Experimental Evaluation of the Height of a Random Set of Points in a *d*-dimensional Cube

Eric Breimer, Mark Goldberg, Brian Kolstad, and Malik Magdon-Ismail

Computer Science Department
Rensselaer Polytechnic Institute
110 Eight Street, Troy NY 12180
{breime, goldberg, kolstb, magdon}@cs.rpi.edu

Abstract. We develop computationally feasible algorithms to numerically investigate the asymptotic behavior of the length $H_d(n)$ of a maximal chain (longest totally ordered subset) of a set of n points drawn from a uniform distribution on the d-dimensional unit cube $\mathbf{V}_d = [0,1]^d$. For $d \geq 2$, it is known that $c_d(n) = H_d(n)/n^{1/d}$ converges in probability to a constant $c_d < e$, with $\lim_{d\to\infty} c_d = e$. For $d = 2$, the problem has been extensively studied, and, it is known that $c_2 = 2$. Monte Carlo simulations coupled with the standard dynamic programming algorithm for obtaining the length of a maximal chain do not yield computationally feasible experiments. We show that $H_d(n)$ can be estimated by considering only the chains that are close to the diagonal of the cube and develop efficient algorithms for obtaining the maximal chain in this region of the cube. We use the improved algorithm together with a linearity conjecture regarding the asymptotic behavior of $c_d(n)$ to obtain even faster convergence to c_d. We present experimental simulations to demonstrate our results and produce new estimates of c_d for $d \in \{3, \ldots, 6\}$.

1 Introduction

Let $\mathbf{V}_d = [0,1]^d$ be the d-dimensional unit cube and let n random points $\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(n)$ be chosen independently from the uniform distribution on $\mathbf{V}_d$. These points form the underlying set of a *random order* $\mathbf{P}_d(n)$ with a partial ordering given by $\mathbf{x}(i) \leq \mathbf{x}(j)$ if and only if $\mathbf{x}_k(i) \leq \mathbf{x}_k(j)$ for all $k = 1, \ldots, d$. The number of elements in a longest chain (totally ordered subset) of $\mathbf{P}_d(n)$ is called the height of $\mathbf{P}_d(n)$. We are interested in the asymptotic behavior of the random variable $c_d(n) = H_d(n)/n^{1/d}$. The problem was intensively studied, experimentally and mathematically, for the case of $d = 2$ (see ([8], [15], [1], [9], [10], [13], [16], [2], [5], [12]). It was shown that $E[L_n]/\sqrt{n} \to 2$. The multidimensional case was initially considered by Steel in [14]. Bollobás and Winkler, [4], proved that $c_d(n)$ converges in probability to a constant c_d ($d \geq 0$). Except for $c_2 = 2$, no other c_d is currently known. Further, it is not even known whether or not the sequence $\{c_d\}$ is monotonically increasing in d. Using Monte Carlo simulation seems to be a natural approach to estimating c_d for $d > 2$. Unfortunately, it was discovered through experiments ([11]), that even for $d = 3$, the estimates

d	$d=2$	$d=3$	$d=4$	$d=5$	$d=6$
c_d	[1.998, 2.002]	[2.363, 2.366]	[2.514, 2.521]	[2.583, 2.589]	[2.607, 2.617]

Table 1. Estimates of c_d. The intervals indicate the estimated range of c_d.

for $c_d(n)$ do not converge for n up to 10^9. The standard dynamic programming algorithm for computing the length of a maximal chain is quadratic in n with memory requirement linear in n. Performing a sufficient number of Monte Carlo simulations for $n \approx 10^{10}$ is not feasible, even with advanced computational resources. The time and memory requirements are too great.

In this paper, we present an approach that addresses both the computational efficiency and the memory issues. Our estimates suggest that $\{c_d\}$ is a monotonically increasing sequence in d, a fact that has previously not been suggested by any experiments.

Our technique combines two ideas. First, generalizing the result from [4], we prove that a maximal chain "r-close" to the diagonal of the unit cube $\mathbf{V}_d = [0,1]^d$ must exist as $n \to \infty$. Second, we design an implementation of the dynamic programming algorithm for computing a longest r-close chain which runs in $O(rn^2)$ time, using $O(rn)$ memory. The parameter $0 < r << 1$ can be decreased to boost the efficiency of the algorithm for very large values of n. The algorithm's efficiency resulted from the fusion of the input generation and the computation of the longest r-close chain into one procedure where only $O(nr)$ points are "at work" at every moment of the computation. The reduction of the running time and the memory allows us to perform a sufficient number of Monte Carlo simulations. Based upon geometrical considerations, we *conjecture* that the maximal chains for different values of r converge at essentially the same rate. The simulations strongly support the conjecture, which together with the data yield the estimates shown in table 1.

The outline of the paper is as follows. First we describes two approaches to the design of experiments and then, we present experimental simulations demonstrating our algorithms. For proofs and other technical details, the reader is referred to the technical report, [6].

Acknowledgment. We are grateful to Catherine McGeogh for introducing us to the problem and sharing the information about her experiments.

2 Location of Maximal Chains

Since a longest chain is only a very small subset of the set of n random points in $\mathbf{V}_d(r)$, we can expect to gain an advantage over the straightforward simulation by restricting our search for such a chain to a small region where we are likely to find the chain. Let $\mathbf{V}_d(r)$ be the region of $\mathbf{V}_d = [0,1]^d$ obtained by translating a cube of side r along the diagonal as shown in Figure 1. The volume $V_d(r)$ of $\mathbf{V}_d(r)$ is given by $V_d(r) = r^d + dr^{d-1}(1-r)$. Let n points be sampled uniformly in $\mathbf{V}_d(r)$ to form a random partially ordered set $\mathbf{P}_d(n,r)$. Denote the height of

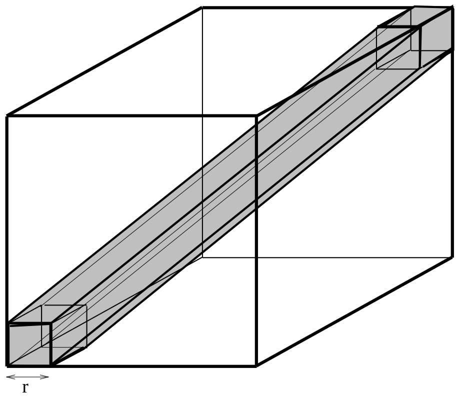


Fig. 1. Diagonal volume element obtained by translation of a cube of side r

this set by $H_d(n,r)$ and define the random variable $c_d(n,r)$ by

$$c_d(n,r) = \frac{H_d(n,r)}{(n/V_d(r))^{1/d}} \tag{1}$$

Intuitively, we expect that if one were to generate $n^* = \lceil n/V_d(r) \rceil$ points in the whole cube $\mathbf{V}_d$, then about n of them would fall in $\mathbf{V}_d(r)$. If n^* is large, we expect a maximal chain to exist in $\mathbf{V}_d(r)$. A formal statement to this effect can be proved, namely that for every $r > 0$, $c_d(n,r)$ converges to c_d. For more details, the reader is referred to [6]. To compute $c_d(n,r)$, there is no need to generate points in the whole cube: we only need to generate points in this diagonal region in a way that would be consistent with having generated a larger number of points in the entire cube. Thus, for a fixed $r > 0$, a Monte Carlo simulation to determine $\overline{c}_d(n,r)$ can be used to suggest a value for c_d, provided n is large. One might hope that the experiment with a large and infeasible n^* that yields a good estimate for c_d would be equivalent to an experiment with a feasible $n \approx n^* V_d(r)$, for some $r > 0$. Thus, our general approach will be to estimate $E[c_d(n,r)]$ for various n and r by Monte Carlo simulation, and use these values to construct a final estimate for c_d.

3 Boosting

Not every combination of n and r is computationally equivalent to an $n^* = \lceil n/V_d(r) \rceil$. In fact, for fixed n, as $r \to 0$, $c_d(n,r) \to (rn)^{1-1/d} d^{1/d}$, which approaches zero. This simply indicates that r is too small for the given n, and it is unlikely that a maximal chain of a random set with n^* points in $\mathbf{V}_d$ is located inside $\mathbf{V}_d(r)$. On the other hand, if n is fixed and r is sufficiently large, but less than 1, then it is likely that about n out of n^* points fall in $\mathbf{V}_d(r)$ and a maximal chain exists within even a smaller distance of the diagonal than r. These observations suggest that given n, there exists an optimal $r = r_{opt}(n)$, which yields[1] a maximal value for $c_d(n, r_{opt})$. In the next section, we present an algorithm for computing $c_d(n, r)$ which requires $O(rnd)$ memory and runs in $O(rn^2d)$ time. Thus, it is of practical importance to select as small an r as possible for which $c_d(n, r)$ approximates c_d. We are thus led to the following experiment

Experiment 1 (Boosting):

1. Select a maximal feasible n.
2. Set $r = 1$, and obtain an estimate for $\bar{c}_d(n, r)$.
3. Decrease r ($r \leftarrow r - \epsilon$, for a suitably chosen ϵ) and re-estimate $c_d(n, r)$. Repeat until $\bar{c}_d(n, r)$ stops increasing.
4. Determine $r_{opt}(n)$, the r after which $\bar{c}_d(n, r)$ started to decrease.
5. If $r_{opt}(n)n^2d$ is much less than the available resources, increase n ($n \leftarrow n + n_0$, for a suitably chosen n_0). Compute $\bar{c}_d(n, r)$. Go back to step 3.
6. Once a maximal n has been reached, and its associated optimal r, $r_{opt}(n)$ has been determined, compute $\bar{c}_d(n, r_{opt})$ to the desired accuracy. Output this value as the estimate for c_d.

As the name of Experiment 1 suggests, given a maximal n for the standard dynamic programming algorithm (determined by computational resources), we can *boost* it up to a higher n^* by going to a smaller r_{opt} and, perhaps, further increasing n as described in Experiment 1, while still using the same amount of computational resources. Our results demonstrate that boosting gives considerable improvement over the straightforward Monte Carlo. We can obtain even faster convergence to c_d by combining our boosting algorithms with a co-convergence conjecture. We discuss this next.

4 Co-convergence

Another approach to estimating c_d is to select a set $\{r_i\}_{i=1}^{T}$ of values for r and consider T sequences $\{\bar{c}_d(n, r_i)\}$ for $i = 1 \ldots T$. Each of these sequences converges to the same limit c_d as $n \to \infty$. The problem now is one of estimating the common limit of these T sequences. Suppose $\bar{c}_d(n)$ ($= c_d(n, 1)$) converges at some rate $f(n)$ to c_d: $\bar{c}_d(n) = c_d - f(n)$. Based on geometrical considerations, it is reasonable to expect that the convergence is also governed by $f(n)$ and, thus, we are lead to *conjecture* that the order of convergence of $c_d(n, r)$ for all $r > 0$ is essentially the same, i.e., there exists a function $\mu(r) > 0$, such that

$$\forall r > 0,\ \bar{c}_d(n, r) = c_d - \mu(r) f(n) + o(f(n)) \tag{2}$$

The goal of Experiment 1 was to obtain the smallest r, namely $r_{opt}(n)$, for which $\mu(r_{opt})$ is a minimum. The conjecture suggests a strong interdependence between the sequences, and we might be able to exploit this interdependence in order to

[1] In fact, our experiments suggest that $c_d(n, r)$ has a single maximum as a function of r.

get a more accurate estimate of c_d. The traditional approach to obtaining the convergence point of the sequence $c_d(n)$ would be to assume that $f(n)$ has a certain form and then obtain a value for c_d consistent with this assumption and with the observed values $\bar{c}_d(n)$. The success of this kind of an approach depends largely on the validity of the assumption on $f(n)$. Our conjecture allows us to estimate c_d without estimating $f(n)$. Given r_i, r_j ($r_i \neq r_j$), and n, (2) implies that the following two equalities hold simultaneously

$$\begin{cases} \bar{c}_d(n, r_i) = c_d - \mu(r_i) f(n) + o(f(n)) \\ \bar{c}_d(n, r_j) = c_d - \mu(r_j) f(n) + o(f(n)) \end{cases}$$

Resolving this system with respect to $f(n)$, we have

$$\bar{c}_d(n, r_i) = (1 - A(r_i, r_j))\, c_d + A(r_i, r_j) \bar{c}_d(n, r_j) + o(f(n)). \tag{3}$$

where $A(r_i, r_j) = \mu(r_i)/\mu(r_j)$. In other words, $\forall i, j \in [1, T]$ $\bar{c}_d(n, r_i)$ linearly depends on $\bar{c}_d(n, r_j)$, up to $o(f(n))$. Further, from the functions $\bar{c}_d(n, r_i)$ and $\bar{c}_d(n, r_j)$, one can estimate $A(r_i, r_j)$ from the slope of this dependence, and $(1 - A(r_i, r_j))\, c_d$ from the intercept. Using these estimates, we can construct an estimate for c_d *without* having to make any statements about $f(n)$. From (3), we see that the convergence to linear behavior is at a rate $o(f(n))$, whereas the convergence of the sequences themselves is at the rate $f(n)$. Thus, the linear behavior will materialize at smaller n than the actual convergence. Hence, we expect to extract more accurate estimates for c_d in this way, given the computational resources. We are thus led to the following experiment.

Experiment 2 (Co-convergence):

1. Select a set $\mathcal{T} = \{r_1, \ldots, r_T\}$ of values for r.
2. Select a set $\mathcal{N} = \{n_1, \ldots, n_L\}$ of values for n. Let $\mathcal{N}_h = \{n_1, \ldots, n_h\}$ for $h = 1, \ldots, L$.
3. Compute $\bar{c}_d(n, r)$, $\forall r \in \mathcal{T}$ and $\forall n \in N$.
4. $\forall i, j \in [1, T]$ ($i \neq j$), perform an analysis on the pair of sequences $\{\bar{c}_d(n, r_i)\}$ and $\{\bar{c}_d(n, r_j)\}$ for $n \in \mathcal{N}_h$ to obtain the slope, $A_d(h, i, j)$, and the intercept, $B_d(h, i, j)$, for $h = 2, \ldots, L$.
5. Evaluate $c_d(h, i, j) = B_d(h, i, j)/(1 - A_d(h, i, j))$ for $h = 1, \ldots, L$.

   /*$c_d(h, i, j)$ is a sequence of estimates for c_d; this sequence should converge to c_d at the rate at which the linear behavior arises, *i.e.*, $o(f(n))$. */
6. Compute $e_d(i, j)$, the value to which $c_d(h, i, j)$ converges with respect to h.
7. Repeat steps 4–6 to compute $e_d(i, j)$ for all $\binom{T}{2}$ distinct pairs $i, j \in \mathcal{T}$.
8. Let $L_d = \min_{i,j \in \mathcal{T}} e_d(i, j)$ and let $U_d = \max_{i,j \in \mathcal{T}} e_d(i, j)$.
9. Output the interval $[L_d, U_d]$ as an estimate for c_d.

5 Algorithms

Both experimental approaches described in the previous section attempt to get more accurate estimates for c_d by effectively "accessing" higher n without actually computing with the higher n. Both techniques rely on efficient algorithms for computing $c_d(n, r)$.

The first task is to generate n points chosen independently from a uniform distribution in $\mathbf{V}_d(r)$. A trivial algorithm is the one that generates random points in $\mathbf{V}_d$ and keeps only those that fall in $\mathbf{V}_d(r)$, continuing until n points in $\mathbf{V}_d(r)$ have been generated. This can be highly inefficient, especially if r is small, as the acceptance rate will be extremely small. A more efficient approach is to generate random points in $\mathbf{V}_d(r)$ itself. This can be done quite efficiently, and further, it is possible to generate the points in a sequential manner so that the dynamic programming algorithm for computing $H_d(n, r)$ keeps in memory only a small portion of the total of n points, those that are necessary for executing the algorithm.

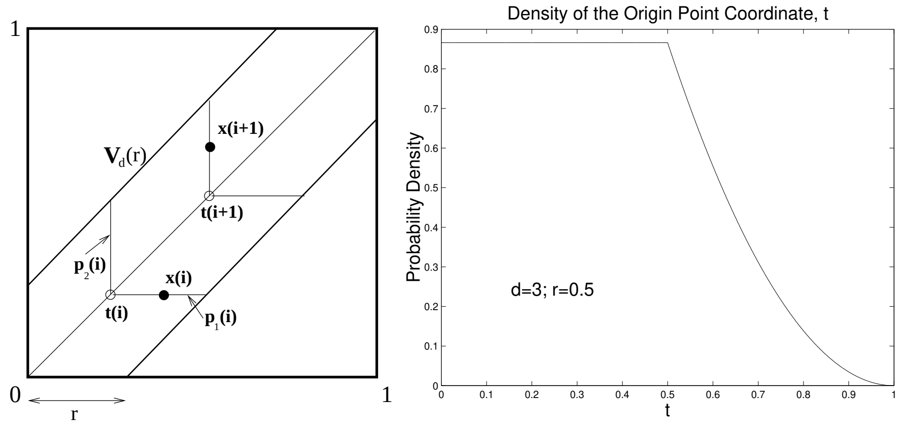
The second task is to compute $H_d(n, r)$. The standard dynamic programming algorithm has computational complexity $O(dn^2)$ [7]. However, since the points are generated in a sequential manner, it is possible to design an algorithm that maintains a *working set* that takes advantage of the sequential point generation. Operations need only be performed on this working set, resulting in a factor of r reduction in both computational complexity and memory requirements.

5.1 Generation of Input Points

The generation of points is illustrated in Figure 2 (a), for the case of $d = 2$. First, an "origin" point, $\mathbf{t}(i) = (t(i), t(i), \ldots, t(i))$, is generated along the diagonal. Then one of the hypercubes $\mathbf{p}_k(i)$, $k = 1, \ldots, d$ of dimension $d - 1$ is chosen at random and a point is generated from a uniform distribution on this hypercube. In this way one can generate n points in $\mathbf{V}_d(r)$. The probability density of the origin point coordinate $t(i)$ is uniform up to $t(i) = 1 - r$ and then decaying like $(1 - t(i))^{d-1}$ for $t(i) > 1 - r$. This density is shown in Figure 2 (b), for the case $d = 3$ and the case $r = 0.5$. Further, instead of generating the origin points in a random order, we generate the n order statistics for the origin points.

Algorithm 1 (Sequential generation of input points):

1. Set $i = 1$ and $t_{prev} = 0$.
2. Generate the i^{th} origin point $\mathbf{t}(i) = (t(i), t(i), \ldots, t(i))$ where $t(i) \in [t_{prev}, 1]$ is the i^{th} order statistic (given that the $(i-1)^{th}$ order statistic was at t_{prev}) of n points generated from the distribution of origin points such as the one shown in Figure 2.
3. Generate a random vector $=\{v_1, v_2, \ldots, v_d\}$ where each v_i is in $[0, \min\{r, 1-t_i\}]$
4. Generate a random integer k from $\{1, 2, \ldots, d\}$ and set $v_k = 0$
5. Generate input point $\mathbf{x}(i) = \{t(i) + v_1, t(i) + v_2, \ldots, t(i) + v_d\}$.
6. Set $t_{prev} = t(i)$, $i = i + 1$ and go back to step 2 if $i \leq n$.



(a) (b)

Fig. 2. (a) The sequential generation of points in $\mathbf{V}_d(r)$ illustrated for the case $d = 2$. (b) Probability density of origin points for d=3, r=0.5

The detailed implementation of step 2 to generate the order statistics for the origin points will be presented in the final paper. The next proposition states two (almost trivial) ordering properties that the points $\mathbf{x}(1), \ldots, \mathbf{x}(n)$ have. While these properties might seem trivial, they are of key importance to the algorithm for computing the height of the n points in $\mathbf{V}_d(r)$, as we shall see in the next section.

Proposition 1 (Ordering Properties). *Suppose points* $\mathbf{x}(1), \ldots, \mathbf{x}(n)$ *are generated according to the algorithm above. Then,*

1. *If* $\mathbf{t}(i) \geq \mathbf{x}(j)$, *then, for all* $k \geq i$, $\mathbf{x}(k) \geq \mathbf{x}(j)$.
2. *If* $i < j$, *it cannot be that* $\mathbf{x}(i) \geq \mathbf{x}(j)$, *more specifically,* $P[\mathbf{x}(i) \geq \mathbf{x}(j)] = 0$.

5.2 Computing $H_d(n, r)$

Although the ordering of the origin points does not guarantee that the projected input points $\{\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(n)\}$ are ordered, it does guarantee the ordering properties stated in proposition 1. Property 2 ensures a newly generated point can never be below a previously generated point. Therefore, the height of newly generated points can be immediately computed without considering future points. At iteration i, property 1 ensures $\mathbf{x}(i)$ has a height of at least $h(\mathbf{x}(j)) + 1$ for all $\mathbf{x}(j) < \mathbf{t}(i)$. Among these input points, the $\mathbf{x}(j)$ with maximum height is identified as the *barrier point* and the rest are discarded. The remaining input points (all $\mathbf{x}(j) \geq \mathbf{t}(i)$ plus the *barrier point*) define the *working set* for iteration $i + 1$. The height of $\mathbf{x}(i + 1)$ can be determined by inspecting only the points in the *working set*.

$H_d(n, r)$ is computed using the following procedure where t and x are the origin point and projected input point as generated by sequential generator described above. $h(x)$ is the height of the maximal chain ending at point x, and $x_{barrier}$ is the *barrier point*.

Algorithm 2 (Computation of $H_d(n, r)$):

1. Set $h_{max} = 1$.
2. Using Algorithm 1, generate $t(1)$ and $\mathbf{x}(1)$ and add $\mathbf{x}(1)$ to the *working set*.
3. Set the height of $\mathbf{x}(1)$ to 1 ($h(\mathbf{x}(1)) = 1$).
4. For $i = 2$ to n
   (a) Using Algorithm 1, generate $t(i)$ and $\mathbf{x}(i)$.
   (b) Initialize the height of $\mathbf{x}(i)$ to 1 ($h(\mathbf{x}(i)) = 1$).
   (c) For every point $\mathbf{x}(j)$ in the *working set*
      i. If $\mathbf{x}(i)$ is above $\mathbf{x}(j)$ and $h(\mathbf{x}(j)) >= h(\mathbf{x}(i))$ then set $h(\mathbf{x}(i)) = h(\mathbf{x}(j)) + 1$.
      ii. If $\mathbf{x}(j)$ is below $t(i)$, if the *barrier point* is defined, and if $h(\mathbf{x}(j)) > h(x_{barrier})$, then remove the current *barrier point* and replace it with $\mathbf{x}(j)$.
      iii. If $h(\mathbf{x}(j)) <= h(x_{barrier})$ then remove $\mathbf{x}(j)$ from the *working set*
      iv. If $\mathbf{x}(j)$ is below $t(i)$ and the *barrier point* is not defined then set $\mathbf{x}(j)$ to be the *barrier point*.
   (d) Add $\mathbf{x}(i)$ to the *working set*.
   (e) If $h(\mathbf{x}(i)) > h_{max}$ then set $h_{max} = h(\mathbf{x}(i))$.
5. Return $h_{max}/n^{1/d}$.

The algorithm computes the height of each $\mathbf{x}(i)$ as the input points are generated. $\mathbf{x}(i)$ is given a default height of 1. Every input point forms a chain of height 1 with itself as the beginning and end points. The new input point is compared with every $\mathbf{x}(j)$ in the *working set* to find the highest chain which can be continued by $\mathbf{x}(i)$. Each $\mathbf{x}(j)$ is also compared with $\mathbf{t}(i)$ to determine if it is below the *barrier*. If $\mathbf{x}(j)$ is below the *barrier*, it either is removed from the working set or (in the case where $h(\mathbf{x}(j)) > h(x_{barrier})$) replaces the current *barrier point*. After iteration i, the *working set* includes only the *barrier point* and the points that are not below the *Barrier* $t(i)$ (see Figure 3). This defines a subset of the $\mathbf{V}_d(r)$ which has a volume $V_d(r)r$. Since n points are uniformly distributed in $\mathbf{V}_d(r)$, it is expected that the *working set* contains rn points. Since the algorithm must iterate over the *working set* for every newly generated point, the expected number of point comparisons is $O(rn^2)$. Therefore, using a very small r not only increases the number of virtual points n^* but decreases the memory requirements and the number of computations by a factor of r. The proof that this algorithm is correct can be found in [6].

6 Experiments

We have performed computer simulations to support the claims that boosting provides a considerable efficiency gain over straightforward Monte Carlo, and

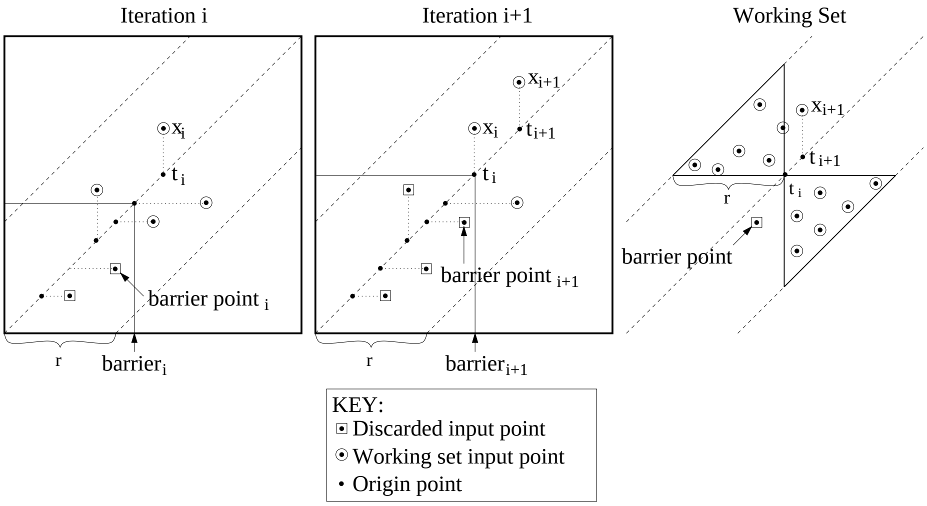


Fig. 3. The *working set* at iteration i and $i+1$

that the co-convergence experiment coupled with the algorithms for boosting lead to rapidly converging estimates for $\{c_d(n,r)\}$. We present results for the case $d = 3$ here. We have also performed experiments for other dimensions, $d = 2, 4, 5,$ and 6, with similar results.

Our first series of simulations used the values for n and r given in the next table to compute $\bar{c}_d(n,r)$.

$$n \in \{50000, 100000, 200000, 300000\}$$

$$r \in \{0.01, 0.02, 0.06, 0.08, 0.10, 0.14, 0.20, 0.30, 1.00\}$$

The dependence of $c_d(n,r)$ on r for various n is shown in Figure 4. The larger the estimate, the closer it is to the true value. It appears that the radius that maximizes $c_d(n,r)$, $r_{opt} \approx 0.08$. With $r = 1$, to obtain an estimate comparable to using r_{opt} and $n = 50000$, one needs $n \geq 300000$ which represents a gain in efficiency by a factor of over 450 in favor of boosting. Further, figure 4 shows that the maximum value of$c_d(r,n)$ is achieved using roughly the same value of r for 4 different values of n. The data indicates that r_{opt} is independent of n which supports the fact that $\mu(r)$ is independent of n, as required by the conjecture.

Our second set of simulations tests co-convergence (according to Experiment 3). We selected n and r according to the following scheme.

$$n_k = \lfloor 10^{2+i_k \times 0.1} \rfloor \text{ for } k \in [0, 40], \text{ and } r \in \{0.01, 0.005, 0.001, 0.0005, 0.0001\} \tag{4}$$

Figure 5(a) demonstrates the convergence behavior of $c_3(n,r)$, for the five values of r. Even for $n = 10^6$, it is not clear to what value these sequences are converging. Given $c_d(n, r_i)$ and $c_d(n, r_j)$ for $n = n_1, \ldots, n_h$, we can estimate

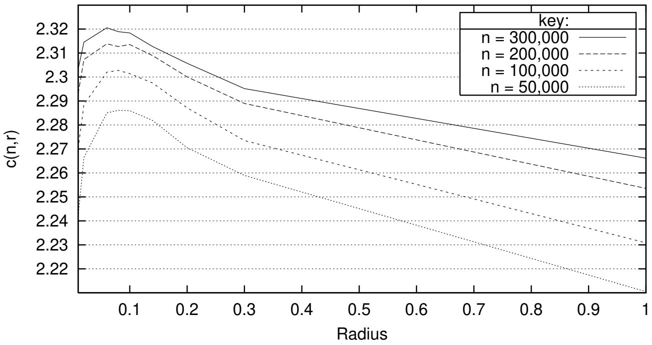
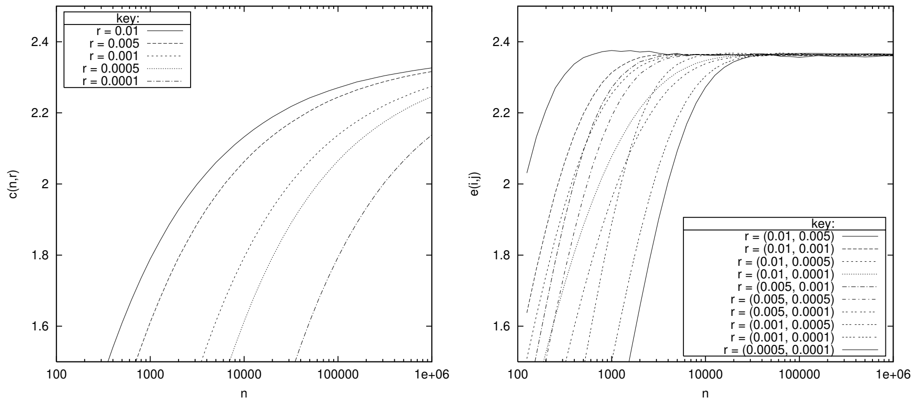


Fig. 4. Dependence of $\overline{c}_d(n, r)$ on r for various n's.



(a) (b)

Fig. 5. (a) Convergence of $\overline{c}_3(n, r)$. (b) Convergence of and $e_3(i, j)$

$A_d(h, i, j)$ and $B_d(h, i, j)$ for $h = 2, \ldots, L$ as in Experiment 3. The estimate given by $B(h, i, j)/(1 - A(h, i, j))$ should converge with respect to h to c_d, independent of which particular pair i, j is used. The behavior of our estimate for every pair of radii is demonstrated in figure 5 (b). It is clear that the curves are all converging to the same value. Further, as expected, the convergence occurs earlier, as compared with the convergence of $c_3(n, r)$ in figure 5 (a), in accordance with the expected $o(f(n))$ behavior of this estimate. We propose a range for c_3 by taking the range of values to which these curves are converging. Similar results have been obtained for the cases $d = 2, 4, 5, 6$, and the estimates for c_d were presented earlier.

References

1. R. M. Baer and P. Brook, *Natural sorting over permutation spaces*, Mat. Comp., Vol. 22 (1968) 385-510
2. J. Baik, P. Deift, and K. Johansonn, *On the distribution of the length of the longest increasing subsequence of random permutations*, J. of the Amer. Math. Soc., Vol. 12, No. 4 (1999), 1119-1178
3. P. Billingsley, *Probability and Measure*, Wiley Series in Probability and Mathematical Statistics, Wiley (1986)
4. B. Bollobás and P. Winkler, *The longest chain among random points in Euclidean space*, Proceedings of the Amer. Math. Soc., Vol. 103, No. 2 (1988) 347-353
5. A. Borodin, *Longest increasing subsequences of random colored permutations*, Electr. J. Comb., Vol. 6, No. 1, R13 (1999)
6. E. Breimer, M. Goldberg, B. Kolstad and M. Magdon-Ismail, *Experimental evaluation of the height of a random set of points in a d-dimensional cube*, Technical Report, Dept. of Computer Science, RPI, Troy, NY 12180, TR # 00-6 (2000)
7. T. Cormen, C. Leiserson, R. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusets; London, England (1990)
8. P. Erdös and G. Szekeres, *A combinatorial problem in geometry*, Compositio Math, Vol. 2 (1935) 463-470
9. J. M. Hammersley, *A few seedlings of research*, Proc. Sixth Berkley Sympo. Math. Stat. Prob., Univ. of California Press, Berkeley, Calif. (1972) 345-394
10. B. Logan and L. Shepp, *A variational problem for random Young tableaux*, Adv. in Math., Vol. 26 (1977) 206 - 222
11. Catherine McGeoch, (*private communication*)
12. A. M. Odlyzko and E.M. Rains, *On longest increasing subsequences in random permutations*, in *Analysis, Geometry, Number Theory: The Mathematics of Leon Ehrenpreis*, E. Grinberg, S. Berhanu, M. Knopp, and G. Mendoza (eds.), Amer. Math. Soc. (to appear 2000)
13. S. Pilpel, *Descending Subsequences of Random Permutations*, J. of Combinatorial Theory, Vol. A 53, No. 1 (1990) 96 - 116
14. J. M. Steele, *Limit properies of random variables associated with a partial order in* $\Re^d$, Ann. Probab., Vol. 5 (1977) 395-403
15. S. M. Ulam, *Monte Carlo calculations in problems of mathematical physics*, in *Modern Mathematics for the Engineer: Second Series*, McGraw-Hill (1961) 261-281
16. A. M. Verśik and S. V. Kerov, *Asymptotics of the Plancherel measure of the symmetric group and the limiting form of Young tableaux*, Dokl. Acad. Nauk SSSR, Vol. 233 (1977) 1024-1028