ON GENERATING RANDOM PERMUTATIONS WITH ARBITRARY DISTRIBUTIONS

by B. J. Oommen and D.T.H. Ng

SCS-TR-138

June 1988

School of Computer Science
Carleton University
Ottawa, Ontario
CANADA K1S 5B6

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada.

ON GENERATING RANDOM PERMUTATIONS WITH ARBITRARY DISTRIBUTIONS†

B. J. Oommen* and D. T. H. Ng*

ABSTRACT

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 [D2, K1] and Moses *et al* [M1] 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 the 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.

Keywords : Combinatorial Problems, Random Permutation Generation, Random Algorithms.

---

† Partially supported by the Natural Sciences and Engineering Research Council of Canada.

* School of Computer Science, Carleton University, Ottawa : K1S 5B6, CANADA.

INTRODUCTION

Let $\mathfrak{R} = \{R_1, R_2, ..., R_M\}$ 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! permutation are generated with equal probability. Thus, in this case, $G(\pi)$ is defined as follows:

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

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 [D2], [K1] and [M1]) 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 way by which the vector of control probabilities can be completely controlled by a single parameter ρ called the degree of uniformity. ρ is a measure of the uniformity of $G(\pi)$, for 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 diffusion of the probability mass among the elements of π.

We also present various simulation results which demonstrate the effect of varying the degree of uniformity, ρ on $G(\pi)$. Various applications of the RPG technique are also described.

II RANDOM PERMUATION 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 lexicographic (based on the **ordered** set $\mathfrak{R}$) [K1] or may be the ordering specified by a permutation listing algorithm [S1]. We require however that π_i 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 X 1 probability vector satisfying:

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

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

s_i represents the probability of generating the element $R_i \in \mathfrak{R}$ in any permutation generation operation.

Let $\mathfrak{V}$ be any subset of $\mathfrak{R}$. We define the conditional control vector $\mathbf{S}|\mathfrak{V}$ 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, $\mathbf{S}|\mathfrak{V}$ consists of normalized probabilities $\{s_i'\}$, where,

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

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

Observe that the vector **S** defined by (2) and (3) are exactly equivalent to $\mathbf{S}|\mathfrak{R}$.

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 $i \neq j$, $p_i, p_j \in \mathfrak{R}$ and $p_i \neq p_j$. We shall generate P by randomly assigning a position for every element of $\mathfrak{R}$. The way by which this is achieved is by computing successively the prefixes of P. Initially p_1 is randomly assigned a value in $\mathfrak{R}$ based on the control distribution **S**. (i.e., $\mathbf{S}|\mathfrak{R}$). 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 $\mathfrak{V} = \mathfrak{R} - \{R_{P(1)}\}$. This is done in a recursive manner by using the conditional distribution of $\mathbf{S}|\mathfrak{V}$ and the process is recursively continued by successively updating $\mathfrak{V}$. For the sake of completeness, the 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 $p_i, p_j \in \mathfrak{R}$.
Method
Begin
    $\mathfrak{V} := \mathfrak{R}$
    For i := 1 to M **Do**
        Select $p_i \in \mathfrak{V}$ based on the distribution $\mathbf{S}|\mathfrak{V}$, as per (4) and (5).
        $\mathfrak{V} := \mathfrak{V} - \{p_i\}$
    EndFor
End Algorithm RPG

Theorem I.

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 $\pi \in \pi$. Finally, if $\mathbf{S} = [1/M, 1/M, \ldots, 1/M]^T$, 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, if in any generation if $p_1 = R_{P(1)} \neq R_i$, then the probability that R_i is in the second position is clearly

$$\frac{s_i}{\sum_{\mathfrak{R} - \{p_1\}} s_j}$$

which again is clearly greater than zero. A similar argument shows that R_i can be in 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 $\mathbf{S} = [1/M, 1/M, ..., 1/M]^T$ the first element p_1 can be any element in $\mathfrak{R}$. 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 $\mathbf{S}|\mathfrak{V}$ 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 II.

Let **S** be the user defined control probability vector. Then, in any permutation that is generated the probability 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 ε {1, 2, ..., 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 ε {1, 2, ..., 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 $\mathfrak{R} \supseteq \mathfrak{W}$. Let $\xi_{u,v}(i)$ be the event that either R_u or R_v is the element which is selected when the For loop index is i. 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 $\xi_{u,v}(i)$ 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 $\xi_{U,V}(\mathfrak{R})$ 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 $\mathbf{S}|\mathfrak{W}$ will contain non-zero elements for s_U' and s_V' since both s_U and s_V are positive. This implies that given $\xi_{U,V}(i)$, since R_U and R_V have not yet been selected, their selection will be based on the conditional control vector $\mathbf{S}|\mathfrak{W}$. In this case the probability that R_U ultimately precedes R_V in the permutation 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 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 $\xi_{U,V}(\mathfrak{W})$ 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). Hence the theorem ! •••

II.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". Indeed, even for small M, an infinite number of calls may be required in any one permutation generation.

Notice that such a "divergent" scenario does not occur if the $\{s_i\}$ are used to describe $G(\pi)$ because the control vector **S** fully defines the pairwise probabilities

${}_uP_v$ for all u, v $\in$ {1, 2, ..., M}. The reverse is, of course, not true. However, if M-1 of the pairwise probabilities are user specified, by using Theorem III, the control vector **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 I.

Let $\mathfrak{A}$ = {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,

$$s_A / (s_A + s_B) = 0.625 \; ; \; s_B / (s_B + s_C) = 0.6 \; ; \text{ and, } \; s_A + s_B + s_C = 1.$$

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 generate to satisfy the user specified pairwise probabilities.

II.2 Single Parameter Specification of G(π)

Till now we have specified G(π) and generated random permutations based on the control vector **S**. However, for the specification of **S** the user has to specify M-1 probabilities. It is often more convenient that **S** is defined by a single parameter, ρ. ρ is called the degree of uniformity. The specification is as follows for $\rho \in (0, 1]$.

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

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 ρ=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. The distribution of G(π) for M=3 and M=4 are tabulated below for various values of ρ. Notice the concentration of the probability mass when ρ=0 and the spread in the mass as ρ increases. Thus, when ρ=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 ρ=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 an enumeration strategy is infeasible. To consider the effect of ρ for large values of M, we have taken the number of

inversions in a permutations to be an indicator on the degree of unsortedness in the permutation*. This [K2] 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 parameterize the degree of uniformity of $G(\pi)$.

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 I : The distribution $G(\pi)$ for M=3 specified in terms of the degree of uniformity. In this case, $\mathfrak{R}$ = {A, B, C}. Observe that when $\rho = 0$, the probability mass is concentrated on π_1=ABC and when ρ=1 the mass is uniformly distributed.

The plot is shown in Figure I for N taking values 5, 10 and 15. In each case, **ten thousand** permutation 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.

---

* 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 [K2].

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-3	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

Table II : The distribution $G(\pi)$ for M=3 specified in terms of the degree of uniformity. In this case, $\mathfrak{R}$ = {A, B, C, D}. Observe that when $\rho=0$, the probability mass is concentrated on π_1=ABCD and when $\rho = 1$ the probability mass is uniformly distributed.

We conclude this section be 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

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

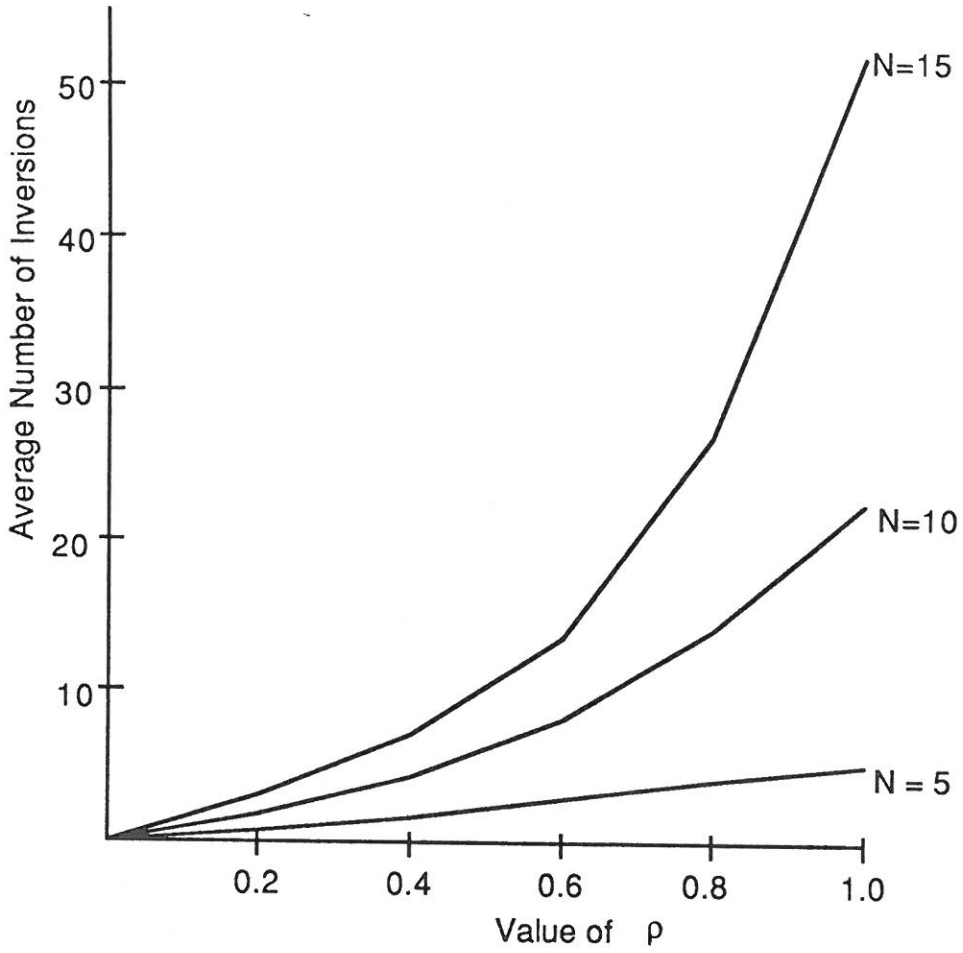

Figure I : A plot of the expected number of inversions as a function of ρ. Note the increase in the expected number of inversions with ρ.

III. 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(MlogM). [K2], [S1].

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 [D1]. 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 [D1].

To actually compare various sorting algorithms Cook and Kim [C1] 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 $\mathfrak{A}$ = {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 I, 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 N=10 and $\rho = 0.2$, although all the M! permutations (518,400) 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 [L1], [S1]. In this problem, salesman is required to start from any city and tour all the cities in his jurisdiction and cover the minimum distance in this endeavor. 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** initial tour can be solved using the RPG technique suggested this paper. Of course, it is not desirable that the starting tour be any permutation of the M cities, because many such tours can definitely be discarded, especially those which have crossings. By assigning a probability measure to the various initial tours, various travelling salesman algorithms can be compared on the basis of the various

random starting tours proposed by the RPG.

IV. CONCLUSIONS

In this paper we have studied the problem of generating random permutations of a set $\mathfrak{R} = \{R_1, R_2, \ldots, R_M\}$. Let π be the set of permutations of $\mathfrak{R}$. 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 the π according to this control vector. The technique generates the random permutation by invoking a RNG exactly M times.

We have also suggested a technique by which the user can specify the control vector **S** in terms of a single parameter ρ, called the degree of uniformity. ρ 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, various applications of the RPG technique are also described which involve the generation of "almost sorted lists" and the generation of the initial tour for an algorithm attempting to solve the Travelling Salesman Problem.

REFERENCES

[C1] Cook,C.R. and Kim, D.J., "Best Sorting Algorithm for Nearly Sorted Files", *Comm. of the ACM,* Vol. 23, 1980, pp.620-624.

[D1] Dijkstra, E.W., "Smoothsort, An Algorithm for Sorting in SITU", *Science of Comp. Prog.*, 1982, pp. 223-233.

[D2] Durstenfeld, R., "Random Permutation", *Comm. of the ACM,* Vol. 7, 1964, pp.420.

[L1] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B.,*The Traveling Salesman Problem* , John Wiley, 1985.

[K1] Knuth, D.E., *The Art of Computer Programming ; Vol. 2 : Seminumerical Algorithms,* Addison Wesley, Reading, MA, Dover, 1981. (Second Edition).

[K2] Knuth, D.E., *The Art of Computer Programming ; Vol. 3 : Sorting and Searching,* Addison Wesley, Reading, MA, Dover, 1981. (Second Edition).

[M1] Moses, L.E. and Oakford, R.V.,*Tables of Random Permutations*, Stanford University Press, 1963.

[S1] Sedgewick R.., *Algorithms,* Addison Wesley, 1988. (Second Edition).

[R1] Ross, S. M., *Stochastic Processes*, Wiley, 1983.

Carleton University, School of Computer Science
Bibliography of Technical Reports
Publications List (1985 -->)

School of Computer Science
Carleton University
Ottawa, Ontario, Canada
KIS 5B6

SCS-TR-66 **On the Futility of Arbitrarily Increasing Memory Capabilities of Stochastic Learning Automata**
_____ B.J. Oommen, October 1984. Revised May 1985.

SCS-TR-67 **Heaps in Heaps**
_____ T. Strothotte, J.-R. Sack, November 1984. Revised April 1985.

SCS-TR-68 **Partial Orders and Comparison Problems**
out-of-print M.D. Atkinson, November 1984. See Congressus Numerantium 47 ('86), 77-88

SCS-TR-69 **On the Expected Communication Complexity of Distributed Selection**
_____ N. Santoro, J.B. Sidney, S.J. Sidney, February 1985.

SCS-TR-70 **Features of Fifth Generation Languages: A Panoramic View**
_____ Wilf R. LaLonde, John R. Pugh, March 1985.

SCS-TR-71 **Actra: The Design of an Industrial Fifth Generation Smalltalk System**
_____ David A. Thomas, Wilf R. LaLonde, April 1985.

SCS-TR-72 **Minmaxheaps, Orderstatisticstrees and their Application to the Coursemarks Problem**
_____ M.D. Atkinson, J.-R. Sack, N. Santoro, T. Strothotte, March 1985.

SCS-TR-73 **Designing Communities of Data Types**
Wilf R. LaLonde, May 1985.
Replaced by SCS-TR-108

SCS-TR-74 **Absorbing and Ergodic Discretized Two Action Learning Automata**
out-of-print B. John Oommen, May 1985. See IEEE Trans. on Systems, Man and Cybernetics, March/April 1986, pp. 282-293.

SCS-TR-75 **Optimal Parallel Merging Without Memory Conflicts**
_____ Selim Akl and Nicola Santoro, May 1985

SCS-TR-76 **List Organizing Strategies Using Stochastic Move-to-Front and Stochastic Move-to-Rear Operations**
_____ B. John Oommen, May 1985.

SCS-TR-77 **Linearizing the Directory Growth in Order Preserving Extendible Hashing**
_____ E.J. Otoo, July 1985.

SCS-TR-78 **Improving Semijoin Evaluation in Distributed Query Processing**
_____ E.J. Otoo, N. Santoro, D. Rotem, July 1985.

Carleton University, School of Computer Science
Bibliography of Technical Reports

SCS-TR-79 **On the Problem of Translating an Elliptic Object Through a Workspace of Elliptic Obstacles**
B.J. Oommen, I. Reichstein, July 1985.

SCS-TR-80 **Smalltalk - Discovering the System**
W. LaLonde, J. Pugh, D. Thomas, October l985.

SCS-TR-81 **A Learning Automation Solution to the Stochastic Minimum Spanning Circle Problem**
B.J. Oommen, October 1985.

SCS-TR-82 **Separability of Sets of Polygons**
Frank Dehne, Jörg-R. Sack, October 1985.

SCS-TR-83 **Extensions of Partial Orders of Bounded Width**
out-of-print
M.D. Atkinson and H.W. Chang, November 1985. See Congressus Numerantium, Vol. 52 (May 1986), pp. 21-35.

SCS-TR-84 **Deterministic Learning Automata Solutions to the Object Partitioning Problem**
B. John Oommen, D.C.Y. Ma, November 1985

SCS-TR-85 **Selecting Subsets of the Correct Density**
out-of-print
M.D. Atkinson, December 1985. To appear in Congressus Numerantium, Proceedings of the 1986 South-Eastern conference on Graph theory, combinatorics and Computing.

SCS-TR-86 **Robot Navigation in Unknown Terrains Using Learned Visiblity Graphs. Part I: The Disjoint Convex Obstacles Case**
B. J. Oommen, S.S. Iyengar, S.V.N. Rao, R.L. Kashyap, February 1986

SCS-TR-87 **Breaking Symmetry in Synchronous Networks**
Greg N. Frederickson, Nicola Santoro, April 1986

SCS-TR-88 **Data Structures and Data Types: An Object-Oriented Approach**
John R. Pugh, Wilf R. LaLonde and David A. Thomas, April 1986

SCS-TR-89 **Ergodic Learning Automata Capable of Incorporating Apriori Information**
B. J. Oommen, May 1986

SCS-TR-90 **Iterative Decomposition of Digital Systems and Its Applications**
Vaclav Dvorak, May 1986.

SCS-TR-91 **Actors in a Smalltalk Multiprocessor: A Case for Limited Parallelism**
Wilf R. LaLonde, Dave A. Thomas and John R. Pugh, May 1986

SCS-TR-92 **ACTRA - A Multitasking/Multiprocessing Smalltalk**
David A. Thomas, Wilf R. LaLonde, and John R. Pugh, May 1986

SCS-TR-93 **Why Exemplars are Better Than Classes**
Wilf R. LaLonde, May 1986

SCS-TR-94 **An Exemplar Based Smalltalk**
Wilf R. LaLonde, Dave A. Thomas and John R. Pugh, May 1986

SCS-TR-95 **Recognition of Noisy Subsequences Using Constrained Edit Distances**
B. John Oommen, June 1986

Carleton University, School of Computer Science
Bibliography of Technical Reports

SCS-TR-96 **Guessing Games and Distributed Computations in Synchronous Networks**
J. van Leeuwen, N. Santoro, J. Urrutia and S. Zaks, June 1986.

SCS-TR-97 **Bit vs. Time Tradeoffs for Synchronous Elections**
M. Overmars and N. Santoro, February 1988.

SCS-TR-98 **Reduction Techniques for Distributed Selection**
N. Santoro and E. Suen, June 1986.

SCS-TR-99 **A Note on Lower Bounds for Min-Max Heaps**
A. Hasham and J.-R. Sack, June 1986.

SCS-TR-100 **Sums of Lexicographically Ordered Sets**
M.D. Atkinson, A. Negro, and N. Santoro, May 1987.

SCS-TR-102 **Computing on a Systolic Screen: Hulls, Contours, and Applications**
F. Dehne, J.-R. Sack and N. Santoro, October 1986.

SCS-TR-103 **Stochastic Automata Solutions to the Object Partitioning Problem**
B.J. Oommen and D.C.Y. Ma, November 1986.

SCS-TR-104 **Parallel Computational Geometry and Clustering Methods**
F. Dehne, December 1986.

SCS-TR-105 **On Adding *Constraint Accumulation* to Prolog**
Wilf R. LaLonde, January 1987.

SCS-TR-107 **On the Problem of Multiple Mobile Robots Cluttering a Workspace**
B. J. Oommen and I. Reichstein, January 1987.

SCS-TR-108 **Designing Families of Data Types Using Exemplars**
Wilf R. LaLonde, February 1987.

SCS-TR-109 **From Rings to Complete Graphs - Q(n logn) to Q(n) Distributed Leader Election**
Hagit Attiya, Nicola Santoro and Shmuel Zaks, March 1987.

SCS-TR-110 **A Transputer Based Adaptable Pipeline**
Anirban Basu, March 1987.

SCS-TR-111 **Impact of Prediction Accuracy on the Performance of a Pipeline Computer**
Anirban Basu, March 1987.

SCS-TR-112 **ϵ-Optimal Discretized Linear Reward-Penalty Learning Automata**
B.J. Oommen and J.P.R. Christensen, May 1987.

SCS-TR-113 **Angle Orders, Regular n-gon Orders and the Crossing Number of a Partial Order**
N. Santoro and J. Urrutia, June 1987.

SCS-TR-114 **An Optimal Algorithm for Detecting Weak Visibility of a Polygon**
Jorg-R. Sack and Subhash Suri, December 1986.

Carleton University, School of Computer Science
Bibliography of Technical Reports

SCS-TR-115 **Time is Not a Healer: Impossibility of Distributed Agreement in Synchronous Systems with Random Omissions**
N. Santoro, June 1987.

SCS-TR-116 **A Practical Algorithm for Boolean Matrix Multiplication**
M.D. Atkinson and N. Santoro, June 1987.

SCS-TR-117 **Recognizing Polygons, or How to Spy**
James A. Dean, Andrzej Lingas and Jörg-R. Sack, August 1987.

SCS-TR-118 **Stochastic Rendezvous Network Performance - Fast, First-Order Approximations**
J.E. Neilson, C.M. Woodside, J.W. Miernik, D.C. Petriu, August 1987.

SCS-TR-120 **Searching on Alphanumeric Keys Using Local Balanced Tree Hashing**
E.J. Otoo, August 1987.

SCS-TR-121 **An O($\sqrt{n}$) Algorithm for the ECDF Searching Problem for Arbitrary Dimensions on a Mesh-of-Processors**
Frank Dehne and Ivan Stojmenovic, October 1987.

SCS-TR-122 **An Optimal Algorithm for Computing the Voronoi Diagram on a Cone**
Frank Dehne and Rolf Klein, November 1987.

SCS-TR-123 **Solving Visibility and Separability Problems on a Mesh-of-Processors**
Frank Dehne, November 1987.

SCS-TR-124 **Deterministic Optimal and Expedient Move-to-Rear List Organizing Strategies**
B.J. Oommen, E.R. Hansen and J.I. Munro, October 1987.

SCS-TR-125 **Trajectory Planning of Robot Manipulators in Noisy Workspaces Using Stochastic Automata**
B.J. Oommen, S. Sitharam Iyengar and Nicte Andrade, October 1987.

SCS-TR-126 **Adaptive Structuring of Binary Search Trees Using Conditional Rotations**
R.P. Cheetham, B.J. Oommen and D.T.H. Ng, October 1987.

SCS-TR-127 **On the Packet Complexity of Distributed Selection**
A. Negro, N.Santoro and J. Urrutia, November 1987.

SCS-TR-128 **Efficient Support for Object Mutation and Transparent Forwarding**
D.A. Thomas, W.R. LaLonde and J. Duimovich, November 1987.

SCS-TR-129 **Eva: An Event Driven Framework for Building User Interfaces in Smalltalk**
Jeff McAffer and Dave Thomas, November 1987.

SCS-TR-130 **Application Frameworks: Experience with MacApp**
John R. Pugh and Cefee Leung, December 1987.

SCS-TR-131 **An Efficient Window Based System Based on Constraints**
Danny Epstein and Wilf R. LaLonde, March 1988.

SCS-TR-132 **Building a Backtracking Facility in Smalltalk Without Kernel Support**
Wilf R. LaLonde and Mark Van Gulik, March 1988.

Carleton University, School of Computer Science
Bibliography of Technical Reports

SCS-TR-134 **Separating a Polyhedron by One Translation from a Set of Obstacles**
Otto Nurmi and Jörg-R. Sack, December 1987.

SCS-TR-135 **An Optimal VLSI Dictionary Machine for Hypercube Architectures**
Frank Dehne and Nicola Santoro, April 1988.

SCS-TR-136 **Optimal Visibility Algorithms for Binary Images on the Hypercube**
Frank Dehne, Quoc T. Pham and Ivan Stojmenovic, April 1988.

SCS-TR-137 **An Efficient Computational Geometry Method for Detecting Dotted Lines in Noisy Images**
F. Dehne and L. Ficocelli, May 1988.

SCS-TR-138 **On Generating Random Permutations with Arbitrary Distributions**
B. J. Oommen and D.T.H. Ng, June 1988.