A NEW ALGORITHM FOR TESTING THE REGULARITY OF A PERMUTATION GROUP

V. Acciaro and M.D. Atkinson

SCS-TR-183, NOVEMBER 1990

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

A new algorithm for testing the regularity of a permutation group

V. Acciaro
Istituto di Scienze dell' Informazione
Bari, Italy
and
M.D. Atkinson
School of Computer Science
Carleton University, Ottawa
Canada K1S 5B6

Abstract. An algorithm is presented for testing whether the group G generated by a given set of m permutations of degree n is regular. The algorithm has a worst case time complexity of $O(m^2n)$. Then a probabilistic modification is proposed which is designed to reduce the execution time in cases where the generating set is redundant. The group parameters which control the execution time of the modified algorithm are discussed.

1. Introduction

For computational purposes it is attractive to represent finite groups as permutation groups on a set $\Omega = \{1, 2, ..., n\}$, the group being specified by a set A of m generating permutations. Of course, by Cayley's theorem, every finite group can be represented in this way. Such a representation can be very compact since most finite groups require only a small number of generators and have permutation representations with n considerably smaller than their order.

Algorithms for computing with the group generated by a given set of permutations have been studied for many years (see [3] for a survey). Initially the main concerns were with practical and efficient programs but in recent years the complexity of such algorithms has been investigated. Some of the algorithms require a strong generating set [7], or a complete labelled branching [5], and these structures cannot, at present, be computed with worst case complexity less than n^5. On the other hand very much more efficient algorithms are known for testing transitivity (folk lore dating from the 1960's) and primitivity [1]. Besides these two problems there is one other notable problem which can be solved without the use of a strong generating set:- the question of whether a permutation group is regular (recall that a transitive group is said to be *regular* if its degree and order are equal or, equivalently, its point stabiliser is the identity subgroup). In lectures given in Oxford in 1973 Sims presented an algorithm for solving this problem whose execution time was $O(m^2n)$ and it eventually appeared in print in [4]. The purpose of this paper is to describe a new algorithm for the regularity problem whose execution time is similarly bounded although in actual practice the new algorithm has a slightly superior performance. Both algorithms therefore deteriorate quadratically as the number of generators for G rises and so behave poorly for groups described by a redundant set of generators. We shall show how our algorithm can be modified to largely avoid this degradation in performance. We discuss the factors which influence the execution time of the modified algorithm and present results which indicate that its execution time is, in many cases, O(mkn) where k is the minimal number of generators required by G.

2. The new regularity algorithm

We shall assume that the group G generated by the set A of m given permutations on $\{1, 2, ..., n\}$ is transitive. This is not a drawback since a transitivity test is very efficient (complexity O(mn)). The new algorithm (and the Sims algorithm also) could be used to test for regularity on each orbit (that is, semi-regularity).

The key ingredient of our algorithm is an efficient implementation of a function that we call EQUALSTABILISERS(α,ε) which decides whether the two stabilisers G_α and G_ε are equal. To see how this may be done recall that, if $\{u_\beta \mid \beta \in \Omega\}$ is a set of permutations with the property that $\alpha u_\beta = \beta$ then this set is a set of representatives for the cosets of G_α, and the set $\{u_\beta g u_{\beta g}^{-1} \mid \beta \in \Omega, g \in A\}$ is the set of Schreier generators for G_α. The coset representatives and Schreier generators may be calculated by the following well known algorithm.

```
Input:      a set A of permutations on $\Omega$ generating a group G, and an element $\alpha \in \Omega$
Output:     representatives $\{u_\beta\}$ for the cosets of $G_\alpha$, and generators for $G_\alpha$
            $\Gamma := \{\alpha\}$;     $\Delta := \{\}$;     $u_\alpha := 1$
            repeat
                choose $\beta \in \Gamma - \Delta$
                for each $g \in A$ do
                    $\gamma := \beta g$
                    if $\gamma \notin \Gamma$ then
                $u_\gamma := u_\beta g$;     $\Gamma := \Gamma \cup \{\gamma\}$
            **endif**
            compute the Schreier generator $u_\beta g u_\gamma^{-1}$
        **endfor**
        $\Delta := \Delta \cup \{\beta\}$
    **until** $\Gamma = \Delta$
```

The correctness of this algorithm follows from the observation that it keeps invariant the assertions

1. $\Delta^g \subseteq \Gamma \subseteq \alpha^G$ for all generators g, and
2. for all $\gamma \in \Gamma$, $\alpha u_\gamma = \gamma$

If, to this algorithm, we added the statement

if $\varepsilon(u_\beta g u_\gamma^{-1}) \neq \varepsilon$ **then** return(*false*)

immediately after $u_\beta g u_\gamma^{-1}$ has been calculated it would return with *false* whenever any generator of G_α failed to fix ε and could only exit naturally if $G_\alpha = G_\varepsilon$. So the algorithm would decide the $G_\alpha \neq G_\varepsilon$ question.

But, as it stands, this procedure would take time on the order of mn^2. We can effect a saving of a factor of n by maintaining an array T with the property that $T[\beta] = \varepsilon u_\beta$. For we have

$$\varepsilon(u_\beta g u_\gamma^{-1}) = \varepsilon \text{ if and only if } T[\beta]g = T[\gamma]$$

At the point that u_γ is calculated as $u_\beta g$ we can define $T[\gamma]$ since $T[\gamma] = \varepsilon u_\gamma = \varepsilon u_\beta g = T[\beta]g$. The test that $T[\beta]g = T[\gamma]$ does not require that the set of coset representatives be calculated at all and we can recast the test that $G_\alpha = G_\varepsilon$ as

EQUALSTABILISERS(α, ε)

Input: a set A of permutations on Ω generating a group G, and elements $\alpha, \varepsilon \in \Omega$

Output: *true* if $G_\alpha = G_\varepsilon$, *false* otherwise

```
    $\Gamma := \{\alpha\}$;     $\Delta := \{\}$;     $T[\alpha] := \varepsilon$
    **repeat**
        choose $\beta \in \Gamma - \Delta$
        **for each** $g \in A$ **do**
            $\gamma := \beta g$
```

```
                if $\gamma \notin \Gamma$ then
                        $T[\gamma] := T[\beta]g$;        $\Gamma := \Gamma \cup \{\gamma\}$
                endif
                if $T[\beta]g \neq T[\gamma]$ then return(*false*)
            endfor
            $\Delta := \Delta \cup \{\beta\}$
        until $\Gamma = \Delta$
        return(*true*)
```

The choosing of $\beta \in \Gamma - \Delta$ can be done in constant time by listing the elements of Γ in an array of which Δ is some initial segment and defining β to be the element of Γ which follows the last element of Δ. Also the test that $\gamma \notin \Gamma$ can be done in constant time by keeping the characteristic vector of the subset Γ of Ω. Consequently the EQUALSTABILISERS function takes time O(mn).

Using this function the regularity algorithm is easy to describe:

```
Algorithm REGULAR
input:     a set A of permutations on $\Omega$
output:    a boolean value (*true* or *false*)
           for each $g \in A$ do
               if not EQUALSTABILISERS($\alpha$, $\alpha g$) then return(*false*)
           endfor
           return(*true*)
```

Lemma 1 If A generates a transitive group G the algorithm REGULAR returns *true* if and only if G is regular. The execution time of the algorithm is $O(m^2n)$.

Proof. The algorithm can return *false* only if it discovers that $G_\alpha \neq G_{\alpha g}$ which certainly implies that G is not regular. If it returns *true* it is because $G_\alpha = G_{\alpha g}$ for all generators g, and therefore, since G is transitive, $G_\alpha = G_\beta$ for all $\beta \in \Omega$; it follows that $G_\alpha = 1$ and so G is regular. It is immediate from the execution time of EQUALSTABILISERS that the execution time is $O(m^2n)$.

We now suppose that m is somewhat larger than the minimal number of generators of G and that the quadratic dependence on m of the REGULAR algorithm results in inconveniently long execution times. We therefore seek to reduce the number of times the EQUALSTABILISERS function is called by modifying the REGULAR algorithm.

The modified algorithm works with equivalence relations on Ω and these are most conveniently represented by their collection of (equivalence) classes. A G-invariant equivalence relation is, of course, just a block system for G. The equivalence relation of most interest is the one defined by

$$\sigma \approx \tau \text{ if and only if } G_\sigma = G_\tau$$

and we shall denote it by Σ^*. Clearly G is regular if and only if $|\Sigma^*| = 1$ (that is, Σ^* is the universal relation with just one class). The algorithm tests regularity by constructing Σ^*.

Given two equivalence relations on Ω we shall say that the second is *coarser* than the first if the classes of the first relation are subsets of the classes of the second. Throughout the algorithm we shall be manipulating a variable Σ which represents an equivalence relation. Initially Σ is the trivial equivalence relation and it is replaced by coarser and coarser relations as the algorithm proceeds. It is convenient to denote the class to which a point $\theta \in \Omega$ belongs by Σ_θ.

The modified algorithm is as follows:

```
Algorithm  MODIFIED_REGULAR
input:     a set A of permutations on $\Omega$
output:    a boolean value (true or false)
           $\Sigma := \{ \{\theta\} \mid \theta \in \Omega \}$
           while $|\Sigma| > 1$ do
               choose $\varepsilon \in \Omega - \Sigma_\alpha$
               if EQUALSTABILISERS($\alpha$, $\varepsilon$) then
                   CLOSE($\alpha$, $\varepsilon$)
               else
                   return(false)
               endif
           endwhile
           return(true)
```

Here the function CLOSE(α, ε) transforms Σ into the least coarse G-invariant equivalence relation strictly coarser than Σ in which α and ε are equivalent. An implementation of it is described below.

Lemma 2 The algorithm MODIFIED_REGULAR returns *true* if and only if G is regular.

Proof. When the algorithm returns *false* it is because it has discovered that $G_\alpha \neq G_\varepsilon$ for some α, ε and this certainly means that G is not regular. When the algorithm returns *true* it is because Σ has become the universal equivalence relation. If we can show that, throughout the algorithm, Σ^* is coarser than Σ it will follow that Σ^* is universal and hence that G is regular. Initially, Σ^* (indeed every equivalence relation) is coarser than Σ. When CLOSE(α, ε) modifies Σ, the points α and ε are equivalent in Σ^* and so Σ^* is a G-invariant equivalence relation coarser than Σ in which α and ε are equivalent. Since CLOSE constructs the least coarse equivalence relation with this property Σ^* remains coarser than the new equivalence relation Σ.

We now consider the implementation and execution time of the CLOSE(α, ε) function. We shall use a method similar to that given in [2] which in turn was a refinement of the algorithm of [1]. This method requires two operations, FIND and UNION, to be applied to the current equivalence relation Σ. The operation FIND(θ) returns a distinguished point within Σ_θ; this distinguished point serves as the name of this equivalence class. The operation UNION(θ,ϕ) replaces Σ by the coarser equivalence relation obtained by uniting the classes Σ_θ and Σ_ϕ. The classes of Σ are represented by rooted trees defined by a 'father' function f:- f(ϕ) is the node immediately above ϕ in the set of trees (or ϕ itself if ϕ is a root). This representation allows an efficient implementation of the FIND and UNION operations. The FIND(ϕ) operation uses the function f to trace a path from ϕ to the root of its tree. The UNION(ϕ,ψ) operation inserts a branch between the roots of the trees containing ϕ and ψ (if these trees are different). The weighting and path compression rules described in [8] are used. The function CLOSE(α, ε) also uses a set C of tree branches which are represented as pairs of end points.

```
CLOSE($\alpha$, $\varepsilon$)
input:      a G-invariant equivalence relation on $\Omega$, and two inequivalent points $\alpha$, $\varepsilon$
output:     another G-invariant equivalence relation on $\Omega$
            $\alpha^*$ := FIND($\alpha$);        $\varepsilon^*$ := FIND($\varepsilon$)
            UNION($\alpha^*$, $\varepsilon^*$)
            C := {($\alpha^*$, $\varepsilon^*$)}
            repeat
                delete some ($\gamma$,$\delta$) from C
                for each g $\in$ A do
```

```
                    $\phi := \gamma g$;          $\psi := \delta g$
                    $\sigma :=$ FIND($\phi$);  $\tau :=$ FIND($\psi$)
                    **if** $\sigma \neq \tau$ **then**
                            UNION($\sigma,\tau$)
                            add ($\sigma,\tau$) to C
                    **endif**
            **endfor**
    **until** C is empty
    return($\Sigma$)
```

Lemma 3 CLOSE(α, ε) returns the least coarse G-invariant equivalent relation which contains all the equivalences of the input relation and in which α and ε are also equivalent.

Proof. CLOSE(α, ε) manipulates a set of trees using UNION and FIND. Let us suppose, temporarily, that path compression is not performed in the FIND operations. Then it is clear that the pairs in the set C all represent tree branches. From the body of the **for** statement it follows that, at the end of each iteration of the **repeat** loop, we have the following condition:

For every branch (σ,τ) of the set of trees which represent the current equivalence relation one of the following holds:

1. $(\sigma,\tau) \in$ C, or

2. σg and τg lie in the same tree, for every $g \in$ A.

But on termination C is empty and so, for all tree branches (σ,τ), σg and τg lie in the same tree, for every $g \in$ A. Thus the output equivalence relation is indeed G-invariant and, because of the initial steps in the CLOSE function, α and ε are equivalent. It is also clear that the output equivalence relation is the least coarse equivalence relation with these properties because the algorithm only joins two classes together when forced to do so to fulfil the properties. Finally, note that the equivalence relation computed by CLOSE(α, ε) is unaffected by how FIND is implemented so that the assumption above that path compression is not used can be removed.

Lemma 4 The execution time of the algorithm MODIFIED_REGULAR is bounded above by O(kmn + mnα(n)) where k is the number of times that EQUALSTABILISERS is called.

Proof. During all executions of CLOSE(α, ε) at most n-1 UNION operations can be performed because each one decreases the number of classes by 1. Thus a total of at most n-1 pairs are placed in C and so at most 2(n-1)m FIND operations are performed in all. Thus, according to the main result of [8], the total time spent in all calls on CLOSE(α, ε) is $O(mn\alpha(n))$, $\alpha(n)$ being a very slow growing function related to the inverse of Ackerman's function. The other significant contribution to the execution time is the time spent in the EQUALSTABILISERS function and, as we have seen, EQUALSTABILISERS takes time O(mn).

It is clear that the quantity k defined in the Lemma 4 cannot exceed the number of divisors of n since each call to CLOSE(α,ε) transforms Σ into a block system whose number of blocks is a divisor of what it was previously. Moreover a simple modification to the algorithm along the lines of the initial version ensures that $k \leq m$. Rather than choose each point ε arbitrarily from $\Omega — \Sigma_\alpha$ we could choose ε from among $\{\alpha g \mid g \in A\}$. This makes very little difference to the performance of the algorithm in practice because it appears that k is usually very close to the minimal number of elements required to generate G anyway; therefore no effort is required to ensure $k \leq m$. In the next section we attempt to partially explain this phenomenon.

3. Expected Time Complexity

We concentrate on the case that G is regular since our modified algorithm (and Sims' algorithm) usually runs considerably faster for non-regular groups.

Let $L_d(G)$ denote the number of d-tuples of elements of G which are a generating set for G and define $\lambda_d(G)=\frac{L_d(G)}{|G|^d}$ to be the probability that a sequence of d elements chosen at random generate G. We then have $|G|^d = \sum_{H \leq G} L_d(H)$ an equation which is often useful for determining $L_d(H)$ for small groups H. The probability that a sequence of group elements $x_1,x_2,\ldots,x_{d-1},x_d$ generates G and $x_1,x_2,\ldots,x_{d-1}$ does not generate G is $\lambda_d(G)-\lambda_{d-1}(G)$; therefore the expected number of elements of G which have to be generated at random before a set of generators is found is $e(G)= \sum_{d=1}^{\infty} d(\lambda_d(G)-\lambda_{d-1}(G))$.

Lemma 5 The expected execution time of the algorithm MODIFIED_REGULAR when applied to a regular group G is $O(mn(\alpha(n)+e(G))$.

Proof. In a regular group elements $g_1,\dots,g_d$ generate G if and only if the smallest block containing $\alpha g_1,\dots,\alpha g_d$ is Ω. Thus the expected number of elements of Ω that would have to be generated before the smallest block containing them is Ω itself is e(G). In fact the points ε are chosen slightly more carefully since each ε is chosen to be outside the block containing the previously chosen points. Thus the expected number of times that EQUALSTABILISERS is called is at most e(G). The expected execution time is therefore the sum of the time $O(mn\alpha(n))$ required by the CLOSE operations and the time O(mn e(G))required by the EQUALSTABILISERS operations.

The quantity e(G) is, of course, very difficult to calculate and so upper estimates of e(G) need to be found. Let d be any integer. From the random process which generates $x_1,x_2,\dots$ we can define a Bernoulli process in which the independent trials are the successive blocks of d elements from this sequence, a trial being successful if it generates G. The expected number of trials required before one is successful is $1/\lambda_d(G)$ and hence $e(G)\leq d/\lambda_d(G)$. Kantor and Lubotzky [6] have recently proved that $\lambda_2(G) \to 1$ as $|G| \to \infty$ for classical simple groups G. At the opposite extreme one can calculate much more exactly in p-groups:

Lemma 6 Let G be a p-group with minimal number of generators d. Then,

(i) $$\lambda_d(G) = \prod_{i=1}^{d}(1 - p^{-i}), \text{ and}$$

(ii) $$\text{if } k\geq 0,\ \lambda_{d+k}(G) = \lambda_d \prod_{i=1}^{k} \frac{p^i - p^{-d}}{p^i - 1}$$

Proof. Let $\Phi(G)$ be the Frattini subgroup of G. Then $G/\Phi(G)$ is an elementary abelian group of order p^d and elements $x_1,\dots,x_r$ of G generate G if and only if their images in $G/\Phi(G)$ generate $G/\Phi(G)$. Thus it is sufficient to prove the lemma for an elementary abelian group of order p^d.

Define μ_{rs} to be the probability that the independently chosen group elements $x_1,\dots,x_r$ generate a subgroup of order p^s. Then, clearly, $\mu_{rs} = 0$ if $r<s$ and $\mu_{r0} = p^{-dr}$. Moreover, if $r\geq s$,

$$\mu_{rs} \quad = \quad P(|<x_1,\dots,x_r>| = p^s \text{ and } |<x_1,\dots,x_{r-1}>| = p^s)$$

$$
\begin{aligned}
&+ P(|\langle x_1,\dots,x_r\rangle| = p^s \text{ and } |\langle x_1,\dots,x_{r-1}\rangle| = p^{s-1})\\
= \quad & P(|\langle x_1,\dots,x_{r-1}\rangle| = p^s \text{ and } x_r \in \langle x_1,\dots,x_{r-1}\rangle)\\
&+ P(|\langle x_1,\dots,x_{r-1}\rangle| = p^{s-1} \text{ and } x_r \notin \langle x_1,\dots,x_{r-1}\rangle)\\
= \quad & \mu_{r-1,s}p^s/p^d + \mu_{r-1,s-1}(1-p^{s-1}/p^d)
\end{aligned}
$$

It may be verified that the solution to this recurrence is

$$\mu_{ss} = \prod_{i=d-k+1}^{d}(1-p^{-i}), \text{ and}$$

$$\mu_{s+k,s} = \frac{\mu_{ss}}{p^{dk}}\prod_{i=1}^{k}\frac{p^{s+i}-1}{p^i-1}, \text{ if } k\geq 0.$$

The lemma follows immediately since $\lambda_{d+k}(G) = \mu_{d+k,d}$.

The formulae in this lemma are rather complicated but there is a handy estimate for λ_d, namely, $\frac{p-1}{p} \geq \lambda_d(G) \geq \frac{p-2}{p-1}$ which demonstrates that $\lambda_d(G) \to 1$ as $p \to \infty$. The upper bound follows immediately from the expression for λ_d and the lower bound is because, for all $k\geq 0$,

$$\lambda_{d+k}(G) = \prod_{i=k+1}^{k+d}(1-p^{-i}) \geq \prod_{i=1}^{\infty}(1-p^{-i}) = \frac{1}{1+\sum_{i=1}^{\infty}p(n)p^{-n}} \geq \frac{1}{1+\frac{1}{2}\sum_{i=1}^{\infty}(2/p)^n} = \frac{p-2}{p-1}$$

where $p(n)$ is the partition function and we have used its generating function and the inequality $p(n) \leq 2^{n-1}$.

As an example of the numerical effectiveness of the formulae in the lemma we note that, for a 4-generator p-group with $p\geq 5$, $\lambda_{6,4}(G)>0.99$.

Acknowledgment. We thank Dr. G. Butler for his helpful comments on an early draft of this paper.

References

1. M.D. Atkinson: An algorithm for finding the blocks of a permutation group, Math. Comp. 29 (1975), 911-913.

2. M.D. Atkinson, R. A. Hassan, M.P. Thorne: Group theory on a micro-computer, in Computational Group Theory (M.D. Atkinson, Ed.), Academic Press, London-New York, 1984, pp.275-280.

3. M.D. Atkinson: A survey of algorithms for handling permutation groups, School of Computer Science Technical Report SCS-TR164, Carleton University, Ottawa, January 1990.

4. J.J. Cannon: A computational toolkit, in Proceedings of the Rutgers Group Theory Year 1983-1984 (Editors M. Aschbacher, D. Gorenstein, R. Lyons, M.E. O'Nan, C.C. Sims, W. Feit), Cambridge University Press, 1984, pp. 1-18.

5. M. Jerrum: A compact representation for permutation groups, J. Algorithms 7 (1986), 60-78.

6. W.M. Kantor, A. Lubotzky: The probability of generating a finite classical group, Geometriae Dedicata 36 (1990), 67-87.

7. C.C. Sims: Computation with permutation groups, in Proc. 2nd Symposium on Symbolic and Algebraic Manipulation (S.R. Petrick, Ed.), 23-28, ACM, New York, 1971.

8. R.E. Tarjan: On the efficiency of a good but not linear set merging algorithm, J. ACM 22 (1975), 215-265.

School of Computer Science, Carleton University
Bibliography of Technical Reports

SCS-TR-141	**Graphically Defining Simulation Models of Concurrent Systems** H. Glenn Brauen and John Neilson, September 1988
SCS-TR-142	**An Algorithm for Distributed Mutual Exclusion on Arbitrary Networks** H. Glenn Brauen and John E. Neilson, September 1988
	SCS-TR-143 to 146 are unavailable.
SCS-TR-147	**On Transparently Modifying Users' Query Distributions** B.J. Oommen and D.T.H. Ng, November 1988
SCS-TR-148	**An O(N Log N) Algorithm for Computing a Link Center in a Simple Polygon** H.N. Djidjev, A. Lingas and J.-R. Sack, July 1988 Available in STACS 89, 6th Annual Symposium on Theoretical Aspects of Computer Science, Paderborn, FRG, February 16-18, 1989, Lecture Notes in Computer Science, Springer-Verlag No. 349
SCS-TR-149	**Smallscript: A User Programmable Framework Based on Smalltalk and Postscript** Kevin Haaland and Dave Thomas, November 1988
SCS-TR-150	**A General Design Methodology for Dictionary Machines** Frank Dehne and Nicola Santoro, February 1989
SCS-TR-151	**On Doubly Linked List ReOrganizing Heuristics** D.T.H. Ng and B. John Oommen, February 1989
SCS-TR-152	**Implementing Data Structures on a Hypercube Multiprocessor, and Applications in Parallel Computational Geometry** Frank Dehne and Andrew Rau-Chaplin, March 1989
SCS-TR-153	**The Use of Chi-Squared Statistics in Determining Dependence Trees** R.S. Valiveti and B.J. Oommen, March 1989
SCS-TR-154	**Ideal List Organization for Stationary Environments** B. John Oommen and David T.H. Ng, March 1989
SCS-TR-155	**Hot-Spot Contention in Binary Hypercube Networks** Sivarama P. Dandamudi and Derek L. Eager, April 89
SCS-TR-156	**Some Issues in Hierarchical Interconnection Network Design** Sivarama P. Dandamudi and Derek L. Eager, April 1989
SCS-TR-157	**Discretized Pursuit Linear Reward-Inaction Automata** B.J. Oommen and Joseph K. Lanctot, April 1989
SCS-TR-158 (revised)	**Parallel Fractional Cascading on a Hypercube Multiprocessor** Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, May 1989 (Revised April 1990)
SCS-TR-159	**Epsilon-Optimal Stubborn Learning Mechanisms** J.P.R. Christensen and B.J. Oommen, June 1989
SCS-TR-160	**Disassembling Two-Dimensional Composite Parts Via Translations** Doron Nussbaum and Jörg-R. Sack, June 1989
SCS-TR-161 (revised)	**Recognizing Sources of Random Strings** R.S. Valiveti and B.J. Oommen, January 1990 Revised version of SCS-TR-161 "On the Data Analysis of Random Permutations and its Application to Source Recognition", published June 1989
SCS-TR-162	**An Adaptive Learning Solution to the Keyboard Optimization Problem** B.J. Oommen, R.S. Valiveti and J. Zgierski, October 1989
SCS-TR-163	**Finding a Central Link Segment of a Simple Polygon in O(N Log N) Time** L.G. Alexandrov, H.N. Djidjev, J.-R. Sack, October 1989

SCS-TR-164	**A Survey of Algorithms for Handling Permutation Groups** M.D. Atkinson, January 1990
SCS-TR-165	**Key Exchange Using Chebychev Polynomials** M.D. Atkinson and Vincenzo Acciaro, January 1990
SCS-TR-166	**Efficient Concurrency Control Protocols for B-tree Indexes** Ekow J. Otoo, January 1990
SCS-TR-167	**A Hierarchical Stochastic Automaton Solution to the Object Partitioning Problem** B.J. Oommen, January 1990
SCS-TR-168	**Adaptive List Organizing for Non-stationary Query Distributions. Part I: The Move-to-Front Rule** R.S. Valiveti and B.J. Oommen, January 1990
SCS-TR-169	**Trade-Offs in Non-Reversing Diameter** Hans L. Bodlaender, Gerard Tel and Nicola Santoro, February 1990
SCS-TR-170	**A Massively Parallel Knowledge-Base Server using a Hypercube Multiprocessor** Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, April 1990
SCS-TR-171	**Parallel Processing of Quad Trees on the Hypercube (and PRAM)** Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, April 1990
SCS-TR-172	**A Note on the Load Balancing Problem for Coarse Grained Hypercube Dictionary Machines** Frank Dehne and Michel Gastaldo, May 1990
SCS-TR-173	**Self-Organizing Doubly-Linked Lists** R.S. Valiveti and B.J. Oommen, May 1990
SCS-TR-174	**A Presortedness Metric for Ensembles of Data Sequences** R.S. Valiveti and B.J. Oommen, May 1990
SCS-TR-175	**Separation of Graphs of Bounded Genus** Ljudmil G. Aleksandrov and Hristo N. Djidjev, May 1990
SCS-TR-176	**Edge Separators of Planar and Outerplanar Graphs with Applications** Krzystof Diks, Hristo N. Djidjev, Ondrej Sykora and Imrich Vrto, May 1990
SCS-TR-177	**Representing Partial Orders by Polygons and Circles in the Plane** Jeffrey B. Sidney and Stuart J. Sidney, July 1990
SCS-TR-178	**Determining Stochastic Dependence for Normally Distributed Vectors Using the Chi-squared Metric** R.S. Valivetei and B.J. Oommen, July 1990
SCS-TR-179	**Parallel Algorithms for Determining K-width- Connectivity in Binary Images** Frank Dehne and Susanne E. Hambrusch, September 1990
SCS-TR-180	**A Workbench for Computational Geometry (WOCG)** P. Epstein, A. Knight, J. May, T. Nguyen, and J.-R. Sack, September 1990
SCS-TR-181	**Adaptive Linear List Reorganization under a Generalized Query System** R.S. Valiveti, B.J. Oommen and J.R. Zgierski, October 1990
SCS-TR-182	**Breaking Substitution Cyphers using Stochastic Automata** B.J. Oommen and J.R. Zgierski, October 1990
SCS-TR-183	**A New Algorithm for Testing the Regularity of a Permutation Group** V. Acciaro and M.D. Atkinson, November 1990