do**
3: $\quad f := f + f_\pi$ /* *Note this also changes* G_f */
4: **end while**

- As we already discussed, this is a scheme rather than a well specified algorithm since we have not said how one chooses an augmenting path (as there can be many such paths).
- And as we have noted, the Ford Fulkerson scheme is a local search algorithm where the neighbourhood $Nbhd(f)$ is the set of flows $\hat{f}$ that can be obtained by adding the flow from an augmenting path.
- This is a somewhat unusual local search algorithm in that any local optimum is a global optimum.

Final topic of the course: Randomization

Randomization

- Unlike our focus on topics such as greedy algorithms, dynamic programming, local search, IP/LP rounding, randomization is not a meta algorithm or algorithmic paradigm.
- Rather, randomization an idea that can be applied in any algorithm.
- We shall consider its use in a variety of applications.

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 the class of decision/search/optimization problems solvable in randomized polynomial time is the same as those solvable in polynomial time.

  - In fact, this seems to be the current wisdom of some experts 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 will show that a very naive use of randomization computes a truth assignment that (in expectation) satisfies a $\frac{2^k-1}{2^k}$ fraction of clauses (in the weighted case, the fraction of the total weight of all clauses).
- Let $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be an exact k-CNF formula.
- Let W_i denote the weight of the clause C_i.
- The algorithm sets each variable randomly (and independently) so that $\mathbb{P}[x_j = \mathit{true}] = \mathbb{P}[x_j = \mathit{false}] = 1/2$.
- Let X_i (resp. $\bar{X}_i$) denote the indicator random variable that is 1 if C_i is satisfied (resp. unsatisfied) and 0 otherwise.
- Let the random variable W_F be the total weight of all satisfied clauses in F.

Proposition

$$\mathbb{E}[W_F] = \left(\sum W_i\right) \cdot \frac{2^k - 1}{2^k}$$

Proving the proposition

Proposition

$$\mathbb{E}[W_F] = \left(\sum W_i\right) \cdot \frac{2^k - 1}{2^k}$$

- Recall **linearity of expectation** says that:

$$\mathbb{E}\left[\sum_i X_i\right] = \sum_i \mathbb{E}[X_i] \qquad \mathbb{E}[c \cdot X] = c \cdot \mathbb{E}[X]$$

- We observe that

Claim

$$\mathbb{E}[\bar{X}_i] = 1/2^k \text{ and hence } \mathbb{E}[X_i] = (2^k - 1)/2^k.$$

- By linearity of expectation, we have

$$\mathbb{E}[W_F] = \left(\sum W_i\right) \cdot (2^k - 1)/(2^k),$$

- **Note:** compare this with the oblivious local search.

Derandomizing 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 derandomize this algorithm by the method of conditional expectations to yield a deterministic (still online) greedy algorithm.
- The method works as follows.
  - 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.

Derandomizing the algorithm (continued)

- Consider the first variable assignment (say to x_1). We have

$$\mathbb{E}[W_F] = \frac{1}{2} \cdot \mathbb{E}[W_F \mid x_1 = \mathit{true}] + \frac{1}{2} \cdot \mathbb{E}[W_F \mid x_1 = \mathit{false}]$$

- 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_1 in each clause to which it belongs.

- Having set x_1 appropriately, we then can consider the next variable always maintaining the weight of satisfied clauses and the number of literals in each unsatisfied clause.

- Historical fact: The derandomized version of this algorithm is called Johnson's algorithm and was shown by Johnson well before it was realized (by Yannakakis) that Johnson's algorithm was the derandomized version.

From good expectation to good probability for almost the expectation

- In some sense the Exact 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 τ, it is "likely" to satisfy a "good" number of clauses.
  - Let's consider the number of unsatisfied clauses.
  - Given the expectation E, we can use Markovs inequality to show that

$$\mathbb{P}[\#\text{ of unsatisfied clauses } \geq c \cdot E] \leq \frac{1}{c}$$

  - For example, if say $c = 8/7$, then the probability is at most $7/8$.

Probability Amplification

- We have just shown that

$$\mathbb{P}[\#\text{ of unsatisfied clauses } \geq c \cdot E] \leq \frac{1}{c}$$

- To drive this probability down, independently repeat the "trial" t times so that the probability of always finding a τ with more than $(8/7) \cdot E$ 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 independent trials is a key aspect of randomized algorithms.

- **Useful fact:** $\left(1 - \frac{1}{t}\right)^t \leq 1/e$ for all t and limits to $1/e$ as $t \to \infty$.