Chapter 1

CONCENTRATION OF MEASURE FOR RANDOMIZED ALGORITHMS: TECHNIQUES AND ANALYSIS

Devdatt Dubhashi *

Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Hauz Khas, New Delhi 110016
India
dubhashi@cs.chalmers.se

Sandeep Sen

Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Hauz Khas, New Delhi 110016
India
dubhashi@cse.iitd.ernet.in

1. INTRODUCTION

Randomized algorithms often turn out to be simpler and faster than their deterministic counterparts. For certain problems, like primality testing, there is no matching polynomial-time deterministic algorithm, so the randomized algorithms are the only practical alternative. The above mentioned advantages are sometimes viewed with skepticism because of the inherent uncertainty in the behavior of the randomized algorithm. The uncertainty may be with regards to the correctness (Monte Carlo), or the variation in running time (Las Vegas). In this chapter we will focus on Las Vegas randomized algorithms that always produce the correct answer, but manifest some variation in the running times. The

*Present address: Department of Computing and Mathematical Sciences, Chalmers University of Technology and Göteborg University, Göteborg, Sweden.

most common measure is the expected running time of the algorithm, where the expectation is over the choice of random bits in the algorithm and not the the input distribution. The natural concern is what is an *acceptable* probability of deviating from the expected running time by a certain amount - say a small constant factor. Roughly speaking, the expected running time implies that the probability of exceeding twice the expected time is 1/2. If the hardware itself is subjected to a similar scrutiny, there is empirical evidence that it has a failure probability as high as $\frac{1}{2^k}$ where k is a constant. Therefore it is not unreasonable to aim for a success probability for randomized algorithms that is about an order of magnitude smaller than this.

Over the past decade, the notion of *high probability* bound has gained wide acceptance. An algorithm is said to have a running time $f(n)$ with high probability (whp), if the following holds

$$\Pr(T(n) > c\alpha f(n)) \leq \frac{1}{n^{\alpha}}$$

for some constant c and any α, where $T(n)$ is the running time for an input of size n. This is also referred to as *inverse polynomial* probability of success. It must be noted that there is an implicit trade-off between the success probability and the running time. An inverse-*exponential* probability is even better, namely,

$$\Pr(T(n) > cf(n)) \leq \frac{1}{2^n}.$$

However, randomized algorithms that succeed with high probability are considered as reliable as any deterministic algorithm.

1.1 CONCENTRATION OF MEASURE

The simplest and most common method to achieve a higher success probability is to make several independent runs of the algorithm. In this chapter, we investigate and analyze techniques for increasing the success probability of randomized algorithms, without sacrificing efficiency. Sometimes this is possible by analyzing known algorithms more carefully, whereas in other cases, a simple modification leads us to substantially improved bounds.

The performance of a randomized algorithm measured by its running time, work performed or space used is a random variable that can potentially take a wide range of values. However, one observes that in actual fact, the observable behavior is often concentrated in a very sharply defined narrow range. This is one manifestation of the deep and widespread phenomenon of concentration of measure, and the one we are particularly interested in. In this chapter we will discuss a number of general tools of varying power and range of applicability for proving results about concentration of measure and illustrate them with applications to algorithms.

1.2 ORGANIZATION

Recurrences are the most common yet powerful tools for analyzing algorithms. In the context of randomized algorithms, the associated recurrences are far more complex because of some underlying stochastic process that the recurrence models. Certain *ad hoc* methods have been used repeatedly in the past to solve some commonly occurring (probabilistic) recurrences till Karp tried to develop a coherent theory. We start our chapter with a discussion on related techniques, results and applications.

Probabilistic inequalities form the crux of analysis of randomized algorithms. As noted above, the measures like running-time (or working space) of a randomized algorithm are essentially a random variables with (possibly) complicated distributions and our primary interest is to get useful bounds on the tails of the distributions. We discuss some of the more commonly used probabilistic inequalities, including situations where the behavior of an algorithm is related to several such random variables with non-trivial dependencies among them.

Next we illustrate applications of a well-known stochastic process called the branching process and derive concentration bounds in a relatively general framework.

Randomized algorithms have been very influential in the area of computational geometry. In the last two sections, we describe techniques for obtaining high-probability bounds for randomized algorithms in computational geometry with applications to sequential and parallel algorithms.

2. PROBABILISTIC RECURRENCES

Karp[21] developed an attractive framework for the analysis of randomized algorithms. Suppose we have a randomized algorithm that on input x, performs "work" $a(x)$ and then produces a subproblem of size $H(x)$ which is then solved by recursion. One can analyze the performance of the algorithm by writing down a "recurrence":

$$T(x) = a(x) + T(H(x)). \tag{1.1}$$

Superficially this looks just the same as the usual analysis of algorithms via recurrence relations. However, the crucial difference is that in contrast with deterministic algorithms, the size of the subproblem produced here, $H(x)$ is a random variable, and so (1.1) is a *probabilistic recurrence* equation.

What does one mean by the solution of such a probabilistic recurrence? The solution $T(x)$ is itself a random variable and we would like as much information about its distribution as possible. While a complete description of the exact distribution is usually neither possible nor really necessary, the "correct" useful analogue to the deterministic solution is a concentration of measure result for $T(x)$. Of course, to do this, one needs some information on the distribution of the subproblem $H(x)$ generated by the algorithm. Karp gives a very easy–to–apply framework that requires only the bare minimum of information on the distribution of $H(x)$, namely (a bound on) the expectation, and yields a concentration result for $T(x)$. Suppose that in (1.1), we have $\mathtt{E}[H(x)] \leq m(x)$ for some function $0 \leq m(x) \leq x$. Consider the "deterministic" version of (1.1) obtained by replacing the random variable $H(x)$ by the deterministic bound $m(x)$:

$$u(x) = a(x) + u(m(x)). \tag{1.2}$$

The solution to this equation is $u(x) = \sum_{i \geq 0} a(m^i(x))$, where $m^0(x) := 0$ and $m^{i+1}(x) = m(m^i(x))$. Karp gives a concentration result around this value $u(x)$:

Theorem 1 (Karp's First Theorem) *Suppose that in (1.1), we have* $\mathtt{E}[H(x)] \leq m(x)$ *for some function* $0 \leq m(x) \leq x$ *and such that* $a(x), m(x), \frac{m(x)}{x}$ *are all non–decreasing. Then*

$$\mathtt{Pr}[T(x) > u(x) + ta(x)] \leq \left(\frac{m(x)}{x}\right)^t.$$

Remark 1 We have stated the result in the simplest memorable form that captures the essence and i essentially correct. However, technically the statement of the theorem above is actually not quite accurate and we have omitted some continuity conditions on the functions involved. These conditions usually hold in all cases where we'd like to apply the theorem. Moreover, as shown in [10], some of these conditions can be discarded at the cost of only slightly weakening the bound. For instance, we can discard the condition that $\frac{m(x)}{x}$ is non–decreasing; in that case, the bound on the right hand side can be essentially replaced by $\left(\max_{0 \leq y \leq x} \frac{m(y)}{y}\right)^t$

Also, in the formulation above, we assumed that the distribution of $H(x)$, the size of the derived subproblem depends only on the input size x. Karp[21] gives a more general formulation where the subproblem is allowed to depend on the actual input instance. Suppose we have a "size" function s on inputs, and on processing an input z, we expend work $a(s(z))$ and get a subproblem $H(z)$ such that $\mathtt{E}[s(H(z))] \le m(s(z))$. The probabilistic recurrence is now

$$T(z) = a(s(z)) + T(H(z)).$$

By considering $T'(x) := \max_{s(z)=x} T(z)$, one can bound this by a recurrence of the earlier form and apply the Theorem to give exactly the same solution. Thus we can apply the Theorem *per se* even in this more general situation.

We illustrate the ease of applicability of this cook–book style recipe by some examples (taken from Karp's paper).

Example 2..1 (Selection) Hoare's classic algorithm for finding the kth smallest element in a n–element set S, proceeds as follows: pick a random element $r \in S$ and by comparing each element in $S \setminus r$ with r, partition $S \setminus r$ into two subsets $L := \{y \in S \mid y < r\}$ and $U := \{y \in S \mid y > r\}$. Then,

- If $|L| \ge k$, recursively find the kth smallest element in L.
- If $|L| = k - 1$, then return r.
- If $|L| < k - 1$, then recursively find the $k - 1 - |L|$th smallest element in U.

The partitioning step requires $n - 1$ comparisons. It can be shown that the expected size of the subproblem, namely the size of L or U is at most $3n/4$, for all k. Thus Karp's Theorem can be applied with $m(x) = 3x/4$. We compute $u(x) \le 4x$. Thus, if $T(n, k)$ denotes the number of comparisons performed by the algorithm, we have the following concentration result: for all $t \ge 0$,

$$\mathtt{Pr}[T(n,k) > 4n + t(n-1)] \le \left(\frac{3}{4}\right)^t.$$

This bound is nearly tight as showed by the following simple argument. Define a *bad* splitter to be one where $\frac{n}{|U|} \ge \log\log n$ or $\frac{n}{|L|} \ge \log\log n$. The probability of this is greater than $\frac{2}{\log\log n}$. The probability of picking $\log\log n$ consecutive bad splitters is $\Omega(\frac{1}{(\log n)^{\log\log\log n}})$. The work done for $\log\log n$ consecutive bad splitters is

$$n + n\left(1 - \frac{1}{\log\log n}\right) + n\left(1 - \frac{1}{\log\log n}\right)^2 + \ldots n\left(1 - \frac{1}{\log\log n}\right)^{\log\log n}$$

which is $\Omega(n \log\log n)$. Compare this with the previous bound using $t = \log\log n$.

Example 2..2 (Luby's Maximal Independent Set Algorithm) Luby [23] gives a randomized parallel algorithm for constructing a maximal independent set in a graph. The algorithm works in stages: at each stage, the current independent set is augmented and some edges are deleted form the graph. The algorithm terminates when we arrive at the empty graph. The work performed at each iteration is equal to the number of edges in the current graph. Luby showed that at each stage, the expected number of edges deleted is at least one–eighth of the number of edges in the complete graph. If $T(G)$ is the number of stages the algorithm runs and $T'(G)$ is the total amount of work done, then we get the concentration results:

$$\mathtt{Pr}[T(G) > \log_{8/7} n + t] \leq \left(\frac{7}{8}\right)^t,$$

$$\mathtt{Pr}[T'(G) > (8+t)n] \leq \left(\frac{7}{8}\right)^t.$$

Example 2..3 (Tree Contraction) Miller and Reif [27] give a randomized *tree contraction* algorithm that starts with a n node tree representing an arithmetic expression and repeatedly applies a randomized contraction operation that provides a new tree representing a modified arithmetic expression.The process eventually reaches a one node tree and terminates. The work performed in the contraction step can be taken to be proportional to the number of nodes in the tree. Miller and Reif show that when applied to a tree with n nodes, the contraction step results in a tree of size at most $4n/5$. However the distribution of the size may depend on the original tree, not just the original size. Define the size function here to be the number of nodes in the tree in order to apply the more general framework. Let $T(z), T'(z)$ denote the number of iterations and the total work respectively when the contraction algorithm is applied to tree z. Then, Karp's Theorem gives the measure concentration results:

$$\mathtt{Pr}[T(z) > \log_{5/4} n + t] \leq (4/5)^t,$$

and

$$\mathtt{Pr}[T'(z) > (5+t)n] \leq (4/5)^t.$$

Under the weak assumptions on the distribution of the input, Karp's First Theorem is essentially tight. However, if one has additional information on the distribution of the subproblem, say some higher moments, then one can get sharper results which will be explored below in § ref-sec:martingales.

Karp also gives an extension of the framework for the very useful case when the algorithm might generate more than one subproblem. Suppose we have an algorithm that on input x performs work $a(x)$ and then generates a fixed number $k \geq 1$ sub–problems $H_1(x), \ldots, H_k(x)$ each a random variable. This corresponds to the probabilistic recurrence:

$$T(x) = a(x) + T(H_1(x)) + \cdots + T(H_k(x)). \tag{1.3}$$

To obtain a concentration result in this case, Karp uses a different method which requires a certain condition:

Theorem 2 (Karp's Second Theorem) *Suppose that in (1.3), we have that for all possible values $(x_1, \ldots, x_k)$ of the tuple $(H_1(x), \ldots, H_k(x))$, we have*

$$\mathtt{E}[T(x)] \geq \sum_i \mathtt{E}[T(x_i)]. \tag{1.4}$$

Then, we have the concentration result: for all x and all $t > 0$,

$$\mathtt{Pr}[T(x) > (t+1)\mathtt{E}[T(x)]] < e^{-t}.$$

The condition (1.4) says that the expected work in processing *any* sub–problems that can result from the original one can never exceed the expected cost of the processing the original instance. This is a very strong assumption and unfortunately, in many cases of interest, for example in computational geometry, it does not hold. Consequently the theorem is somewhat severely limited in its applicability. A rare case in which the condition is satisfied is for

Example 2..4 (Quicksort) Hoare's Quicksort algorithm is a true classic in Computer Science: to sort a set S of n items, we proceed as in the selection algorithm from above: select a random element $r \in S$ and by comparing it to every other element, partition S as into the sets L of elements less than x and U, the set of elements at least as big as r. Then, recursively, sort L and U. Let $Q(n)$ denote the number of comparisons performed by Quicksort on a set of n elements. Then $Q(n)$ satisfies the probabilistic recurrence:

$$T(n) = n - 1 + Q(H_1(n)) + Q(H_2(n)),$$

where $H_1(n) = |L|$ and $H_2(n) = |U|$. For Quicksort we have "closed-form" solutions for $q_n := \mathtt{E}[Q(n)]$which imply that $q_n \geq q_i + q_{n-i-1} + n - 1$ for any $0 \leq i < n$, which is just the condition needed to apply Karp's Second Theorem. Thus we get the concentration result:

$$\mathtt{Pr}[Q(n) > (t+1)q_n] \leq e^{-t}.$$

Actually one can get a much stronger bound by applying Karp's First Theorem suitably! Charge each comparison made in Quicksort to the non–pivot element, and let $T(\ell)$ denote the number of comparisons charged to a fixed element when Quicksort is applied to a list ℓ. Use the natural size function $s(\ell) := |\ell|$, which gives the number of elements in the list. Then we have the recurrence, $T(\ell) = 1 + T(H(\ell))$, where $s(H(\ell) = |\ell|/2$ since the sublist containing the fixed element (when it's not the pivot) has size uniformly distributed in $[0, |\ell|]$. So applying Karp's First Theorem, we have that for $t \geq 1$,

$$\texttt{Pr}[T(\ell) > (t+1)\log|\ell|] \leq (1/2)^{t\log|\ell|} = |\ell|^{-t}.$$

Thus any fixed element in a list of n elements is charged at most $(t+1)\log n$ comparisons with probability at least $1 - n^{-t}$. The total number of comparisons is therefore at most $(t+1)n\log n$ with probability at least $1 - n^{t-1}$.

This is an inverse polynomial concentration bound. In a later section we shall get a somewhat stronger and provably optimal bound on the concentration.

It would naturally be of great interest to extend the range of Karp's Second Theorem by eliminating the restrictive hypothesis. For instance, it would be of interest to extend the Theorem under the kind of assumptions in Karp's First Theorem.

3. CHERNOFF–HOEFFDING BOUNDS

Perhaps the most basic and widely used tools for the analysis of randomized algorithms are the *Chernoff–Hoeffding* (CH) bounds on sums of bounded independent random variables. Unlike elementary probabilistic inequalities like Markov and Chebychev, the CH technique make use of the entire moment generating function $M(t) := \texttt{E}[e^{tX}]$ of a random variable X, and provides much stronger bounds.

$$\begin{aligned} \texttt{Pr}[X > s] &= \texttt{Pr}[e^{tX} > e^{ts}] \\ &\leq \frac{\texttt{E}[e^{tX}]}{e^{ts}} \quad \text{using Markov's inequality} \end{aligned} \tag{1.5}$$

By minimizing the right hand side w.r.t. t, we obtain the Chernoff bound for $\texttt{Pr}[X \geq s]$.

Theorem 3 (Chernoff–Hoeffding Bounds) *Let $X_1, \ldots, X_n$ be independent random variables taking values in the unit interval $[0,1]$ and let $X := X_1 + \cdots + X_n$. Let $p_i = \texttt{E}[X_i], i \in [n]$ and let $p := (p_1 + \cdots +$*

$p_n)/n, q := 1 - p$. *Then*

$$\mathtt{Pr}[X > (p+t)n], \mathtt{Pr}[X < (p-t)n] \leq \exp\left(-nH\left((p+t, q-t) \mid (p,q)\right)\right),$$

where H is the **relative entropy**,

$$H\left((p_1, \ldots, p_r), (q_1, \ldots, q_r)\right) := \sum_i p_i \log \frac{p_i}{q_i}.$$

This is not the usual way of stating the CH bounds, nor is it the most useful; we give more useful forms for applications later below. The reasons for stating the bound above are: First, it is the strongest form of the bound from which all the others below can be derived.

Second, it gives the most insight into the bound form which it can be easily remembered. Think of the X_i as identical binary (0/1) variables, like n independent tosses of the same coin. Then X counts the number of heads observed in the n independent coin tosses. From the *a priori* probability, we expect to see np heads. What is the chance that the observed or *a posteriori* number of heads is $(p+t)n$? That is, what is the chance that the *a posteriori* distribution appears as one deriving from a $(p+t, q-t)$ coin when the actual *a priori* coin has a (p,q) distribution? One expects that the probability of this event drops exponentially in the relative entropy "distance" between the *a priori* and *a posteriori* distributions.

Finally, this viewpoint opens the way for many fruitful and far–reaching extensions to the so–called "large deviation" theory. We just give a flavor of this: suppose we have an experiment with r different outcomes with corresponding *a priori* probabilities $p_1, \ldots, p_r$. In coin tossing, $r = 2$ with p_1 and p_2 being the probabilities of heads and tails respectively. If the experiment is tossing a die, $r = 6$ and the probabilities $p_1, \ldots, p_6$ would reflect how the die is loaded (for a fair die, each $p_i = 1/6$). If we now perform n independent and identical trials, what is the probability that the the observed distribution will resemble $(q_1, \ldots, q_r)$? The answer is roughly, that this probability once again falls exponentially in the relative entropy distance. Some forms of the CH bounds that are most useful for applications are summarized in the next theorem; they can be deduced from Theorem 3.

Theorem 4 *Let $X := X_1 + \cdots + X_n$ where $X_i, i \in [n]$ are independent random variables taking values in the unit interval* $[0,1]$. *Then:*

- $$\mathtt{Pr}[X > \mathtt{E}[X] + t], \mathtt{Pr}[X < \mathtt{E}[X] - t] < e^{-2t^2/n}.$$

- *For* $\epsilon > 0$,

$$\mathtt{Pr}[X > (1+\epsilon)\mathtt{E}[X]] < \exp\left(\frac{-\epsilon^2}{3}\mathtt{E}[X]\right), \qquad \mathtt{Pr}[X < (1-\epsilon)\mathtt{E}[X]] < \exp\left(\frac{-\epsilon^2}{2}\mathtt{E}[X]\right).$$

4. APPLICATIONS OF CH BOUNDS

Example 4..1 (Skip List)

Skip-list is a data structure introduced by Pugh [33] as an alternative to balanced binary search trees for handling dictionary operations on ordered lists. The underlying idea is to substitute complex book-keeping information used for maintaining balance conditions for binary trees by random sampling techniques. It has been shown by Pugh [33] that, given access to random bits, the *expected* search time in a skip-list of n elements is $O(\log n)$ [1] which compares very favourably with balanced binary trees. Moreover, the procedures for insertion and deletion are very simple which makes this data-structure a very attractive alternative to the balanced binary trees.

Since the search time is a stochastic variable (because of the use of randomization), it is of considerable interest to determine the bounds on the tails of its distribution. Often, it is crucial to know the behavior for any individual access rather than a chain of operations since it is more closely related to the real-time response.

Review of Skip-lists. We briefly review the basic data-structure proposed by Pugh. This data-structure is maintained as a hierarchy of sorted linked-lists. The bottom-most level is the entire set of keys S. We denote the linked list at level i from the bottom as L_i and let $|L_i| = N_i$. By definition $L_0 = S$ and $|L_0| = n$. For all $0 \leq i$, $L_i \subset L_{i-1}$ and the topmost level, say level k has constant number of elements. Moreover, correspondences are maintained between common elements of lists L_i and L_{i-1}. For a key with value E, for each level i, we denote by T_i a tuple (l_i, r_i) such that $l_i \leq E \leq r_i$ and $l_i, r_i \in L_i$. We call this tuple *straddling pair* (of E) in level i.

The search begins from the topmost level L_k where T_k can be determined in constant time. If $l_k = E$ or $r_k = E$ then the search is successful else we recursively search among the elements $[l_k, r_k] \bigcap L_0$. Here $[l_k, r_k]$ denotes the closed interval bound by l_k and r_k. This is done by searching the elements of L_{k-1} which are bounded by l_k and r_k. Since both $l_k, r_k \in L_{k-1}$, the *descendence* from level k to $k-1$ is easily achieved in $O(1)$ time. In general, at any level i we determine the tuple T_i by walking through a portion of the list L_i. If l_i or r_i equals E then we are done else we repeat this procedure by *descending* to level $i-1$.

In other words, we refine the search progressively until we find an element in S equal to E or we terminate when we have determined (l_0, r_0). This procedure can also be viewed as searching in a tree that has variable degree (not necessarily two as in binary tree).

Of course, to be able to analyze this algorithm, one has to specify how the lists L_i are constructed and how they are dynamically maintained under deletions and additions. Very roughly, the idea is to have elements in i-th level point to approximately 2^i nodes ahead (in S) so that the number of levels is approximately $O(\log n)$. The time spent at each level i depends on $[l_{i+1}, r_{i+1}] \bigcap L_i$ and hence the objective is to keep this small. To achieve these conditions on-line, Pugh [33] uses the following elegant method. The nodes from the bottom-most layer (level 0) are chosen with probability p (for the purpose of our discussion we shall assume $p = 0.5$) to be in the first level. Subsequently at any level i, the nodes of level i are chosen to be in level $i+1$ independently with probability p and at any level we maintain a simple linked list where the elements are in sorted order. If $p = 0.5$, then it is not difficult to verify that for a list of size n, the *expected* number of elements in level i is approximately $n/2^i$ and are spaced about 2^i elements apart. The expected number of levels is clearly $O(\log n)$, (when we have just a trivial length list) and the expected space requirement is $O(n)$.

To insert an element, we first locate its position using the search strategy described previously. Note that a byproduct of the search algorithm are all the T_i's. At level 0, we choose it with probability p to be in level L_1. If it is selected, we insert it in the proper position (which can be trivially done from the knowledge of T_1), update the pointers and repeat this process from the present level. Deletion is very similar and it can be readily verified that deletion and insertion have the same asymptotic run time as the search operation. So we shall focus on this operation.

Analysis. To analyze the run-time of the search procedure, we look at it backwards, i.e., retrace the path from level 0. The search time is clearly the length of the path (number of links) traversed over all the levels. So one can count the number of links one traverses before climbing up a level. In other words the expected search time can be expressed in the following recurrence (from [33])

$$C(k) = (1-p)(1 + C(k)) + p(1 + C(k-1))$$

where C(k) is the expected cost for climbing k levels. From the boundary condition C(0) $= 0$, one readily obtains $C(k) = k/p$. For $k = O(\log n)$, this is $O(\log n)$. The recurrence captures the crux of the method in the following manner. At any node of a given level, we climb up if this node has been chosen to be in the next level or else we add one to the cost of the present level. The probability of this event (climbing up a level) is p which we consider to be a success event. Now the entire search procedure can be viewed in the following alternate manner. We are tossing a coin which turns up heads with probability p - how many times should we toss to come up with $O(\log n)$ heads ? Each head corresponds to the event of climbing up one level in the data structure and the total number of tosses is the cost of the search algorithm. We are done when we have climbed up $O(\log n)$ levels (there is some technicality about the number of levels being $O(\log n)$ but that will be addressed later). The number of heads obtained by tossing a coin N times is given by a Binomial random variable X with parameters N and p. Using Chernoff bounds from Theorem 4, for $N = 15\log n$ and $p = 0.5$, $\texttt{Pr}[X \leq 1.5\log n] \leq 1/n^2$ (using $\epsilon = 9/10$ in equation 1). Using appropriate constants, we can get rapidly decreasing probabilities of the form $\texttt{Pr}[X \leq c\log n] \leq 1/n^\alpha$ for $c, \alpha > 0$ and α increases with c. These constants can be fine tuned although we shall not bother with such an exercise here.

We thus state the following lemma.

Lemma 1 *The probability that access time for a fixed element in a skip-list data structure of length n exceeds $c\log n$ steps is less than $O(1/n^2)$ for an appropriate constant $c > 1$.*

Proof We compute the probability of obtaining fewer than k (the number of levels in the data-structure) heads when we toss a fair coin ($p = 1/2$) $c\log n$ times for some fixed constant $c > 1$. That is, we compute the probability that our search procedure exceeds $c\log n$ steps. Recall that each head is equivalent to climbing up one level and we are done when we have climbed k levels. To bound the number of levels, it is easy to see that the probability that any element of S appears in level i is at most $1/2^i$, i.e. it has turned up i consecutive heads. So the probability that any fixed element appears in level $3\log n$ is at most $1/n^3$. The probability that $k > 3\log n$ is the probability that at least one element of S appears in $L_{3\log n}$. This is clearly at most n times the probability that any fixed element survives and hence probability of k exceeding $3\log n$ is less than $1/n^2$.

Given that $k \leq 3\log n$ we choose a value of c, say c_0 (to be plugged into equation 1 of Chernoff bounds) such that the probability of obtaining fewer than $3\log n$ heads in $c_0\log n$ tosses is less than $1/n^2$. The search algorithm for a fixed key exceeds $c_0\log n$ steps if one of the above events fail; either the number of levels exceeds $3\log n$ or we get fewer than

$3\log n$ heads from $c_0 \log n$ tosses. This is clearly the summation of the failure probabilities of the individual events which is $O(1/n^2)$. $\Box$.

Theorem 5 *The probability that the access time for any arbitrary element in skip-list exceeds $O(\log n)$ is less than $1/n^\alpha$ for any fixed $\alpha > 0$.*

Proof: A list of n elements induces $n+1$ intervals. From the previous lemma, the probability P that the search time for a fixed element exceeding $c \log n$ is less than $1/n^2$. Note that all elements in a fixed interval $[l_0, r_0]$ follow the same path in the data-structure. It follows that for any interval the probability of the access time exceeding $O(\log n)$ is n times P. As mentioned before, the constants can be chosen appropriately to achieve this. $\Box$

It is possible to obtain even tighter bounds on the space requirement for a skip list of n elements. From Pugh [33] it is known that the expected space is $O(n)$. Moreover it is clear that it does not exceed $O(n \log n)$ with probability $1 - 1/n^2$ (no element survives more than $O(\log n)$ levels with this probability from the previous lemma). One can obtain a much stronger bound by viewing the entire skip list structure as a stochastic experiment each node corresponds to a Bernoulli trial that turns up heads (similar to obtaining the obtaining the query bound). Each element is replicated till the the trial turns up tails. Since there are n elements, the number of nodes corresponds to the number of Bernoulli trials required to obtain n tails. This is a negative binomial distribution and one can use Chernoff bounds (Theorem 4) directly to obtain the following result.

Theorem 6 *For any constant $\alpha > 0$, the probability of the space exceeding $2n + \alpha \cdot n$, is less than $\exp^{\Omega(\alpha^2 n)}$.*

Example 4..2 (Randomized Search Trees and Backward Analysis)

Randomized Search Trees. The class of binary (dynamic) search trees is perhaps the first introduction to non-trivial data-structure in computer science. However, the update operations, although asymptotically very fast are not the easiest to remember. The rules for *rotations* and the *double-rotations* of the AVL trees, the splitting/joining in B-trees and the color-changes of red-black trees are often complex, as well as their correctness proofs. The *Randomized Search Trees* (also known as randomized treaps) provide a practical alternative to the Balanced BST. We still rely on rotations, but no explicit balancing rules are used. Instead we rely on the magical properties of random numbers.

The Randomized Search Tree (RST) is a binary tree that has the keys in an in-order ordering. In addition, each element is assigned a priority (Wlog, the priorities are unique) and the nodes of the tree are heap-ordered based on the priorities. Simultaneously, the key values follow in-order numbering. It is not difficult to see that for a given assignment of priorities, there is exactly one tree. If the priorities are assigned randomly in the range $[1, N]$ for N nodes, the expected height of the tree is *small*. This is the main crux of the following analysis of the performance of the RSTs.

Let us first look at the way search time using a technique known as *backward analysis*. For that we (hypothetically) insert the N elements in a decreasing order of their *priorities* and then count the number of elements that Q can *see* during the course of their insertions. This method (of assigning the random numbers on-line) makes arguments easier and the reader must convince himself that it doesn't affect the final results. Q can *see an element* N_i if there are no previously inserted elements in between.

Observation 1 *The tree constructed by inserting the nodes in order of their priorities (highest priority is the root) is the same as the tree constructed on-line.*

Observation 2 *The number of nodes Q sees is exactly the number of comparisons performed for searching Q. In fact, the order in which it sees corresponds to the search path of Q.*

Theorem 7 *The expected length of search path in RST is $O(H_N)$ where H_N is the N-th harmonic number.*

In the spirit of backward analysis, we pretend that the tree-construction is being reversed, i.e. nodes are being deleted starting from the last node. In the forward direction, we would count the expected number of nodes that Q sees. In the reverse direction, it is the number of times Q's visibility changes (convince yourself that these notions are identical). Let X_i be a Bernoulli rv that is 1 if Q sees N_i (in the forward direction) or conversely Q's visibility changes when N_i is deleted in the reverse sequence. Let X be the length of the search path.

$$X = \sum X_i$$

$$E[X] = E[\sum X_i] = \sum E[X_i]$$

We claim that $E[X_i] = \frac{2}{i}$. Note that the expectation of a Bernoulli variable is the probability that it is 1. We are computing this probability over all permutations of N being equally likely. In other words, if we consider a prefix of length i, all subsets of size i are equally likely. Let us find the probability that $X_i = 1$, *conditioned* on a *fixed* subset $N^i \subset N$. Unconditioning is easy if probability that $X_i = 1$ does not depend on N^i itself. So, given a fixed N^i, all N^{i-1} are equally likely, so the probability that $X_i = 1$ is the probability that one of the (maximum two) neighboring elements was removed in the reverse direction. The probability of that is less than $\frac{2}{i}$ which is independent of any specific N^i. So, the unconditional probability is the same as conditional probability - hence $E[X_i] = \frac{2}{i}$. The theorem follows as $\sum_i E[X_i] = 2\sum_i \frac{1}{i} = O(H_N)$.

The X_i's defined in the previous proof are independent but not identical. So, we can apply Chernoff-Hoeffding bounds to to obtain strong tail-estimates for deviation from the expected bound. From Theorem 4 , it follows that

Theorem 8 *The probability that the search time exceeds* $2\log n$ *comparisons in a randomized treap is less than* $O(1/n)$.

A similar technique can be used for counting the number of rotations required for RST during insertion and deletions. Backward analysis is a very elegant technique for analyzing randomized algorithms, in particular in a paradigm called randomized incremental construction.

Example 4..3 (Random Permutation)

A randomized algorithm for generating a random permutation becomes a necessity and we would like to do this efficiently. A natural scheme would be to make a pass through the n elements and try to find a random assignment for each element in the range $[1, n]$. Since the mapping must be 1-1, we have to make sure that no two elements are mapped to the same index, namely there is no *collision*. For this we can use the following algorithm to generate a random permutation Π for elements $\{1, 2 \ldots n\}$.
for each element i do

1. generate a random number in range $[1, n]$, say j. If $\Pi^{-1}(j)$ is less than i repeat the step (there is a collision).
2. Set $\Pi(i)$ to j.

The running time of this algorithm depends on the number of times Step 1 is repeated. A simple analysis shows that for the i-th element, the expected number of repetitions is $\frac{n}{n-i+1}$ for the i-th element yielding a total expected value of $O(n \log n)$. It is known that the time bound is concentrated around its expected value (from the analysis of *coupon collector's problem*). A simple modification makes the expected running time linear. In the first step, generate random numbers in the range

$[1, 2n]$ and in the end, compress the indices back to $[1, n]$ (keeping the relative order unchanged). The expected number of repetitions for i-th element is always less than 2. Since the trials are independent, we can apply Chernoff bounds to show that the running time is concentrated around the mean with inverse exponential probability. This technique is also known as *Random Assignment* and is very useful for parallel algorithms, like parallel integer sorting[34].

5. CH BOUNDS WITH NEGATIVE AND LIMITED DEPENDENCE

In many applications, we are required to give concentration bounds on a sum of variables which fail to be independent. In such cases, we cannot apply the CH bounds directly, and it is of great interest to seek conditions under which they may be salvaged. In this section, we give two general scenarios in which this can be done.

5.1 NEGATIVE DEPENDENCE

One form of dependence in which CH bounds can be intuitively expected to hold is *negative dependence*: when one set of variables is "high", a disjoint set is "low". For, in this kind of systematic dependence, variations of different variables around their means will tend to cancel each other so that their sum is highly concentrated.

We now define two useful formal notions of negative dependence. Random variables $X_1, \ldots, X_n$ satisfy:

(-R) **negative regression** if for every two disjoint subsets I and J of the index set $[n]$ and for every function f which is non–decreasing in each co–ordinate, $\mathtt{E}[f(X_i, i \in I) \mid X_j = x_j]$ is non–increasing in each $x_j, j \in J$.

(-A) **negative association** if for for every two disjoint subsets I and J of the index set $[n]$ and for all functions f and g which are non–decreasing in each co–ordinate,

$$\mathtt{E}[f(X_i, i \in I)g(X_j, j \in J)] \leq \mathtt{E}[f(X_i, i \in I)]\mathtt{E}[g(X_j, j \in J)].$$

The usefulness of these definitions lies in the fact that first they obtain in many natural situations in the analysis of algorithms (some examples follow) and second that

Theorem 9 *The Chernoff–Hoeffding bounds can be applied to sums of variables that satisfy either $(-R)$ or $(-A)$.*

Two of the simplest and most useful examples where these notions of dependence obtain are:

- The *hypergeometric distribution*: the number of red balls in a sample of n balls, drawn *without replacement* from an urn containing N balls, $M \leq N$ of which are red. An intuitive *coupling* argument (that can be esrtablished by more formal arguments) in the context of hypergeometric distribution is of the following kind. Consider a sample of n balls drawn from two populations of N balls each, one of which has m red balls and the other $m' > m$ red balls. It is intuitively clear that the distribution of red balls in the sample drawn from the latter stochastically dominates the distribution drawn from the former.

  Let us sketch how to verify quickly, the seemingly complicated condition $(-R)$. By symmetry, one can assume that the set J represents the first set of trials and the set I the last. Then the conditioning $X_j = x_j, j \in J$ simply represents the sampling experiment with $N - |J|$ balls containing $M - \sum_{j \in J} x_j$ red balls. Since this last number is non–increasing in each $x_j, j \in J$, a simple coupling argument gives the result.

- The *balls and bins experiment*: m (non–identical) balls are thrown independently at random into n (non–identical) bins. Arbitrary probabilities $p_{i,k}$ are assigned to the event that ball k lands in in bin i. Of the various sets of variables of interest in this experiment are the *occupancy numbers* $B_i, i \in [n]$ giving the number of balls in each bin and the *empty bin indicators* $Z_1, \ldots, Z_n$ which are 1 or 0 accordingly as the corresponding bins are empty or not. Both these sets of variables satisfy $(-R)$ as well as $(-A)$. Once again, one can exploit symmetry to verify the $(-R)$ property in the case when all bins and balls are identical. The conditioning $B_j = b_j, j \in J$, simply reduces in this case, to a balls and bins experiment involving fewer balls, namely $m - \sum_{j \in J} b_j$ thrown into the balls indexed by I, and the result follows by an easy coupling. In the general case, i.e. when neither the bins nor the balls are identical, the result is surprisingly non–trivial, see [12].

5.2 LIMITED INDEPENDENCE

Recall that random variables $X_1, \ldots, X_n$ are fully independent if $\mathtt{Pr}[X_i = x_i, i \in [n]] = \prod_i prob[X_i = x_i]$. To reduce the randomness requirements in many randomized algorithms, we are interested in weaker notions of independence. The random variables $X_i, i \in [n]$ are said to be k–

wise independent ($k \leq n$) if for each subset $I \subseteq [n]$ of size at most k, $\texttt{Pr}[X_i = x_i, i \in I] = \prod_{i \in I} prob[X_i = x_i]$.

Suppose we have a set of variables $X_1, \ldots, X_n$ which are k–wise independent, where k is some positive even integer. To derive a concentration result for the sum $X = \sum_i X_i$ around its mean μ, we employ the k–th *moment inequality*:

$$\begin{aligned} \texttt{Pr}[|X - \mu| > t] &= prob[(X-\mu)^k > t^k], \text{since } k \text{ is even} \\ &< \frac{\texttt{E}[(X-\mu)^k]}{t^k}, \text{by Markov's inequality.} \end{aligned} \quad (1.6)$$

To estimate $\texttt{E}[(X-\mu)^k]$, we observe that by expanding and using linearity of expectation, we only need to compute $\texttt{E}[\prod_{i \in S}(X_i - \mu_i)]$ for multi–sets S of size k. By the k–wise independence property, this is the same as $\texttt{E}[\prod_{i \in S}(\hat{X}_i - \mu_i)]$, where $\hat{X}_i, i \in [n]$ are fully independent random variables with the same marginals as $X_i, i \in [n]$. Turning the manipulation on it's head, we now use Chernoff–Hoeffding bounds on $\hat{X} := \sum_i \hat{X}_i$:

$$\begin{aligned} \texttt{E}[(\hat{X}-\mu)^k] &= \int_0^\infty \texttt{Pr}[(\hat{X}-\mu)^k > t] dt \\ &= \int_0^\infty \texttt{Pr}[|\hat{X}-\mu| > t^{1/k}] dt \\ &< \int_0^\infty e^{-2t^{2/k}/n} dt, \quad \text{using CH bounds} \\ &= (n/2)^{k/2} \frac{k}{2} \int_0^\infty e^{-y} y^{k/2-1} dy \\ &= (n/2)^{k/2} \frac{k}{2} \Gamma(k/2-1) \\ &= (n/2)^{k/2} (k/2)! \end{aligned}$$

Now using Stirling's approximation for $n!$ gives the estimate:

$$\texttt{E}[(\hat{X}-\mu)^k] \leq 2e^{1/6k} \sqrt{\pi t} \left(\frac{nk}{e}\right)^{k/2},$$

which in turn, plugged into (1.6) gives the following version of a tail estimate valid under limited i.e. k–wise dependence:

$$\texttt{Pr}[|X-\mu| > t] \leq C_k \left(\frac{nk}{t^2}\right)^{k/2},$$

where $C_k := 2\sqrt{\pi k} e^{1/6k} \leq 1.0004$. This derivation is due to Bellare and Rompel [1].

Using this same basic method with other estimates of C.H. bounds, one can derive other bounds that may be more suitable for the application at hand (for example see [40]).

6. MARTINGALES AND THE METHOD OF BOUNDED DIFFERENCES

Martingales and the so–called Method of Bounded Differences [26] are a very powerful and versatile technique for proving concentration of measure results for arbitrary functions of several variables. In the general set–up, we have a function f and random variables $X_1, \ldots, X_n$ and we would like to establish a concentration result for $f(X_1, \ldots, X_n)$. The power and versatility of the method is highlighted by the fact that

- The random variables $X_1, \ldots, X_n$ need not be independent. This is crucial for applications in the analysis of algorithms where the involved random variables are often the result of complex interactions, and hence often fail to be independent.

- The function f is allowed to be arbitrarily complex and in fact, one does not even require an explicit description of it! Only certain weak conditions are placed on the function, essentially that the function is "well–behaved" or "smooth" in the sense that one can control the effect of individual variables on the function. That is, if we change the values of only a few variables (keeping the remaining fixed), then the resulting change in the value of the function can be bounded well.

We will now give very briefly, the basic definitions and concepts we require. We will give an elementary and simplified account which is sufficient for all our purposes rather than the most general one possible. A sequence of random variables $\mathbf{Y} = Y_0, Y_1, \ldots$ is a martingale with respect to a sequence of random variables $\mathbf{X} = X_0, X_1, \ldots$ if for each $i \geq 0$,

- $Y_i = f_i(X_j, j \leq i)$ for some function f_i.

- $\mathtt{E}[Y_{i+1} \mid X_j, j \leq i] = Y_i$.

Intuitively, the variables $X_1, X_2, \ldots$ are the basic underlying random variables in the stochastic process at hand and their values are being revealed or "exposed" in the indicated sequence. The variables $Y_1, Y_2, \ldots$ are functions of the basic underlying variables which we observe in the indicated sequence as we gather more and more information about the random outcomes of the stochastic system at hand. The first condition merely says that the value of the observation at stage i, Y_i is completely determined by all the random outcomes $X_1, \ldots, X_i$ revealed upto this stage. The second condition says that the value of observations in the future cannot be predicted, on average, better than the currently observed value.

Example 6..1 (Tossing coins or Gambling) Consider repeated independent trials of tossing a fair coin. The basic underlying random variables are $X_i, i \geq 1$ where X_i is 1 if we get heads on the ith trial and 0 otherwise. Let the observed variables be the partial sums $Y_i := \sum_{j \leq i} X_j$. This representation already shows the first condition. For the second condition, we have,

$$\begin{aligned} \mathtt{E}[Y_{i+1} \mid X_j, j \leq i] &= \mathtt{E}[Y_i + X_{i+1} \mid X_j, j \leq i] \\ &= Y_i + \mathtt{E}[X_{i+1} \mid X_j, j \leq i] \\ &= Y_i + \mathtt{E}[X_{i+1}], \quad \text{since the trials are independent,} \\ &= Y_i, \quad \text{since the coin is fair.} \end{aligned}$$

Thus the partial sums $Y_i, i \geq 1$ are a martingale with respect to the trial variables $X_i, i \geq 1$. This is the original example from gambling, where the X_i variables give the outcome of the ith gamble and Y_i represents the fortune of the gambler after the ith trial.

The basic inequality is due independently to Azuma and (implicitly) Hoeffding:

Theorem 10 (Azuma's Inequality) *Let* $\mathbf{Y} = Y_0, Y_1, \ldots$ *be a martingale with respect to the variables* $\mathbf{X} = X_0, X_1, \ldots$. *Suppose further that there are non–negative reals* $c_i, i \geq 0$ *such that the following* **bounded differences condition** *holds on the the martingale differences:*

$$|Y_i - Y_{i-1}| \leq c_i, \quad i \geq 1.$$

Then

$$\mathtt{Pr}[Y_n > Y_0 + t], \mathtt{Pr}[Y_n < Y_0 - t] \leq \exp\left(\frac{-2t^2}{\sum_{i \leq i \leq n} c_i^2}\right).$$

A standard way to define a martingale sequence is via the so–called *Doob process*: let f be an arbitrary function and let $X_1, \ldots, X_n$ be arbitrary random variables. Then the sequence

$$Y_i := \mathtt{E}[f(X_1, \ldots, X_n) \mid X_1, \ldots, X_i], \quad 0 \leq i \leq n,$$

is a martingale sequence with respect to $0 =: X_0, X_1, \ldots, X_n$. Applying Azuma's inequality to the Doob martingale gives us the first version of the so–called Method of Bounded Differences, which we shall refer to as the Method of Martingale Differences:

Theorem 11 (Method of Martingale Differences) *Let $X_1, \ldots, X_n$ be arbitrary random variables and let f be an arbitrary function. Suppose there exist non–negative reals $c_i, i \in [n]$ such that*

$$|\mathtt{E}[f \mid X_1, \ldots, X_i] - \mathtt{E}[f \mid X_1, \ldots, X_{i-1}]| \leq c_i. \qquad (1.7)$$

Then

$$\mathtt{Pr}[f > \mathtt{E}[f] + t], \mathtt{Pr}[f < \mathtt{E}[f] - t] \leq \exp\left(\frac{-2t^2}{\sum_{i \leq i \leq n} c_i^2}\right).$$

An averaging argument yields a slightly weaker form of the method which is often more convenient:

Theorem 12 (Method of Bounded Average Differences) *Let $X_1, \ldots, X_n$ be arbitrary random variables and let f be an arbitrary function. Suppose there exist non–negative reals $c_i, i \in [n]$ such that for any two values a, a' that X_i can assume,*

$$\mathtt{E}[f \mid \mathbf{X}_{i-1}, X_i = a] - |\mathtt{E}[f \mid \mathbf{X}_{i-1}, X_i = a']| \leq c_i. \qquad (1.8)$$

Then

$$\mathtt{Pr}[f > \mathtt{E}[f] + t], \mathtt{Pr}[f < \mathtt{E}[f] - t] \leq \exp\left(\frac{-2t^2}{\sum_{i \leq i \leq n} c_i^2}\right).$$

A dramatically simpler *avatara* of the method emerges if we place a very natural condition on f: we will say that f is *Lipschitz* with non–negative constants $c_i, i \in [n]$ if for all $\mathbf{a}, \mathbf{a}'$ that differ only in the ith co–ordinate,

$$f(\mathbf{a}) - f(\mathbf{a}')| \leq c_i. \qquad (1.9)$$

This is in fact the usual definition of the Lipschitz condition when the underlying metric is the weighted Hamming metric:

$$d_H(x, y) := \sum_i c_i [x_i \neq y_i].$$

Under this condition we obtain the best known version of the method:

Theorem 13 (Method of Bounded Differences) *Let $X_1, \ldots, X_n$ be* **independent** *random variables and let f be Lipschitz with non–negative constants $c_i, i \in [n]$. Then,*

$$\mathtt{Pr}[f > \mathtt{E}[f] + t], \mathtt{Pr}[f < \mathtt{E}[f] - t] \leq \exp\left(\frac{-2t^2}{\sum_{i \leq i \leq n} c_i^2}\right).$$

We shall illustrate the three versions of the method with applications below. First however, we make some general remarks on a relative comparison of the three versions:

- The Method of Bounded Differences is the most convenient one to use in applications, for it only involves checking the Lipschitz condition (1.9) on the function f. There are however, two limitations of the method: first the underlying variables are required to be independent, and second, the resulting bound can often be very weak.

- It may be the case in an application that f satisfies the Lipschitz condition with small constants in most cases, but there are some pathological low probability cases in which the constant is much larger. In this case, the Method of Bounded Differences unfairly penalizes the function with the worst case constants rather than the "average case constants". The Method of Bounded Average Differences replaces the worst case constants with a version of "average case constants", according it the name. Since the resulting constants can be much smaller than the constants required in the Method of Bounded Differences, the bound obtained is correspondingly much stronger.

- The Method of Martingale Differences is the most powerful version of the method. While it might not differ perceptibly from the Method of Bounded Average Differences in the bounds obtained, it can be more convenient to apply in certain situations.

6.1 COUPLING

In order to make effective use of the Method of Average Bounded Differences, we need to get a good handle on the bound (1.8), for the difference in the expected values of a function under two different conditioned distributions. A very useful technique for this is the method of *coupling.* Suppose that we can find a joint distribution $\pi(\mathbf{Y}, \mathbf{Y}')$ such that the marginal distribution for Y is the same as the distribution of $\mathbf{X}$ conditioned on $\mathbf{X}_{i-1} = \mathbf{a}_{i-1}, X_i = a_i$ and the marginal distribution for $\mathbf{Y}'$ is the same as the distribution $\mathbf{X}$ conditioned on $\mathbf{X}_{i-1} = \mathbf{a}_{i-1}, X_i = a'_i$. Such a joint distribution is called a coupling of the two original distributions. Then,

$$\begin{aligned}|\mathtt{E}[f \mid \mathbf{X}_{i-1} = \mathbf{a}_{i-1}, X_i = a_i] - \mathtt{E}[f \mid \mathbf{X}_{i-1} = \mathbf{a}_{i-1}, X_i = a'_i]| &= \\ |\mathtt{E}_\pi[f(\mathbf{Y})] - \mathtt{E}_\pi[f(\mathbf{Y}')]| = |\mathtt{E}_\pi[f(\mathbf{Y}) - f(\mathbf{Y}')]| & \qquad (1.10)\end{aligned}$$

If the coupling π is well–chosen so that $|f(\mathbf{Y}) - f(\mathbf{Y}')|$ is usually very small, we can get a good bound on the difference (1.8). For example, suppose that

- For any sample point $(\mathbf{y}, \mathbf{y}')$ we have $|f(\mathbf{y}) - f(\mathbf{y}')| \leq d$ for some constant $d > 0$; and
- For most sample points $(\mathbf{y}, \mathbf{y}', f(\mathbf{y}) = f(\mathbf{y}')$. That is, $\pi[f(\mathbf{Y}) - f(\mathbf{Y}')] \leq p$, for some $p << 1$.

Then, we can conclude using (1.9) that

$$|\mathtt{E}[f \mid \mathbf{X}_{i-1} = \mathbf{a}_{i-1}, X_i = a_i] - \mathtt{E}[f \mid \mathbf{X}_{i-1} = \mathbf{a}_{i-1}, X_i = a_i']| \leq pd.$$

We shall construct suitable couplings to bound the difference in (1.8).

7. APPLICATIONS OF MARTINGALE METHODS

Example 7..1 (Bin Packing) In the *bin packing* problem, we are given items of sizes $a_1, \ldots, a_n$ with $0 \leq a_i \leq 1$ for each $i \in [n]$, and we are required to "pack" them into a minimum number of bins, each of unit capacity. There are many known heuristics for this problem, which in unlikely to be solvable efficiently because it is NP-complete. One such is *first-fit*: put the next item in the first bin in which it can fit. Consider the average case instance of this problem, where each a_i is drawn independently and uniformly at random from $[0, 1]$. A general theorem from the theory of *subadditive functionals* [43] shows that the optimum number of bins, $B_n := B(a_1, \ldots, a_n)$ satisfies the limit law:

$$\mathtt{Pr}[\lim_n \frac{B_n}{n} = \gamma] = 1,$$

for some constant $\gamma > 0$. In the limit, therefore, the expectation will also be γn. In any case, for any finite n as well, one can easily deduce a sharp concentration bound on $\mathtt{E}[B_n]$ using the simplest form of the method of bounded differences. Just observe that $B(a_1, \ldots, a_n)$ can change by at most 1 if one changes only one item size while keeping the others fixed. In fact this is true also of the number of bins delivered by the first fit heuristic. Thus for either of these random variables, we have the concentration result:

$$\mathtt{Pr}[|B_n - ep[B_n]| > t] \leq \exp\left(-2t^2/n\right),$$

which for $t = \delta\mathtt{E}[B_n]$ gives an exponentially decreasing probability.

Example 7..2 (Quicksort) We shall sketch the application of the Method of Martingale Differences to Quicksort. This application is interesting because it is a very natural application of the method and yields a provably optimal tail bound. While conceptually simple, the details required to obtain the full bound are messy, so we shall confine ourselves to indicating the basic method.

Recall that Quicksort can be modeled as a binary tree T, corresponding to the partition around the pivot element performed at each stage. With each node v of the binary tree, we associate the list L_v that needs to be sorted there. At the outset, the root r is associated with $L_r = L$, the input list, and if the the pivot element chosen at node v is X_v, the lists associated with the left and right children of v are the sublists of L_v consisting of, respectively, all elements less than X_v and all elements greater than X_v (for simplicity, we assume that the input list contains all distinct elements). Now, the number of comparisons performed by Quicksort on the input list L, Q_L is a random variable given by some function f of the random choices made for the pivot elements, $X_v, v \in T$:

$$Q_L = f(X_v, v \in T).$$

We shall now expose the variables $X_v, v \in T$ in the natural top–down fashion: level–by–level and left to right within a level, starting with the root. Let us denote this (inorder) ordering of the nodes of T by $<$. Thus, to apply the Method of Martingale Differences, we merely need to estimate for each node $v \in T$,

$$|\mathtt{E}[Q_L \mid X_w, w < v] - \mathtt{E}[Q_L \mid X_w, w \leq v]|.$$

A moment's reflection shows that this difference is simply

$$|\mathtt{E}[Q_{L_v}] - \mathtt{E}[Q_{L_v} \mid X_v]|,$$

where L_v is the list associated with v as a result of the previous choices of the partitions given by $X_w, w < v$. That is, the problem reduces to estimating the difference between the expected number of comparisons performed on a given list when the first partition is specified and when it is not. Such an estimate is readily available for Quicksort via the recurrence satisfied by the expected value $q_n := \mathtt{E}[Q_n]$, the expected number of comparisons performed on a input list of length n. If the first partition (which by itself requires $n-1$ comparisons) splits the list into a left part of size $k, 0 \leq k < n$ and a right part of size $n-1-k$, the expected number of comparisons is $n-1+q_k+q_{n-1-k}$ and the estimate is:

$$|q_n - (n-1+q_k+q_{n-k-1})| \leq n-1.$$

We shall plug this estimate into the Method of Bounded Differences: thus, if $\ell_v := |L_v|$ is the length of the list associated with node v, then we need to estimate $\sum_v \ell_v^2$. This is potentially problematical, since these lengths are themselves random variables! Suppose, that we restrict attention to levels $k \geq k_1$ for which we can show that $\ell_v \leq \alpha n$ for some parameters k_1 and α to be chosen later; then summing over these levels, level by level,

$$\begin{aligned}\sum_v \ell_v^2 &= \sum_{k \geq k_1} \sum_{h(v)=k} \ell_v^2 \\ &\leq \sum_{k \geq k_1} \sum_{h(v)=k} \alpha n \ell_v \\ &= \sum_{k \geq k_1} \alpha n \sum_{h(v)=k} \ell_v \\ &\leq \sum_{k \geq k_1} \alpha n^2.\end{aligned}$$

Next we are faced with yet another problem: the number of levels, which itself is again a random variable! Suppose we can show for some $k_2 > k_1$, that the tree has height no more than k_2 with high probability. Then the previously computed sum reduces to $(k_2 - k_1)\alpha n^2$. X Finally all that remains is to choose the parameters carefully: suppose we choose k_1 and α so that the maximum size of the list associated with a node at height at least k_1 exceeds αn with probability at most p_1 and and k_2 so that the overall height of the tree exceeds k_2 with probability at most p_2. This can be done in an elementary way by using the fact that the size of the list at a node at depth $k \geq 0$ is explicitly given by $n \prod_{1 \leq i \leq k} Z_i$, where each Z_i is uniformly distributed in $[0, 1]$. Then the final result will be:

$$\texttt{Pr}[Q_n > q_n + t] < p_1 + p_2 + \exp\left(\frac{-2t^2}{(k_2 - k_1)\alpha n^2}\right).$$

We choose the parameters to optimize this sum of three terms. The result whose details are messy (see [25]) is:

Theorem 14 *Let* $\epsilon = \epsilon(n)$ *satisfy* $1/\ln n < \epsilon \leq 1$. *Then as* $n \to \infty$,

$$\texttt{Pr}[|\frac{Q_n}{q_n} - 1| > \epsilon] < n^{-2\epsilon \log \log n}.$$

This bound is slightly better than an inverse polynomial bound and can be shown to be essentially tight.

8. DISTRIBUTED EDGE COLORING

Vizing's Theorem shows that every graph G can be edge colored sequentially in polynomial time with Δ or $\Delta + 1$ colors, where Δ is the maximum degree of the input graph (see, for instance, [3]). The proof is in fact a polynomial time sequential algorithm for achieving a $\Delta + 1$ coloring.

It is a challenging open problem whether colorings as good as these can be computed fast in a distributed model. Here one has a set of processors connected by an arbitrary network of communication channels. Data is distributed throughout the network amongst the processors. The processors are required to co–operate together in some fashion to compute some function of all the data. For this, they communicate data between each other via the communication channels of the network. In the graph coloring problem, the network of processors is required to compute an edge coloring of itself.

In **many** realistic situations, it is the communication that is expensive rather than internal computation at any single processor. Hence, in the model, communication is at a premium rather than computation. One is therefore required to design algorithms that minimize communication. This poses a *locality* constraint: the algorithm should require each processor to gather only local information i.e. from processors in a small neighborhood around itself.

The difficulty posed by this locality constraint is often compounded by the problem of symmetry-breaking; if every process has the same view of the network– i.e. the same input– it In the absence of such a result one might aim at the more modest goal of computing reasonably good colorings, instead of optimal ones. By a trivial modification of a well-known *vertex* coloring algorithm of Luby it is possible to edge color a graph using $2\Delta - 2$ colors in $O(\log n)$ rounds (where n is the number of processors) [24].

We shall present and analyze two classes of simple localized distributed algorithms that compute near optimal edge colorings. Both algorithms proceed in a sequence of rounds. In each round, a simple randomized heuristic is invoked to color a significant fraction of the edges successfully. The remaining edges are passed over to succeeding rounds. This continues until the number of edges is small enough to employ a brute–force method at the final step. For example, the algorithm of Luby mentioned above can be invoked when the degree of the graph becomes small i.e. when the condition $\Delta \gg \log n$ is no longer satisfied.

8.1 TWO CLASSES OF ALGORITHMS

One of the classes of algorithms involves a standard reduction to bipartite graphs described in [32]: the graph is split into two parts T ("top") and B (bottom). The bipartite graph $G[T, B]$ induced by the edges connecting top and bottom vertices is colored by invoking the Algorithm P described below. The algorithm is then invoked recursively in parallel on $G[T]$ and $G[B]$, the graphs respectively induced by the top and bottom vertices. Both graphs are colored using the same set of colors. Thus it suffices to describe the algorithm used for coloring bipartite graphs.

We describe the action carried out by both algorithms in a single round. For the second class of algorithms, we describe the action only for bipartite graphs; additionally, each vertex knows whether it is top or bottom. At the beginning of each round, there is a palette of fresh new available colors, $[\Delta]$, where Δ is the maximum degree of the graph at the current stage. For simplicity we will assume that the graph is Δ–regular.

Algorithm I(Independent): Each edge *independently* picks a color. This *tentative* color becomes permanent if there are no conflicting edges picking the same tentative color at either endpoint.

Algorithm P(Permutation): There is a two step protocol:

- Each bottom vertex, in parallel, makes a *proposal* independently of other bottom vertices by assigning a random *permutation* of the colors to their incident edges.
- Each top vertex, in parallel, then picks a *winner* out of every set of incident edges that have the same color. Tentative colors of winner edges become final.
- The *losers*– edges who are not winners– are decoloured and passed to the next round.

For the purposes of the high probability analysis below, the exact rule used for selecting the winner edge is unimportant – it can be chosen arbitrarily from any of the edges of the relevant color; we merely require that it should not depend on edges of different colors. This is another illustration of the power of the martingale method.

It is apparent that both algorithms are truly distributed. That is to say, each vertex need only exchange information with the neighbors to execute the algorithm. This and its simplicity make the algorithms amenable for implementations in a distributed environment. Algorithm I is used with some more modifications in a number of edge coloring algorithms [11, 15]. Algorithm P is exactly the algorithm used in [32].

We focus all our attention in the analysis of one round of both algorithms. Let Δ denote the maximum degree of the graph at the beginning of the round and Δ' denote the maximum degree of the leftover graph. One can easily show that both algorithms, $\mathtt{E}[\Delta' \mid \Delta] \leq \beta\Delta$, for some constant $\beta < 1$. For algorithm I, $\beta = 1 - e^{-2}$ while for algorithm P, $\beta = 1/e$. The goal is to show that this holds with high probability. This is done in § 8.2 after the relevant tools – the Martingale inequalities – are introduced in the next section.

For completeness, we sketch a calculation of the total number of colors $\mathrm{BC}(\Delta)$ used by Algorithm P for the bipartite coloring of a graph with maximum degree Δ: with high probability, it is,

$$\begin{aligned} \mathrm{BC}(\Delta) &= \Delta + \frac{(1+\epsilon)\Delta}{e} + \frac{(1+\epsilon)^2\Delta^2}{e} + \dots \\ &\leq \frac{1}{1-(1+\epsilon)e}\Delta \approx 1.59\Delta \text{ for small enough } \epsilon. \end{aligned}$$

To this, one should add $O(\log n)$ colors at the end of the process. As can be seen by analyzing the simple recursion describing the number of colors used by the outer level of the recursion, the overall numbers of colors is the same $1.59\Delta + O(\log n)$, [32].

8.2 HIGH PROBABILITY ANALYSES

Top Vertices. The analysis is particularly easy when v is a top vertex in Algorithm P. For, in this case, the incident edges all receive colors independently of each other. This is exactly the situation of the classical balls and bins experiment: the incident edges are the "balls" that are falling at random independently into the colors that represent the "bins". One can apply the method of bounded differences in the simplest form. Let $T_e, e \in E$, be the random variables taking values in $[\Delta]$ that represent the tentative colors of the edges. Then the number of edges successfully colored around v is a function $f(T_e, e \in N^1(v))$, where $N^1(v)$ denotes the set of edges incident on v.

It is easily seen that this function has the *Lipschitz* property with constant 1: changing only one argument while leaving the others fixed only changes the value of f by at most 1. Note that this is true *regardless of the rule for choosing winners*, as long as this rule does not depend on edges of different colors. This will also be true of the remaining analyses below and illustrates once again, the power of the martingale methods.

Moreover, the variables $T_e, e \in N^1(v)$ are independent when v is a "top" vertex. Hence, by the method of bounded differences in the simplest form, we get the following sharp concentration result by plugging into Theorem 13:

Theorem 15 *Let v be a top vertex in algorithm P and let f be the number of edges around v that are successfully colored in one round of the algorithm. Then,*

$$\mathtt{Pr}[|f - \mathtt{E}[f]| > t] \leq \exp\left(\frac{-2t^2}{\Delta}\right),$$

For $t := \epsilon\Delta$ $(0 < \epsilon < 1)$, this gives an exponentially decreasing probability for deviations around the mean. If $\Delta \gg \log n$ then the probability that the new degree of any vertex deviates far from its expected value is inverse polynomial, i.e. the new max degree is sharply concentrated around its mean.

Other Vertices: The Difficulty. The analysis for the "bottom" vertices in Algorithm P is more complicated in several respects. It is useful to see why so that one can appreciate the need for using a more sophisticated tool such as the Method of Bounded Average Differences. To start with, one could introduce an indicator random variable X_e for each edge e incident upon a bottom vertex v. These random variable are not independent however. Consider a four cycle with vertices v, a, w, b, where v and w are bottom vertices and a and b are top vertices. Let's refer to the process of selecting the winner (step 2 of the algorithm P) as "the lottery". Suppose that we are given the information that edge va got tentative color red and lost the lottery— i.e. $X_{va} = 0$— and that edge vb got tentative color green. We'll argue intuitively that given this, it is more likely that $X_{vb} = 0$. Since edge va lost the lottery, the probability that edge wa gets tentative color red increases. In turn, this increases the probability that edge wb gets tentative color green, which implies that edge vb is more likely to lose the lottery. So, not only are the X_e's not independent, but the dependency among them is particularly malicious.

One could hope to bound this effect by using the MOBD in it simplest form. This is also ruled out however, for two reasons. The first is that the tentative color choices of the edges around a vertex are not independent. This is because the edges incident on vertex are assigned a permutation of the colors. The second reason applies also to algorithm I where all edges act independently. The new degree of v, a bottom vertex in algorithm P or an arbitrary vertex in algorithm I, is a function $f = f(T_e, e \in N(v))$, where $N(v)$ is the set of edges at distance at most 2 from v. Thus f depends on as many as $\Delta(\Delta - 1) = \Theta(\Delta^2)$ edges. Even if f is Lipshitz with constants $d_i = 2$, this is not enough to get a strong enough bound because $d = \sum_i d_i^2 = \Theta(\Delta^2)$. Applying the method of bounded differences in the simple form, Theorem 13, would give the

bound
$$\texttt{Pr}[|f-\texttt{E}[f]|>t] \le 2\exp\left(-\frac{t^2}{\Theta(\Delta^2)}\right).$$
This bound however is useless for $t=\epsilon\texttt{E}[f]$ since $\texttt{E}[f]\approx\Delta/e$.

We will use the Method of Bounded Average Differences, Theorem 12, to get a much better bound. We shall invoke the two crucial features of this more general method. Namely that it does not need the underlying variables to be independent [2], and that, second that as we shall see, it allows us to bound the effect of individual random choices with constants much smaller than those given by the MOBD in simple form.

Let's now move on to the analysis. A similar analysis applies to both cases: when v is a bottom vertex in algorithm P or an arbitrary vertex in algorithm I. Let $N^1(v)$ denote the set of "direct" edges– i.e. the edges incident on v– and let $N^2(v)$ denote the set of "indirect edges" that is, the edges incident on a neighbor of v. Let $N(v) := N^1(v)\bigcup N^2(v)$. The number of edges successfully colored at vertex v is a function $f(T_e, e\in N(v))$. Note that in Algorithm P, even though f seems to depend on edges at distance 3 from v via their effect on edges at distance 2, f can still be regarded as a function of the edges in $N(v)$ only (i.e. f is fixed by giving colors to all edges in $N(v)$ regardless of what happens to other edges) and hence only these edges need be considered in the analysis.

Let us number the variables so that the direct edges are numbered *after* the indirect edges (this will be important for the calculations to follow). We need to compute
$$\lambda_k := |\texttt{E}[f \mid \mathbf{T}_{k-1}, T_k = c_k] - \texttt{E}[f \mid \mathbf{T}_{k-1}, T_k = c'_k]|. \tag{1.11}$$
We decompose f as a sum to ease the computations later. Introduce the indicator functions $f_e, e\in E$:
$$f_e(\mathbf{c}) := \begin{cases} 1; & \text{if edge } e \text{ is successfully coloured in colouring } \mathbf{c},\\ 0; & \text{otherwise.}\end{cases}$$
Then $f=\sum_{v\in e} f_e$.

Hence we are reduced, by linearity of expectation, to computing for each $e\in N^1(v)$,
$$|\texttt{Pr}[f_e = 1 \mid \mathbf{T}_{k-1}, T_k = c_k] - \texttt{Pr}[f_e = 1 \mid \mathbf{T}_{k-1}, T_k = c'_k]|.$$

For the computations that follows we should keep in mind that in algorithm P bottom vertices assign colors independently of each other. This implies that in either algorithm, the color choices of the edges incident upon a neighbor of v are independent of each other. In Algorithm I, *all* edges have their colors assigned independently.

General Vertex in Algorithm I. To compute a good bound for λ_k in (1.11), we shall construct a suitable coupling of the two different conditioned distributions. The coupling $(\mathbf{Y}, \mathbf{Y}')$ is almost trivial: $\mathbf{Y}$ is distributed as $\mathbf{T}$ conditioned on $\mathbf{T}_{k-1}, T_k = c_k$ and $\mathbf{Y}'$ is identically equal to $\mathbf{Y}$ except that $\mathbf{Y}'_k = c'_k$. It is easily seen that by the independence of all tentative colors, the marginal distributions of $\mathbf{Y}$ and $\mathbf{Y}'$ are exactly the two conditioned distributions $[\mathbf{T} \mid \mathbf{T}_{k-1}, T_k = c_k]$ and $[\mathbf{T} \mid \mathbf{T}_{k-1}, T_k = c'_k]$ respectively.

Now let us compute $|\mathtt{E}[f(\mathbf{Y}) - f(\mathbf{Y}')]|$.

- First, let us consider the case when $e_1, \ldots, e_k \in N^2(v)$, i.e. only the choices of indirect edges are exposed. Let $e_k = (w, z)$, where w is a neighbor of v. Then for a direct edge $e \neq vw$, $f_e(\mathbf{y}) = f_e(\mathbf{y}')$ because in the joint distribution space, $\mathbf{y}$ and $\mathbf{y}'$ agree on all edges incident on e. So we only need to compute $|\mathtt{E}[f_{vw}(\mathbf{Y}) - f_{vw}(\mathbf{Y}')]|$. To bound this simply, we observe first that $f_{vw}(\mathbf{y}) - f_{vw}(\mathbf{y}') \in [-1, 1]$ and second that $f_{vw}(\mathbf{y}) = f_{vw}(\mathbf{y}')$ unless $y_{vw} = c_k or c'_k$. Thus we can conclude that

  $$\mathtt{E}[f_{vw}(\mathbf{Y}) - f_{vw}(\mathbf{Y}')]| \leq \mathtt{Pr}[Y_e = c_k \vee Y_e = c'_k] \leq \frac{2}{\Delta}.$$

  In fact one can do a tighter analysis using the same observations. Let us denote $f_e(\mathbf{y}, y_{w,z} = c_1, y_e = c_2)$ by $f_e(c_1, c_2)$. Note that $f_{vw}(c_k, c_k) = 0$ and similarly $f_{vw}(c'_k, c'_k) = 0$. Hence

  $$\begin{aligned}
  \mathtt{E}[f_e(\mathbf{Y}) - f_e(\mathbf{Y}') \mid z] = & \\
  & (f_{vw}(c_k, c_k) - f_{vw}(c'_k, c_k))\mathtt{Pr}[Y_e = c_k] + (f_{vw}(c_k, c'_k) - f_{vw}(c'_k, c'_k))\mathtt{Pr}[Y_e = c'_k] \\
  = & (f_{vw}(c_k, c'_k) - f_{vw}(c'_k, c_k))\frac{1}{\Delta}
  \end{aligned}$$

  (Here we used the fact that the distribution of color around v is unaffected by the conditioning around z and that each color is equally likely.) Hence $|\mathtt{E}[f_e(\mathbf{Y}) - f_e(\mathbf{Y}')]| \leq \frac{1}{\Delta}$.

- Now let us consider the case when $e_k \in N^1(v)$, i.e. choices of all indirect edges and of some direct edges have been exposed. In this case, we merely observe that f is Lipshitz with constant 2: $|f(\mathbf{y}) - f(\mathbf{y}')| \leq 2$ whenever $\mathbf{y}$ and $\mathbf{y}'$ differ in only one co–ordinate. Hence we can easily conclude that $|\mathtt{E}[f(\mathbf{Y}) - f(\mathbf{Y}')]| \leq 2$.

Overall,

$$\lambda_k \leq \begin{cases} 1/\Delta; & \text{for an edge } e_k \in N^2(v), \\ 2; & \text{for an edge } e_k \in N^1(v) \end{cases},$$

and we get

$$\sum_k \lambda_k^2 = \sum_{e \in N^2(v)} \frac{1}{\Delta^2} + \sum_{e \in N^1(v)} 4 \leq 4\Delta + 1.$$

We thus arrive at the following sharp concentration result by plugging into Theorem 12:

Theorem 16 *Let v be an arbitrary vertex and let Let f be the number of edges successfully colored around v in one stage of algorithm I. Then,*

$$\mathtt{Pr}[|f - \mathtt{E}[f]| > t] \leq 2 \exp\left(-\frac{t^2}{2\Delta + \frac{1}{2}}\right).$$

Jaikumar Radhakrishnan observed that a similar result can be obtained very simply for Algorithm I by applying Theorem 13: regard f as a function of Δ variables: $\mathbf{T}_w, (v, w) \in E$, where $\mathbf{T}_w := (T_e, w \in e)$ is the vector of tentative colors of edges around w. Since f is Lipshitz with constant 2 with respect to each of these (vector valued) variables, we get the bound:

$$\mathtt{Pr}[|f - \mathtt{E}[f]| > t] \leq 2 \exp\left(-\frac{t^2}{2\Delta}\right).$$

Bottom Vertices in Algorithm P. Once again, to compute a good bound for λ_k in (1.11), we shall construct a suitable coupling [3] of the two different conditioned distributions $\mathbf{T}_{k-1}, T_k = c_k$ and $\mathbf{T}_{k-1}, T_k = c'_k$. Suppose e_k is an edge zy where z is a bottom vertex. The coupling $(\mathbf{Y}, \mathbf{Y}')$ in this case is the following: $\mathbf{Y}$ is distributed as $\mathbf{T}$ conditioned on $\mathbf{T}_{k-1}, T_k = c_k$ and $\mathbf{Y}'$ is identically equal to $\mathbf{Y}$ except on the edges incident on z, where the colors c_k and c'_k are switched. We can think of the distribution as divided into two classes: on the edges incident on a vertex other than z, the two variables $\mathbf{Y}$ and $\mathbf{Y}'$ are identically equal. In particular, when z is not v, they have the same uniform distribution on all permutations of colors on the edges around v. However, on the edges incident on z, the two variables differ on exactly two edges where the colors c_k and c'_k are switched. It is easily seen that by the independence of different vertices, the marginal distributions of $\mathbf{Y}$ and $\mathbf{Y}'$ are exactly the two conditioned distributions $[\mathbf{T} \mid \mathbf{T}_{k-1}, T_k = c_k]$ and $[\mathbf{T} \mid \mathbf{T}_{k-1}, T_k = c'_k]$ respectively.

Now let us compute $|\mathtt{E}[f(\mathbf{Y}) - f(\mathbf{Y}')]|$. Recall that f was decomposed as a sum $\sum_{v \in e} f_e$. Hence by linearity of expectation, we need only bound each $|\mathtt{E}[f_e(\mathbf{Y}) - f_e(\mathbf{Y}')]|$ separately.

- First, let us consider the case when $e_1, \ldots, e_k \in N^2(v)$, i.e. only the choices of indirect edges are exposed. Let $e_k = (w, z)$ for a neighbor w of v. Note that since

$$\mathtt{E}[f(\mathbf{Y}) - f(\mathbf{Y}')] = \mathtt{E}[\mathtt{E}[f(\mathbf{Y}) - f(\mathbf{Y}') \mid \mathbf{Y}_e, \mathbf{Y}'_e, z \in e]],$$

it suffices to bound $|\mathtt{E}[f(\mathbf{Y}) - f(\mathbf{Y}') \mid \mathbf{Y}_e, \mathbf{Y}'_e, z \in e]|$. Hence, fix some distribution of the colors around z. Recall that $\mathbf{Y}_{w,z} = c_k$ and $\mathbf{Y}'_{w,z} = c'_k$. Suppose $\mathbf{Y}_{z,w'} = c'_k$ for some other neighbor w' of z. Then by our coupling construction, $\mathbf{Y}'_{z,w'} = c_k$ and on the remaining edges $\mathbf{Y}$ and $\mathbf{Y}'$ agree identically. Moreover, by the independence of the other vertices, the distributions of $\mathbf{Y}$ and $\mathbf{Y}'$ on the remaining edges conditioned on the distribution around z is unaffected. let us denote the conditioned joint distribution by $[(\mathbf{Y}, \mathbf{Y}') \mid z]$. We thus need to bound $|\mathtt{E}[f(\mathbf{Y}) - f(\mathbf{Y}') \mid z]|$.

Then for a direct edge $e \notin vw, vw'$, $f_e(\mathbf{y}) = f_e(\mathbf{y}')$ because in the joint distribution space, $\mathbf{y}$ and $\mathbf{y}'$ agree on all edges incident on e. So we only need to compute $|\mathtt{E}[f_e(\mathbf{Y}) - f_e(\mathbf{Y}')]|$ for $e \in vw, vw'$. To bound this simply, we observe that for either $e = vw$ or $e = vw'$, first, $f_e(\mathbf{y}) - f_e(\mathbf{y}') \in [-1, 1]$ and second that $f_e(\mathbf{y}) = f_e(\mathbf{y}')$ unless $y_e = c_k or c'_k$. Thus we can conclude that

$$\mathtt{E}[f_e(\mathbf{Y}) - f_e(\mathbf{Y}')]| \leq \mathtt{Pr}[Y_e = c_k \vee Y_e = c'_k] \leq \frac{2}{\Delta}.$$

Thus taking the two contributions for vw and vw' together, $|\mathtt{E}[f(\mathbf{Y}) - f(\mathbf{Y}') \mid z]| \leq \frac{4}{\Delta}$.

In fact one can do a tighter analysis using the same observations. Let us denote $f_e(\mathbf{y}, y_{w,z} = c_1, y_e = c_2)$ by $f_e(c_1, c_2)$. Note that $f_{vw}(c_k, c_k) = 0 = f_{vw}(c'_k, c'_k)$ and similarly $f_{vw'}(c_k, c_k) = 0 = f_{vw'}(c'_k, c'_k)$. Thus, for $e = vw$ or $e = vw'$,

$$\begin{aligned}
&\mathtt{E}[f_e(\mathbf{Y}) - f_e(\mathbf{Y}') \mid z] = \\
&\qquad (f_e(c_k, c_k) - f_e(c'_k, c_k))\mathtt{Pr}[Y_e = c_k] + (f_e(c_k, c'_k) - f_e(c'_k, y'_e = c'_k))\mathtt{Pr}[Y_e = c'_k] \\
&= \quad (f_e(c_k, c'_k) - f_e(c'_k, c_k))\frac{1}{\Delta}
\end{aligned}$$

Hence $|\mathtt{E}[f_e(\mathbf{Y}) - f_e(\mathbf{Y}') \mid z]| \leq \frac{1}{\Delta}$. Taking the two contributions for edges vw and vw' together, $|\mathtt{E}[f(\mathbf{Y}) - f(\mathbf{Y}') \mid x]| \leq \frac{2}{\Delta}$.

- Now let us consider the case when $e_k \in N^1(v)$, i.e. choices of all indirect edges and of some direct edges have been exposed. In this case, we observe again that $|f(\mathbf{y}) - f(\mathbf{y}')| \leq 2$ since $\mathbf{y}$ and $\mathbf{y}'$ differ on exactly two edges. Hence we can easily conclude that $|\mathtt{E}[f(\mathbf{Y}) - f(\mathbf{Y}')]| \leq 2$.

Overall,

$$\lambda_k \leq \begin{cases} 2/\Delta; & \text{for an edge } e_k \in N^2(v), \\ 2; & \text{for an edge } e_k \in N^1(v) \end{cases},$$

and we get

$$\sum_k \lambda_k^2 = \sum_{e \in N^2(v)} \frac{4}{\Delta^2} + \sum_{e \in N^1(v)} 4 \leq 4(\Delta + 1).$$

We thus arrive at the following sharp concentration result by plugging into Theorem 12:

Theorem 17 *Let v be an arbitrary bottom vertex and let f be the number of edges successfully colored around v in one stage of algorithm P. Then,*

$$\mathtt{Pr}[|f - \mathtt{E}[f]| > t] \leq 2 \exp\left(-\frac{t^2}{2\Delta + 2}\right).$$

Comparing this with the corresponding bound for Algorithm I, we see that the failure probabilities for both algorithms are almost identical. For $t = \epsilon\Delta$, both a probability that decreases exponentially in Δ. As remarked earlier, if $\Delta \gg \log n$, this implies that the new max degree is sharply concentrated around the mean (with failure probability inverse polynomial in n). The constant in the exponent here is better than the one in the analysis in [32].

Extensions. It is fairly clear that the method extends more generally to cover similar scenarios in distributed computing. We sketch such a general setting: One has a distributed randomized algorithm that requires vertices to assign labels to themselves and incident edges. Each vertex acts independently of the others, and furthermore is symmetric with respect to the labels (colors). The function of interest, f depends only on a small local neighborhood around some vertex v, is Lipschitz and satisfies some version of the following *locality* property: the labels on vertices and edges far away from v only effect f if certain events are triggered on nearer vertices and edges; these triggering events correspond to the setting of the nearer vertices and edges to specific values. For example, in edge coloring, the color of an indirect edge only affects f if the incident direct edge has the same color. One can extend the same arguments as above virtually intact for this general setting. This encompasses all the edge coloring algorithms mentioned above as well as the vertex coloring algorithms in [28] and [16].

9. BRANCHING PROCESSES AND APPLICATIONS

We give a very compact account of the theory of branching process which closely follows the description in Feller [13].

9.1 SUMS OF RANDOM NUMBER OF VARIABLES

Let $\{X_k\}$ be a sequence of i.i.d. random variables with a common distribution function $\Pr[X_k = j] = f_j$ and the generating function $f(s) = \sum_i f_j s^j$. We are often interested in the sums

$$S_N = X_1 + X_2 \ldots X_N$$

where N itself is a random variable (independent of X_js). Let $\Pr[N = i] = g_i$ be the distribution of N and $g(s) = \sum_i g_i s^i$ be the generating function. For the distribution $\Pr[S_N = j] = h_j]$, we can write using the law of conditional probabilities

$$\Pr[S_N = j] = \sum_{n=0}^{n=\infty} \Pr[N = n] \cdot \Pr[X_1 + X_2 \ldots + X_n = j]$$

since N and X_k's are independent. For a fixed n, the distribution of $X_1 + X_2 \ldots + X_n$ is given by the n fold convolution of f_i and it is somewhat easier to deal with the generating function given by $f^n(s)$. The generating function of S_N can be written as

$$h(s) = \sum_{j=0}^{\infty} h_j s^j = \sum_{n=0}^{\infty} g_n f^n(s)$$

Lemma 2 *The generating function of the sum $S_N = X_1 + X_2 \ldots + X_N$ is given by $g(f(s))$ where g and f are the generating functions of N and X_i respectively.*

9.2 BRANCHING PROCESSES

We can think of branching processes as a tree (possibly infinite), where each node is associated with a distribution function p_k of having k children. Let the mean $\mu = \sum_k k \cdot p_k$. All the nodes whose distance to the root is n form the n-th generation.

Let Z_n denote the size of the n-th generation and P_n denote the generating function of the probability distribution. $Z_0 = 1$ and

$$P_1(s) = P(s) = \sum_{k=0}^{\infty} p_k s^k.$$

We would like to write a recurrence for P_s using the observation that Z_n can be partitioned into *clans* depending on its ancestor in the first generation. Thus Z_n is the sum of Z_1 random variables $Z_n^{(k)}$. Clearly $Z_n^{(k)}$ has the same probability distribution as Z_{n-1} (as the probability distribution at each node are identical and independent). From Lemma 2

$$P_n(s) = P(P_{n-1}(s)) \tag{1.12}$$

Here are some important questions that are relevant to branching process.

- Is the branching process finite ?
- What is the expected size of Z_n ?
- What is the expected size of progeny $Y_n = 1 + Z_1 + Z_2 \ldots Z_n$ and the *total* progeny (if the branching process is finite).

We will state some of the important results known in the context of the above issues.

Theorem 18 *If $\mu \leq 1$, the branching process becomes extinct with probability rapidly converging to 1. If $\mu > 1$, then the probability x_n that the branching process terminates at or before the n-th generation converges to the (unique) root of the equation $x = P(x)$, $x < 1$.*

Proof Sketch: If x_n denotes the probability $\Pr[Z_n = 0]$, then $x_n = P_n(0)$. Clearly $x_1 = p_0$ and from the recursive relation for $P_n(s)$, it follows that

$$x_n = P_n(0) = P(P_{n-1}(0)) = P(x_{n-1})$$

Notice that $x_1 < x_2 < \ldots$ since $x_2 = P(x_1) = P(p_0) > P(0) = x_1$, so there must exist a limit $x \leq 1$ such that $x = P(x)$.

It can be argued that this is the extinction probability.

Remark 2 *By solving the recurrence $x_n = P(x_{n-1})$, we can obtain the* rate of extinction *as a function of n.*

Theorem 19 *The expected size of the n-th generation is μ^n.*

Proof. $E[Z_n] = P_n'(1)$. From equation 1.12 , and chain rule, it follows that

$$P_n'(1) = P'(1) \cdot P_{n-1}'(1) = \mu \cdot E[Z_{n-1}].$$

The result follows by induction. ∎

Remark : The previous theorem gives us a clue about whether the branching process would die out (except for the case $\mu = 1$).

Theorem 20 *The generating function of the total number of descendents is given by the (unique) root of the equation*

$$t = s \cdot P(t)$$

when the branching process is finite. The mean progeny is

$$\frac{1}{1-\mu}.$$

Proof Sketch: Let $Y_n = 1 + Z_1 + Z_2 \ldots + Z_n$ be the total number of descendents up to and including the n-th generation. The total progeny is Y_n for $n \to \infty$. We will denote the generating function of Y_n by $R_n(s)$. Since $Y_1 = 1 + Z_1$, we have $R_1(s) = s \cdot P(s)$. As in the case of $P_n(s)$, we can write down a recurrence for $R_n(s)$ by adding the root.

$$R_n(s) = s \cdot P(R_{n-1}(s)). \tag{1.13}$$

Notice that for $s < 1$,

$$R_2(s) = sP(R_1(s)) < sP(s) = R_1(s)$$

because of the monotonic nature of the generating functions P and R_n. Inductively, we can argue that $R_n(s) < R_{n-1}(s)$ and so $R_n(s)$ must decrease monotonically to a limit $t(s)$ satisfying equation 1.13, namely,

$$t(s) = s \cdot P(t(s))$$

It can be shown that $t(s)$ is unique and $t(1) = 1$ when the branching process terminates; in fact it is equal to the extinction probability.

By differentiating the previous equation, we obtain, $t'(1) = \frac{1}{1-\mu}$.

Example 9..1 (Min cuts)

A *cut* of a given (connected) graph $G = (V, E)$ is set of edges which when removed disconnects the graph. An $s - t$ cut must have the property that the designated vertices s and t should be in separate components. A *mincut* is the minimum number of edges that disconnects a graph and is sometimes referred to as *global* mincut to distinguish is from $s - t$ mincut. The weighted version of the mincut problem is the natural analogue when the edges have non-negative associated weights.

A cut can also be represented by a set of vertices S where the cut-edges are the edges connecting S and $V - S$.

It was believed for a long time that the mincut is a harder problem to solve than the $s-t$ mincut - in fact the earlier algorithms for mincuts determined the $s-t$ mincuts for all pairs $s, t \in V$. The $s-t$ mincut can be determined from the $s-t$ maxflow flow algorithms and over the years, there have been improved reductions of the global mincut problem to the $s-t$ flow problem, such that it can now be solved in one computation of $s-t$ flow.

In a remarkable departure from this line of work, first Karger[19], followed by Karger and Stein [20] developed faster algorithms (than maxflow) to compute the mincut with *high probability*. The algorithms produce a cut that is very likely the mincut, i.e., these are Monte Carlo algorithms. Unfortunately, there is yet no known matching verification algorithms. We will describe an algorithm that runs in time $O(n^2 polylog(n), (n = |V|)$ which is nearly best possible for dense graphs[4] This algorithm exploits some properties of branching processes.

The contraction algorithm. The basis of the algorithm is the procedure contraction described below. The fundamental operation $contract(v_1, v_2)$ replaces vertices v_1 and v_2 by a new vertex v and assigns the set of edges incident on v by the union of the edges incident on v_1 and v_2. We do not merge edges from v_1 and v_2 with the same end-point but retain them as multiple edges. Notice that by definition, the edges between v_1 and v_2 disappear.

> **Procedure Contraction(t)**
> Input: A multigraph $G = (V, E)$
> Output: A t partition of V
>
> **Repeat** until t vertices remain
>
> choose an edge (v_1, v_2) at random
> **contract**(v_1, v_2)

Procedure $Contraction(2)$ produces a cut. Using the observation that, in an n-vertex graph with a mincut value k, the minimum degree of a vertex is k, the following can be shown quite easily.

Lemma 3 *The probability that a specific mincut $\mathcal{C}$ survives at the end of Contraction(t) is at least $\frac{t(t-1)}{n(n-1)}$.*

Therefore $Contraction(2)$ produces a mincut with probability $\Omega(\frac{1}{n^2})$.

Lemma 4 *A single iteration of the Procedure Contraction can be carried out in $O(n)$ steps.*

This is done by using an adjacency graph representation (see Karger [19] for details). Therefore using the Procedure Contract to produce mincut is somewhat expensive since we need to repeat it about n^2 times. Instead, we run Procedure $Contraction(\sqrt{n}/2)$ twice independently and repeat it recursively on the contracted graphs. The Algorithm is described below.

Algorithm Fastmincut
Input: A multigraph $G = (V, E)$
Output: A cut $\mathcal{C}$

1. Let $n := |V|$.
2. If $n \leq 6$ then compute mincut of G directly else
   - 2.1 $t := \lceil 1 + n/\sqrt{2} \rceil$.
   - 2.2 Call $Contraction(t)$ twice (independently) to produce to graphs H_1 and H_2.
   - 2.3 Let $C_1 =$ **Fastmincut** (H_1) and $C_2 =$ **Fastmincut** (H_2).
   - 2.4 $\mathcal{C} = \min\{C_1, C_2\}$

The running time of algorithm **Fastmincut** satisfies the following recurrence

$$T(n) = 2T\left(\lceil 1 + n/\sqrt{2} \rceil\right) + O(n^2)$$

which yields $T(n) = O(n^2 \log n)$. Perhaps a more interesting question is to ascertain the probability with which **Fastmincut** returns a mincut. The probability that a mincut survives in H_1 after Step 2.2 is

$$\frac{(\lceil 1 + n/\sqrt{2} \rceil)(\lceil 1 + n/\sqrt{2} \rceil - 1)}{n(n-1)} \geq \frac{1}{2}$$

from Lemma 3. The same argument applies to H_2 independently. Therefore we can view the recursive algorithm as a branching process where any node can have zero, one or two children depending on if the mincut survived in zero, one or both children. The distribution function at each node can be approximated by a binomial distribution with two trials, each with success probability greater than $1/2$ (i.e. mean $\mu \geq 1$). Since the algorithm has roughly $2 \log n$ levels of recursion, we can restate the survival probability of the mincut as the complement of the extinction probability at the $2 \log n$ generation.

From Theorem 18, the extinction probability of a branching process with mean $\mu > 1$ converges to the solution of $x = P(x)$ where P is the generating function of the probability distribution. Here, we can approximate $P(s)$ by $\frac{1}{4}+\frac{1}{2}s^2+\frac{1}{4}s^2$. Solving for x yields $x=1$ which is an asymptotic solution but does not give us much information about the rate of convergence. For this, we need to solve the recurrence

$$x_n = P(x_{n-1})$$

where x_i is the extinction probability of the i-th generation. Substituting our generating function and simplifying yields

$$x_n = \frac{1}{4}(1+x_{n-1})^2$$

The solution to this recurrence is $x_n = \Theta(\frac{1}{n})$. So the survival probability of the mincut (after $2\log n$) levels of recursion is $\Omega(\frac{1}{\log n})$. Repeating the procedure (with independently chosen random bits) $m \log n$ times increases the probability of finding the mincut to $1-\exp^{-m}$.

Example 9..2 (Fractional Cascading)

The problem of searching for a key in many ordered lists arises very frequently in computational geometry (see [5] for applications). Chazelle and Guibas [4] introduced fractional cascading as a general technique for solving this problem. Their work unified some earlier work in this area and gave a general strategy for improving upon the naive method of doing independent searches for the same key in separate lists. In brief, they devised a data-structure that would enable searching for the same key in n lists in time $O(\log M + n)$ where M is the size of the longest list. If N is the total size of all the lists then this data structure can be built in $O(N)$ preprocessing time and take $O(N)$ space.

Here, we give an alternate implementation of their data-structure that uses randomization. While retaining the salient features of their data-structure, we are able to simplify its construction considerably to an extent that is practical. The motivation of the new technique has been derived from the success of skip-lists (Pugh [33]).

Description of Fractional Cascading. In this section, we give a brief description of the problem setting and the approach taken by Chazelle and Guibas. Consider a fixed graph $G=(V,E)$ of $|V|=n$ vertices and $|E|=m$ edges. The graph G is undirected and connected and does not contain multiple edges. Each vertex v has a catalog C_v and associated with each edge e of G is a range R_e.

A catalog is an ordered collection of records where each record has an associated value in the set $\Re \cup \{\infty, -\infty\}$. The records are stored in a non-decreasing order of their values and more than one record can have the same value. A catalog is never empty (has at least a ∞ and a $-\infty$). A range is an interval of the form $[x, y], [-\infty, y], [x, \infty], [-\infty, \infty]$. The graph G together with the associated catalogs and ranges is called a catalog graph. This is the basic structure to which fractional cascading is applied.

For notational purposes, if the value k is an end-point of the range $R_{u,v}$, then k appears as the value of some record in both C_u and C_v. Moreover, if two ranges $R_{u,v}$ and $R_{v,w}$ have an end-point in common it appears twice in C_v. The space required to store a catalog graph is $N = \sum_{v \in V} |C_v|$. This includes the space to store the graph itself. A catalog graph G is said to be *locally bounded* by degree d if for each vertex v and each value of $x \in \Re$ the number of edges incident on v whose range includes x is bounded by d.

The input to a query is an arbitrary element k in the universe and a connected subtree $\Pi = (\bar{V}, \bar{E})$ such that $k \in R_e$ for all edges e in the subtree and $\bar{V} \subset V, \bar{E} \subset E$. The output of the query for each vertex $v \in \bar{V}$ is an element y such that $predecessor(y) < k \leq y$. We shall refer to such a pair of elements as *straddling pair* of k.

Theorem 21 (Chazelle-Guibas) *Let G be a catalog graph of size N and locally bounded degree d. In $O(N)$ space and $O(dN)$ time it is possible to construct a data-structure which allows multiple look-ups (query) in a subtree of size p in in time $O(pd + \log N)$. If d is fixed this is optimal. In addition, if the underlying catalog graph G is restructured to a constant degree graph, then the search time and the preprocessing time can be improved to $O(p \log d + \log N)$ and $O(N)$ respectively.*

Anatomy of the Data Structure. Our data-structure is very similar to [4] in the sense that we retain their idea of using augmented catalogs A_v for every vertex v such that $C_v \subset A_v$. But we shall use a different method for its construction. An augmented catalog A_v is also a linear list of records whose values form a sorted multiset. Augmented catalogs in neighboring nodes of G will contain a number of records with common values. The corresponding records are linked together to correlate locations in the two catalogs. The objective is that given the location of a record in A_v, we would be able to find its location (the straddling pair) in the augmented catalog of a neighbor of v in constant additional time.

More formally, for each node u and an edge e connecting u and v in G, we maintain a list of 'bridges' from u to v, B_{uv}, which is an ordered subset of records in A_v and lying in the range R_e. The end-points of R_e are the first and last records of B_{uv}. In node v, we maintain for each bridge in B_{uv} a companion bridge in B_{vu}. The value of a record is distinct from the record. Moreover each bridge is associated with a unique edge of G, implying that if a given value in A_u is chosen to be a bridge in both B_{uv} and B_{uw}, then it is duplicated and stored in different records of A_u.

A pair of consecutive bridges associated with the same edge $e = (u, v)$ defines a *gap*. Let a_u and b_u be two consecutive bridges in B_{uv} and a_v (respectively b_v) be the companion bridges in B_{vu}. If $\mathbf{value}(a_u) < \mathbf{value}(b_u)$, then the gap of b_u includes all elements of A_u positioned strictly between a_u and b_u and all elements of A_v between a_v and b_v (the bridges are not included). By definition, gap of b_v is the same as gap of b_u. One of the key strategy used by [4] is to maintain the invariant that no gap exceeds $6d$ -1 in size.

We now take a closer look at the information maintained with each record. Both C_v and A_v are maintained as linked lists. A record of C_v have the fields *key* and *up-pointer*. The key contains the value and the up-pointer is a pointer to the next record. A_v has several other fields :

(1) key: stores the value k of record r.
(2) C-pointer: holds a pointer $\nu(r)$, the successor of r in C_v.
(3) up-pointer, down-pointer : pointers to successor and predecessor in A_v.

In addition a bridge element also has pointers to its companion bridge and also the label of the edge for which it is a bridge (If r is a bridge in B_{uv} then it stores the label uv).

To answer a multiple look-up query (x, Π) where x is the key value and Π is the subtree, one begins by locating x in the first node of the path Π and then use the following properties:

Lemma 5 (CG) *: If we know the position of value x in A_v, we can compute the position of x in C_v in one step.*

This can be done by using the C-field.

Lemma 6 *: If we know the position of value x in A_v and $e = (v, w)$ is an edge of G such that $x \in R_e$, then we can compute the position of x in A_w in $O(|Gap_e(x)|)$ time. $Gap_e(x)$ is the set of elements in the gap (corresponding to edge e) that x belongs to.*

From the position of x in A_v follow up-pointers until a bridge is found which connects to A_w.Note that Chazelle-Guibas [4] maintained the invariant that all gap sizes are less than $6d$ which yields a search time of $O(\log N + d|\Pi|)$. This invariant was maintained during the construction of all the augmented catalogs which is done incrementally. Their algorithms start with empty catalog and then for each vertex v, the records of C_v are inserted in an increasing order into A_v. Between any two insertions, the gap invariants are restored. Note that a single insertion into A_v can alter the gaps leading to insertion of a new bridge which introduces new records and this could continue as a long chain of events.

We suggest the following modification. Instead of explicitly maintaining gap invariants, we choose a newly inserted element r in A_v to be a bridge with probability p (we shall determine the exact value later). This is repeated for every edge incident on v and whose ranges cover r. If r is chosen to be a bridge for an edge (u, w), it leads to the insertion of a a new record in A_w and (possibly even in A_v if r is already a bridge). These new records are treated exactly the same way as described above (i.e. choose it with certain probability to be a bridge element). For each new record in the augmented catalog, we initialize the following fields:

(i) C-pointer: Can be determined from the predecessor and the successor elements in the augmented catalog. If this element came from C_v, then update the C-pointers for all the elements in A_v between this record and the previous element of C_v.
(ii) Initialize the edge field.

For each new record of C_v, this process is continued until there are no more bridges to insert.

Analysis. Let us first analyze the running time for a multiple look-up query. Given the position of x in A_v we follow the up-pointers until we find a bridge b_v which connects to A_u and then traverse the down-pointers until we locate the straddling pair. Since every element is chosen to be bridge element with probability p, the expected length of a gap is $2/p$. Thus from Lemma 1, the expected search time is $\mathrm{O}(\log N + |\Pi|/p)$.

Moreover, if $|\Pi| \geq \log N$, we can show that the search time is $\tilde{O}(\log N + |\Pi|/p)$ using the observation that the search time is a sum of $O(\log N)$ independent random variables with a geometric distribution and parameter p (Sen [41]). There is however an oversight in the above reasoning since the query is a tree of size $|\Pi|$ which can assume various forms. In particular, Π can be any of the $\mathcal{S}$ positional trees of maximum degree d. It can be shown that $|\mathcal{S}| < 2^{O(d|\Pi|)}$ so that if d is constant then the search time holds with high likelihood. Notice that there are $O(N)$ choices for the root of this tree and $O(N)$ combinatorially distinct search paths given the search tree (corresponding to the $O(N)$ intervals induced by all the key values).

The more interesting aspect of the analysis is to bound the time and space during the data-structure construction. If we look more closely at the way the records are added to the augmented catalogs, the underlying stochastic process can be modeled by a branching process. The root corresponds to an inserted record from C_v and the number of children correspond to the bridges that are created by this record. Each bridge is created with probability p, which corresponds to two children. For each new record inserted from the C_v's the time and the space needed is proportional to the total progeny of this branching process. Each node can have upto 2d children where the number of children is a random variable which takes values 0, 2, 4 ..$2d$. The probability that this random variable takes value $2k$ is the same as the probability that a binomial random variable with parameters (d, p) takes value k (i.e. there are k bridges created). The mean μ of this distribution is clearly $2pd$ and the generating function $G(s)$ can be worked out as $(q + ps^2)^d$. Here $q = 1-p$.

From Theorem 18, a branching process is finite if $\mu < 1$. Hence we choose p such that $2pd < 1$, that is $p < 1/2d$. This gives an expected gap length of greater than $4d$. From Theorem 20, if the generating function for the total progeny is denoted by $t(s)$, then $t = sG(t)$. In our case $G(t) = (q + pt^2)^d$. Moreover, the mean total progeny is $\frac{1}{1-\mu}$, which is $\frac{1}{1-2pd}$ in our case. If we choose $p = 1/3d$, this yields a mean progeny of 3. This in turn implies a total expected space bound of $O(\sum_{v \in V} C_v)$ which is $O(N)$.

We can get stronger bounds by estimating the probability of deviating from the mean value. The usual procedure is to use Chernoff bounds but in our case it is complicated by the fact that we cannot get an explicit generating function for $d > 2$ (since it involves solving equations of high degree). Instead we take an indirect approach. The total space and time for data-structure construction is the sum of N independent and identical random variables $Y_i, 1 \leq i \leq N$, each of which is the total progeny of a branching process. If $A = \sum_i X_i$, then from Chernoff Bounds,

$$\Pr[A \geq X] \leq s^{-X} t(s)^N, \text{ for } s \geq 1$$

where $t(s)$ is the generating function for each Y_i. For $X > cN$, for some fixed c, this can be rewritten as

$$\Pr[A \geq cN] \leq \left((\frac{t(s)}{s^c}) \right)^N$$

We shall prove that for some $s > 1$, there exists a constant c such that $t(s)/s^c < 1$. Let $F(s,t) = t - s(q + pt^2)^d$. Then

$$F_t(s,t) = 1 - 2tpds(q + pt^2)^{d-1}$$

We have used the notation f_y to denote the partial derivative of f with respect to variable y. Hence $F_t(1,1) = 1-2pd$ and for $2pd < 1$, $F_t(1,1) \neq 0$. For completeness, we state the following theorem.

Theorem 22 (Implicit Function Theorem) *Let f(x,y) be continuously differentiable in D. Let (x_o, y_o)be any point in D such that $f_y(x_o, y_o) \neq 0$. Then there exist numbers $\delta > 0$ and $\epsilon > 0$ and a continuously differentiable function $g(x)$ defined for $|x - x_o| \leq \delta$ and $|y - y_o| \leq \epsilon$, then $f(x,y) = f(x_o, y_o) \Leftrightarrow y = g(x)$.*

From *Implicit Function theorem*, it follows that there exists a neighborhood of $(s = 1, t = 1)$, such that

$$F(s,t) = F(1,1) \Leftrightarrow t = t(s).$$

Since $F(1,1) = 0$, i.e., $t(s) = sG(t(s))$ for $t = 1, s = 1$, there is a value $s > 1 + \epsilon$ for which $t(s) < 1 + \delta$ for some $\epsilon, \delta > 0$. By choosing c large enough, $t(s)/s^c$ can be made less than 1 and hence the probability of deviation from mean decreases as $1/2^{O(N)}$.

For each new record in A_v, we need $O(d)$ time to determine if it will be a bridge with respect to any of the d (maximum) neighbors. Moreover inserting a new bridge takes time proportional to gap-size whose expected value is also $O(d)$. To complete the analysis for the time bound for building the data-structure, we have to ensure that the total number of C-pointer updates is also $O(N)$. For any new record inserted into A_v from C_v, the total number of C-pointer updates can be bound by the total number of records (i.e. the space bound). The records of C_v are inserted in an increasing order and so any record in A_v has its C-pointer updated at most once (not including when it is first created). Hence we state the following result

Theorem 23 *Let G be a catalog graph of size N and locally bounded degree d. Our algorithm constructs a data-structure for iterative search in O(N) space and $O(dN)$ time to do iterative search in time $\tilde{O}(\log N + d|\Pi|)$. The bounds for preprocessing time and space hold with probability $1 - 2^{-O(N)}$.*

Remarks

1. The bounds for preprocessing time and search time can be improved to $O(N)$ and $\tilde{O}(\log N + \log d|\Pi|)$ respectively by using the same modifications as Chazelle-Guibas to restructure the catalog graph to a fixed degree graph.

2. The constant associated with the asymptotic bounds for preprocessing time is lower than the deterministic construction.

3. Chazelle-Guibas also arrived at the figure $4d$ for minimum gap size from the observation that otherwise their analysis yields infinite time and space bound for construction. However, they give

examples where the actual algorithm halts even when they use gap sizes of less than $4d$. Our analysis captures a more fundamental reason for this phenomenon. Although the mean $\mu < 1$, guarantees that the process is finite, the process dies out with probability x where $x = G(x)$ even when $\mu > 1$. So one can have a gap length of less than $4d$ and still terminate. The motivation for this is clearly a reduction in search time which is inversely proportional to the gap size.

4. To allow insertions/deletions from the catalogs, our procedure for maintaining the augmented catalogs readily dynamize. The arguments for query-time and the space-bound remain identical to the static case. Unfortunately (as in the case of [4]), the bottleneck is maintaining the correspondence between the A_v and C_v. In particular, we are unable to analyze the number of C-pointer updates in the case of inserting or deleting a record from C_v. Using the priority queue of Fries et al. [14] to maintain this correspondence, we get similar bounds. Both the search time and update times are off by an $O(\log\log N)$ factor from the best possible. For the special case of keys chosen randomly, this method leads to optimal dynamic algorithm (Sen [42]).

10. RANDOM SAMPLING AND POLLING IN COMPUTATIONAL GEOMETRY

Divide-and-conquer is certainly the most commonly used technique for designing parallel algorithms. The idea is analogous to sequential algorithm design where the original problem is sub-divided into smaller subproblems and then the solutions of the subproblems are combined to obtain a solution to the original problem. The smaller subproblems are solved recursively until a sub-problem size becomes smaller than a predetermined threshold. At this stage a direct (usually brute-force) method is used to solve it. For analyzing this procedure it usually suffices to write down the recurrence equation for the time complexity as:

$$T(n) = \begin{cases} \max_i T(n_i) + g(n) & \text{if } n_i > K \\ F(n) & \text{otherwise} \end{cases}$$

where K is a predetermined threshold, n_i is the size of the ith subproblem, F is the complexity of a direct algorithm and $g(n)$ is the cost of dividing the problem and recombining the solutions.

For a number of problems, $\sum_i n_i = n$ and hence it is the size of the largest subproblem (which is of size less than n) that is crucial for determining the running time of the algorithm. The equations for the processor and the space bounds can be written similarly, and the processor complexity is the maximum number of processors used at any step of the algorithm. Since there is a trade-off between the number of processors used and the running time, sometimes it becomes necessary to write a recurrence using two variables namely, the problem size and the number of processors used.

A generalization of the above procedure is to allow for *expected* bounds where it is possible to write down the recurrence equation for expected bound with respect to a specific resource. In the above equation for time bound, we can associate a distribution with the size of the largest subproblem, and the solution would be the expected running time of the algorithm. The exact distribution is often hard to ascertain and in most cases only the expectation is known. For some special forms, one can solve these *probabilistic recurrence relations* satisfactorily as discussed in section 2.

In the context of computational geometry, sorting can be looked upon as a one-dimensional problem. The basic tools that we will develop in this section will enable us to extend some of the techniques to higher-dimensional problems. In some sense, this exercise can be viewed as (although readers are cautioned against over-simplified conclusions) expanding the basic paradigm of quicksort-like algorithms. For the most of this section, the techniques discussed are very general without reference to any particular problem. We shall present some interesting applications in the next section to specific problems where algorithms will be presented more formally.

10.1 RANDOM SAMPLING

The divide-and-conquer mechanism that we will use is based on partitioning the problem using a random subset of the input. Recall that in quicksort, the input was partitioned by splitters that were randomly chosen elements of the input. For the higher-dimensional problems, it is not obvious how these splitters partition the input in the absence of a linear ordering.

Example 1 : For concreteness, consider the problem of constructing the *trapezoidal map* of a given set N of line segments. These segments partition the plane into regions which may have complicated shapes. By passing vertical lines through every end-point and intersection, these regions get partitioned into trapezoids or triangles (degenerate trapezoids). The vertical lines are not allowed to cross line segments. See Figure 1.1 for an illustration.

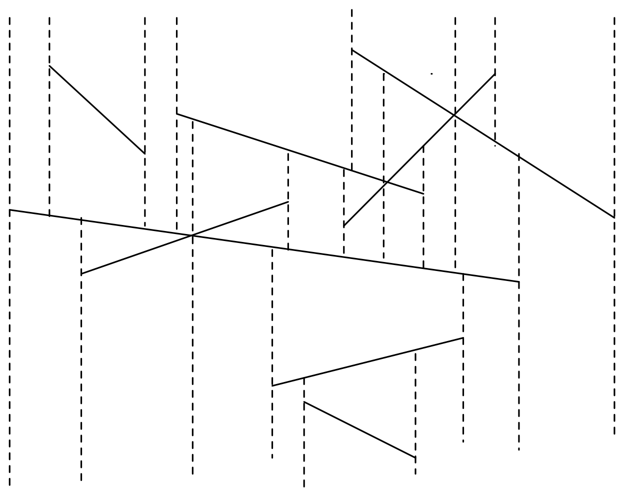


Figure 1.1 Trapezoidal map of a set of line segments

We will denote the trapezoidal map of a set S by $\mathcal{T}(S)$ and the set of trapezoids in $\mathcal{T}(S)$ as $H(S)$. By choosing a random subset of line segments $R \subset N$, it is not clear how to partition N as there is no natural ordering between segments (like points in a line). Since it is a two-dimensional problem, a natural solution is to consider the two-dimensional partitions induced by R. For example, consider $\mathcal{T}(R)$. For any trapezoid $\triangle \in H(R)$, consider $\mathcal{T}(N)$ restricted within $\triangle$. The union of $\triangle \cap \mathcal{T}(N)$ for all $\triangle \in \mathcal{T}(R)$ contains all the relevant information about $\mathcal{T}(N)$. Thus we can recursively compute construct $\mathcal{T}(N)$ within each $\triangle \in H(R)$. We will denote the line segments intersecting a trapezoid $\triangle \in H(R)$ by $L(\triangle)$ so that recursively we compute $\mathcal{T}(L(\triangle)$. From our earlier discussion, a bound on $\max\{L(\triangle\}$ will be crucial for the running time of the algorithm. In addition, the quantity $\sum_{\triangle} L(\triangle)$ is also important as a segment $s \in N$ can intersect many $\triangle$s implying that this quantity could exceed n. So it represents the *blow-up* in the overall problem size in a recursive call and would affect the overall efficiency of any recursive algorithm.

Remark 3 *In the case of quicksort, we did not have to worry about it since an element would belong to exactly one interval.*

Example 2: Consider the problem of computing the intersection of a set N of half-spaces in three dimensions. If we adopt the previous approach, we choose a random sample R of half-spaces and construct the intersection $\cap R$. For convenience, we assume it is non-empty and contains the origin in its interior. Unlike the previous case, we do not have regions analogous to the trapezoids. With some thought, it is not difficult to come up with sub-divisions analogous to trapezoids. For example, the pyramids formed by joining the origin to every vertex of the intersection. For technical reasons that will become clear later, we will like to have these regions, which we shall call *ranges* defined by a constant number of input objects or equivalently having constant size. A pyramid defined as above can have a fairly large base if the corresponding face (of $\cap R$ is large). By further subdividing the bases using parallel translates of a fixed plane, we can restrict the pyramid bases to be trapezoids (or triangles), so that these have constant size. The intersection $\cap N$ can be constructed recursively within these pyramids. In this context, $L(\triangle)$ denotes the set of half-spaces whose defining half-planes intersect a pyramid $\triangle$.

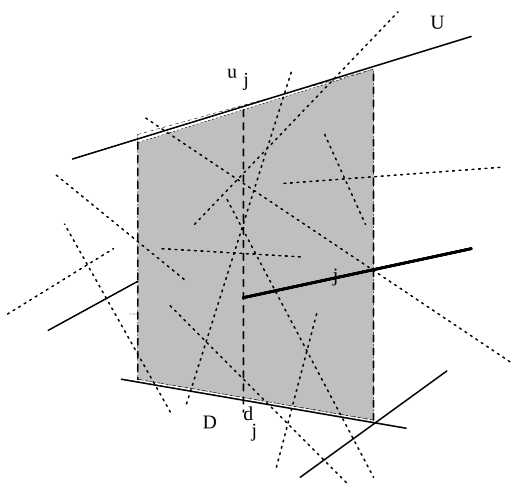
The previous examples would have given the reader some intuition about how random sampling gives rise to a natural class of divide-and-conquer algorithms in computational geometry. It is not difficult to come up with a suitable definition of *range* in the context of a given problem. To prove any interesting results about these algorithms, we have to bound the quantities $\max\{L(\triangle\}$ and $\sum_{\triangle} L(\triangle)$. The former is crucial to bound the depth of recursion, which determines parallel running time and the latter will determine the efficiency of this approach. We shall prove some useful bounds for these quantities for fairly general situations. From here, we will use $\triangle$ to denote a range in an abstract setting where a range is a subset of the Euclidean space E^d defined appropriately in the context of the problem in E^d. We will require that a range be defined by at most a constant number of input objects, say b which bounds the possible number of ranges by a polynomial in n, namely $O(n^b)$. In the case of trapezoidal maps, the reader can verify that b is 4. We will use $l(\triangle)$ to denote $|L(\triangle)|$ which will be referred to as the *conflict size* of $\triangle$. Note that we are interested in those ranges $\triangle$ where $\triangle$ is a range in $\mathcal{T}(R)$. The following result gives a bound on $l(\triangle)$ for a random sample R.

Lemma 7 ([7, 17]) *For some suitable constant k_1 and large n,*

$$\Pr[\max_{\triangle \in H(R)} l(\triangle) \geq k_1(n/r)\log r] \leq 1/4$$

where the probability is taken over all possible choices of the random sample R such that $|R| = r$.

Remark 4 *The lemma uses sampling in a slightly different manner than what we had discussed previously. However, it can be seen easily that*



t]

Figure 1.2 Segments intersecting $\triangle_j$

$|R|$ is $r + o(r)$ with high probability and the same bound would hold for a random sample of size r exactly. In the subsequent discussions, we will not distinguish between these sampling schemes.

The next result bounds the expected value of $\sum_{\triangle} l(\triangle)$.

Lemma 8 ([7, 35]) *For some suitable constant k and large n,*

$$\Pr[\sum_{\triangle \in H(R)} l(\triangle) \geq kn] \leq 1/4$$

where the probability is taken over all possible choices of the random sample R.

Lemma 9 ([7, 31]) *For $|N| = n$ and a random sample R obtained by choosing every element of N independently with probability r/n,*

$$E[\sum_{\triangle} l^c(\triangle)] = O(\left(\frac{n}{r}\right)^c \cdot E[|H(R)|])$$

where $E[|H(R)|]$ is the expected number of ranges in $\mathcal{T}(R)$ and c is a fixed positive integer (independent of n).

Reif and Sen[36] give an alternate proof. The above bound is useful in situations where the algorithm's performance can be expressed as $E[\sum_{\triangle} l^c(\triangle)]$.

10.2 CONVERTING EXPECTED BOUNDS TO HIGH-PROBABILITY

If the sampling lemma only guarantees a *good* sample with constant probability. The usual method used is that we repeat sampling till we obtain a good sample - the expected number is trials is $O(1)$ in our case. For this we must have a verification procedure $\mathcal{V}$. We can improve the probability of obtaining a good sample further by increasing the number of trials. That is, by choosing $p(n)$ independent sets of samples, we can be assured of a good sample with probability $1 - \frac{1}{2^{p(n)}}$ (at least one of the samples is good). The problem is clearly the extra cost of resampling, namely, we have to invoke the verification procedure $p(n)$ times. To avoid the extra cost, we can think about an *approximate verification* procedure that is much faster and gives us a very good estimate of how good a certain sample is. This is the basic philosophy of *polling*. Instead of testing the goodness of a sample with respect to the entire input (of size n), we actually 'poll' a small fraction of the input, which is typically about $\frac{1}{p(n)}$ - the exact value would depend on the desired success probability and the cost of $calV$. We will discuss some of the technical details in the context of random sampling in computational geometry.

As discussed in the beginning of this section, the probabilistic recurrence relation arising from a recursive randomized algorithm are often hard to solve without the knowledge of the tail distributions. The parallel algorithm's running time is the maximum of the resulting recursive calls unlike the sequential algorithm where it is a sum of the running times of the recursive procedures. The latter becomes easier to bound from the linearity of expectations. No such nice properties are known about the maximum of expected values and it does appear to depend on the tail estimates. This informal discussion is a motivation for the results obtained in this subsection; we do not claim that this is the only way to get around the problem of bounding maximum of expected values.

The naive random sampling gives us a high-probability bound on the maximum size of a subproblem (Lemma 7) and only an expected bound on the sum of subproblems (Lemma 9). We can restate these results in the following manner.

Lemma 10 *For suitable constants k_{total} and k_{max} the following conditions hold with probability at least 1/2 for a sample where each of the n input elements has been selected independently with probability r/n.*

(i) The maximum size of a subproblem is less than $k_{max}\frac{n}{r}\log n$
(ii) The sum of the subproblems is less than $k_{total}\frac{n}{r} \cdot E[|H(R)|]$.

Proof. From Lemma 8 and Markov's inequality we can choose k_{total} such that the probability that (ii) fails is at most 1/3 (i.e., sum of subproblems is thrice the expected value). Choose k_{max} such that failure probability in Lemma 7 is no more than $1/n$. For sufficiently large n, $1/n + 1/3$ is less than $1/2$. Thus the probability that both (i) and (ii) are satisfied is at least $1/2$. ∎

Remark 5 *We can make additional conditions like bounding the higher moments hold with probability at least 1/2 by the above argument. A sample that satisfies a required set of conditions will be called a good sample. In general we may assume that there are some (fixed) number of properties that a good sample must satisfy and a random sample is good with constant probability, say* $1/2$.

As a consequence of the previous claim, if we repeat the sampling $p(n)$ times, the probability that the conditions are not satisfied for all samples is less than $2^{-p(n)}$. That is, if we choose independently $p(n) = O(\log n)$ sets of samples, one of them is good with very high likelihood. We assume that $p(n) = \Omega(\log n)$ henceforth. Let us assume that we have a verification procedure $\mathcal{V}$ that verifies if a sample is good using $v(n)$ operations for n input elements. Therefore, to determine if a sample is 'good', we have to run $\mathcal{V}$ $O(\log n)$ times for a total of $v(n) \log n$ operations. Depending on $p(n)$ and $v(n)$, this could make an algorithm inefficient. To make resampling more efficient, we can run $\mathcal{V}$ using only a fraction of the input for each of the samples. For example, we can estimate the size of a subproblem by looking at a fraction of the input.

For exposition, we describe it for the special case of trapezoidal maps - for other problems the same arguments apply with minimal modifications, by replacing trapezoids with appropriate ranges.

We choose $c_0 \cdot \frac{n}{p(n) \cdot f}$ input segments randomly from the n input segments for some constant c_0 and a parameter f. (the actual values will be determined from the required success probability and the efficiency of the verification algorithm). Let X_i^j be the number of segments intersecting trapezoid $\triangle_i$ corresponding to sample R_j, $1 \leq j \leq p(n)$. A_i^j be the number of segments intersecting $\triangle_i$ out of the $c_0 \cdot \frac{n}{p(n) \cdot f}$ randomly chosen input segments for the same sample. Clearly, A_i^j is a binomial random variable with parameters $c_0 \cdot \frac{n}{p(n) \cdot f}$ (number of trials) X_i^j/n (probability of success).

Assuming that X_i^j is greater than $\bar{c} \cdot p(n) \cdot p(n) \cdot f$, for some constant $\bar{c}$, we can apply Chernoff bounds to estimate X_i^j within a constant multiplicative factor with probability exceeding $1 - \frac{1}{2^{p(n)}}$ (recall $p(n) = \Omega(\log n)$). Since we do it only for $\frac{1}{p(n) f}$ of the input segments, the total number of operations for the $p(n)$ random subsets can be bounded by $O(v(n/(p(n)\cdot f))\cdot p(n))$. In most cases, we want to ensure that this quantity is $O(v(n))$ and accordingly choose f.

When $X_i^j < \bar{c}\ p(n)\cdot p(n)\cdot f$, the estimates are not as accurate and we must ensure that this does not violate the desired properties of the sample. In the case of line segment intersections and the two properties of Lemma 10 , we must ensure that

$\bar{c}\ p(n)\cdot p(n)\cdot f \leq maxsize$, and
$\bar{c}\ p(n)\cdot p(n)\cdot f|H(R)| \leq maxsum$.

More formally, for the case $X_i^j < \bar{c}\ p(n)\cdot p(n)\cdot f$ (by invoking Chernoff bounds), for any $\alpha > 0$ (α is a function of c_0), there exists a c_1, independent of n,

$$\Pr[A_i^j \ \leq\ \alpha c_1 X_i^j/p(n)f)] \ \leq\ \frac{1}{2^{p(n)}}$$

and

$$\Pr[A_i^j \geq c_2\alpha c_0\cdot X_i^j/(p(n)f] < \frac{1}{2^{c_0 p(n)}} < \frac{1}{2^{p(n)}} \text{ for } c_0 > 1).$$

From the last two inequalities, X_i^j is bounded by $L^j = A_i^j p(n)\cdot f/c_0c_2\alpha$ from below, and by $U^j = A_i^j p(n)\cdot f/c_1\alpha$ from above. With appropriate choice of the constants, this condition holds with the desired probability (as defined in section 2.1) for all X_i^j simultaneously. In the context of trapezoidal map, we do the following procedure simultaneously for all the samples R_j and choose the sample R^{j_o} using the following simple test:

Procedure Polling

- *Input:* Samples $R_1\ldots R_m$ where $m = O(p(n))$.
- *Output:* A good sample R^{j_o}.
- *Notation:* Let $N^j = \sum A_i^j$ and the let actual number of intersections be denoted by T^j and the upper and lower bounds obtained from N^j by U^j and L^j respectively. We will use S to denote $k_{total}n/rE[|H(R)|]$.

If $S > U^j$ (clearly good) **then** accept sample R^j (since $S\ \geq\ U^j\ \geq\ T^j$),
if $S\ \leq\ L^j$ then the sample is 'bad' (since $S\ \leq\ L^j\ \leq\ T^j$),
if $L^j\ \leq\ S\ \leq\ U^j$, (choose the best) then accept the sample R^{j_o} for which N^{j_o} is minimum. Since both S and T^{j_o} lie in this interval this guarantees that $T^{j_o}\ \leq\ c_3\cdot S$ where $c_3 = U^j/L^j$ which is a constant.

Recall, that from our earlier discussion at least one of the samples would satisfy conditions 1 or 3 with probability $1 - O(\frac{1}{2^{p(n)}})$. Conditions 1 and 3 will fail if all the $p(n)$ are bad or if our estimates are inaccurate and the probability of either of the events is bounded by $\frac{1}{2^{p(n)}}$. We summarize as follows :

Lemma 11 (Polling lemma) *Using procedure* **Polling***, we can obtain a sample that is 'good' with probability* $1 - O(\frac{1}{2^{p(n)}})$ *where a naive random sample is known to be 'good' with probability 1/2. Given a verification procedure* $\mathcal{V}$ *that runs in* $O(v(n))$ *operations for* n *elements, this procedure uses* $O(v(n/p(n))p(n))$ *operations.*

For trapezoidal map we can obtain following kinds of bounds

- With probability $1 - \frac{1}{n}$, the maximum size of the subproblem is less than $O(n/rpolylog(n))$. Choose $p(n) = 2\log n$.
- With probability $1 - \frac{1}{2^{n^\delta}}$, $(\delta < 1)$ the maximum subproblem size is less than $O(n^{1+\epsilon}/r)$. Choose $p(n) = n^\delta$.

The sum of the subproblem sizes is $O(n)$ with probabilities $1 - \frac{1}{n}$ and $1 - \frac{1}{2^{n^\delta}}$ respectively.

In the following two sections we illustrate the use of Polling on two algorithms - a sequential iterative algorithm, and a parallel divide-and-conquer algorithm. We will obtain success probability $1 - 1/n$, although stronger bounds can be obtained as shown here.

11. AN OUTPUT SENSITIVE 3-D CONVEX-HULL ALGORITHM

The first known optimal $O(n \log h)$ time output-sensitive algorithm for three dimensional convex hull, where n and h are input and output sizes respectively was a randomized algorithm described in Clarkson and Shor [7]. The expected runing time is over the choice of random bits in the algorithm but did not have any associated tail estimates. Below we show that the use of Polling yields high-probability bounds for practically the same algorithm.

11.1 BRIEF OVERVIEW OF THE ALGORITHM FOR CONVEX HULLS

The problem of constructing the convex hull of points in three dimensions is well known to be equivalent to the problem of finding the *intersection of half-spaces*. Here we give an algorithm for the latter which implies a solution for the former.

Let us denote the input set of half-spaces by S and their intersection by $P(S)$. We construct the intersection $P(R)$ of a random sample R of r half-spaces and filter out the redundant half-spaces i.e. the half-spaces which do not contribute to $P(S)$. Without loss of generality, we can assume that the origin lies inside the intersection. Take an arbitrary (fixed) plane T and partition each face of $P(R)$ into trapezoids using the translates of T that pass through the vertices of the face. We further partition trapezoids into triangles. The convex closure of the origin O with a triangle from the cutting of the faces defines a region which we call the *cones*. These cones will be intersected by bounding planes of a number of half-spaces that were not chosen in the sample. We say that a half-space *intersects* a cone if its bounding plane intersects the cone. We say that a half-space *conflicts* with a cone if its bounding plane intersects the cone. A cone is said to be *critical* if it contains an output point. We delete the half-spaces which do not intersect with any critical cone. The procedure is repeated on the reduced problem.

To prove any interesting result we must determine how quickly the problem size decreases. The random sampling lemmata in the next section show that for a large sample ($> \Omega(h^2)$) the size of the problem decreases very quickly.

11.2 RANDOM SAMPLING PROPERTIES

Let $H(R)$ denote the set of cones induced by a sample R and let $H^*(R)$ denote the set of critical cones. We will denote the set of half-spaces intersecting a cone $\triangle \in H(R)$ by $L(\triangle)$ and its cardinality $|L(\triangle)|$ by $l(\triangle)$. $L(\triangle)$ will also be referred to as the *conflict list* of $\triangle$ and $l(\triangle)$, its *conflict size*. We will use Lemma 8 and Lemma 7 to bound the size of the reduced problem.

A sample is 'good' if it satisfies the properties of Lemma 8 and Lemma 7 simultaneously. Clearly, a sample is 'good' with probability at least 1/16. Using the method of Polling, we can do the following.

Lemma 12 *We can find a sample R which satisfies both Lemma 8 and Lemma 7 simultaneously with high probability. Moreover this can be done in $O(\log r)$ time and $O(n \log r)$ work with high probability.*

Since $|H^*(R)| \leq h$, a 'good' sample clearly satisfies the following property also

Lemma 13 *For a 'good' sample R,*

$$\sum_{\triangle \in H^*(R)} l(\triangle) = O(nh \log r / r)$$

where $|R| = r$ and $H^*(R)$ is the set of all cones that contain at least one output point.

This will be used repeatedly in the analysis to estimate the non-redundant half-spaces whenever $h \leq r/\log r$.

11.3 ALGORITHM

We give below an algorithm for convex hull and in the following section we analyse each step in details.

Let S be the input set of n half-spaces. The algorithm is iterative. Let n_i (respectively r_i) denote the size of the problem (respectively the sample size) at the i^{th} iteration with $n_1 = n$. A typical iteration of our algorithm is shown in Figure 1.3. Repeat the procedure until $r_i > n^\epsilon$ (this condition guarantees that the sample size is never too big) or $n_i < n^\epsilon$ for some fixed ϵ between 0 and 1. If $n_i < n^\epsilon$ then solve the problem directly else do one more iteration and solve the problem directly.

11.4 OVERVIEW OF ANALYSIS

Let l be the iteration when the sample size exceeds h^2 for the first time. The analysis is actually divided into two phases - work done before the l-th iteration and the work done after iteration l. The work done in Step 1 is bounded by $O(n_i \log r_i)$ with high probability from the polling procedure. Step 3a can be done using an intersection detection for every plane followed by finding all the regions that a half space intersects. These take time $O(n_i \log r_i)$ and $\sum \triangle \in H(R)|L(\triangle)|$ respectively. The second term is $O(n_i)$ from Lemma 8. The work done in Step 3b can be bounded by $O(\sum_{\triangle \in H(R)} |L(\triangle)| \log h_i^*)$ which is $O(n_i \log h_i^*)$ from the property of a good sample (Lemma 12). By our convention, $\log(h_i^*)$ is $O(\log h)$ for $i \leq l$. The remaining steps can be done in $O(n_i)$ time.

The running time of the algorithm is

$$\sum_{i<l} n_i(\log r_i + h_i^*) + \sum_{i \geq l} n_i(\log r_i + h)$$

with high probability. The high probability follows from the use of polling.

The first term can be bounded by $\sum_{i<l} n \log r_i^*$ where $r_i^* \geq (r_{i-1}^*)^2$ and $r_i^* \leq h^2$. Thus this term can be bounded by $O(n \log h)$ with high probability (a geometric sum with leading term $O(n \log h)$).

Procedure Rand(i)

1. Use Resampling and Polling to choose a 'good' sample R of size $r_i =$ constant for $i = 1$ and $\max\{r_{i-1}^2, h_{i-1}^*\}$ for $i > 1$ where h_i^* is the maximum output size of a 2d problem (defined later in step 3).

2. Solve the problem for R.

3. Define regions on the basis of the solution obtained in Step 2 — explained in Section 11.1. Let $H(R)$ denote the set of regions induced by R. We say that an input half-space is *redundant* if it does not contribute to the output. Filter out the redundant input as follows.

   (a) For every half space, find out the regions that it intersects.

   (b) Call a region *critical* if it contains an output. Denote the set of critical regions by $H^*(R)$. Find out the set $H^*(R)$ - these are regions that contain only parts of edges/faces of $P(S)$. This can be verified by computing the (two dimensional) polygonal intersection of the faces of $\triangle \in H(R)$ with S. An output sensitive algorithm (like [22, 2]) must be used for this step. The two dimensional polygonal structure of the intersections on the faces of $\triangle$ can be used to verify whether the region $\triangle$ contains any output vertex. The maximum number of vertices of any polygon defines h_i^* (which is a lower bound on the output size of $P(S)$.

   (c) Delete a half-space if it does not belong to $\cup_{\triangle \in H^*(R)} L(\triangle)$.

4. The input for the next iteration is $\cup_{\triangle \in H^*(R)} L(\triangle)$.

   Size of the reduced problem for the next iteration is $n_{i+1} = |\cup_{\triangle \in H^*(R)} L(\triangle)|$.

   Increment i.

end.

Figure 1.3

For the second term, we notice that $r_i \geq h^2$, and so we can simplify it to $\sum_{i \geq l} n_i \log r_i$. Moreover, for $i \geq l$, from Lemma 13

$$n_i \log r_i = O(\frac{n_{i-1}}{r_{i-1}} \log^2 r_i) \leq n_{i-1}/2$$

for $r_i \geq C$ for some constant C. So this term can be bounded by $O(n)$ as n_is decrease geometrically (from Lemma 13).

Thus the entire algorithm terminates in $O(n \log h)$ steps with high probability. The high probability can be improved further to inverse exponential probability by using a stronger tail estimate for polling. The original version of this algorithm described by Clarkson and Shor [7] could only guarantee expected bounds since they only relied on straightforward sampling.

Theorem 24 *Algorithm Rand(i) computes the convex hull of n points in $O(n \log h)$ steps with probability $1 - \frac{1}{2^{n^\epsilon}}$ for some $\epsilon > 0$.*

12. RANDOMIZED DIVIDE-AND-CONQUER

The tools and techniques developed in the previous two sections will be used to design a very general scheme for parallel divide-and-conquer. Random sampling will be used to achieve fairly even partitioning of the problem and the analysis will be done using various properties of random sampling proved earlier. We will illustrate the general methodology using the problem of computing trapezoidal map of line segments that can intersect only in the end-points. The parallel model that will be used is CREW unless otherwise mentioned.

12.1 TRAPEZOIDAL MAP CONSTRUCTION

This problem is the same as defined in previous section except that the segments are non-intersecting except possibly in the end-points. For every end-point, we want to determine the segment lying immediately above and below. This problem is called the *vertical visibility map* and is particularly interesting because of its close connection to triangulation. The algorithm we will describe will construct the visibility map of a given set N of line segments. A very high-level description is as follows:

Algorithm Vertical Visibility

1. Select a 'good' sample R of size $O(n^\epsilon)$ ($\epsilon > 0$ is a constant that will be determined in the analysis). By 'good' sample, we imply that the conditions of Lemma 10 hold. We use the technique of polling to do this step efficiently.

2. Construct $\mathcal{T}(R)$ using a brute-force approach.
3. For each segment of N, determine the trapezoids of $\mathcal{T}(R)$ that it intersects by a procedure we will describe shortly. (The same procedure will be used as the verification algorithm for Polling in step 1).
4. For each trapezoid $\triangle \in \mathcal{T}(R)$, we apply a clean-up phase called *Filtering* to discard some of the segments in $L(\triangle)$. As we will show later, this phase is crucial for bounding the processor complexity.
5. If $l(\triangle) > C$, for some predefined threshold C, then call the algorithm recursively on $L(\triangle)$ **else** solve the problem directly (We assume that a suitable algorithm already exists - usually a brute-force method suffices).

We now look at the individual steps in some details. The procedures in steps 1-3 are actually quite related. For $\epsilon < 1/2$, the following brute-force method works. For every endpoint, we draw a vertical line and order the segments intersecting this line (sorting in the Y direction of the intersections suffices which can be done in $O(\log n)$ time using n processors for every end-point). From this information, we can determine the segment lying immediate above and below every end-point. From this, we can also compute easily for each segment all the end-points for which it is visible from above (and below). In fact we order the vertical projections of these end-points by sorting. Constructing individual trapezoids can be done by 'walking' around the vertices of a trapezoid using the successor information from the previous computation. So the entire computation can be done in $O(\log n)$ time using $n^\epsilon \cdot n^\epsilon \leq n$ processors.

Partitioning the problem. To determine the segments intersecting a trapezoid, we use a *locus* based approach. This approach involves considering each query as a higher dimensional point and partitioning the underlying space into regions providing the same answer. Thus the query problem is reduced to a point location problem, given sufficient preprocessing time and space. The problem at hand involves preprocessing the trapezoidal subdivisions (induced by the sample) in such a manner that given the end-points of any segment we should be able to list the regions it intersects in $O(\log n)$ time using $\lceil k/\log n \rceil$ processors where k is the number of regions that it intersects. The preprocessing for n segments can be done in $O(\log n)$ time using $\mathrm{O}(n^c)$ processors, where c is a fixed constant. Thus any sample of size less than $n^{1/c}$ will suffice. See Reif and Sen [38] for details of this step which we omit from this discussion.

Clean-up for processor bound. Recall that a good sample implies the total sub-problem size is within a constant factor of $n/r \cdot E[|H(R)|]$. In this case (for non-intersecting segments), $H(R)$ is $O(|R|)$, i.e., the total subproblem size will be $\leq kn$ for some constant k. So, after recursion depth i, the total subproblem size can be bounded only by k^i. This algorithm has a recursion depth $O(\log\log n)$ and hence the total sub-problem size could become $\Omega(n\text{polylog}(n)$. This quantity is related to the number of processors; so we wish to bound it by $O(n)$. This is achieved by the following clean-up phase that we call *Filtering*

After partitioning the segments into the trapezoidal regions $\triangle$, we group the segments $L(\triangle)$ into two categories:

(a) $L^1(\triangle)$:Segments (part-segments) that have at least one end-point in the region
(b)$L^2(\triangle)$: Segments that span the region (horizontally)

Notice that number of segments of type (a) is less than $2n$ and $L^2(\triangle)$ in trapezoid $\triangle$, can be completely ordered (with respect to y-coordinate) within $\triangle$. So for the end-points of type $L^1(\triangle)$, a straight-forward binary search suffices to find the nearest segments among $L^2(\triangle)$ in the vertical direction. Consequently, we do not further preprocess the segments in $L^2(\triangle)$, i.e. we can leave them out from further recursive calls. Thus, the total size of the subproblems at any level of the recursive call is no more than $2n$. Processor allocation is achieved by simply allocating processors in a region equal to the number of end-points lying in it. This ensures that the algorithm does not require more than $O(n)$ processors for any step.

Analysis. The algorithm is recursive and we have shown that every step can be done in $\tilde{O}(\log n_i)$ steps for a subproblem of size n_i using n_i processors. The total processor requirement never exceeds n. The reader can verify that the time complexity T_i at depth i satisfies the preconditions of Theorem 26. Hence it follows that

Theorem 25 *Algorithm Vertical Visibility executes in $\tilde{O}(\log n)$ steps using n CREW processors.*

12.2 THE GENERAL STRATEGY

The algorithm developed for visibility maps in the previous section can be used for a number of other problems with some modifications required in the context of the problem. The following gives a bird's eye-view of this approach.

1. **Good-sampling** We use the method of Polling to choose a sample satisfying certain properties that holds with only constant probability for a naive sample.
2. **Partitioning** For computing the subproblems for a given sample. This step is also used in conjunction with the previous step for verifying

if a sample is good. The most common partitioning method used is doing point-location in a parameterized space.
3. **Filtering** Keep the total problem size at any stage of the algorithm within the given processor bounds by pruning individual subproblems before recursively solving them. This step is the most problem-dependent and depends on the geometric properties of the problem.
4. Solve sub-problems recursively if the size exceeds a certain threshold.

This approach has yielded optimal speed-up algorithms for many problems, the most notable being 3-D convex hull for which no matching deterministic algorithm is known.

We shall now prove the following useful result that gives tail estimates for a class of randomized divide and conquer algorithms :

Theorem 26 *Given a process-tree which has the property that a procedure at depth i from the root takes time T_i such that*

$$\Pr[T_i \quad \geq \quad kc\alpha \log n(\epsilon_0)^i] \leq \quad 2^{-(\epsilon_0)^i c\alpha \log n}$$

then, all the leaf-level procedures are completed in $\tilde{O}(\log n)$ time.

Proof. Setting $t_i = k(\epsilon_0)^i \log n\alpha(c - c_o)$, where c_o is some constant, we obtain

$$\Pr[T_i \quad \geq \quad k\alpha c(\epsilon_0)^i \log n \ + \ t_i\ ] \quad \leq \quad 2^{-(\epsilon_0)^i c\alpha \log n} \leq 2^{-t_i/k}.$$

If T is the total time for this worst case chain of nested calls and $m = 1/(1\text{-}\epsilon_0)$, the probability that T exceeds $mk\alpha \log nc_o + t$ is less than the sum of the probability of events where $\sum_i t_i = t$, $t > 0$, and $\mu = mk\alpha \log nc_o$. We shall compute the probability that $\sum_i t_i = t$.
$\prod_{\sum t_i = t} 2^{-t_i} \quad \leq \quad \sum 2^{-t/k}$ over $t^{O(\log\log n)}$ tuples.
Thus $\Pr[T > km\alpha \log nc_o + t] < 2^{-t/k + O(\log t \log\log n)}$.
Using $t \geq km\alpha(c - c_o)\log n$, for large values of n and $m > 1$, we can rewrite the above expression as

$$\Pr[T > km\alpha c \log n] < 2^{-\alpha(c-c_o)\log n}.$$

For $c > 4c_o$, i.e., $c - c_o > 3/4c$, we have the following required bound,

$$\Pr[T > \alpha logn] \leq 2^{-(3/4)c\alpha \log n} \leq n^{-c_1\alpha},$$

assuming that k, m, and c are larger than 1. ∎

Remark The proof of this theorem critically uses high probability tail estimates that is possible because of Polling. It is unlikely that any meaningful bound can be obtained only from expected bounds as we are trying to bound the *maximum* of the procedures spawned from any given node.

12.3 FURTHER READING

Random sampling was introduced into parallel computational geometry by Reif and Sen [39] (conference version) independently around the same time as the seminal papers of Clarkson [6, 8] and Haussler and Welzl [17]. All these papers primarily exploited the ε-net property, namely the (almost) even partitioning of the problem using a random subset. The subsequent papers of Clarkson [9] and Clarkson and Shor [7] refined the techniques considerably and developed very general and elegant techniques for designing geometric algorithms, namely, the randomized divide-and-conquer and randomized incremental construction. Mulmuley [29, 30], extended randomized incremental construction for dynamic settings and obtained impressive results for random updates. The textbook of Mulmuley [31] gives an excellent description of these general techniques. One of the most tantalizing open problems in this area, is to obtain concentration measure for the generic randomized incremental construction.

A number of very general properties of random-sampling were proved in Clarkson and Shor [7] that simplified or/and improved existing algorithms. The technique of Polling was first introduced by Reif and Sen [39] (conference version). It is an efficient method to find a sample with certain desirable properties that succeeds with high probability. In particular, the high-probability bounds are critical for divide-and-conquer based parallel algorithms. Even for many sequential algorithms based on Clarkson and Shor's paradigm, Polling can be used to bound the variance of the (otherwise only expected) running times without loss of efficiency as illustrated by the output-sensitive 3-D convex hull algorithm.

For a more detailed treatment of the basic parallel routines, the reader is referred to the textbook of Ja'Ja' [18], and Reif[37].

Notes

1. Note that all logarithms are to base 2 unless otherwise mentioned.

2. Jaikumar Radhakrishnan pointed out that one can redefine variables in the analysis of Algorithm P to make them independent; however, this is unnecessary since Theorem to be applied does not need independence. Moreover, in general, it may not always be possible to make such a redefinition of variables. But the general method will still apply.

3. Edgar Ramos and Jaikumar Radhakrishnan both pointed out an error in an earlier analysis, and the latter also independently developed the analysis given below.

4. A more recent algorithm of Karger improves this to $O(|E|polylog(n)$ using a more sophisticated Monte Carlo algorithm.

References

[1] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. *Proc. of the 35th Annual Symposium on Foundations of Computer Science*, pages 276 – 287, 1994.

[2] B. Bhattacharya and S. Sen. On a simple pactical output-sensitive randomized planar convex-hull algorithm. *Journal of Algorithms*, 25:177 – 193, 1997.

[3] B. Bollabas. *Graph Theory: An Introductory Course.* Springer Verlag, 1980.

[4] B. Chazelle and L. Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1:133 – 162, 1986.

[5] B. Chazelle and L. Guibas. Fractional cascading: Ii. applications. *Algorithmica*, 1:163 – 191, 1986.

[6] K. Clarkson. A randomized algorithm for closest point queries. *SIAM Journal on Computing*, 17:830–847, 1988.

[7] Kenneth L Clarkson and Peter W Shor. Applications of random sampling in computational geometry ii. *Discrete Comp. Geom.*, 4:387–421, 1989.

[8] K.L. Clarkson. New applications of random sampling in computational geometry. *Discrete and Computational Geometry*, pages 195 – 222, 1987.

[9] K.L. Clarkson. Applications of random sampling in computational geometry ii. *Proc of the 4th Annual ACM Symp on Computational Geometry*, pages 1 – 11, 1988.

[10] D. Dubhashi and S. Chaudhuri. Probabilistic recurrence relations revisited. *Theoretical Computer Science*, 181:45 – 56, 1997.

[11] D. Dubhashi, D. Grable, and A. Panconesi. Near optimal distributed edge colouring via the nibble method. *Theoretical Computer Science* Special ESA 95 Issue, 203:225 – 251, 1998.

[12] D. Dubhashi and D. Ranjan. Balls and bins: a study in negative dependence. *Random Structures and Algorithm*, 13:99 – 124, 1998.

[13] W. Feller. *An introduction to probability theory and applications, Volume I.* John Wiley, 1968.

[14] O. Fries, K. Mehlhorn, and S. Nåher. Dynamization of geometric data structures. In *Proceedings of the ACM Symp. on Computational Geometry*, pages 168 – 176, 1985.

[15] D. Grable and A. Panconesi. Near optimal distributed edge colouring in $o(\log \log n)$ rounds. *Random Structures and Algorithms*, 10:385 – 405, 1997.

[16] D. Grable and A. Panconesi. Brooks and vizing colorings. In *Proc of the SODA*, 1998.

[17] D. Haussler and E. Welzl. ϵ-nets and simplex range queries. *Discrete and Computational Geometry*, 2(2):127 – 152, 1987.

[18] Ja Ja Joseph. *An Introduction to Parallel Algorithms.* Addison-Wesley Publishing Company, 1992.

[19] D. Karger. Global min-cuts in rnc and other ramifications of a simple min-cut algorithm. In *Proc. of SODA*, pages 21 – 30, 1993.

[20] D. Karger and C. Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In *Proc. of the ACM STOC*, pages 757 – 765, 1993.

[21] R. Karp. Probabilistic recurrence relation. *J. ACM*, 41:1136 – 1150, 1994.

[22] D G Kirkpatrick and R Seidel. Output-size sensitive algorithms for finding maximal vectors. *Proc. of ACM Symp. on Computational Geometry*, 1985.

[23] M. Luby. 'a simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036 – 1053, 1986.

[24] M. Luby. Removing randomness in parallel without a processor penalty. *J. Computer and Systems Sciences*, 47:250 – 286, 1993.

[25] C. McDiarmid and R. Hayward. Large deviations for quicksort. *Journal of Algorithms*, 21:476 – 507, 1996.

[26] C.J. McDiarmid. On the method of bounded differences. *Surveys in Combinatorics, London Mathematical Society Lecture Notes Series*, 141, 1989.

[27] G. Miller and J. Reif. Parallel tree contraction and its applications. *Random Structures and Algorithm*, 5:47 – 72, 1989.

[28] M. Molloy and B. Reed. A bound on the strong chromatic index of a graph. *J. Comb. Theory (B)*, 69:103 – 109, 1997.

[29] K. Mulmuley. Randomized multidimensional search trees: Dynamic sampling. *Proc. of the 7th ACMSymp. on Computational Geometry*, pages 121–131, 1991.

[30] K. Mulmuley. Randomized multidimensional search trees: lazy balancing and dynamic shuffling. *Proc. of the 32nd IEEE Foundations of Computer Science*, pages 180–196, 1991.

[31] Ketan Mulmuley. *Computational Geometry : An Introduction through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[32] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the chernoff–hoeffding bounds. *SIAM J. Computing*, 26:350 – 368, 1997.

[33] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33:668 – 676, 1990.

[34] S. Rajasekaran and J. Reif. Optimal and sublogarithmic time radomized parallel sorting, algorithms. *SIAM Journal on Computing*, 18:594–607, 1989.

[35] S. Rajasekaran and S. Sen. *Random sampling Techniques and parallel algorithm design.* J.H. Reif editor. Morgan, Kaufman Publishers, 1993.

[36] J. Reif and S. Sen. *Parallel COmputtaional Geometry : An approach using randomization.* J. Sack and J. Urrutia eds. Elsevier, 1999.

[37] J.H. Reif, editor. *Synthesis of Parallel Algorithms.* Morgan, Kaufman Publishers, 1993.

[38] J.H. Reif and S. Sen. Optimal randomized parallel algorithms for computation al geometry. *Algorithmica*, 7(7):91 – 117, 1992.

[39] J.H. Reif and S. Sen. Optimal randomized parallel algorithms for computational geometry. *Algorithmica*, 7:91 – 117, 1992.

[40] J. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math.*, 8:223 – 250, 1995.

[41] S. Sen. Some observations on skip lists. *Information Processing Letters*, 39:173 – 176, 1991.

[42] S. Sen. Fractional cascading revisited. *Journal of Algorithms*, 19:161 – 172, 1995.

[43] J.M. Steele. *Probability Theory and Combinatorial Optimization.* CBMS-NSF Regional Conference Series in Applied Mathematics, 69 Society for Industrial and Applied Mathematics, 1997.