On Generating Random Permutations with Arbitrary Distributions†

B. J. OOMMEN AND D. T. H. NG
School of Computer Science, Carleton University, Ottawa: K1S 5B6, Canada

Let $\mathfrak{R} = \{R_1, R_2, \ldots, R_M\}$ *be an ordered set of M elements where* $R_i < R_j$ *whenever* $i < j$. *Let* π *be the set of permutations of* $\mathfrak{R}$. *We consider the problem of randomly generating the elements of* π *according to a distribution* $G(\pi)$. *Various algorithms including those due to Durstenfeld*[3,5] *and Moses et al*[7] *are available for the case when the distribution* $G(\pi)$ *is a uniform distribution (i.e., where all the elements of* π *are generated with equal probability). In this paper we consider the case when the distribution* $G(\pi)$ *is not necessarily uniform. We present a strategy for specifying the distribution* $G(\pi)$ *and propose a technique for generating the elements of* π *according to the distribution* $G(\pi)$. *Applications of the technique to generate 'almost sorted lists' and in the Travelling Salesman Problem have been presented. Finally, simulation results have been included which demonstrate the power of the Random Permutation Generation (RPG) technique.*

Received October 1988, *revised February* 1989

1. INTRODUCTION

Let $\mathfrak{R} = \{R_1, R_2, \ldots, R_M\}$ be an ordered set of M distinct elements, where $R_i < R_j$ if $i < j$. Let π be the set of permutations of $\mathfrak{R}$. We are interested in randomly generating elements of π based on an underlying user-defined distribution $G(\pi)$. This problem is called the Random Permutation Generation (RPG) problem.

If the distribution $G(\pi)$ is uniform, the various $M!$ permutations are generated with equal probability. Thus, in this case, $G(\pi)$ is defined as follows:

$$g_i \equiv G(\pi_i) = \text{Prob}[\pi_i \text{ is generated}] = 1/M!. \qquad i = 1, 2, \ldots, M!. \quad (1)$$

If all the values of $G(\pi_i)$ are not equal G is said to be non-uniform.

Observe that if M is small (typically $M \leq 6$) any arbitrary distribution $G(\pi)$ can be specified by a set of simple assignment statements. This obviously entails the enumeration of g_i for $1 \leq i \leq M!$. In such a case, the Random Permutation Generation (RPG) based on the distribution $G(\pi)$ could be achieved by a **single** call to a Random Number Generator (RNG). The technique to do this is simple and straightforward. The interval $[0, 1]$ is subdivided into $M!$ sub-intervals of width g_i, and the random permutation generated is the one in whose subinterval the generated random **number** falls.

When M is large, such a strategy is infeasible primarily because the machine architecture does not permit the generation of a **number** to the degree of precision required. In such a case, one has to generate the random permutation in a more expensive way – i.e., the RPG scheme needs to invoke the RNG more than once. If M is large and $G(\pi)$ is **uniform**, various algorithms have been reported (See Refs 3, 5, 7) to yield equally likely random permutations. The techniques that have been reported require M calls to the uniform RNG.

If M is large, however, and G is non-uniform the problem is two-fold. For the first part, the user is left with the problem of specifying g_i for $1 \leq i \leq M!$. Additionally, once this has been done, the question of generating the permutations based on the $\{g_i\}$ has to be addressed.

In this paper, we shall address both these problems. We shall first propose a technique to specify the probability associated with the various permutations. Although this specification strategy does not permit the user to explicitly specify and control **all** the $M!$ values of g_i (which the user should be grateful for), it does let the user specify $M - 1$ distinct quantities using which he can control the various probabilities $\{g_i\}$. These $M - 1$ quantities are specified in terms of a probability vector called the **control vector**. After specifying it we shall present an algorithm to achieve the RPG based on the latter vector.

Although, in general, the user can completely define the control vector, we believe that in many applications it is easier for the user if this vector is defined in terms of a single parameter. To render this possible, we have suggested a few ways by which the vector of control probabilities can be completely specified in terms of a single parameter called the degree of uniformity. In one specific case the parameter is called ρ, and it can be shown that if $\rho = 0$ the entire probability mass lies on a single permutation. Furthermore, if ρ is unity, $G(\pi)$ is uniform. In between these boundary values ρ can take any value in the open interval $(0, 1)$. A higher value of ρ implies a more equal diffusion of the probability mass among the elements of π.

Alternative single parameter distributions for $G(\pi)$ are also defined in the paper, and in each case the degree of uniformity has been appropriately defined. The paper also contains various simulation results which demonstrate the effect of varying the degree of uniformity on $G(\pi)$. Applications of the RPG technique are also described.

† Partially supported by the National Sciences and Engineering Research Council of Canada. A preliminary version of this paper was presented at the 1989 ACM Computer Science Conference in Louisville, Kentucky.

2. RANDOM PERMUTATION GENERATION

Throughout this paper we assume that the various permutations in π are ordered so that it makes sense to refer to π_i and π_j, where $\pi_i, \pi_j \in \pi$. The ordering may be lexicogrphic (based on the **ordered** set $\Re$)[5] or may be the ordering specified by a permutation listing algorithm.[8] We require however that π_1 and $\pi_{M!}$ are the unique permutations as follows:

$$\pi_1 = R_1 R_2 \ldots R_{M-1} R_M$$

$$\pi_{M!} = R_M R_{M-1} \ldots R_2 R_1.$$

We define the control vector S as an $M \times 1$ probability vector satisfying:

$$S = [s_1, s_2, \ldots, s_M]^T, \text{ where,} \quad (2)$$

$$\sum_{i=1}^{M} s_i = 1. \quad (3)$$

s_i is the probability of generating the element $R_i \in \Re$ in any permutation generation operation.

Let $\mathfrak{V}$ be any subset of $\Re$. We define the conditional control vector $S|\mathfrak{V}$ as the vector of normalized probabilities in which only the quantities corresponding to the elements of the set $\mathfrak{V}$ have non-zero values. Thus, $S|\mathfrak{V}$ consists of the normalized probabilities $\{s_i'\}$, where,

$$s_i' = 0 \qquad \text{if } R_i \notin \mathfrak{V} \quad (4)$$

$$= \frac{s_i}{\sum_{R_i \in \mathfrak{V}} s_i} \qquad \text{otherwise.} \quad (5)$$

Observe that the vector S defined by (2) and (3) is exactly equivalent to $S|\Re$, and thus these above normalized probabilities are indeed conditional probabilities appropriately conditioned.

The technique for generating the permutation is now described. For the sake of explanation we define $P \in \pi$ be the randomly generated permutation, where, if P is the string $p_1 p_2 p_3 \ldots p_M$, then, for all $i \neq j$, $p_i, p_j \in \Re$ and $p_i \neq p_j$. We shall generate P by randomly assigning an element of $\Re$ for every position in P. The way by which this is achieved is by computing successively the prefixes of P. Initially p_1 is randomly assigned a value in $\Re$ based on the control distribution S (i.e., $S|\Re$). By this we mean that for all i, R_i is selected with a probability of s_i. Let us assume that p_1 is assigned the value $R_{P(1)}$. Clearly, the string $p_2 \ldots p_M$ has to be computed using the symbols in $\mathfrak{V} = \Re - \{R_{P(1)}\}$. This is done in a recursive manner by using the conditional distribution of $S|\mathfrak{V}$ and the process is recursively continued by successively updating $\mathfrak{V}$.

We note in passing that although the algorithm has been defined in terms of generating prefixes of the permutation P, it can just as powerfully be defined in terms of assigning positions to the elements of $\Re$. The reason for the current assignment strategy will be clear in Section 2.2.

For the sake of completeness, the above procedure is algorithmically described below. The properties of the algorithm are proved subsequently.

Algorithm RPG ***(S)***
Input: The control vector S satisfying (2) and (3).
Output: A permutation $P = p_1 p_2 p_3 \ldots p_M$, where for all $i \neq j$, $p_i \neq p_j$, and $p_i, p_j \in \Re$.
Method
Begin
  $\mathfrak{V} := \Re$
  For $i := 1$ **to** ***M Do***
    Select $p_i \in \mathfrak{V}$ based on the distribution $S|\mathfrak{V}$, as per (4) and (5).
    $\mathfrak{V} := \mathfrak{V} - \{p_i\}$
  EndFor
End Algorithm RPG

Observe that generating the permutations based on the above algorithm automatically implies that the algorithm disallows all possible **distributions** for $G(\pi)$. But clearly, for large values of M, even the explicit definition of $G(\pi)$ is unattainable. Furthermore, unlike the distribution used in the currently known algorithms, $G(\pi)$ need not necessarily be uniform. Indeed, by merely specifying $M - 1$ independent control quantities, a very large subset of the possible set of distributions can be included in the generating process.

We now prove the properties of Algorithm RPG.

Theorem 1.

The generation of a particular permutation requires at most M invocations of a RNG. Furthermore, if every $s_i > 0$, there is a positive probability of generating every permutation in π. Finally, if $S = [1/M, 1/M, \ldots, 1/M]^T$, the Algorithm RPG generates all the $M!$ permutations with a uniform distribution.

Proof.

The first assertion involving the number of calls of the RNG required is obvious from Algorithm RPG.

If every $s_i > 0$, there is a finite positive probability that R_i can be assigned to p_1. However, in any generation if $p_1 \neq R_i$, then the probability that R_i is in the second position is clearly

$$s_i \Big/ \sum_{\Re - \{p_1\}} s_j$$

which again is clearly greater than zero. A similar argument shows that R_i can be any of the M positions. The second assertion is thus true since the argument is true for all R_i.

To prove the final assertion we note that if $S = [1/M, 1/M, \ldots, 1/M]^T$ the first element p_1 can be any element in $\Re$. Furthermore every element is chosen with an equal probability. The result follows by using a simple recursive argument observing that for all $\mathfrak{V}$ the non-zero components of $S|\mathfrak{V}$ consists of $|\mathfrak{V}|$ elements all of which have the value $1/|\mathfrak{V}|$.

□

The final theorem describes the way by which the control vector controls the form of the permutation that is generated.

Theorem 2.

Let S be the user defined control probability vector. Then, in any permutation that is generated the prob-

ability that R_u precedes R_v is exactly the ratio $s_u/(s_u + s_v)$.

Proof.

Let R_u and R_v be any two distinct elements with indices $u, v \in \{1, 2, \ldots, M\}$, and let their corresponding control probabilities be s_u and s_v respectively. Since the theorem is trivial for the case when either (or both) these probabilities are zero, with no loss of generality we assume that both s_u and s_v are positive. It is required to prove that in this case, the probability of generating a permutation in which R_u precedes R_v is $s_u/(s_u + s_v)$. We shall prove the theorem by performing an induction on the length of the prefix of the generated permutation P.

For the indices $u, v \in \{1, 2, \ldots, M\}$, let $\xi_{u,v}(\mathfrak{W})$ be the event that either R_u or R_v is the element which is selected from $\mathfrak{W}$, where $\mathfrak{R} \supseteq \mathfrak{W}$. Also, let $\xi_{u,v}(i)$ be the event that position i contains the first appearance of R_u or R_v (i.e., neither of them have occurred before, but the ith position contains one of them). Note then that $\xi_{u,v}(1), \ldots, \xi_{u,v}(M-1)$ are mutually exclusive and collectively exhaustive events. By virtue of the generation scheme R_u ultimately precedes R_v if it is selected **before** R_v, since if that is the case, if it is in position p_i, then R_v will be in position p_j where $i < j$.

We first consider the case when $\mathfrak{W} = \mathfrak{R}$. Given $\xi_{u,v}(1)$, R_u ultimately precedes R_v in the permutation every time R_u is the **first** element selected. Clearly, the probability that p_1, the first element selected is R_u is s_u because the RNG is uniform. Similarly, the probability that R_v ultimately precedes R_u in the permutation is the probability that $p_1 = R_v$, and this occurs with a probability s_v. Thus the conditional probability of R_u preceding R_v given $\xi_{u,v}(1)$ is $s_u/(s_u + s_v)$.

We now consider the case when $\mathfrak{W} \supseteq \mathfrak{V}$, where $\mathfrak{V} = \{R_u, R_v\}$ and $\mathfrak{R} \supseteq \mathfrak{W}$. Using (4) and (5), the conditional control vector $\boldsymbol{S}|\mathfrak{W}$ will contain non-zero elements for s'_u and s'_v since both s_u and s_v are positive. Since R_u and R_v have not yet been selected, their selection will be based on the conditional control vector $\boldsymbol{S}|\mathfrak{W}$. The conditional probability that R_u ultimately precedes R_v given $\xi_{u,v}(M + 1 - |\mathfrak{W}|)$ is the probability that the first element selected from $\mathfrak{W}$, $p_{M+1-|\mathfrak{W}|}$, is R_u. Again, since the RNG is uniform, this occurs with a probability s'_u. Similarly, the **conditional** probability that R_v ultimately precedes R_u in the permutation given $\xi_{u,v}(M + 1 - |\mathfrak{W}|)$ is the probability that $p_{M+1-|\mathfrak{W}|}$ is R_v. This occurs with a probability s'_v. Thus the conditional probability of R_u preceding R_v given $\xi_{u,v}(M + 1 - |\mathfrak{W}|)$ is obviously $s'_u/(s'_u + s'_v)$, which again is $s_u/(s_u + s_v)$ since the normalizing constant for s'_u and s'_v are exactly the same and is defined by (5).

To complete the proof we note that since $\xi_{u,v}(1), \ldots, \xi_{u,v}(M-1)$ are mutually exclusive and collectively exhaustive events, using the laws of total probability,

$$\text{Prob}(R_u \text{ precedes } R_v) = \sum_i \text{Prob}[R_u \text{ precedes } R_v | \xi_{u,v}(i)] \cdot \text{Prob}[\xi_{u,v}(i)].$$

Since $\text{Prob}[R_u \text{ precedes } R_v | \xi_{u,v}(i)]$ is the same for all i,

$$\text{Prob}(R_u \text{ precedes } R_v) = \text{Prob}[R_u \text{ precedes } R_v | \xi_{u,v}(i)] \cdot \sum_i \text{Prob}[\xi_{u,v}(i)]$$

$$= s_u/(s_u + s_v)$$

and the result follows. □

2.1. Alternate Specifications of $G(\pi)$: Pairwise probabilities

An alternative way of specifying a distribution $G(\pi)$ would be to specify the probability of R_u preceding R_v in a permutation. Let ${}_uP_v$ be this probability. Clearly, ${}_uP_v = 1 - {}_vP_u$, and hence, if the user so desired, he could specify these $M(M-1)/2$ probabilities (i.e., $\{{}_uP_v$ for all $u, v\}$) to completely define his distribution $G(\pi)$ on π. Observe that if these $M(M-1)/2$ probabilities were specified, by making $M(M-1)/2$ uniform RNG invocations, the order of every pair of elements would be explicit and the random permutation generated.

The problem with this technique however is that the invocations could lead to contradictory conclusions. Thus, if we consider the set $\mathfrak{R} = \{A, B, C\}$, the RNG based on ${}_AP_B$ may impose the condition that A precedes B. Similarly, the RNG based on ${}_BP_C$ may impose the condition that B precedes C. Notice that this could imply that the permutation generated is 'ABC'. However, if the RNG based on ${}_AP_C$ required that C preceded A, the entire random permutation generation process would have to be repeated because the orders specified by the pairwise positioning of the elements are contradictory. Indeed, in the more general case, of the $2^{M(M-1)/2}$ possible outcome scenarios, only $M!$ of these would lead to valid permutations. Clearly, as M is large the process could be 'divergent'. It is easy to see that even for small M, an infinite number of calls may be required in any one permutation generation.

Such a 'divergent' scenario does not occur if the $\{s_i\}$ are used to describe $G(\pi)$ because the control vector $\boldsymbol{S}$ fully defines the pairwise probabilities ${}_uP_v$ for all $u, v \in \{1, 2, \ldots, M\}$. The reverse is, of course, not true. However, if $M-1$ of the pairwise probabilities are user specified, by using Theorem 2, the control vector $\boldsymbol{S}$ can be fully and **uniquely** defined and the RPG technique described in this section can be used to generate the permutation P. The following example clarifies this.

Example 1

Let $\mathfrak{R} = \{A, B, C\}$ and let the two user defined pairwise probabilities be:

$${}_AP_B = 0.625; \qquad {}_BP_C = 0.6$$

Then, since ${}_AP_B = s_A/(s_A + s_B)$ and ${}_BP_C = s_B/(s_B + s_C)$, we have,

$$s_A/(s_A + s_B) = 0.625; s_B/(s_C + s_B) = 0.6;$$

and,

$$s_A + s_B + s_C = 1.$$

This implies that $s_A = 0.5$, $s_B = 0.3$ and $s_C = 0.2$.

Using this as the control vector, permutations can be generated to satisfy the user specified pairwise probabilities.

2.2. A single parameter specification of $G(\pi)$

Till now we have specified $G(\pi)$ and generated random permutations based on the control vector S. However, for the specification of S the user has to specify $M-1$ probabilities. Often, it is more convenient that S be defined by a single parameter. We shall describe one such single parameter specification of $G(\pi)$.

Let ρ be a real number in the interval $(0, 1]$. Then a single parameter specification of $G(\pi)$ is defined in terms of ρ as:

$$s_i = \rho \,.\, s_{i-1} \qquad \text{for } i = 2, \ldots, M.$$

ρ is called the degree of uniformity.

Observe that if $\rho \to 0$, s_1 will be arbitrarily larger than s_2. Thus the probability that s_1 precedes s_2 can be as close to unity as desired. Also, in general s_i will be arbitrarily larger than s_{i+1} implying that in the limit as $\rho \to 0$, the probability mass will be completely concentrated on $\pi_1 = R_1 R_2 \ldots R_M$ with probability 1. Furthermore, if $\rho = 1$, all the s_i's are equal implying from Theorem I that the probability mass is **equally** dispersed between all the elements of π.

To demonstrate the effect of ρ on the distribution of $G(\pi)$ the latter has been tabulated below for various values of ρ and for the values of $M = 3, 4$. Notice the concentration of the probability mass when $\rho = 0$ and the spread in the mass as ρ increases. Thus, when $\rho = 0.2$, the probability mass on the permutation 'ABC' is 0.672043, and this mass decreases to 0.45788 and 0.22769 as ρ increases to 0.4 and 0.8 respectively. As expected, the distribution is uniform when ρ is unity. Similarly, when $\rho = 0.2$, and $M = 4$, the probability mass on the permutation 'ABCD' is 0.53850, and this mass decreases to 0.28194 and 0.07713 as ρ increases to 0.4 and 0.8 respectively. Again, the distribution is uniform when $\rho \to 1$.

Obviously for large values of M such as enumeration strategy is infeasible. To consider the effect of ρ for large values of M, we have taken the number of inversions in a permutation to be an indicator on the degree of unsortedness in the permutation*. This[6] is quite a natural measure. Thus, in any generated permutation, since we can count the number of inversions, a plot of the expected number of inversions as a function of ρ would demonstrate the power of ρ to parametrize the degree of uniformity of $G(\pi)$.

* An inversion in a permutation $\rho = p_1 p_2 \ldots p_M$ is a pair (p_i, p_j) where $p_j < p_i$ although $i < j$. For the properties of inversions, see [ref. 6].

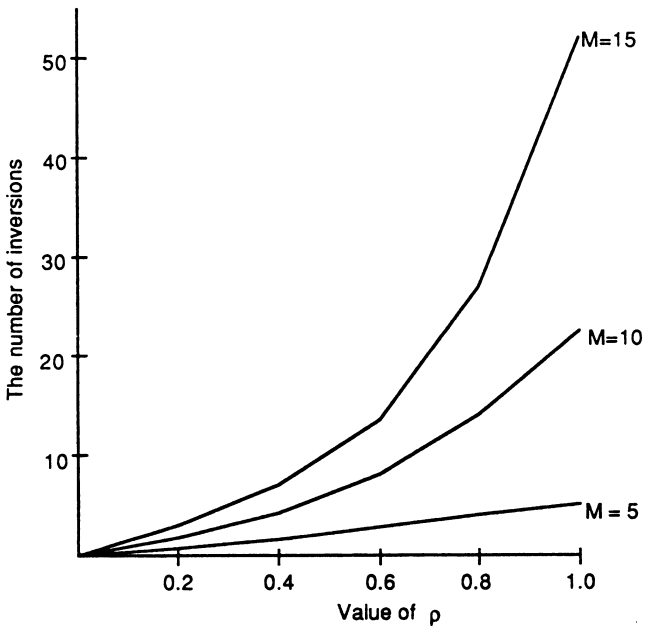

Figure 1. A plot of the expected number of inversions as a function of ρ. Note the increase in the expected number of inversions with ρ.

The plot is shown in figure 1 for M taking values 5, 10 and 15. In each case, **ten thousand** permutations were generated for **every** value of ρ so that a good estimate of the average number of inversions could be obtained. Note the monotonicity of the index as ρ increases. In each case the average number of inversions has a value of zero when ρ is zero, and the quantity increases to the value of $M(M-1)/4$ as ρ increases to unity.

We conclude this subsection by observing that if the user did not choose to have the probability mass concentrated on $\pi_1 = R_1 R_2 \ldots R_M$ for $\rho = 0$, but rather choose to have it concentrated on some other permutation $Q = q_1 q_2 \ldots q_M$, where $Q \in \pi$, this can be trivially achieved by assigning

$$s_{q_i} = \rho \,.\, s_{q_{i-1}} \qquad i = 2, \ldots, M, \text{ and } \rho \in (0, 1].$$

2.3. Alternative single parameter specifications of $G(\pi)$

The parameter ρ called the degree of uniformity has been defined so as to render the set $\{s_i\}$ a geometric

Table 1. The distribution $G(\pi)$ for $M = 3$ specified in terms of the degree of uniformity. In this case, $\Re = \{A, B, C\}$. Observe that when $\rho = 0$, the probability mass is concentrated on $\pi_1 = ABC$ and when $\rho = 1$ the mass is uniformly distributed.

π \ ρ	0	0.2	0.4	0.6	0.8	1
ABC	1	0.672043	0.45788	0.31888	0.22769	0.16667
ACB	0	0.134409	0.18315	0.19133	0.18215	0.16667
BAC	0	0.155086	0.22104	0.22509	0.19992	0.16667
BCA	0	6.2035E-3	3.5367E-2	8.1032E-2	0.12795	0.16667
CAB	0	2.0882E-2	7.326E-2	0.11480	0.14572	0.16667
CBA	0	5.3763E-3	2.9304E-2	6.8878E-2	0.11658	0.16667

Table 2. The distribution $G(\pi)$ for $M = 4$ specified in terms of the degree of uniformity. In this case, $\mathfrak{A} = \{A, B, C, D\}$. Observe that when $\rho = 0$, the probability mass is concentrated on $\pi_1 = ABCD$ and when $\rho = 1$ the probability mass is uniformly distributed.

π \ ρ	0	0.2	0.4	0.6	0.8	1
ABCD	1	0.53850	0.28194	0.14654	7.7130E-2	4.1667E-2
ABDC	0	0.10770	0.11278	8.7926E-2	6.1704E-2	4.1667E-2
ACBD	0	0.12427	0.13611	0.10344	6.7724E-2	4.1667E-2
ACDB	0	4.9707E-3	2.1778E-2	3.7239E-2	4.3343E-2	4.1667E-2
ADBC	0	2.1540E-2	4.5111E-2	5.2755E-2	4.9363E-2	4.1667E-2
ADCB	0	4.3080E-3	1.8044E-2	3.1653E-2	3.9490E-2	4.1667E-2
BACD	0	0.12743	0.14374	0.10935	6.9961E-2	4.1667E-2
BADC	0	2.5486E-2	5.7494E-2	6.5610E-2	5.5969E-2	4.1667E-2
BCAD	0	6.0681E-2	3.0260E-2	5.1797E-2	5.3304E-2	4.1667E-2
BCDA	0	4.8545E-5	1.9366E-3	1.1188E-2	2.7292E-2	4.1667E-2
BDAC	0	1.1763E-3	1.1102E-2	2.7788E-2	3.9315E-2	4.1667E-2
BDCA	0	4.7051E-5	1.7764E-3	1.0003E-2	2.5162E-2	4.1667E-2
CABD	0	2.5512E-2	5.8014E-2	6.6987E-2	5.7178E-2	4.1667E-2
CADB	0	1.0205E-3	9.2823E-3	2.4115E-2	3.6594E-2	4.1667E-2
CBAD	0	5.2644E-3	2.5299E-2	4.4952E-2	4.9615E-2	4.1667E-2
CBDA	0	4.2115E-5	1.6192E-3	9.7910E-3	2.5403E-2	4.1667E-2
CDAB	0	1.7688E-4	3.0764E-3	1.2299E-2	2.6673E-2	4.1667E-2
CDBA	0	3.5377E-5	1.2306E-3	7.3793E-3	2.1338E-2	4.1667E-2
DABC	0	4.3080E-3	1.8044E-2	3.1653E-2	3.9490E-2	4.1667E-2
DACB	0	8.6159E-4	7.2177E-3	1.8992E-2	3.1592E-2	4.1667E-2
DBAC	0	9.9415E-4	8.7111E-3	2.2343E-2	3.4674E-2	4.1667E-2
DBCA	0	3.9766E-5	1.3938E-3	8.0437E-3	2.2192E-2	4.1667E-2
DCAB	0	1.7232E-4	2.8871E-3	1.1395E-2	2.5274E-2	4.1667E-2
DCBA	0	3.4464E-5	1.1548E-3	6.8371E-3	2.0219E-2	4.1667E-2

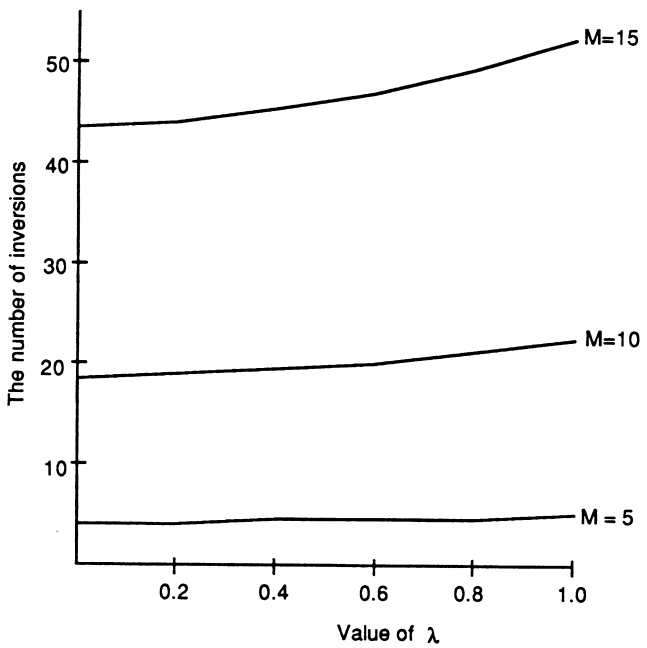

Figure 2. A plot of the expected number of inversions as a function of λ. Note the increase in the expected number of inversions with λ.

progression. Thus R_1 has a highest probability of being p_1, and R_M the smallest probability of being p_1.

This is not the only way by which $G(\pi)$ can be specified using a single parameter. Indeed, the set $\{s_i\}$ can also be defined as a set of diminishing probabilities where the quantities decrease as per an arithmetic progression. In such a case, if $\lambda \in (0, 1)$, we define

$$s_i' = N/2 + (N - i + 1)(1 - \lambda).$$

Furthermore, we fully define the control vector $\{s_i\}$ as,

$$s_i = s_i' \Big/ \sum_{i=1}^{M} s_i'.$$

Observe that for any value of λ, it is impossible to obtain s_i to be arbitrarily larger than s_{i+1} for $M > 2$. In this case, to avoid repetition, we have not considered the case of $M = 3$ and 4 as in the case discussed earlier. However, for large values of M, we have plotted (in Figure 2) the average number of inversions as a function of λ. In each case the average number of inversions increases to the value of $M(M - 1)/4$ as λ increases to unity.

To conclude this subsection, we note that $\{s_i\}$ need not merely be geometrically varied (as in section 2.2) or arithmetically varied (as in this subsection). Indeed,

$\{s_i\}$ can be varied as per any user-defined progression, such as the Zipf law or an arithmetico-geometric progression. Thus the set of distributions permitted by our RPG algorithm is indeed a very large subset of the set of all possible distributions.

3. APPLICATIONS OF RANDOM PERMUTATION GENERATION

One of the most basic problems studied in computer science is that of sorting M elements. It is well known that sorting algorithms can be made very efficient, and that in particular, sorting can be achieved with the **average** time of $O(M \log M)$.[6,8]

However, if the elements to be sorted are in an 'almost sorted' order, some of the 'worst' algorithms turn out to be extremely efficient. Indeed, Sedgewick has even asserted that Insertion Sort (which has a worst case complexity of $O(M^2)$) has a linear time complexity if the list is initially 'almost sorted'. In this connection, probably the most impressive of the various sorting algorithms is SmoothSort, due to Dijkstra[2]. The latter has a worst case complexity of $O(M \log M)$ and it has a complexity of $O(M)$ for 'almost sorted' lists. Furthermore, the complexity increases 'smoothly' as the degree of unsortedness increases.[2]

To actually compare various sorting algorithms Cook and Kim[1] tested a variety of sorting algorithms experimentally. However, the biggest problem in such studies is to generate the data in the testing phase. Indeed, if $\Re = \{A, B, C, D, E\}$, it is easy to see that if 'ABCDE' is the sorted list then 'BACDE' is 'almost sorted'. The question is then one of randomly generating such lists.

It is a desirable feature that the algorithm generating 'almost sorted' lists must permit the generation of every element of π. The reasons for this is because although Insertion Sort is good for almost sorted lists, it performs very poorly for 'almost unsorted' lists. Thus, if a user intends to use Insertion Sort as his sorting mechanism, he would have to reckon with the fact that the scheme would be excellent if the list is 'almost sorted' but it would perform poorly if the list is unsorted. The question is now one of understanding that although 'EDCBA' is an unsorted list, this *could* be the input to the algorithm although the probability of this occurring as a list to be sorted may be very small. The strategy which we have proposed in this paper permits the generation of almost sorted lists. Indeed, as shown in Figure 1, a small value of ρ (say $\rho = 0.2$) yields the 'almost sorted' lists with a relatively high probability, but also distributes the rest of the probability mass among the other permutations in such a way that the less sorted lists have less of a probability of being generated. Notice if $M = 10$ and $\rho = 0.2$, although all the $M!$ permutations (3,628,800) can be generated, the average number of inversions generated is but 1.84. We believe that our strategy will be extremely powerful in such scenarios.

Another application of the technique presented in this paper is in the travelling salesman problem.[4,8] In this problem, a salesman is required to start from any city and tour all the cities in his jurisdiction and cover the minimum distance in this endeavour. The problem is known to be NP-Complete and a host of approximate solutions have been suggested. One such solution uses the concept of simulated annealing. In this solution, the algorithm starts with a tour and using the principles of 'controlled annealing', attempts to find a superior tour. The algorithm assumes the knowledge of an initial tour obtained, for example, using various diameters of the points. The question of computing an initial **random** tour can be solved using the RPG technique suggested in this paper. By assigning a probability measure to the various initial tours, various travelling salesman algorithms can be compared by using the ensemble averages obtained from the results of having the algorithms run from the random starting tours proposed by a RPG algorithm.

Observe that in this case the designer of the algorithm must choose an appropriate value of ρ or λ (or the control vector S) which assigns to tours which are initially not worthwhile pursuing a negligible probability measure. He should then take this into consideration to possibly re-index the cities and determine a suitable value of ρ or λ so as to make such a probability assignment possible.

4. CONCLUSIONS

In this paper we have studied the problem of generating random permutations of a set $\Re = \{R_1, R_2, \ldots, R_M\}$. Let π be the set of permutations of $\Re$. We have concentrated on randomly generating the elements of π according to a distribution $G(\pi)$ when the latter distribution is not necessarily uniform. We have first proposed a strategy for specifying the distribution $G(\pi)$ in terms of a control probability vector S, and then proceeded to present a technique for generating the elements of π according to this control vector. The technique generates the random permutation by invoking a RNG exactly M times.

We have also suggested a few techniques by which the user can specify the control vector S in terms of a single parameter called the degree of uniformity. In one case this parameter is given by ρ and it quantifies the uniformity of $G(\pi)$. If $\rho = 0$ the entire probability mass lies on a single permutation and if ρ is unity, $G(\pi)$ is uniform. In between these boundary values ρ can take any value in the open interval $(0, 1)$. A higher value of ρ implies a more equal spread of the probability mass among the elements of π. The effects of changing ρ on the distribution $G(\pi)$ has been demonstrated by presenting various simulation results.

Finally, two applications of the RPG technique are also described which involve the generation of 'almost sorted lists' and the generation of the initial tours for an algorithm attempting to solve the Travelling Salesman Problem.

ACKNOWLEDGEMENTS

We are very grateful to an anonymous referee for his careful reading of the paper and for helping us tighten the proof of Theorem II.

REFERENCES

1. C. R. Cook and D. J. Kim, Best Sorting Algorithm for Nearly Sorted Files, *Comm. of the ACM*, **23**, 620–624 (1980)
2. E. W. Dijkstra, Smoothsort, An Algorithm for Sorting in SITU, *Science of Comp. Prog.*, pp. 223–233 (1982).

3. R. Durstenfield, Random Permutation, *Comm. of the ACM,* 7, 420 (1964).
4. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, *The Travelling Salesman Problem,* John Wiley (1985).
5. D. E. Knuth, *The Art of Computer Programming; Vol. 2: Seminumerical Algorithms,* Addison Wesley, Reading, MA, Dover (Second Edition) (1981).
6. D. E. Knuth, *The Art of Computer Programming; Vol. 3: Sorting and Searching,* Addison Wesley, Reading, MA, Dover (Second Edition) (1981).
7. L. E. Moses and R. V. Oakford, *Tables of Random Permutations,* Stanford University Press (1963).
8. R. Sedgewick, *Algorithms,* Addison Wesley (Second Edition) (1988).
9. S. M. Ross, *Stochastic Processes,* Wiley (1983).

Announcement

2–5 JUNE 1991

HAWAII

Fourth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-91), Waiohai Hotel, Kauai, Hawaii, June 2–5, 1991. Sponsored by ACM/SIGART, and The University of Tennessee Space Institute, in cooperation with AAAI, IEEE Computer Society, International Association of Knowledge Engineers, and Canadian Society for Computational Studies of Artificial Intelligence, International Neural Network Society, and ECCAI.

For further information contact:
Dr. Moonis Ali, Conference General Chairman, The University of Tennessee Space Institute, MS15, B.H. Goethert Parkway, Tullahoma, TN 37388, U.S.A. Tel: (615) 455-0631 ext. 236. Fax (615) 454-2354.
E-mail: ALIF@UTSIV1.BITNET

12–16 NOVEMBER 1990

NIMES, FRANCE

Neural Networks and their Applications

The Second International Workshop on "Neural Networks & their Applications", held in November 1989, attracted over 500 visitors, attendees and speakers from both the industrial world and research laboratories of some twenty countries. It was also the occasion for twenty-five companies and research organizations to exhibit their products and prototypes.

Fortified by this success, the Neuro-Nîmes Workshop will be held at Nîmes from November 12 to 16, 1990.

In a field where fashion may perhaps tend to obscure seriousness, it can sometimes be difficult to distinguish what should and what should not be taken into account, and it is with the focus closely on reality and efficiency that Neuro-Nîmes '90 will present a selection of the leading R&D work on connectionism, neural networks and their applications. Participants will be able to benefit from a unique opportunity to take stock of current methods and techniques and be in a position to foresee the short term and medium-term industrial applications.

A meeting place for engineers from industry and research workers, Neuro-Nîmes will comprise four complementary events:

- a scientific and technical conference
- a series of tutorials
- an exhibition of commercially available products and advanced prototypes
- an industrial forum

Information

Correspondence and requests for information should be addressed to the General Chairman: Jean-Claude Rault
EC2-269-287, rue de la Garenne - 92024 Nanterre Cedex - France. Tel: +33.1 47 80 70 00 - Telex: 612 469 - Fax: +33.1 47 80 66 29