Robin Moser makes Lovász Local Lemma Algorithmic!

Notes of Joel Spencer

1 Preliminaries

The idea in these notes is to explain a new approach of Robin Moser [1] to give an algorithm for the Lovász Local Lemma. This description is of the approach as modified and improved by Gábor Tardos. We don't strive for best possible or most general here. In particular, we stick to what is called the symmetric case.

Lets start with a particular and instructive example. Let x_i, $1 \leq i \leq n$ be Boolean variables. Let C_j, $1 \leq j \leq m$ be clauses, each the disjunction of k variables or their negations. For example, with $k = 3$, $x_8 \vee \overline{x_{19}} \vee x_{37}$ would be a typical clause. We say two clauses overlap, and write $C_i \sim C_j$, if they have a common variable x_k, regardless of whether the variable is negated or not in the clauses. A set of clauses is called mutually satisfiable if there exists a truth assignment of the underlying variables so that each clause is satisfied or, equivalently, if the $\wedge$ of the clauses is satisfiable.

Theorem 1.1 *Assume, using the above notation, that each clause overlaps at most d clauses (including itself). Assume*

$$2^{-k}\frac{d^d}{(d-1)^{d-1}} \leq 1 \tag{1}$$

Then the set of clauses is mutually satisfiable. Moreover (and this is the new part) there is an algorithm that finds an assignment for which each clause C_j is satisfied that runs in linear time in n, with k, d fixed.

Here is a more general setting. Let Ω be a set of size n. For $v \in \Omega$ let X_v be independent random variables. For $1 \leq j \leq m$ let $e^j \subseteq \Omega$ and let B_j be an event that depends only on the values X_v, $v \in e^j$. We say two events overlap, and write $B_i \sim B_j$, if $e^i \cap e^j \neq \emptyset$.

Theorem 1.2 *Assume, using the above notation, that each event overlaps at most d events (including itself). Assume*

$$\Pr[B_j] \leq p \text{ for all } j \tag{2}$$

[1] Moser is a graduate student (!) at ETH, working with Emo Welzl

and that

$$p\frac{d^d}{(d-1)^{d-1}} \leq 1 \tag{3}$$

Then

$$\wedge_{j=1}^{m}\overline{B_j} \neq \emptyset \tag{4}$$

Moreover (and this is the new part) there is an algorithm that finds an assignment of the X_i for which each $\overline{B_j}$ holds that runs in linear time in n, with k, d fixed.

To say the implication of Theorem 1.1 from Theorem 1.2 consider a random assignment X_v of the variables x_v. That is, each X_v independently takes on the values true, false with probabilities one half. The "bad" event B_j is then that the clause C_j is not satisfied, which has probability 2^{-k}. The event that none of the bad events occur is nonempty. By Erdős Magic, there is a point in the probability space, which is precisely an assignment of truth values, such that no bad event occurs, which is precisely that the clauses are all simultaneously satisfied.

We will write the proof in the more general form, but the example of Theorem 1.1 is a good one to keep in mind.

The time of the algorithm actually will depend on some data structure assumptions which we omit.

`The Moser-Tardos (ML) Algorithm`.

1. Give X_v random values from their distributions.
2. WHILE some B_j holds.
3. PICK some B_j that holds.
4. Reset the X_v, $v \in e_j$, independently
5. END WHILE

The selection mechanism for PICK can be arbitrary. For definiteness, we may pick the minimal j for which B_j holds, but it doesn't affect the proof. We just need some specified mechanism for PICK.

As this is a randomized algorithm, its output may be and will be considered a random variable. Let B_t, e_t be the event and underlying set in the t-th iteration of the WHILE loop. We shall refer to this as *time t* in the running of ML. We define the LOG of the running of the algorithm to be the sequence $e_1, \ldots, e_t, \ldots$. A priori, there is no reason to believe that this algorithm will actually terminate, and so the LOG might be an infinite sequence. On the other extreme, the initial random assignment might work in which case LOG would be the null sequence.

For convenience we let $H = \{e^1, \ldots, e^m\}$ so that the $e \in H$ are just the possible values of the e_t.

For $e \in H$ let $COUNT[e]$ denote the number of times e appears in LOG, that is, the number of times t for which $e = e_t$. A priori this could be infinite. But our main result is:

Theorem 1.3

$$E[COUNT[e]] \leq \frac{1}{d-1} \tag{5}$$

Given this result, linearity of expectation gives

Theorem 1.4 *The expected length of LOG is at most $\frac{m}{d-1}$ where m is the number of events.*

As each event overlaps at most d events, each $v \in \Omega$ can be in at most d events, and so $m \leq nd$. Theorem 1.4 then gives that the expected length of LOG is linear in the size of Ω. This is why we call the MT algorithm linear time, though in particular instances one would need further assumptions about the data structure.

The remainder of the argument is a proof of Theorem 1.3.

Given a running of MT with the LOG of size at least t we define $TREE[t]$ to be a rooted tree with vertices labelled by the $e \in H$. (Note: Several vertices may have the same label.) The root of $TREE[t]$ is e_t. Now we construct the tree by reverse induction from $i = t-1$ to $i = 1$. (When $t = 1$ the tree has only the root e_1.) For a given i we check whether there is a j, $i < j \leq t$, such that e_i, e_j overlap and e_j has already been placed in the tree. If there is no such j we go on to the next i, that is, we do not put e_i in the tree. If there is such a j select that j for which e_j is lowest (that is, furthest from the root, this part is important!) and add to the tree by making e_i a child of e_j. In case of ties, use an arbitrary tiebreaker, for example, pick that j with the smallest index.

$TREE[t]$ gives a concise description of those e_i that are relevant to e_t. It has certain key tautological properties.

• The $TREE[t]$ are all different.

Reason: If $s < t$ and $TREE[s], TREE[t]$ were equal, they would have to have the same root $e = e_s = e_t$. In creating $TREE[t]$ each time $e_i = e$ for $1 \leq i \leq t$ there will be another node e in the tree. (When $i < t$ as e_i does overlap e_t it is placed in the tree.) That is, e appears in the tree precisely the number of times in appears in $e_1, \ldots, e_t$. When $e = e_s = e_t$, however, these numbers will be different for $TREE[s], TREE[t]$ as all the copies of e in $TREE[s]$ are in $TREE[t]$ and e_t is in $TREE[t]$ but not $TREE[s]$.

• The $e \in TREE[t]$ on the same level of the tree do not overlap.

Reason: Suppose $r < s$ and $e_r, e_s \in TREE[t]$ and suppose they did overlap. When e_r is placed in the tree it is placed as low as possible. Since e_s is already in the tree it is placed on the level below e_s or even lower.

• When $e_r, e_s \in TREE[t]$ overlap and $r < s$, e_r is lower than e_s.

Reason: Above.

• Let $v \in \Omega$ and let $f_0, \ldots, f_s$ be the nodes of $TREE[t]$ that contain v. Order these by the depth of the node in the tree with the first being the furthest from the root. (From above there will be no ties.) Then the fs will be in this order in the LOG.

Reason: Say $0 \leq i < j \leq s$. After f_j was placed in $TREE[t]$ the later (in creating $TREE$) f_i overlaps f_j and so is placed on the level below f_j or even lower.

• Furthermore, there will be no other e in the LOG that contain v and come before f_s.
Reason: All such e overlaps f_s and so would be placed in the $TREE$.

Let T be a rooted tree with vertices labelled by $e \in H$ and such that when f is a child of e, f, e overlap. For each such T let $OCCUR[T]$ be the event that $T = TREE[t]$ for some t. Let $|T|$ denote the number of nodes of T.

Theorem 1.5
$$\Pr[OCCUR[T]] \leq p^{|T|} \tag{6}$$

While the proof of Theorem 1.5 is short, it is subtle and we begin with two simple examples. Suppose T consists solely of the root e. Then $OCCUR[T]$ means that at some time t in the running of ML an event B_t held and values X_i, $i \in e_t = e$ were changed. But it further means that there had been no e_s overlapping e before this time. That is, the values X_i, $i \in e_t = e$, were unchanged from their original values. Thus for $OCCUR[T]$ to hold it is necessary that B_t holds with the original values of the X_i and this occurs with probability at most p. Now suppose T consists solely of the root e and a child f. Suppose f arises at time s and then e arises at time $t > s$. The subtlety arises with those $i \in \Omega$ lying in both e and f. For these we must distinguish the original value of X_i and the revised value after time s. It is necessary that B_s hold with the original values of X_i and this occurs with probability at most p. It is further necessary that B_t holds using the revised values X_i for $i \in e \cap f$ and the original values for the other $i \in e$. This also occurs with probability at most p and, more importantly, that probability that both B_s and B_t hold is at most p^2. This is because in looking at B_t we are looking at different "coin flips" for the values X_i.

We argue the general case for Theorem 1.5 by preprocessing the randomness. That is, each variable $v \in \Omega$ independently makes a countable number of evaluations of X_v, labelled $x_v^0, x_v^1, \ldots$. Following this preprocessing ML is deterministic. When v needs to reset X_v for the i-th time, it takes the value x_v^i, x_v^0 being the original evaluation. We call i above the evaluation number. Preprocessing is a powerful tool in analyzing random algorithms, though it has no affect on the actual running of the algorithms.

While $TREE[t]$ does not determine LOG, or even the precise order of the appearances of the nodes of $TREE[t]$ in the LOG, it does determine the order of appearance of the nodes $f_0, \ldots, f_s$ containing any given vertex v. When the ML algorithm reached f_i, v had value x_v^i. That is, for each $e \in TREE[t]$ and each vertex $v \in e$ there is an evaluation number i so that when ML reaches e the vertex v had value x_v^i. Critically, this i is determined by the $TREE$ (even though many values of LOG could yield the same $TREE$) and the particular node e and vertex v. For $OCCUR[T]$ it is *necessary* that

For each $e \in TREE$ the associated bad event B occurs where for each $v \in e$, v is using the i-th evaluation with i the evaluation number as determined above.

For each $e \in TREE$ this occurs with probability at most p as the various evaluations are independent. Moreover, critically, the various events B are mutually independent as for each v one uses different evaluation number for each one. Thus the probability that all the B hold at their various "times" is at most $p^{|T|}$, completing Theorem 1.5.

Now we turn to proving Theorem 1.3 by bounding $E[COUNT[e]]$. As the values $TREE[t]$ are tautologically distinct, no T can be counted more than once in $COUNT[e]$ and so

$$E[COUNT[e]] = \sum_T \Pr[OCCUR[T]] \leq \sum_T p^{|T|} \tag{7}$$

where the sum is over all finite trees with root e, and the second inequality is from Theorem 1.5.

We bound the Right Hand Side of 7 using standard methods from algebraic combinatorics. Let `PURE` denote the infinite rooted tree in which every node has precisely d children. Let `ACTUAL` be the infinite tree rooted at e in which f has as children all g which overlap it. Our basic assumption is that each node in `ACTUAL` will have at most d children and so `ACTUAL` is a subtree of `PURE`. Hence the sum for subtrees T of `ACTUAL` is at most the sum for subtrees of `PURE`. Here we have an exact result.

Theorem 1.6 *Assume*

$$p \leq \frac{(d-1)^{d-1}}{d^d} \tag{8}$$

Then, taking the sum over all subtrees of `PURE` *rooted at the root of* `PURE`*,*

$$y = \sum_T p^{|T|} \tag{9}$$

is finite, and is that unique y, $0 \leq y \leq \frac{1}{d-1}$, satisfying

$$y = p(1+y)^d \tag{10}$$

`Proof:` Assumming the sum is finite, we find 10 by considering the terms when there are i children of the root in T. Each subtree then gives a contribution of y and they are independent so the total contribution is y^i. Thus

$$y = p\sum_{i=0}^{d} \binom{d}{i} y^i = p(1+y)^d \tag{11}$$

In general let y_s be the sum 9 over all T of depth at most s. Then $y_0 = p$ and 11 becomes

$$y_{s+1} = p\sum_{i=0}^{d} \binom{d}{i} y_s^i = p(1+y_s)^d \tag{12}$$

and basic analysis shows that this sequence approaches a fixed point y as desired.