IIT Kanpur

CS396A: Undergraduate Project-II

Approximation Algorithms using Randomized Rounding

Author:
Utkarsh Patange
11493

Guide:
Dr. Surender Baswana

April 15, 2014

Contents

Abstract

Approximation algorithms provide a novel way to solve otherwise intractable problems, e.g. problems those are NP-hard. Due to the growing belief that $P \neq NP$ and the need to solve many NP hard problems, approximation algorithms are fast gaining importance. One way to obtain such algorithms is to solve a relaxed version of the problem, one that can be solved efficiently, and then randomly rounding its solution to obtain the solution to the original problem. We look at problems that can be solved using linear programming first, and then look at a very powerful tool - Lovász Local Lemma - that has many applications in proving bounds on the approximation ratio/running time of various algorithms.

0.1 Introduction

It is well known that many optimization problems are intractable, i.e. they are NP hard. Such problems still needs to be solved efficiently. Due to there being no solution for finding their exact solution (there is no solution to such problems unless $P = NP$, which is not believed to be trued), we turn to designing algorithms that will provide us with some approximate solutions. The aim here is to find algorithms that will always give an output which is *proven* to be appropriately close to the actual optimum solution of the problem. In chapter 1, we look at linear programming problems, which we will later use in chapter 2 to find randomized rounding algorithms that will give an approximate solution to some problems. After that, in chapter 3 we will state and prove the general Lovász local lemma. We will present its two proofs: a non-constructive (existential) proof that appeared in [LE75],[AS00] and a constructive proof that later appeared in [MT10]. In chapter 4, we discuss an application of the local lemma.

Chapter 1

Linear Programming

1.1 Introduction

Linear programming is concerned with optimizing (minimizing or maximizing) a linear function in, say, n variables subject to, say, m linear inequalities called constraints. There are a number of techniques to solve linear programming problems like simplex algorithms, ellipsoidal method. Simplex algorithm can take exponential time in the worst case, but works better than ellipsoidal method (studied by Leonid Khachiyan, 1979) - which is polynomial time algorithm - in practice.

The material presented here was taken from Wikipedia, which in turn had taken it from [Mur83].

1.2 LP Duality

An important feature of an LP problem is the duality. That is, for every minimization problem there is a corresponding maximization problem associated with it and solving any one of them would answer the other. For example, let us take the following problem:

$$\begin{array}{lllll} \textit{maximize} & P & = & x_1 + 7x_2 - 5x_3 & \\ \textit{subject to} & p_1 & = & 5x_1 - x_3 & \leqslant \ 24 \\ & p_2 & = & 8x_1 + x_2 + 3x_3 & \leqslant \ 7 \\ & p_3 & = & -x_1 + x_2 + 2x_3 & \leqslant \ 32 \\ & & & x_1, x_2, x_3 & \geqslant \ 0 \end{array}$$

Suppose, we multiply p_1, p_2, p_3 by positive y_1, y_2, y_3 respectively to get a polynomial Q' such that the coefficients of x_i's in P are less than or equal to those in Q', then P will be bounded by Q' which will in turn be bounded by $Q = 24y_1 + 7y_2 + 32y_3$. Note that, y_i's need to be positive so that the sign of inequality doesn't change while adding. minimizing Q will give the tightest upper bound on P and hence will be the maximum value P can attain. So, we get another optimization problem (minimization this time) which can be described the following polynomials. The constraint inequalities ensure that

coefficients of x_i's in P are less than or equal to those in Q.

$$\begin{array}{lllll} \textit{minimize} & Q & = & 24y_1 + 7y_2 + 32y_3 & \\ \textit{subject to} & q_1 & = & 5y_1 + 8y_2 - y_3 & \geqslant \ 1 \\ & q_2 & = & y_2 + y_3 & \geqslant \ 7 \\ & q_3 & = & -y_1 + 3y_2 + 2y_3 & \geqslant \ -5 \\ & & & y_1, y_2, y_3 & \geqslant \ 0 \end{array}$$

This minimization problem is called the dual of the earlier maximization problem. If the minimum value attained by Q is r, then we have:

$$P \leqslant Q' \leqslant Q$$

which holds true for all values of x_i's and y_i's and hence will hold true even for those y_i's for which Q is minimum, i.e.

$$P \leqslant Q' \leqslant r$$

Thus the minimum value of Q is the maximum value of P. Hence, to solve any linear programming problem, one can also solve it's dual which in turn is another linear programming problem.

1.3 Simplex Algorithm

The most practically used method to solve a linear programming problem is the simplex algorithm. Simplex algorithm is applied on linear programs in the standard form. In this form, there is an objective function to be minimized, there is a set of equations (and not inequalities) and every variable in the equation is positive. Every linear programming problem can be converted to the standard form.

This algorithm relies on the observation that any extremum of any linear function must lie on at least one vertex of the polyhedron defined by the constraints. The polyhedron will be in n dimensions and will have m surfaces each corresponding one inequality. Also, it will always be a convex polyhedron, i.e. a line joining any two points inside the polyhedron will be entirely enclosed by the polyhedron.

1.3.1 Idea

In essence, the algorithm computes the function f, that is to be optimized, at any vertex of the polyhedron. Then checks if it is decreasing along some edge. If yes, it chooses the other vertex at the end of this edge and repeats the procedure until there is no edge along which f increases. If the edge in question doesn't have another vertex, i.e. it is infinite, then f is unbounded below and the problem has no solution. The value of f at this vertex is the answer to the problem. The algorithm will terminate since there can only be finite number of vertices to a polyhedron. It can be proved that if on a vertex, f is not minimum, then there must be at least one edge from that vertex along which f decreases. Thus, the algorithm will give correct answer for any input.

1.3.2 Conversion to standard form

First we look at a method to convert any given linear program to the standard form. For any inequality of the form $s \leqslant a$ we introduce a variable $t \geqslant 0$, and replace the inequality by $s + t = a$. Similarly, $s \geqslant a$ would be replaced by $s - t = a$. Also, for a particular variable z that is unbounded, we take two variables x, y such that $x \geqslant 0, y \geqslant 0$ and substitute z by $x - y$. Thus, any given linear program can be converted to its standard form.

1.3.3 Applying the simplex algorithm

Given a linear program in the standard form:

$$\begin{array}{lllll} \textit{minimize} & c^T x & & & \\ \textit{subject to} & Ax & = & b & \\ & x_i & \geqslant & 0 & \forall i \in [1, n] \end{array}$$

where $c = (c_1, c_2, \ldots c_n)$ is a column vector with coefficients of variables x_i in the objective function, $x = (x_1, x_2, \ldots, x_n)$ is the column vector of variables, A and b are matrix and vector representing the constraint equations in the linear program respectively. To solve the linear program, we need a way to find the value of objective function at the vertices of the polyhedron (polytope for higher dimensions) defined by the constraint equations. For that, we need a way to identify vertices of the polytope.

Looking at the matrix as a combination of column vectors, it can be observed that $A_1x_1 + A_2x_2 + A_3x_3 + \ldots + A_nx_n = b$ where A_i is i^{th} column of A. If $x = \alpha$ is a solution to this problem, and say, $\alpha_i \neq 0 \quad \forall i \in [1, n]$, then $\{A_i | i \in [1, n]\}$ is a linearly independent set. Because if it weren't and, say, A_n was a linear combination of the others, then we would surely find many more points satisfying the same equation as a result of more degrees of freedom. If some α_i were to be 0, the same can be said about other A_i's for which α_i's aren't 0.

By using Graham-Schimdt orthonormalization method, we can obtain a set of orthogonal unit column vectors which are a linear combinations of the original column vectors. This can be used to find out a set of n linearly independent vectors. If there is no such set, then there is no vertex to the polytope and hence, no solution to the problem. Solving this set by gaussian elimination (By converting the matrix to identity matrix by row operations) method will give us a vertex on the polytope.

Now, we need a method to traverse an edge of the polytope that starts from this vertex. To do this, we first distinguish between variables that were 0 in the first vertex from those that weren't.After the gaussian elimination applied above, some variables were being multiplied by columns of identity matrix. Let's call these variables "basic variables" and others "non-basic variables". If we convert a non-basic variable to basic and a non-basic to basic, we will get another set of linearly independent vectors and thus another vertex of the polytope. This can be done simply by choosing a non-basic column i and converting it to I_r that is, the r^{th} column of the identity matrix by row operations. This process will change the r^{th} column of previous identity matrix which can thus be made non-basic for next iteration. Thus, we get to an adjacent vertex of the polytope.

To identify whether we got to an optimal solution or not, we need a way to check if the objective function will decrease or increase on swapping a pair of

basic and non-basic variables. This can be done by looking at the coefficients of the corresponding variables. If the coefficient of some non-basic variable in the objective function is negative, this variable should be chosen as the "Entering variable". Once the entering variable is chosen, the "leaving variable" can be chosen by the requirement that the resulting solution be feasible.

It must be noted that the algorithm terminates if there is no choice for entering variable. In this case, the solution at hand is the optimal solution. Also, r is chosen such that r^{th} entry in A_i is positive. If there are no positive entries, then any value assigned to the entering variable will be feasible and it will obviously decrease the objective function (since the coefficient was negative for the entering variable). Thus, in this case there is no lower bound for the objective function.

If there are multiple possible choices for the entering and/or leaving variable, any one of them can be chosen. Though, there are some methods that optimize the running time by narrowing down the choices, the worst case running time could still be exponential.

Chapter 2

Randomized Rounding problems

2.1 A Wiring Problem

This problem and its solution is given in [MR97] and is a great example of how randomized rounding can be applied to solve problems.

2.1.1 Problem Statement

There is a $\sqrt{n} \times \sqrt{n}$ grid of unit squares which needs to be wired. We are given some wires and the squares they should connect. The wires can only turn at right angle and at most once. At all times, the wires run parallel to the grid boundaries. So, the wires will have to pass boundaries between adjacent squares to go to the destination. Let w be the maximum number of wires that pass through a certain boundary. The problem is to minimize w.

First we convert it into an equivalent Integer Linear Programming (ILP) Problem

2.1.2 Equivalent ILP Problem

Since only a single right turn is allowed, there may be a maximum of two routes possible for each wire to go from one square to another. We use the variables x_{i0} and x_{i1} to denote which route we chose. $x_{i0} = 1$ if we go horizontally first, and 0 otherwise. $x_{i1} = 1$ if $x_{i0} = 0$. On fixing the values of x_{i0} and x_{i1}, we obviously know which boundaries the i^{th} wire will pass. So we introduce sets for each boundary b:

$T_{b0} = \{i|$ net i passes through b if $x_{i0} = 1\}$
$T_{b1} = \{i|$ net i passes through b if $x_{i1} = 1\}$

Thus, we get the following ILP:

$$\begin{array}{lccl} \text{Minimize} & w & & \\ \text{Subject to,} & x_{i0} + x_{i1} & = & 1\,\forall i \\ & x_{i0}, x_{i1} & \in & \{0,1\}\forall i \\ & \sum_{i\in T_{b0}} x_{i0} + \sum_{i\in T_{b1}} x_{i1} & \leq & w\,\forall b \end{array}$$

To get a relaxed version which can be solved in polynomial time, we replace the integer constraints by $x_{i0}, x_{i1} \in [0,1]$. But, now we need to map this to the original problem.

2.1.3 Mapping LP solution to Original Problem

Since the LP problem is a relaxed version of the ILP one, we have: $w_{LP} \leq w_{ILP}$. After mapping the solution for the LP to the original problem, let the reported answer be w_{rep}. This can be found out from the wiring generated by the mapped solution. We also need to prove that w_{rep} is provably not very bigger than w_{ILP}.

Applying randomized rounding, we report $x_{ij,rep} = 1$ with probability $x_{LP}\,\forall i$ and $\forall j \in \{0,1\}$ and 0 otherwise. This ensures that if x_{ij} is close to 0 for some i and j, there will be high probability for it to be reported as 0.

Let $w_{rep}(b)$ denote the number of wires through a boundary b. By definition, we have:

$$\begin{array}{rcl} w_{rep}(b) & = & \sum_{i\in T_{b0}} x_{i0,rep} + \sum_{i\in T_{b1}} x_{i1,rep} \\ E[w_{rep}(b)] & = & \sum_{i\in T_{b0}} E[x_{i0,rep}] + \sum_{i\in T_{b1}} E[x_{i1,rep}] \\ & = & \sum_{i\in T_{b0}} x_{i0,LP} + \sum_{i\in T_{b1}} x_{i1,LP} \\ & \leq & w_{LP} \end{array}$$

Since all $x_{ij,rep}$ are independent Bernoulli random variables (with probability $x_{i0,LP}$ each), we can apply Chernoff Bound. Thus, we get:

$$\begin{array}{rcl} Pr(w_{rep}(b) \geq (1+\delta)w_{LP}) & \leq & Pr(w_{rep}(b) \geq (1+\delta)\mu) \\ & \leq & \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} \end{array}$$

Now, we have proved that $w_{rep}(b)$ probabilistically very close to w_{LP}. By union theorem and the fact that there are less than $2n$ boundaries to a $\sqrt{n}\times\sqrt{n}$ grid, we get: $Pr(w_{rep} \geq (1+\delta)w_{LP}) \leq 2n\left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{w_{LP}}$. and since $w_{LP} \leq w_{ILP}$

$$\begin{array}{lrcl} Pr(w_{rep} \geq (1+\delta)w_{ILP}) \leq Pr(w_{rep} \geq (1+\delta)w_{LP}) & & \leq & 2n\left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{w_{LP}} \\ \text{Taking } \delta = 1 & Pr(w_{rep} \geq 2w_{ILP}) & \leq & 2n\left(\frac{e}{2^2}\right)^{w_{LP}} \\ \text{For } w_{LP} = c\log n \text{ it becomes,} & Pr(w_{rep} \geq 2w_{ILP}) & \leq & 2n \times n^{-c} \\ \text{But for constant } w_{LP}\text{,say 20,} & Pr(w_{rep} \geq (1+\delta)\times 20) & \leq & 2n\left(\frac{e^{20\delta}}{(1+\delta)^{20(1+\delta)}}\right) \\ \text{We need } \delta \geq 2e-1 & & \leq & 2ne^{-c\delta^2/4} \\ \text{We need } \delta = \Omega(\sqrt{\log n}) \text{ for similar bound. Here the probability} & & \leq & 2n \times n^{-k} \end{array}$$

Thus, we have achieved a tight upper bound on the probability that our reported answer to the wiring problem using randomized rounding is not very different from the actual answer. It must be noted, however, that the tightness of the bound depends upon the actual answer w_{LP}. If w_{LP} is higher the bound is tighter as opposed to when w_{LP} is low.

2.2 Minimum Cut about Two Points

The cut defined here is different from the cut defined in flow related problems. This problem is an excellent example in which the randomized solution is, not approximately, but exactly equal to the actual solution. This appeared in [Sri99]

2.2.1 Problem Statement

Given an undirected weighted graph G and two special vertices s and t, the problem is to cut the graph such that s and t are not connected after the cut. Of all the possible such cuts, we need to minimize the total weight of the edges in the cut. We will first design an Integer Linear Programming problem and solve its relaxed version (which will be a linear programming problem and has polynomial time algorithms). This solution will then be mapped to the original problem using randomized rounding.

2.2.2 Equivalent ILP Problem

Given a cut, We take indicator variables x_i for each vertex setting $x_i = 0$ if the vertex is reachable from s and 1 otherwise. Hence $x_s = 0, x_t = 1$. z_{ij} is an indicator variable for the edges indicating whether the edge crosses the cut or not. The equivalent integer linear programming problem can be stated as follows:

$$\begin{array}{rlll} \text{Minimize} & \multicolumn{2}{l}{\sum_{0 \le i,j \le n} c_{ij} z_{ij}} \\ \text{Subject to,} & z_{ij} & \geq & x_i - x_j \\ & z_{ij} & \geq & x_j - x_j \\ & x_s & = & 0 \\ & x_t & = & 1 \\ & x_i & \in & \{0,1\}\, \forall i \\ & z_{ij} & \in & \{0,1\}\, \forall i \forall j \end{array}$$

It can be easily seen that for any optimal solution, $z_{ij} = |x_i - x_j|$ will hold. To solve this ILP problem, we relax the integer constraints and solve the LP problem by known polynomial time algorithms. The LP solutions will then have to be mapped to integer feasible solutions (following the constraints) to find the solution to the original problem

2.2.3 Mapping LP solutions to Original Problem

Relaxing the integer constraints to $x_i \in [0,1]$ and $z_{ij} \in [0,1]$, let us say we get the solution x_i^* and z_{ij}^* with objective function value $y*$. Obviously, $y^* \leq y$ where y is the optimum solution to the original problem.

To map the ILP solution back to the original problem by randomized rounding, we choose a number $u \in [0,1)$ randomly uniformly. We report $x_i = 0$ if $x_i^* \leq u$ and $x_i = 1$ otherwise. Even in this linear programming problem, in an optimum solution $z_{ij}^* = |x_i^* - x_j^*|$ will hold. But, to make the mapped solution feasible in the original problem instance, we will report $z_{ij} = |x_i - x_j|$. Now, z_{ij} will be 1 if $u \in [min(x_i, x_j), max(x_i, x_j)]$. Thus, $\mathrm{E}(z_{ij}) = |x_i^* - x_j^*| = z_{ij}^*$.

Now, the reported answer to the original problem is $y = \sum_{0 \leq i,j \leq n} c_{ij} z_{ij}$. We can find the expected value of the above expression by linearity of expectation:

$$\begin{aligned} E[y] &= E[\sum_{0 \leq i,j \leq n} c_{ij} z_{ij}] \\ &= \sum_{0 \leq i,j \leq n} c_{ij} E[z_{ij}] \\ &= \sum_{0 \leq i,j \leq n} c_{ij} z^*_{ij} \\ &= y^* \end{aligned}$$

This mapped solution follows all the constraints of the original ILP problem. Since answer to the original problem must have objective function value $OPT \geq y^*$. Since $E[y] = y^*$, by pigeon-hole principle we must have $y \leq y^*$ for some $u \in [0, 1)$. But $OPT \leq y$ since OPT is, after all the optimum objective function value. Hence, we get $y = OPT = y^*$.

This is an example where the solution to the relaxed version of the ILP problem is also the actual solution to the same problem. Hence, we don't need to come up with any probability bounds on the answer.

2.3 Maximum Satisfiability

This is a problem where we apply two approaches to find an approximate solution to a given problem and then merge the two solutions probabilistically to get a better approximation to the optimal solution. This problem is given in [Sri99]

2.3.1 Problem Statement

Given m clauses in n variables and weights w_i associated with each clause, we need to maximize the sum of weights of satisfied clauses.

Clearly, this is an NP-hard problem since SAT can be reduced to Max-SAT. All we have to do is to find the solution to Max-SAT (with $w_i = 1 \forall i \in [n]$ and check if this satisfies all the clauses.

2.3.2 Applying Randomized Rounding

In this case, we will explore two different methods for randomized rounding. One will work when $|C_i|$ is small and other when it is large. Here $|C_i|$ denotes the number of literals in i^{th} clause C_i. For a given problem we will choose any one of the method randomly uniformly and prove that this process will give a good approximation ratio in expectation.

Method 1

Here, we will simply assign values from {true,false} to every variable randomly uniformly and independently of the others. Clearly,

$$p_{i,1} = Pr[C_i \ is \ satisfied] = 1 - 2^{-|C_i|}$$

Obviously, this method will produce good enough results only when individual $|C_i|$ are large enough.

Method 2

This method uses a technique called Arithmetization to convert the satisfiability problem to ILP problem. Given m clauses each having variables from the set $\{x_i | i \in [n]\}$, let us define two sets for each clause: $P(i) = \{j | x_j$ appears unnegated in $C_i\}$ and $N(i) = \{j | x_j$ appears negated in $C_i\}$. Letting $z_i \in \{0, 1\}$ we need to maximize $\sum w_i z_i$ subject to,

$$\forall i, z_i \leq \sum_{j \in P(i)} x_j + \sum_{j \in N(i)} (1 - x_j) \tag{2.1}$$

Here, x_j is set to 1 if the corresponding boolean variable is set to true in the Max-SAT problem and 0 otherwise.

Again applying LP relaxation (allowing $z_i \in [0, 1]$), we obtain the solution in the form of x_j^* and z_j^*. Letting the objective function value for this LP problem be y^*, we have y^* as an upper bound on OPT which is the optimum value for the objective function in the original ILP problem.

To map the LP solution to the original problem, we will assign $x_j = 1$ with probability x_j^*.

Without loss of generality assuming that all the variables appearing in C_i are non-negated, from $z_i \in [0, 1] \Rightarrow z_i^* \leq 1$ and equation 2.1 we get:

$$z_i^* = min \left\{ \sum_{j \in P(i)} x_j^*, 1 \right\} \tag{2.2}$$

We have, $Pr[C_i \text{ is satisfied}] = 1 - \prod_{j \in P(i)} (1 - x_j^*)$. From equation 2.2, it is clear that this probability will be minimum when all x_j^*'s are set to the same value (From the A.M. $\geq$ G.M. inequality). From equation 2.2 this minimum value can be attained when $x_j^* = z_i^* / |C_i|$. Thus, we get:

$$p_{i,2} = Pr(C_i \; is \; satisfied) \geq 1 - (1 - z_i^* / |C_i|)^{|C_i|} \tag{2.3}$$

Clearly, for a fixed z_i^*,$p_{i,2}$ decreases monotonically as $|C_i|$ increases.

Now, we need to choose one of the two methods randomly uniformly and analyse the how well this process does. We get,

$$\begin{aligned} Pr(C_i \; is \; satisfied) &= (p_{i,1} + p_{i,2})/2 \\ &\geq 1 - \frac{\left(2^{-|C_i|} + \left(1 - \frac{z_i^*}{|C_i|}\right)^{|C_i|}\right)}{2} = G \end{aligned}$$

To find a lower bound for G consider the function:

$$\begin{aligned} f(l, x) &= 1 - \frac{\left(2^{-l} + \left(1 - \frac{x}{l}\right)^l\right)}{2} - \frac{3x}{4} \\ \frac{\partial f(l,x)}{\partial x} &= \frac{\left(1 - \frac{x}{l}\right)^{l-1}}{2} - \frac{3}{4} \\ &\leq \frac{1}{2} - \frac{3}{4} && \text{Putting } x = 0 \\ \therefore \frac{\partial f(l,x)}{\partial x} &\leq 0 && \forall x \in [0, 1] \end{aligned}$$

Thus, $f(l, x)$ decreases with as $x \in [0, 1]$ increases. Hence, its minimum value must occur at $x = 1$. We have, $f(1, 1) = f(2, 1) = 0$ and for $l \geq 3$, $2^{-l} \leq 1/8$ and $(1 - 1/l)^l \leq 1/e$. Hence, $f(l, 1) \geq 1 - \frac{1}{16} - \frac{1}{2e} - \frac{3}{4} \geq 0 \; \forall l \in \mathbb{Z}$. Since

minimum value of $f(l, x)$ is also greater than 0 for a fixed l, $f(l, x) \geq 0 \ \forall x$ and therefore $G \geq \frac{3}{4}z^*$.

Thus, by linearity of expectation, expected value of objective function for reported solution $= \sum_i (3/4)z^* = (3/4)y^* \geq (3/4)OPT$.

It can also be observed that the bound achieved by this analysis cannot be improved. If we have all the 4 possible clauses (having equal weights) in 2 variables x_1 and x_2, any setting of values to x_1 and x_2 will give $OPT = 3$, but setting all of them to 0.5 will give $y^* = 4$. Hence, there are situations where $OPT = (3/4)y^*$. This means if we guaranteed $y_{rep} \geq f(y^*) > (3/4)y^*$, it will fail in this case since y_{rep} is always $\leq OPT (= (3/4)y^*$ in this case).

Chapter 3

Lovasz Local Lemma: Proof

3.1 Lemma statement

The asymmetric Lovász local lemma [LE75] states the following:

Theorem 3.1.1. *Let $\mathbb{A}$ be a finite set of events in a probability space. For $A \in \mathbb{A}$ let $\Gamma(A) \subset \mathbb{A}$ such that A is independent from the collection of events $A \setminus (\{A\} \cup \Gamma(A))$. If $\exists x : \mathbb{A} \longrightarrow (0,1)$:*

$$\forall A \in \mathbb{A} : \Pr[A] \leq x(A) \prod_{B \in \Gamma(A)} (1 - x(B))$$

then

$$\Pr(\cap_{A \in \mathbb{A}} \overline{A}) \geq \prod_{A \in \mathbb{A}} (1 - x(A))$$

In particular, the probability is positive

First we study the non-constructive proof of theorem 3.1.1 given in [AS00]

3.2 Non-constructive Proof[1]

We prove the asymmetric version of the Lemma. By using the principle of mathematical induction we prove that $\Pr\left(A| \cap_{B \in S} \overline{B}\right) \leq x\left(A\right) \forall S \subset \mathcal{A}, A \notin S$. The induction here is applied on the size (cardinality) of the set S. For base case $S = \phi$ the statement obviously holds since $\Pr\left(A_i\right) \leq x\left(A_i\right)$. We need to show that the inequality holds for any subset of $\mathcal{A}$ of a certain cardinality given that it holds for all subsets of a lower cardinality.

Let $S_1 = S \cap \Gamma(A), S_2 = S \setminus S_1$. We have from Bayes' theorem

$$\Pr\left(A| \cap_{B \in S} \overline{B}\right) = \frac{\Pr\left(A \cap \cap_{B \in S_1} \overline{B}| \cap_{B \in S_2} \overline{B}\right)}{\Pr\left(\cap_{B \in S_1} \overline{B}| \cap_{B \in S_2} \overline{B}\right)}$$

[1] The same proof has been added by me in the wikipedia article of Lovász Local Lemma (https://en.wikipedia.org/wiki/Lovász_Local_Lemma) under the username Uspatange on 31^{st} March 2014

We bound the numerator and denominator of the above expression separately. For this, let $S_1 = \{B_{j1}, B_{j2}, \ldots, B_{jl}\}$. First, exploiting the fact that A does not depend upon any event in S_2.

$$\begin{aligned} Numerator &\leq \Pr\left(A| \cap_{B\in S_2} \overline{B}\right) \\ &= \Pr(A) \leq x(A) \prod_{B\in\Gamma(A)} (1 - x(B)) \end{aligned} \tag{3.1}$$

Expanding the denominator by using Bayes' theorem and then using the inductive assumption, we get

$$\begin{aligned} Denominator &= \Pr\left(\overline{B}_{j1}| \cap_{t=2}^{l} \overline{B}_{jt} \cap_{B\in S_2} \overline{B}\right) \cdot \Pr\left(\overline{B}_{j2}| \cap_{t=3}^{l} \overline{B}_{jt} \cap_{B\in S_2} \overline{B}\right) \cdot \ldots \\ &\quad \ldots \cdot \Pr\left(\overline{B}_{jl}| \cap_{B\in S_2} \overline{B}\right) \\ &\geq \prod_{B\in S_1} (1 - x(B)) \end{aligned} \tag{3.2}$$

The inductive assumption can be applied here since each event is conditioned on lesser number of other events, i.e. on a subset of cardinality less than $|S|$. From equations 3.1 and 3.2, we get

$$\Pr\left(A| \cap_{B\in S} \overline{B}\right) \leq x(A) \prod_{B\in\Gamma(A)-S_1} (1 - x(B)) \leq x(A)$$

Since value of x is always in (0,1). Note that we have essentially proved $\Pr\left(\overline{A}| \cap_{B\in S} \overline{B}\right) \geq 1 - x(A)$. To get the desired probability, we write it in terms of conditional probabilities applying Bayes' theorem repeatedly. Hence,

$$\Pr\left(\overline{A_1} \cap \ldots \cap \overline{A_n}\right) = \Pr\left(\overline{A_1}|\overline{A_2} \cap \ldots \overline{A_n}\right) \cdot \Pr\left(\overline{A_2}|\overline{A_3} \cap \ldots \overline{A_n}\right) \cdot \ldots \cdot \Pr\left(\overline{A_n}\right) \leq \prod_{A\in\mathcal{A}} (1 - x(A))$$

Which is what we had intended to prove.

3.3 Constructive Proof

The proof stated above doesn't give a way to find such a setting in which none of the undesirable events happen. Now we study a constructive proof for theorem 3.1.1 given in [MT10]. The proof given here asumes certain things about the nature of the events and gives a randomized algorithm with an expected bound on the running time that gives a way to find such a setting.

3.4 Assumptions and Terms Used

To get an algorithmic access to the problem, we assume that all the events can be determined by some subset of a finite collection of mutually independent random variables in a fixed probability space. This indeed is the case in almost all known applications of the Lovász Local Lemma.

Terms used:

1. $\Omega :=$ A fixed probability space
2. $\mathbb{P} :=$ Set of mutually independent random variables in Ω
3. $\mathbb{A} :=$ Set of events determined by some $S \subseteq \mathbb{P}$
4. An evaluation of variables in S *violates* $A \in \mathbb{A}$ if it makes A happen
5. $vbl(A) :=$ The unique minimal $S \subseteq \mathbb{P}$ that determines A
6. $G_{\mathbb{A}}(\textit{Dependency Graph}) := (\mathbb{A}, E)$ where E is a set of undirected edges and $(A, B) \in E$ if $A \neq B, vbl(A) \cap vbl(B) \neq \phi$
7. $\Gamma(A) := \Gamma_{\mathbb{A}}(A)$ is the neighborhood of A in G
8. $\Gamma^+(A) := \Gamma(A) \cup \{A\}$

Given the family $\mathbb{A}$, we need to find an algorithm to efficiently find such an evaluation that does not violate any of the events $A \in \mathbb{A}$

3.5 Pseudo Code

```
for P ∈ ℙ do
    v_P ← a random evaluation of P
end
while ∃A ∈ 𝔸 : A is violated when (P = v_P : ∀P ∈ ℙ) do
    pick an arbitrary violated event A ∈ 𝔸
    for P ∈ vbl(A) do
        v_P ← a new random evaluation of P
    end
end
return (v_P)_{P∈ℙ}
```

Algorithm 1: lllfind

It is obvious that if algorithm 1 terminates, it would have definitely found the required evaluation of the random variables. We need to prove that this algorithm not only terminates, but it terminates quickly i.e. there is some bound on the number of iterations.

We will prove that the randomized algorithm 1 resamples and event $A \in \mathbb{A}$ at most an expected $x(A)/(1-x(A))$ times before it terminates. For this purpose, we use the concept of Witness Trees and Execution Logs.

3.6 Execution Logs and Witness Trees

Let's say we fix the method to choose the next event that is to be resampled among all of the events that are violated during an iteration. Let the sequence in which we resample the events be C called an execution log. That is, C is a sequence in which the t^{th} entry is the event that is sampled in the t^{th} step. Thus C now depends entirely upon the random evaluations of the different random variables in $\mathbb{P}$.

A witness tree τ is a finite rooted tree each of whose vertices are labeled by an event from $\mathbb{A}$. We will denote label of a vertex v by $[v]$ and the set of vertices by $V(\tau)$. Each children of a vertex u must receive labels from $\Gamma^+([u])$. We call τ a proper witness tree if all siblings have distinct labels.

We associate a witness tree with each step of the log. The intuition is that the witness tree will *justify* the resampling done in that step. For a log C we define $\tau_C(t)$ for each step t in C. We will construct $\tau_C(t)$ incrementally as follows:

```
1  τ_C^(t)(t) ← an isolated vertex labeled C(t)
2  for i ← t − 1 to 1 do
3  |  ρ_i ← {v ∈ τ_C^(i+1)(t) | C(i) ∈ Γ⁺([v])}
4  |  τ_C^(i+1) = τ_C^(i)
5  |  if ρ_i <> φ then
6  |  |  v ← a vertex in ρ_i that is farthest from root
7  |  |  u ← new vertex
8  |  |  [u] ← C(i)
9  |  |  child_{i+1}(v) ← child_i(v) ∪ u
10 |  end
11 |  τ_C(t) ← τ_C^(1)(t)
12 |  return τ_C(t)
13 end
```

Algorithm 2: Construct (C, t)

Due to the second step in the construction, we can say that no two vertices having labels that are dependent on each other can ever be siblings. This also proves that any witness tree $\tau_C(t)$ will be proper for any log C and step t in C.

We say that a witness tree τ appears in a log C if for some step t $\tau = \tau_C(t)$.

Theorem 3.6.1. *Let τ be a fixed witness tree and C the (random) log produced by algorithm 1. Then,*

1. *If τ appears in C, then τ is not only proper, but τ has mutually independent labels at each depth.*
2. *The probability that τ appears in C is at most $\prod_{v \in V(\tau)} \Pr([v])$*

Proof. Let us assume $\tau = \tau_C(t)$ for some step t. For a vertex $v \in V(\tau)$ we define:

1. $d(v) :=$ Distance from root to v, also called depth of v
2. $q(v) := \max_{i \leq t}\{i | v \in V(\tau_C^i(t))\}$. That is $q(v) > q(u)$ means that v was added to τ earlier than u in algorithm 2

Note that $q(v) > q(u)$ and $[u] \in \Gamma^+(v)$ means u must be added to the tree at a position farther from root than v as per algorihm 2. That is, $d(u) > d(v)$. This implies, among other things, that τ must be proper.

We define a procedure called $\tau-$check to prove the 2nd result in theorem 3.6.1. We say τ passes the check if check(τ) returns true.

```
S ← Sequence of vertices in the reverse order of their being visited in
  BFS traversal
pass ← true
for i in 1 to |V(τ)| do
    v ← S_i
    P ← vbl([v])
    v_P ← random evaluation of variables in P
    if v_P does not violates [v] then
        pass ← false
    end
end
return pass
```

Algorithm 3: check(τ)

It is obvious from algorithm 3 that $\Pr(\tau \text{ passes } \tau-\text{check}) = \prod_{v \in V(\tau)} \Pr([v])$. To link the $\tau-$check with the log, we assume that both of them are run on same set of random sources. We will formalize this definition shortly. If we prove that whenever τ appears in C, it passes the check on the same random source, then we have proved the lemma. This follows from Bayes' theorem.

Let $E_1 :=$ event that τ appears in C and $E_2 :=$ check(τ) passes. Assuming E_1 implies E_2 (so, $\Pr(E_2|E_1) = 1$) We get,

$$\begin{aligned} \Pr(E_2|E_1) \cdot \Pr(E_1) &= \Pr(E_1|E_2) \cdot \Pr(E_2) \\ \therefore \Pr(E_1) &\leq \Pr(E_2) \\ &= \prod_{v \in V(\tau)} \Pr([v]) \end{aligned}$$

So, to prove the second result in theorem 3.6.1, we only need to prove that $\Pr(E_2|E_1) = 1$.

To establish a relation in $\tau-$check and C, we formalize the random selections. In particular, we define for every $P \in \mathbb{P}$ a sequence $S_P = P_0, P_1, P_2, \ldots P_i \ldots$ and say that envoking a random source for the i^{th} time assigns $P = P_{i-1}$. Also, same set of random sources are used for algorithms 1 and 3. Note that once the sources are used for algorithm 1, they are all reset before using in 3, that is, a particular source S_P will output P_0 both when it is used for the first time in algorithm 1 and when it is used for the first time in algorithm 3. This establishes a relationship between the two random operations and now we will prove that E_1 implies E_2.

Assuming E_1 has happened, that is $\tau = \tau_C(t)$ for some step t, to prove that $\tau-$check passes, we need to look at what must have happened while executing a particular iteration of algorithm 1. When the algorithm considers a vertex v, and takes a random evaluation of $vbl([v])$, for every $P \in vbl([v])$ it must have called upon their respective random sources S_P. Taking $W_v(P) := \{w \in V(\tau) | d(w) > d(v), P \in vbl([w])\}$ for any random variable $P \in \mathbb{P}$, it can be seen that when v is considered in algorithm 1, due to the reverse BFS ordering of the $\tau-$check, the source S_P has already been used *exactly* $|W_v(P)|$ times. This is because no vertex u with $d(u) = d(v)$ can have $[v] \in \Gamma^+(u)$ by theorem 3.6.1. Thus, S_P will return $P^{|W_v(P)|}$ when $\tau-$check uses S_P while considering vertex $v \in V(\tau)$.

Now we need to decide if $[v]$ is violated when S_P returns $P^{|W_v(P)|}$. Consider algorithm 1. In step $q(v)$ algorithm 1 chooses $[v]$ to resample (By definition of $q(v)$). Prior to step $q(v)$ P was reevaluated for all $w : P \in vbl(w)$ and $q(w) < q(v)(\equiv d(w) > d(v))$ and no other vertex. This implies, prior to $q(v)$ P was reevaluated *exactly* once $\forall w \in W_v(P)$ and once more at the beginning of algorithm 1. Hence, P contained the value $P^{|W_v(P)|}$. Since there was a need to resample $[v]$ in step $q(v)$, $[v]$ is violated when P is assigned $P^{|W_v(P)}$ $\forall P \in vbl([v])$ (the same argument can be applied to all of the random variables $[v]$ depends upon). Therefore, $[v]$ is violated in both algorithm 1 and 3 and $\Pr(E_2|E_1) = 1$ and we get the second result in theorem 3.6.1.

3.7 Random Generation of Witness trees

Let $\mathcal{T}_A := \{\tau | \tau$ is a witness tree rooted at $A\}$. Let us consider two witness trees $\tau_1, \tau_2 \in \mathcal{T}_A$ that occur in an execution log C. Our claim is that $\tau_1 \neq \tau_2$. This is because according to the construction algorithm 2, τ_1 and τ_2 must correspond to two distinct steps in the logs say, t_1 and t_2 respectively. If $t_1 < t_2$, number of vertices labeled A in τ_2 must be greater than those labeled A in τ_1. Thus, no two witness trees rooted at A can be same.

Let $N_A :=$ the random variable that counts the number of times A appears in the execution log C. This basically denotes how many times we had to resample event A in algorithm 1. From the above observation, N_A is also equal to the number of witness trees rooted at A that occur in C. To prove that the algorithm terminates, it suffices to prove an upper bound on the expected value of N_A.

From theorem 3.6.1 and linearity of expectation, we already have an inequality for the expected value of N_A, viz. $E(N_A) = \sum_{\tau \in \mathcal{T}_A} \Pr(\tau \text{ appears in the log } C) \leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} \Pr([v])$. But we have now way whatsoever to find this sum. Our approach from now onwards will be to determine the term under the $\prod$ sign. Note that we already have a bound for this term due to the assumption in theorem 3.1.1. We will now show that this term also equals *probability* of some event and then using these two equations determine the sum.

To this end, we define a random process (called Galton-Watson Process) to generate a witness tree rooted at a particular label A. Initially, we take our tree to be just the singleton vertex labeled A. In each subsequent iteration, we consider each $v \in V(\tau)$ and independently add to v a child labeled $B \in \Gamma^+(v)$ with probability $x(B)$ and exclude B with probability 1- $x(B)$. The process continues till there is an iteration in which no vertex is added to the tree.

Theorem 3.7.1. *If $x'(B) := x(B) \prod_{C \in \Gamma(B)} (1 - x(C))$, then the probability p_τ of the Galton-Watson process generating exactly the tree τ is given by*

$$p_\tau = \frac{1 - x(A)}{x(A)} \prod_{v \in V(\tau)} x'([v]) \tag{3.3}$$

Proof. For $v \in V(\tau)$ let $\mathcal{W}_v := \{A \in \Gamma^+([v]) | v$ does not have a child labeled $A\}$.

We can now represent p_τ as:

$$p_\tau = \prod_{v \in V(\tau) \setminus \{A\}} \left(x([v]) \prod_{u \in \mathcal{W}_v} (1 - x[u]) \right)$$

This is because every vertex other than the root A is generated as somebodies child, and for each vertex v that is generated, none of the events in $\mathcal{W}_v$ became the child of v. To get rid of $\mathcal{W}_v$, we can replace it by $\Gamma^+([v])$. In doing so, we multiplied the probability by $1-x(B)$ even for those events that were generated. To correct this, we divide by this term for every vertex that is generated. To get:

$$\begin{aligned} p_\tau &= \prod_{v \in V(\tau) \setminus \{A\}} \left(\frac{x([v])}{1 - x([v])} \prod_{u \in \Gamma^+([v])} (1 - x[u]) \right) \\ &= \frac{1 - x(A)}{x(A)} \prod_{v \in V(\tau)} \left(\frac{x([v])}{1 - x([v])} \prod_{u \in \Gamma^+([v])} (1 - x[u]) \right) \end{aligned}$$

Replacing $\Gamma^+([v])$ by $\Gamma([v])$, we get

$$\begin{aligned} p_\tau &= \frac{1 - x(A)}{x(A)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in \Gamma([v])} (1 - x[u]) \right) \\ &= \frac{1 - x(A)}{x(A)} \prod_{v \in V(\tau)} x'([v]) \end{aligned}$$

Thus we get equation 3.3.

Now, we have all the theorems and inequalities needed to prove an upper bound on the expected number of times $A \in \mathbb{A}$ is resampled ($=E[N_A]$). We get:

$E[N_A]$	$=$	$\sum_{\tau \in \mathcal{T}_A} \Pr(\tau \text{ appears in the log } C)$	From linearity of expectation
	$\leq$	$\sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} \Pr([v])$	From theorem 3.6.1
	$\leq$	$\sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} x'([v])$	From assumption in theorem 3.1.1
	$=$	$\sum_{\tau \in \mathcal{T}_A} \frac{x(A)}{1-x(A)} p_\tau$	From equation 3.3 in theorem 3.7.1
	$=$	$\frac{x(A)}{1-x(A)} \sum_{\tau \in \mathcal{T}_A} p_\tau$	$\frac{x(A)}{1-x(A)}$ can be taken out of the sum
	$\leq$	$\frac{x(A)}{1-x(A)}$	Since Galton-Watson process generates exactly one tree at a time. But the process may never terminate Which is the reason for the inequality

Thus every event is resampled at most an expected $\frac{x(A)}{1-x(A)}$ number of times in algorithm 1. This concludes our constructive proof of the Lovász local lemma.

Chapter 4

Lovasz Local Lemma: Applications

We proved the general Lovász local lemma in both constructive and non-constructive fashion. Although the constructive proof assumed that all the events can be determined by an underlying set of mutually independent random variables, this assumption is justified since it is indeed the case in almost all of the practical applications of the lemma. Here we consider two such problems: Packet routing and Job shop scheduling. The analysis of these problems appeared in [Sri99].

4.1 Packet Routing

In computer networks, it is often the case that the optimal path for a packet from a source to destination is known beforehand and we need to route multiple packets such that there is no edge (link) that has more than one packets passing through it at any instant of time. This corresponds to the problem of packet routing.

4.1.1 Problem Statement

Given an undirected graph $G = (V, E)$, a set of n packets numbered 1 through n and pathsP_i corresponding to i^{th} packet, we need to schedule the packets in such a way that for any edge $e \in E$, at any instant of time there is *at most* one packet that traverses that edge in any direction. There may be multiple schedules that satisfy this constraint, of all these schedules we need to report the schedule of minimum *makespan*, that is total time required for every packet to reach its destination. Here we will assume that it takes any packet a unit time to traverse any edge.

For every edge $e \in E$, we define $C_e := \{i | e \in P_i\}$ and for each packet i we define $D_i := \{e | e \in P_i\}$. We call congestion $c := \max_{e \in E} C_e$ and dilation $d := \max_{i \in [n]} D_i$. Clearly, we cannot have a schedule that takes lesser time than $\max\{c, d\}$. This is because every packet must traverse every edge in its corresponding path and even if all paths are disjoint the longest path packet will take time $= d$ to reach its destination. Similarly, since at any time, only one packet can traverse any

particular edge at any instance, we need at least $|C_e|$ units of time for an edge e to be free (that is no packet will traverse edge e after this time). Hence we get a lower bound on the schedule that can exist. Here we will prove that if the paths are *edge simple* then there exists a schedule of makespan $O(c+d)$. This proof appeared in [LMR94].

Note that we can also have an upper bound on the optimum schedule. we mean that there is also a brute force scheduling approach based on the greedy strategy: If at any instance, no packet is traversing on an edge (u, v), and a packet needs to go from u to v then send the packet from u to v. Clearly, this gives a schedule of makespan $O(cd)$.

4.1.2 Randomized Schedule

We will first produce a random schedule and then derandomize it to prove that there exists a schedule that takes $O(c+d)$ time. Let us pick a suitable number a, we will fix a later. For every packet introduce a random delay $d_i \in \{0, 1, 2, \ldots ac\}$. Every packet is now scheduled to leave its source after a delay d_i and then it proceeds to follow its prescribed path to reach the target without stopping anywhere. Of course this schedule may be invalid, that is, there might be some edge having more than one packet traversing it at some point of time. We need to change the schedule to suit these edges.

For this we need to split the current schedule in different time frames. Note that this (possibly invalid) schedule takes time at most $ac + d$ units. We will divide this schedule in contiguous time frames of length $b \log c$ each starting at time 1, again for a suitable b. Our motive is to prove, using the LLL, that every edge has a congestion of at most $b \log c$ in every time frame with positive probability (i.e. such assignment of random delays exists). If this is true, then we have essentially broken the main problem down to $\frac{ac+d}{b \log c}$ subproblems. We can then proceed to solve each of these subproblems, one corresponding to each frame, and then string their solutions together to get the schedule for the original problem. Clearly, the dilation in each frame-subproblem becomes $b \log c$ and so we have (for cleaner notation, we assume that $c = d$.) :

- $T(c) = \frac{ac}{b \log c} T(b \log c)$ where $T(c)$ denotes the makespan of the optimum schedule having congestion = dilation = c
- $T(k) = k^2$ where k is a constant. This is from the greedy brute force schedule that always exists.

Solving the above recurrence, we get $T(c) = \frac{c}{b} \cdot a^{O(\log^*(c))}$. It is a known fact that $\log^* x \leq 6$ for all practical purposes and so we get a schedule of desired length.

Now all we need to do is to prove using Lovász local lemma that there exists such an assignment of random delays to the packets so that in all of the frames, the congestion on any edge is $O(b \log c)$.

4.1.3 Applying LLL

To apply the Lovász local lemma, we need to define some undesirable events in which dependency of every event on other events is bounded. The natural choice of undesirable events here is to have a set of events E_e for a frame e

denoting that there is some frame having congestion more than $b \log c$ *due to* edge e, that is e is the edge used by more than $b \log c$ packets in some frame. We need to analyze the probability $\Pr(\cap_e \overline{E_e})$ and prove that it is positive.

For any edge e in a frame F, let $E'(e, F)$ be the event that the edge e has a congestion $\geq b \log c$ in frame F. It can be seen that the random variable $C(e, F)$ counting the number of packets passing through an edge e in frame F has an expected value $E[C(e, F)] \leq (b \log c) \times c/ac = (b \log c)/a$ (Since a frame has length $b \log c$). If a packet traverses an edge e in a particular frame F, it can never traverse the same edge twice. This is because of our assumption that paths are edge simple. Due to this observation, we can express $C(e, F)$ as a sum of indicator random variables each corresponding to a different packet indicating whether or not that packet traverses e in frame F. Hence we can apply Chernoff bound and union bound to get:

$$\begin{aligned}
\Pr[E_e] \leq \sum_F \Pr(E'(e, F)) &= \sum_F \Pr\left(C(e, F) \geq a \times \frac{b \log c}{a}\right) \\
&\leq \frac{ac + d}{b \log c} \times \frac{e^{\frac{(a-1)b \log c}{a}}}{a^{b \log c}} \\
&\leq \left(\frac{e}{a}\right)^{b \log c} = c^{-b \log(a/e)}
\end{aligned}$$

By choosing a and b as we please, we can make the bound arbitrarily small inverse polynomial in c. Now we analyze the dependency among various events E_e. Note that to apply the Lovász local lemma, all we need to do now is to bound the number of events E_e for any edge e is dependent upon. Number of packets using an edge e in any frame will depend upon number of packets using another edge f only if *both* of these edges are in the path of *some* packet. Thus, E_e is dependent upon a maximum of cd other events.

Finally to successfully apply the Lovász local lemma, we need to find a mapping x from our set of events to $(0, 1)$ that satisfy the conditions required for theorem 3.1.1. Suppose we take $\forall e \; x(E_e) = \frac{1}{1+cd}$ (this is the same reduction to convert an asymmetric form of lll to the symmetric form as per [AS00]). We need to choose a and b such that:

$$\begin{aligned}
\Pr[E_e] \leq \frac{(cd)^{cd}}{(1 + cd)^{1+cd}} &= \frac{1}{1 + cd} \times \left(1 - \frac{1}{1 + cd}\right)^{cd} \\
&\leq \frac{1}{1 + cd} \times \frac{1}{e}
\end{aligned}$$

So, we need to pick a and b such that

$$\begin{aligned}
c^{-b \log(\frac{a}{e})} &\leq \frac{1}{e(1 + cd)} \\
\log c \times b \log \frac{a}{e} &\geq \log(e(1 + cd))
\end{aligned}$$

Thus it is possible to choose $a = e^2(1 + cd)$ and $b \geq \frac{1}{\log c}$ so that it is possible to apply the Lovász local lemma. Note here that our choice of a and b also affect the length of the schedule we get. The expression for which came out to be $\frac{c}{b} \cdot a^{O(\log^*(c))}$. Since after fixing a, we can choose b as large as we please, we can take $b = \max\left(a^{O(\log^*(c))}, \frac{1}{\log c}\right)$ so that we still get an $O(c + d)$ length schedule.

4.1.4 Constructive vs non-constructive proof

We have only proved the existence of the desired schedule. Due to [MT10] we are also now able to provide a randomized constructive algorithm that will output a correct schedule. According to the assumptions used in the proof, we need to justify that the undesirable events considered here can be determined by an underlying set of mutually independent random variables.

In this problem, this becomes quite easy due to our initial random assignment of delays that was already independent for different events. Thus we can take different random variables for different packets. Each of these random variables can take an integer in $\{1, 2, \ldots, ac\}$. After assigning the random delays, the remaining process is determined and we see that all of the undesirable variables can be evaluated (whether they happened or not can be deduced) from the values set to these random variables.

4.2 Job Shop Scheduling

While scheduling jobs on computers, it might so happen that a particular job requires that it be scheduled in a specific order on different machines. Say first on a Supercomputer, then on a computer running x86 architecture, then on a computer that has a very old version of some operating system and so on. If there are multiple such jobs that need to be executed in such a way, we need to schedule them so as to minimize the total time required (or some other parameter depending upon the resources available. Here we will focus on minimizing the total time required to finish all of the jobs). This problem is analyzed in the survey [Sri99].

4.2.1 Problem Statement

Given a set of n jobs $J_1, J_2, \ldots, J_n$ on m machines numbered 1 through m, and for each job, a sequence of operations where each operation $O_{J_i j}$ is a tuple (i, t_{ij}) which denotes that this operation must be carried out on machine i for an uninterrupted *integral* amount of time t_{ij}. We need to schedule all of the jobs in such a way that:

- Every operation of every job is finished
- No machine runs more than one operation at any point of time
- Every job must be executed in the prescribed sequence of its operations on given machines

Subject to the above conditions, we need to minimize the total time required to finish all of the jobs. Here we will only consider the case where the jobs given are *acyclic*. That is, one job never has to be processed on the same machine more than once. Also, we will assume that all of the operations take same amount of time.

4.2.2 Similarity with Packet Routing

On closer inspection, this problem is quite similar to packet routing. The jobs correspond to the packets we had to take from source to destination. The par-

ticular sequence of operations corresponds to the prescribed paths in packet routing problem. The intermediate state of the jobs wherein the job has completed some of its operations and needs to wait for the next machine to be idle to go further corresponds to the nodes in the network. The constraints are also the same after this mapping. Also note that here we are considering *acyclic* instance of the problem and in case of packet routing as well, we had considered only the case where paths were edge-simple.

We have also assumed that the length of times of all operations are the same, thus we can scale down this instance to another instance wherein each operation takes unit time and convert this problem to an instance of the general packet routing problem to get a schedule of length $O(c + d)$ where c, d are defined in the packet routing problem. After restoring the solution to its original length, we get a schedule of length $\alpha O(c + d)$ where α is the scaling factor we used. This result was shown in [LMR94].

4.2.3 Further Advances

The usage of the lll for this problem ends here. By using some different methods, we can get a good approximate solution for the case where the operations are not necessarily of the same length. The following theorem has been proved in [SSW94] and has been discussed in [Sri99].

Theorem 4.2.1. *For general job-shop scheduling, there exists an algorithm that outputs a schedule taking total time* $O((P_{max}+\prod_{max})\cdot\frac{\log m\mu}{\log\log m\mu}\cdot\log(\min(m\mu, p_{max})))$

Here the terms used are:

- P_{max} := maximum total time of processing needed for a job. Analogous to dilation in packet routing
- $\prod_{max}$:= maximum amount of time a machine needs to complete all its assigned operations. Analogous to congestion in packet routing.
- p_{max} := maximum processing time for any operation
- μ := maximum number of operations per job

It should be noted that the theorem not only provides an existence of the proposed schedule, but also gives an efficient algorithm to find such a schedule. Even in the case of packet routing, such an algorithm was possible due to the constructive proof of Lovász local lemma proved in [MT10].

Chapter 5

Conclusion

We first studied the basics of linear programming. Then we saw various approximation algorithms most of which were solved using linear programming. We even saw one problem that gave the exact optimum solution after randomized rounding as opposed to two others wherein we had a proven guarantee on the ratio of the actual solution and the output given by the approximation algorithm.

In the last two chapters, we focused mainly on a tool called Lovász local lemma. We saw in detail how a problem of packet routing can be solved using the local lemma. First we only proved that a very close to optimum schedule of packets exists, but later, due to the constructive proof of the lemma (that appeared in [MT10]), we were also able to give a randomized algorithm that could compute such a schedule efficiently.

Bibliography

[AS00] Naga Alon and Joel H. Spencer. *The probabilistic method*. Wiley-Interscience, New York, 2000.

[LE75] László Lovász and Paul Erdös. Problems and results on 3-chromatic hypergraphs and some related questions. Number v. 2 in Colloquia mathematica Societatis János Bolyai, pages 609–627. North Holland, April 1975.

[LMR94] Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[MR97] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. In Allen B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 141–161. CRC Press, 1997.

[MT10] Robin A. Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *J. ACM*, 57(2), 2010.

[Mur83] Katta Murty. *Linear programming*. Wiley, New York, 1983.

[Sri99] A Srinivasan. Approximation algorithms via randomized rounding: a survey. pages 9–71, 1999.

[SSW94] David B. Shmoys, Clifford Stein, and Joel Wein. Improved approximation algorithms for shop scheduling problems. *SIAM J. Comput.*, 23(3):617–632, 1994.