Algorithmic version of the local lemma

Uri Feige

January 27, 2010

1 The local lemma

Let $\mathcal{A}$ be a collection of random events $A_1 \dots A_m$. For event A_i, let $\Gamma(A_i)$ be a minimal set of events that A_i depends on in the sense that A_i is independent of all events in $\mathcal{A} \setminus \{\Gamma(A_i) \cup A_i\}$ (information on events not in $\Gamma(A_i) \cup A_i$ does not affect the probability of event A_i happening).

The general version of the Lovasz local lemma [3] (see also [1]) is as follows.

Theorem 1 *If there is an assignment $0 < x_i < 1$ satisfying for every i:*

$$Pr[A_i] \leq x_i \prod_{j | A_j \in \Gamma(A_i)} (1 - x_j)$$

then the probability that no event A_i happens is at least $\prod_{i=1}^{m}(1 - x_i) > 0$.

Observe that if the events were independent, the probability that no bad event happens would have been exactly $\prod_{i=1}^{m}(1 - Pr[A_i])$. Hence in a sense, replacing the $Pr[A_i]$ values by the larger x_i values is the penalty paid in the conclusion of the local lemma for the variables not being truly independent.

In many applications, the following uniform version of the local lemma suffices.

Theorem 2 *Let $\mathcal{A}$ be a collection of random events each happening with probability p. Assume that every event depends on at most d events (including itself). Namely, $|\Gamma(A_i)| + 1 \leq d$. Then if $pde < 1$ then with positive probability no event happens.*

Proof: The uniform version follows from the general version by picking $x_i = pe$ for every i. Then we have

$$x_i \prod_{j | A_j \in \Gamma(A_i)} (1 - x_j) \geq pe(1 - pe)^{d-1} \geq pe(1 - 1/d)^{d-1} \geq p$$

$\square$

Here is an application to the local lemma.

Theorem 3 *Every k-CNF formula with n variables in which each variable appears in less than $2^k/ek$ clauses is satisfiable.*

Proof: Consider a random assignment to the variables, and let A_i be the event that clause C_i is not satisfied. Then $Pr[A_i] = 2^{-k}$. Note that removing all events that correspond to clauses that share variables with the clause correspond to event A_i leaves A_i independent of all remaining events. Each clause shares variables with less than $1 + k(2^k/ek - 1) < 2^k/e$ clauses including itself. Hence the uniform version of the local lemma implies that there is positive probability that the formula is satisfied by the random assignment, and hence some assignment is satisfying. $\square$

Note that the randomized algorithm implicit in the proof of Theorem 3 has exponentially small success probability, and hence will need to be repeated an exponential number of times until it is expected to produce a satisfying assignment. Finding a polynomial time ("constructive") version for the local lemma has been an open question for many years. The first breakthrough in this respect was by Joseph Beck [2], which gave a constructive version (with weaker parameters, and that applies only in some cases). Another recent breakthrough by Robin Moser (extended further in [4]) managed to produce a constructive version that among other things provides a polynomial time algorithm for the problem in Theorem 3.

2 An algorithmic version

The algorithmic versions of the local lemma do not always apply, but they do apply in most interesting cases. We shall show a random polynomial time algorithm for satisfying sparse k-CNF formulas as in Theorem 3. This algorithm can be derandomized and generalized to other applications, but this will not be discussed here. The analysis (of the expected running time) that we provide is not the tightest possible, since we try to keep it simple and intuitive. For stronger results, see [4].

Let $x_1, \ldots, x_n$ be the variables of a k-CNF formula and let m be the number of clauses. Assume that each clause shares variables with at most d clauses (including itself). Let $p = 2^{-k}$ be the probability that a clause is not satisfied by a random assignment to the variables. We assume that $pde < 1$ and show a simple randomized algorithm for satisfying the k-CNF formula.

1. Pick a random assignment for the variables, where each variable is set to true independently with probability 1/2.
2. As long as there is some clause that is not satisfied, pick an arbitrary clause that is not satisfied, and assign fresh random value to all its variables.

We shall show that the above algorithm terminates with a satisfying assignment in expected polynomial time. The expectation is taken over the random coin tosses of the algorithm. These are produced dynamically throughout the algorithm. It will be more convenient for us to view them in an equivalent way, as if they are produced statically before the algorithm begins. For this we imagine that there is a table R with r rows (r can be thought of as infinite or very large) and n columns. The columns are indexed by the variables. Initially, one fills the table by random and independent 0/1 values. Thereafter, whenever the algorithm needs a fresh random value for a variable, in considers the column corresponding to the variable, and uses the first yet unused entry in that column. All probabilistic statements that we make are with respect to the random initial contents of R.

Consider now the sequence of unsatisfied clauses visited by the algorithm, $C_1, C_2, \ldots$. Note that C_i and C_j for $i \neq j$ may refer to the same clause. Consider clause C_t that is visited in step t of the algorithm. How do we determine which k entries of the table R gave the value of its variables that made it unsatisfied at time step t? We know which columns they belong to, but the row that each value belongs to depends on the history of the execution of the algorithm. In particular, different variables might take their value from different rows.

The part of the execution of the algorithm that is relevant to C_t can be viewed as a tree T_t labelled by clauses. It is build by backward induction from step t towards step 1. At step i for $t \geq i \geq 1$ we have the tree T_t^i. It is derived from tree T_t^{i+1} as follows. If clause C_i does not intersect any of the clauses in T_t^{i+1} then $T_t^i = T_t^{i+1}$. Else, C_i is appended to the clause C_j deepest in the tree T_t^{i+i} that shares a variable with C_i.

The tree T_t^1 determines uniquely from which locations in R the random values for the variables of each of the clauses C_i are taken, for *all* clauses C_i in the tree (and not just for C_m). Simply start with the deepest node in the tree (breaking ties arbitrarily, which is justified by Proposition 4), allocate to the respective clause the values from the respective columns in the first row of R, erase these values and shift the respective columns down by one row, and repeat.

Proposition 4 *If two clauses C_i and C_j with $i < j$ are at the same depth in T_t^1, then they are disjoint.*

Proof: Otherwise, C_i could have been placed at greater depth, as a descendant of C_j. □

For the random values in R to make the generation of the tree T_t^1 *feasible*, it must be the case that all clauses in the tree are not satisfied by the respective values in R. Let q be the number of clauses in T_t^1. The probability of the event that the tree T_t^1 is feasible is exactly p^q. Note that even if given R, the tree T_t^1 is feasible, it does not imply that the tree is actually generated by the algorithm. For example, in step t there might be two overlapping unsatisfied clauses, and the algorithm might choose any one of them, avoiding the creation of the other respective feasible tree.

Lemma 5 *Given the table R and $q \geq 1$, for the algorithm to run for qm steps it must be the case that R contains a feasible tree of size at least q.*

Proof: Among the first qm clauses visited by the algorithm, at least q of them represent the same clause. All q copies of the same clause must belong to the tree associated with the last appearance of this clause in the sequence of visited clauses. □

It follows that if R is such that no feasible tree of size q or more exists, the algorithm must terminate after at most qm steps. (Observe that it might be the case that no feasible tree of size q exists, but a feasible tree of larger size does exist, since each step of the algorithm adds a root to a tree, and not a leaf, and this root might have degree larger than 1.)

Let us count how many legally labelled trees of size q there are at all. A tree is legally labelled if it is a rooted tree labelled by clauses, every two adjacent nodes are labelled by clauses that share a variable, and the clauses corresponding to any two nodes in the same level of the tree do not intersect. In particular, this implies that any node in the tree has at most d immediate children (as it intersects at most d different clauses).

Pick an arbitrary order among the m clauses. There are m ways of choosing the root. We represent the remaining tree as a Boolean vector of length dq with $q-1$ entries that are 1. For every clause chosen into the tree, we append to the end of the vector d coordinates, representing the (at most) d neighbors of the clause (and some of the coordinates might not represent any clause at all if there are less than d neighbors). Their initial value is 0. Scan the coordinates one by one, and for each coordinate that represents a clause that is in the tree, set the corresponding entry in the vector to 1 (and append d coordinates). It can readily be seen that any legal tree can be represented in such a fashion, and that any vector corresponds to at most one legal tree. Hence the number of legal trees with q nodes is at most $m\binom{dq}{q-1} \leq m(de)^q$.

Hence the probability over R that there is a feasible legal tree of size Q or more is at most $m\sum_{q=Q}^{\infty}(de)^q p^q = m(dpe)^Q/(1-dpe)$. For large enough Q, (e.g., $Q = \Theta(\log m)$ suffices if dpe is bounded away from 1), the probability that a feasible legal tree of size at least Q exists tends to 0. This implies both that the formula is satisfiable, and that the randomized algorithm will find a satisfying assignment in expected $O(mQ)$ steps.

References

[1] Noga Alon and Joel Spencer. The Probabilistic Method. Wiley-Interscience Series in Discrete Mathematics and Optimization.

[2] Joseph Beck. An algorithmic approach to the Lovasz Local Lemma. Random Structures and Algorithms, 2 (1991), pp, 343-365.

[3] Paul Erdos and Laszlo Lovasz. Problems and results on 3-chromatic hypergraphs and some related questions. In Infinite and Finite Sets, volume 11 of Colloq. Math. Soc. J. Bolyai, pages 609-627. North-Holland, 1975.

[4] Robin Moser and Gabor Tardos. A constructive proof of the general Lovasz Local Lemma. Technical Report Corr.abs/0903.0544, Computing Research Repository, 2009.