Lecture 3: Averaging argumets in randomized algorithms; derandomization

Bogdan Warinschi. Updated for CSE 203, Spring 2003 by Russell

February 17, 2004

The theme of this class is: randomness can be an essential tool for devising algorithms, even when it later can be eliminated from them without much cost.

1 MAXSAT

Simple random algorithm for $MAXSAT$

Consider a set of literals:$X = \{x_1, x_2, ..., x_n\} \cup \{\neg x_1, \neg x_2, ..., \neg x_n\}$. A clause over this set is a boolean formula $l_1 \vee l_2 \vee ... \vee l_k$. The **MAXSAT** problem is defined as follows:

- **Input**: A set of clauses $\mathcal{C}$, $|\mathcal{C}| = m$ from a set of variables X;
- **Output**: A boolean assignment to the variables in X that satisfies the maximum number of clauses in $\mathcal{C}$;

It is well known that this problem is NP-hard, and even more, it is NP-hard to approximate within some constant. However some simple approximation algorithms exist.

For example, assigning random boolean values to the literals yields a $(1 - \frac{1}{2^k})$-approximation algorithm for **MAX** $\geq k$-**SAT**(a version of the **MAX-SAT** where all clauses have size $\geq k$). This can be seen easily, since each clause is satisfied with probability $\geq 1 - \frac{1}{2^k}$. The expected number of clauses to be satisfied, E, is $(1 - \frac{1}{2^k})m$ which is obviously grater or equal to $(1 - \frac{1}{2^k})$ fraction of the optimum. Moreover, if $p = Pr\{\#satisfied\ clauses\ \geq\ (1 - \frac{1}{2^k})m\}$ then $p > \frac{1}{m}$. This follows from the fact that $E \leq pm + ((1-p)(1 - \frac{1}{2}^k)m - 1)$.

Derandomizing the algorithm

Derandomizing using k-way independence properties of modular polynomials

A method that is commonly used in derandomizing algorithms is the following. Fix a prime $p > n > k$, and pick $a_0, a_1, ...a_{k-1}$ at random from $\{1, 2, ..., p-1\}$. Then the modular polynomial $h(X) = a_{k-1}X^{k-1} + ... + a_1 X + a_0 (mod\ p)$ has the following property (k-wise independence): let $x_i = h(i) \bmod 2$. Then knowing $x_{i_1}, x_{i_2}, ...x_{i_{k-2}}$, gives no information about the value of $x_{i_{k-1}}$, which is still uniformly distributed (possibly with a slight bias of $1/p$).

The previous algorithm can be derandomized by assigning values to $x_1, ...x_n$ according to this method (using such a polynomial), combined with exhaustive search over the space of such polynomials (space which is of order $O(n^k)$). That the probability of a clause being satisfied is $\geq 1 - 1/2^k$ depends only on the first k variables in the clause, so it is still (approximately) true for the above k-wise independent setting of the variables.

Derandomizing using the method of conditional expectations

For a fixed set of clauses $\mathcal{C}$, for a variable x we denote by $\mathcal{C}_x$ the set of clauses obtained from $\mathcal{C}$ by assigning to x the value true. Similarly, denote by $\mathcal{C}_{\overline{x}}$ the set obtained from $\mathcal{C}$ by assigning to x the value false. Also, denote by $\#_x$ the number of clauses which are satisfied if x is assigned true, and $\#_{\overline{x}}$ the number of clauses that are satisfied if x is assigned false.

We can express the expected number of clauses to be satisfied in two ways: $E(\mathcal{C}) = \sum_{C\in\mathcal{C}} (1 - \frac{1}{2^{|C|}})$, and also $E(\mathcal{C}) = \frac{1}{2}(E(\mathcal{C}_x) + \#_x)) + \frac{1}{2}(E(\mathcal{C}_{\overline{x}}) + \#_{\overline{x}})$.

This leads to the following algorithm:

(1) pick a variable x
(2) compute $E_1 = E(\mathcal{C})$, $E_2 = E(\mathcal{C}_x) + \#_x$,$E_3 = E(\mathcal{C}_{\overline{x}}) + \#_{\overline{x}})$
(3) if $E_1 \leq E_2$ then set x to true else set x to false;
(4) go to (1) if there are unassigned variables.

This deterministic algorithm always achieves at least the expectation, $(1 - 1/2^k)m$ clauses satisfied, because the expected number of satisfied clauses only increases from step to step. But it is still based on a probabilistic analysis.

1.1 A small improvement

The worst problem for the above algorithm is unit clauses, clauses that are just a variable or its negation. We can modify the above algorithm as follows: For each variable x, define its *sign* to be positive if the number of positive unit clauses x is at least the number of negated unit clauses $\neg x$, and negative otherwise. Let $1 > p > 1/2$ be a parameter to be determined later. For each variable independently, if the sign is positive, set it to 1 with probability p and to 0 with probability $1 - p$, and, if negative, to 0 with probability p and 1 with probability $1 - p$.

Note that every clause involving two or more literals is satisfied with probability at least $1 - p^2$. For any variable with $k_1 > k_2$ positive and negative occurences, the expected number of satisfied unit clauses is at least $pk_1 + (1 - p)k_2 \geq pk_1$, and similarly if x has more negated occurances. Note that the maximum possible number of satisfied unit clauses is k_1. So summing over all clauses in the maximum assignment and in the expectation, first for clauses greater than 2, then by unit clauses summed by variable, $E(\#satisfiedclauses) \geq min(1 - p^2, p)OPT$. To get the best ratio, we set $1 - p^2 = p$ and get $p = (\sqrt{5} - 1)/2$ which is around .6. This gives us a .6 approximation algorithm for this problem, which can be derandomized by either of the above methods.

2 Achieving the Average for Max Bisection

Consider the Max Bisection problem: Given a weighted, undirected graph $G = (V, E)$ with an even number of nodes, find a partition of V into two sets S, T with $|S| = |T|$ that maximizes the total weights of edges in $Cut(S, T)$. Give a *deterministic* algorithm that is guaranteed to find solutions to this problem within a factor of 2 of optimal. (Hint: this is very close to the Max Cut problem. The algorithms might also be very close.)

We first give a probability distribution on bisections, whose expected total weight is at least 1/2 the total weight. Arbitrarily label the nodes $a_1, b_1, ..a_{n/2}, b_{n/2}$. The probability distribution is: Independently for each $1 \leq i \leq n/2$, with probability 1/2 put $a_i \in S, b_i \in T$ and with probability 1/2 put $a_i \in T, b_i \in S$. Note that each edge between a_i and b_i is always in the cut, and each edge of any other form is in the cut with probability 1/2. So the expected cut weight is at least 1/2 the total weights.

We'll use the method of conditional expectations to make a deterministic algorithm with the same guarantee. Let S_i, T_i be the two sets after we have made the first i decisions above. Let E_i be the set of edges with at most one endpoint in $a_1, b_1, ..a_i, b_i$. Let M_i be the set of all edges between a_j and b_j for some $j > i$. and $a_{i+1}, b_{i+1}...a_n, b_n$. Then $E[Cut(S,T)|S_i,T_i] = \sum_{e \in Cut(S_i,T_i) \cup M_i} w(e) + 1/2 \sum_{e \in E_i - M_i} w(e)$. Since E_i and M_i depend only on i, not on S_i, T_i, the only part of this formula that depends on our decisions is $\sum_{e \in Cut(S_i,T_i)} w(e)$. So by maximizing the partial cut, we also maximize the expected total cut. Fix S_{i-1}, T_{i-1}, and let $SA_i = S_{i-1} \cup \{a_i\}, TA_i = T_{i-1} \cup \{b_i\}, SB_i = S_{i-1} \cup \{b_i\}, TB_i = T_{i-1} \cup \{a_i\}$. Then $E[Cut(S,T)|S_{i-1},T_{i-1}] = 1/2 E[Cut(S,T)|SA_i,TA_i] + 1/2 E[Cut(S,T)|SB_i,TB_i]$, and $E[Cut(S,T)|SA_i,TA_i] - E[Cut(S,T)|SB_i,TB_i] = Cut(SA_i,TA_i) - Cut(SB_i,TB_i) = \sum_{t \in T_{i-1}} w(\{a_i,t\}) - w(\{b_i,t\}) - \sum_{s \in S_{i-1}} w(\{a_i,s\}) - w(\{b_i,s\})$. So one of the two placements does not decrease this expectation, and we can determine which placement has the larger expectation is $O(n)$ time.

Thus, if we compute each S_i, T_i from S_{i-1}, T_{i-1} by placing a_i, b_i in the way that maximizes the partial cut $Cut(S_i,T_i)$, as above, our algorithm takes $O(n^2)$ time total to compute a cut that achieves the initial expectation, which is at least $1/2$ the total weights of all edges. Thus, this algorithm produces a cut that is at least half the weight of the maximum cut in the graph.