Article Submitted to Journal of Symbolic Computation

Computing Isomorphisms of Association Schemes and its Application

Izumi Miyamoto[1]

[1]*Department of Computer Science, Yamanashi University, 4-3-11, Takeda, Kofu, 400-8511, Japan*

1. Introduction

Using computers, Hanaki and I were classifying association schemes of small order for a past few years (Hanaki and Miyamoto (1998,1998,2000),Hanaki (1999)). In the present paper we will discuss about computing isomorphisms of association schemes. We compute isomorphisms of an association scheme using its relation matrix defined in the following section. A relation matrix can be seen an adjacency matrix of a labeled graph. If we consider an isomorphism from an association scheme to itself, the isomorphism may be given by a permutation on the set of vertices of the labeled graph. In our case we generally consider that an isomorphism may permute also the labels of the graph. The most familiar examples of association schemes are given by transitive permutation groups on the set of vertices. Then an element of the transitive group is an isomorphism of the former type. An element of the normalizer of the group is also an isomorphism and it may permute the labels. The author gives an algorithm to speed up to compute normalizers of permutation groups as an application of computing isomorphisms of association schemes in Miyamoto (2000). In the present paper we will also give some more inspection about this algorithm. In the following section we will define association schemes and will introduce necessary properties. For more basic properties, readers may refer to Bannai and Ito (1984). For graph isomorphisms and computing normalizers readers may also refer to Mckay (1981) and Theißen (1997).

A program computing isomorphisms of labeled graphs is not familiar and if we do not consider the algebraic property of association schemes, we have to compute combinatorially large numbers of permutations on the labels in some graphs. In fact our old program was sufficient to compute an isomorphism from one association scheme to another in the classification of the isomorphism classes stated above but sometimes it took too much time to compute all the isomorphisms of an association scheme to itself. The labeled graph given by an association scheme is regular with respect to each label. This restricts the permutations of isomorphisms on the labels. If an association scheme is defined by a transitive group

of order equal to its degree (i.e. regular permutation group) or equal to twice of its degree, then the valency of each label is equal to one in the former case or equal to one or two in the latter case. As we mentioned above, isomorphisms are related to the normalizer of a group if the association scheme is defined by it. An association scheme defined by a regular group is just the regular representation of the group. So it is purely a group theoretic object and the group algorithm programming language `GAP4` computes its normalizer rather quickly in this case. But in the other case neither `GAP4` nor Magma2.5 can compute the normalizer so quickly as is shown in tables in Section 4. Although we used some more properties like orders of group elements in our old program, the performance was not good. We improved our program to speed up the computation of isomorphisms using the algebraic property of association schemes. Our program is written in the group algorithm programming language `GAP`.

2. Association schemes

Let $X = \{x_1,\ x_2, \cdots, x_n\}$ be the set of vertices. Then an association scheme $(X, \{R_i\}_{0\leq i\leq d})$ is defined as follows.

Definition: $(X, \{R_i\}_{0\leq i\leq d})$ is an *association scheme* if and only if

1. $R_0 = \{(x, x)|x \in X\}$,
2. $\{R_0, R_1, \cdots, R_d\}$ is a partition of $X \times X$,
3. for all R_i there exists $R_{i'}$ such that $R_{i'} = \{(y, x)|(x, y) \in R_i\}$,
4. for all R_i, R_j, R_k and for all $(x, y) \in R_k$, there exists a constant number p_{ijk} such that

$$p_{ijk} = \#\{z \in X|(x, z) \in R_i, (z, y) \in R_j, (x, y) \in R_k\}.$$

Then the number n of the vertices X is called the order of the association scheme and R_i is called a relation. If we fix an arrangement of the vertices $X = \{x_1,\ x_2, \cdots, x_n\}$, we can define matrices $A_0,\ A_1, \cdots, A_d$ by

$$\text{the } (j, k)\text{-entry of } A_i = \begin{cases} 1 & \text{if } (x_j, x_k) \in R_i \\ 0 & \text{otherwise} \end{cases}$$

The matrix A_i is called an adjacency matrix of the association scheme The adjacency matrices satisfy the following properties and these properties are equivalent to the definition of an association scheme.

1. A_i has $\{0,\ 1\}-$entries. A_0 =identity matrix.
2. $A_0 + A_1 + \cdots + A_d = J$(all 1 matrix)
3. For all i there exists i' such that ${}^tA_i = A_{i'}$.
4. $A_iA_j = \sum_{0\leq k\leq d}\ p_{ijk}A_k$

Then A_i has constant column and row sum $p_{ii'0}$. The sum is called its *valency*. The matrix $A = \sum_{0 \leq k \leq d} kA_k$ is called the relation matrix of the association scheme. By an association scheme A we mean the association scheme of which relation matrix is A in the present paper.

Let B be another association scheme. Then two association schemes A and B are isomorphic if $B = P^{-1}A^{\sigma}P$ for some permutation matrix P and some permutation σ on the set $\{1, 2, \cdots, d\}$ of relation numbers, where A^{σ} is the matrix obtained by rewriting each entry i to i^{σ} in A. The automorphism group of A is the permutation group generated by the permutation matrices P satisfying $A = P^{-1}AP$ and it is denoted by $Aut(A)$.

3. Computing isomorphisms of association schemes

We consider an association scheme by its relation matrix, when we compute isomorphisms. We say simply an isomorphism if it is an isomorphism from an association scheme to itself. We use a backtrack algorithm to compute isomorphisms. For an association scheme A, firstly we compute $Aut(A)$, which can be seen as the automorphism group of a labeled graph. So we applied usual algorithms computing the automorphism group of a graph to our case. Set $H = Aut(A)$. We simply set $X = \{1, 2, \cdots, n\}$. We consider a chain of stabilizers $H \supseteq H_1 \supseteq \cdots \supseteq H_{1,2,\cdots i-1} \supseteq H_{1,2,\cdots,i-1,i} \supseteq \cdots \supseteq H_{1,2,\cdots,n} = 1$. The basic strategy is to obtain a set of the representatives of the cosets in $H_{1,2,\cdots,i-1}/H_{1,2,\cdots,i-1,i}$ for each $i \in 1, 2 \cdots, n$. So we control the backtrack algorithm not to compute all the elements but to compute only one element in each coset stated above. We will show how to compute an automorphism which is a permutation on $\{1, 2, \cdots, n\}$. Suppose that a portion of a permutation is constructed up to the following stage which may be extended to an automorphism.

$$\begin{pmatrix} i_1 & i_2 & \cdots & i_r & * & \cdots & * \\ 1 & 2 & \cdots & r & r+1 & \cdots & n \end{pmatrix}$$

The set $\{r+1, \cdots, n\}$ is partitioned so that any two vertices x and y in each cell satisfy that both (j, x) and (j, y) lie in a same relation for each j in $\{1, 2, \cdots, r\}$. This means that the (j, x)-entry and the (j, y)-entry of the relation matrix A coincide with each other for each j in $\{1, 2, \cdots, r\}$. This partition is computed in advance and we call this the r-th partition of A. Then the set of the $n - r$ vertices in the $*$-part of the permutation is similarly partitioned so that any two vertices x' and y' in each cell satisfy that the (i_j, x')-entry and the (i_j, y')-entry of the relation matrix A coincide with each other for each i_j in $\{i_1, i_2, \cdots, i_r\}$. If an automorphism is constructed from this portion-of-permutation, the $*$-part should be arranged so that these two partitions coincide with each other, which means that the first r rows of A is invariant under the action of this portion-of-permutation. In fact this was done in our program when i_r was chosen correctly. Now we choose a vertex i_{r+1} from the cell of the latter partition which corresponds to the cell of the r-th partition containing the vertex $r + 1$. Then we

will check the partitions of the next stage. The partition of the next stage is computed by isolating the vertex i_{r+1} from the cell and taking the refinement of the partition of the last stage by the values of the $(i_{r+1}, *)$-entries of A for all $*$ not contained in $\{i_1, i_2, \cdots, i_{r+1}\}$. If this new $*$-part can be re-arranged as above so that the refined partition coincides with the $(r+1)$-th partition, then we will go forward to the next stage. Otherwise we choose a next candidate from the cell and if all the candidates are exhausted, then the backtrack algorithm works. We note that this re-arrangement leaves each cell of the partition in the previous stage invariant. So it preserves the invariance of the first r rows of A by the portion-of-permutation.

Secondly we compute isomorphisms which move relations. There are $d+1$ relations in A. The first relation which is numbered by 0 is the identity relation. So it remains fixed by any isomorphism. Hence an isomorphism is given by a couple of permutations (σ, g), where σ permutes the relations $\{1, 2, \cdots, d\}$ and g permutes $X = \{1, 2, \cdots, n\}$. We note that (σ, g) is an automorphism if and only if it is an isomorphism and $\sigma = 1$. If (σ, g) and (τ, h) are isomorphisms, then $(\sigma\tau, gh)$ is an isomorphism. So the set of all the isomorphisms defines two groups. One consists of the first entries of the (σ, g)'s which acts on the relations and the other consists of the second entries which acts on X. Hence in order to obtain the first group, we only compute a set of the representatives of the similar cosets for the group on the relations as those for the group on X in the previous paragraph. If (σ, g) and (σ, h) are isomorphisms, then $(1, gh^{-1})$ is an automorphism. Since we have computed $Aut(A)$ in the previous paragraph, it is sufficient to compute one isomorphism (σ, g) for each σ from the representatives. We compute each σ as a permutation on the relations which leaves the constants p_{ijk} invariants and then try to find a g which gives an isomorphism by a similar method stated in the previous paragraph. If we can not find a g, then the σ is rejected and we proceed to compute next σ by the backtrack algorithm.

We compute isomorphisms using partitions as above. Suppose that we have computed a permutation on the relations up to the following stage.

$$\begin{pmatrix} i_1 & i_2 & \cdots & i_r & * & \cdots & * \\ 1 & 2 & \cdots & r & r+1 & \cdots & d \end{pmatrix}$$

The set $\{r+1, \cdots, d\}$ is partitioned in advance and the $*$-part is also partitioned and arranged so that these two partitions coincide with each other. The r-th partition of this case is obtained so that any relations u and v in each cell satisfy $p_{ujk} = p_{vjk}$ for all $k \in \{1, 2, \cdots, r\}$ and for all $j \in \{1, 2, \cdots, k\}$ in our program. Choose a relation i_{r+1} in the cell of the latter partition which corresponds to the cell of the former partition containing the relation $r+1$. We compute the partition of the $*$-part of the next stage by isolating the relation i_{r+1} from the cell of the last partition and taking its refinement by the values $p_{*i_t i_{r+1}}$ with $1 \leq t \leq r+1$ for all $*$ not contained in $\{i_1, i_2, \cdots, i_{r+1}\}$. The partition for the set $\{r+2, \cdots, d\}$ is computed similarly in advance and we try to arrange the new $*$-part so that the partitions of this stage coincide with each other, leaving

each cell of the previous partition invariant in order to maintain the coincidence of the first r relations of A by the computing permutation. The first partition is given by the valencies $p_{ii'0}$ and $i = i'$ or not.

4. About the computation of isomorphisms

The notion of an association scheme may not be so familiar. There are no references about algorithms to compute isomorphisms of association schemes.

Let G be a transitive group on the set X. G acts on $G \times G$ by moving (i, j) to (i^g, j^g) in $X \times X$ for $g \in G$. Then the orbits of G on $X \times X$ give an association scheme (cf. Bannai and Ito (1984)). The normalizer of G in a group K is defined by

$$N_K(G) = \{h \in K | h^{-1}Gh = G\}.$$

Hence any element h in the normalizer of G moves each orbit of G on $X \times X$ to another, or it may fix some orbits. So h induces a permutation τ on the set of the orbits of G on $X \times X$ and the couple (τ, h) is an isomorphism of the association scheme defined by G. Hence the group given by the permutations on X of the isomorphisms contains the normalizer. Let A be the association scheme given by G, let N be the normalizer of G in the symmetric group on X and let I be the group of the permutations on X of the isomorphisms of A. Then $N = N_I(G)$. We can compute N by using software package `GAP4` or Magma2.5 directly. Our program computing isomorphisms is written in `GAP4` language. We compute I and then compute $N_I(G)$ using `GAP4`. We examine our algorithm computing isomorphisms by comparing these two methods of computing normalizers. As a consequence the algebraic structure is not used generally in computing normalizers in `GAP` or Magma and our method is from 10 to 100 times as fast as direct computation by `GAP` or Magma in some cases shown below.

Suppose that the order $|G|$ of G is equal to the size $n = |X|$ of X. This means that G is regular on X and all the valency of A is 1. In this case A is merely the regular representation of G. So it is not necessary to consider association schemes intensionally. Hence we considered groups with $|G| = 2|X|$. In this case the valencies of A are at most 2. So if the algebraic structure of the orbits of G on $X \times X$ is not considered, combinatorial explosion may occur. We can construct such groups by taking the coset representation of a group of order $2|X|$ by its subgroup of order 2 not contained in its center. In `GAP` library there are many groups of small orders. In particular we used some groups of order 64 and 128 from the `GAP` library to construct transitive permutation groups of degree $X = 32$ and 64 respectively.

We computed 97 examples of degree 32. The some timings of computations are in the table for degree 32. The number in the column "No" in the table means that the group comes from the group of the number in the `GAP` library. The "I+G" column shows the timings of computation using isomorphisms. The "GAP" column shows the timings of the direct computation by `GAP4` and the

"Magma" column by Magma2.5. The times are in seconds and we used 300MHz Pentium II machine under Linux. The 88 normalizers are computed by "I+G" within from 2 to 5 seconds. The remaining 9 cases are shown in the table. By GAP 12, 22, 39 and 59 normalizers are computed within 1, 2, 5 and 10 seconds respectively. By Magma the timing varies very much in every computation but 27, 55, 61, 65, 75 and 86 cases are within 0.2, 1, 2, 5, 10 and 20 seconds respectively in one trial. The two columns under "Magma" in the table show that the computation was done twice in some cases. We also give a timing table for

Table 1: Table for degree 32

No.	I+G	GAP	Magma		No.	I+G	GAP	Magma	
69	12	6	0.3		36	3	421	77	
73	21	8	8		37	3	3387	5	
75	12	6	0.4		60	4	531	0.2	
77	30	3	5		90	4	236	6	
78	12	24	0.4		91	4	354	8	
89	12	2	0.4		93	4	750	78	
114	8	4	0.2		100	4	885	313	
130	12	11	4831	25	137	4	694	12	27
133	12	9	1	12	139	4	651	300	13

groups of degree 64. We computed by GAP twice and by Magma three or four times. In some cases the computation was interrupted and the timing till the interruption is shown using a symbol >.

Table 2: Table for degree 64

No.	I+G	GAP		Magma			
1201	24	268	264	>2614	706	151	155
1202	26	197	192	1393	683	>15792	723
1203	27	141	141	1155	>25628	15620	5263
1204	26	22105	21602	651	>19686	1177	2784
1205	37	2857	2799	9633	>16751	5	4537
1206	31	241	242	263	6564	995	
1207	290	23723	23430	4013	828	>21584	4000
1208	299	1617	1605	>12406	2142	166	
1209	31	77	76	1527	486	23	
1210	300	3066	3036	8484	481	232	
1211	31	138	137	71	390	549	
1212	164	35263	34830	515	321	>26433	148
1213	37	59	59	11	5926	>10520	5338
1214	34	11197	11072	20	3367	>34105	1040
1215	111	7964	7894	>7915	545	21137	>9181
1216	47	>34412		>33171	>17824	>10767	

5. An application

In this section we will give some more inspection about the algorithm given in Miyamoto (2000). As is shown in Section 3 if G is a transitive permutation group on X, then the normalizer of G is contained in the group of isomorphisms of the association scheme defined by G. If G is not transitive on X, then the constituent of G on each orbit defines an association scheme. Let $\mathcal{O}_1, \mathcal{O}_2, \cdots, \mathcal{O}_r$ be the orbits of G in X and $(G, \mathcal{O}_i)$ denotes the permutation group defined by the action of G in $\mathcal{O}_i$. Let $A^{(i)}$ be the association scheme defined by $(G, \mathcal{O}_i)$, let $I^{(i)}$ be the group of isomorphisms of $A^{(i)}$ and let g_{ij} be an isomorphism from $A^{(i)}$ to $A^{(j)}$ if they are isomorphic. Let I be the group generated by all $I^{(i)}$ and g_{ij}. In Miyamoto (2000) the following lemma speeds up the computation of the normalizers of permutation groups. In this note we give the lemma in a slightly modified form as is used in our program to compute normalizers.

LEMMA 5.1: *Let K be a permutation group on Ω. Let L be a permutation group containing K, let F be a tuple $[p_1, p_2, \cdots, p_r]$ of points in Ω and let G^i be the stabilizer of the subset $[p_1, p_2, \cdots, p_i]$ of F as a tuple in G for $i = 1, 2, \cdots, r$. Let I^i be the group of isomorphisms of the association schemes on $\Omega \backslash [p_1, p_2, \cdots, p_i]$ defined by G^i as stated above. Set $I^0 = I$, $G^0 = G$ and set $I^{\{0..i\}} = I^0 \cap I^1 \cap \cdots \cap I^i$. Suppose that $G^i \cap L$ and $I^{\{0..i\}} \cap L$ have a common orbit and that the point p_{i+1} is contained in the orbit for $i = 0, 1, \cdots, r-1$. Then the normalizer of G in L is generated by $G \cap L$ and the normalizer of G in $I^{\{0..r\}} \cap L$ and the normalizer of G in K is obtained as the intersection of K and the above normalizer in L.*

Suppose that we would like to compute the normalizer of G in K. The group L in the lemma is usually chosen to be K itself or the symmetric group, which does not mean computing the intersection $I \cap L$ in our program, in order to be able to apply the lemma well. Our program computing normalizers follows the lemma and the normalizer of G in $I^{\{0..r\}} \cap L$ for some r is computed by the `GAP`-command `Normalizer`. Our program is written in `GAP` programming language. Normalizers are computed by a backtrack algorithm searching the normalizing elements. If the lemma works, then we can restrict the searching space. We use a backtrack algorithm to compute the isomorphisms of association schemes. As in the lemma we compute the intersections of groups and they are also computed by a backtrack algorithm. However these backtrack algorithms are computed rather quickly.

If G is transitive, then the lemma works at least once, that is, $r = 1$, if $G \subseteq L$. There are the list of transitive groups of degree up to 22 in `GAP` library (Hulpke, 1996). We will show the general performance of our program computing the normalizers of the transitive groups in the symmetric groups in the following tables. We used 700 MHz Pentium III machine under Linux for computing the tables. We compare our method with the direct computation by `GAP` or Magma command `Normalizer`. The timings are in seconds. The columns "as" show the timings by our method. The columns "`GAP`" show the timings by `GAP4` and

those "Magma" by Magma 2.5. In Table 3 and 5 the columns "number" show the numbers of the transitive groups of which normalizers in the symmetric group are computed within the time range denoted in the columns "time", and the columns "average" show the average computing times. In Table 3 minimum, maximum and overall average times are also shown. Table 3 shows the computation for degree 18, Table 5 for degree 20, 21 and 22, and Table 4 shows the total times for computing all the normalizers of the transitive groups of degree from 5 to 19 except 18. When the degree of the transitive group is greater than 19, we interrupted to compute in 20 minutes.

Table 3: Normalizers of transitive groups of degree 18 in S_{18}

	as		GAP		Magma	
time	number	average	number	average	number	average
< 1	910	0.333	883	0.094	879	0.086
$1 \leq * < 10$	73	1.472	33	1.855	18	3.16
$10 \leq$	0	-	67	5913	86	5300
min		0.14		0.01		0.009
max		2.4		30645		34199
all		0.42		392		430

As is seen in Table 3, most groups are computed very quickly by direct computation using GAP or Magma command Normalizer. In such cases the direct computation is from 5 to 10 times as fast as our method. But there also exist a couple of groups for which the normalizers are hard to compute. In these cases our method is usually more than 100 times as fast as the direct computation. If the degrees are small, Table 4 shows that almost all groups are computed quickly by the direct computation.

Table 4: Normalizers of transitive groups in the symmetric groups

degree	number	as	GAP
5	5	0.21	0.02
6	16	0.72	0.1
7	7	0.38	0.06
8	50	3.77	0.61
9	34	4.29	0.44
10	45	6.59	0.74
11	8	0.77	0.17

degree	number	as	GAP
12	301	55.86	8.74
13	9	1.04	0.15
14	63	15.79	69.97
15	104	30.4	29.37
16	1954	595	1300
17	10	3.12	0.28
19	8	1.72	0.18

In the lemma if $K = S_n$, the symmetric group of degree n, then we have no choice for the group L. We will consider a case $K \cong A_n$, the alternating group.

Table 5: Normalizers of transitive groups of degree n =20, 21, 22 in S_n

		as		GAP	
degree	time	number	average	number	average
20	< 1200	1117	0.83	1091	5.15
	$1200 \leq$	0	-	26	≥ 1200
21	< 1200	164	1.31	143	40
	$1200 \leq$	0	-	21	≥ 1200
22	< 1200	59	1	46	32.3
	$1200 \leq$	0	-	13	≥ 1200

Let G be `TransitiveGroup(22,48)`, the 48th transitive group of degree 22 in the `GAP` library. Then $G \cong M_{11} \wr C_2$, a wreath product of the Mathieu group of degree 11 by a cyclic group of order 2. In this case the C_2-part does not contain an even permutation and so $G \cap K$ is not transitive. It is obtained in Miyamoto (2000) that $I \cong S_{11} \wr C_2$. This group contains an even permutation that interchanges the two blocks of length 11. Hence $I \cap K$ is transitive. Therefore if we choose $K = L$, then there exists no common orbit of $G \cap K$ and $I \cap K$ and the lemma does not work at all. It was hard to compute the normalizer in $I \cap K$. If we choose $L = S_{22}$, the symmetric group, then the lemma works well and the computation proceeds smoothly, which is shown in Table 1 in Miyamoto (2000). The intersection of this normalizer and K can be computed quickly by the `GAP` command `Intersection` and we obtain the desired normalizer.

Let H =`TransitiveGroup(7,4)`, the fourth transitive group of of degree 7 which is doubly transitive of order 7×6, and let C =`Group((1,2,3))`, a cyclic group of order 3. Let G =`WreathProduct(`H, C`)`, the wreath product of H by C, which is isomorphic to `TransitiveGroup(21,109)`. Let K be the wreath product of S_7 by C. Then I is isomorphic to the wreath product of S_7 by S_3 and $G \subseteq K \subseteq I$ holds. The stabilizers of one point in G and K are isomorphic to $C_6 \times H \times H$ and $S_6 \times S_7 \times S_7$ respectively. Both of them have the orbits of lengths 1, 6, 7 and 7. Then we can easily see that sequence of stabilizers in G and K have common orbits of lengths 21, 7, 7, 6, 6 and 6, which means $r = 6$ in the lemma. Hence if we set $L = K$, the lemma works well. Oppositely if we set $L = S_{21}$, we can apply the lemma only twice and the lengths of the common orbits of the stabilizers in K and I are of lengths 21 and 6, since the stabilizer of a point in I is isomorphic to $S_6 \times (S_7 \wr C_2)$ and has orbits of lengths 1, 6, and 14. The computing times differ by about 8 times.

We can make similar examples for larger degrees in both cases and then the computing times differ by much more. In our program we can choose the correct groups in these cases heuristically.

References

Bannai, E., Ito, T., *Algebraic Combinatorics I : Association Schemes*, (1984). Benjamin/Cummings, Menlo Park, CA,

The GAP Group, *GAP – Groups, Algorithms and Programming, Version 4*, (1997). Lehrstuhl D f Mathematik, Rheinisch Westfälische Technisch Hochschule, Aachen, Germany and School of Mathematical and Computational Sciences, U. St.Andrews Scotland,

Hanaki, A., Data of association schemes, published at WWW http://math.shinshu-u.ac.jp/~hanaki, (1999).

Hanaki, A., Miyamoto, I., Classification of association schemes with 16 and 17 vertices, *Kyushu J. Math.*, (1998). **52**, 383–395. Hanaki, A., Miyamoto, I., Classification of association schemes with 18 and 19 vertices, *Korean J. Comp. App. Math.*, (1998). **5**, 543–551. Hanaki, A., Miyamoto, I., Classification of primitive association schemes of order up to 22, *Kyushu J. Math.*, (2000). **54**, 81–86.

Hulpke, A., Konstruktion transitiver Permutationsgruppen Dissertation, RWTH-Aachen, (1996).

Mckay, B., Practical graph isomorphism, *Proceedings of the Tenth Manitoba Conference on Numerical Mathematics and Computing, Vol. 1(Winnipeg, Man., 1980) 45–87. Congressus Numerantium 30*, (1981). 45–87.

Miyamoto, I., Computing normalizers of permutation groups efficiently using isomorphisms of association schemes, *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation*, C. Traverso, (2000). 200–204. ACM Press,

Theißen, H., Eine Methode zur Normalisatorberechnung in Permutationsgruppen mit Anwendungen in der Konstruktion primitiver Gruppen. Dissertation, RWTH-Aachen, (1997).