SIAM J. ALG. DISC. METH.
Vol. 2, No. 1, March 1981

0196-5212/81/0201-0007 $01.00/0

A FAST ALGORITHM FOR FINDING STRONG STARTERS*

J. H. DINITZ† AND D. R. STINSON‡

Abstract. A *strong starter* (of order n) in an additive Abelian group G of odd order $n = 2t+1$ is a set $S = \{\{x_1, y_1\}, \{x_2, y_2\}, \cdots, \{x_t, y_t\}\}$ which satisfies the following properties:

(i) $\{x_1, x_2, \cdots, x_t, y_1, y_2, \cdots, y_t\} = G\backslash\{0\}$,

(ii) $\{\pm(y_1 - x_i) | \{x_i, y_i\} \in S\} = G\backslash\{0\}$,

(iii) $x_i + y_i \neq x_j + y_j$ if $i \neq j$, and $x_i + y_i \neq 0$, for any i.

We present a fast algorithm for finding strong starters in Abelian groups.

1. Introduction. Strong starters are used extensively in the construction of Room squares and Howell designs. A *Howell design* $H(n, 2t)$, with $t \leqq n \leqq 2t-1$, is a square array of side n, where cells are either empty or contain an unordered pair of elements chosen from a set X of size $2t$ such that:

(1) each member of X occurs exactly once in each row and column of the array, and

(2) each pair of elements of X occurs in at most one cell of the array.

A *Room square* of side n (n odd) is an $H(n, n+1)$. It follows that, in this case, each pair of elements of X occurs in exactly one cell of the array. Much research has been done concerning Room squares; see, for example [10] and [14]. Strong starters are related to Room squares by the following theorem of Horton [7].

THEOREM 1.1. *If there exists a strong starter of order n, then there exists a Room square of side n.*

Anderson [1], [2] has shown that for the case of Howell designs, the existence of a strong starter of order n which satisfies certain other (technical) properties implies the existence of many $H(n, 2t)$, $(n+1)/2 \leqq t \leqq n$.

For the above reason, strong starters have been investigated by several people. Some infinite classes of strong starters are known. See for example, Mullin and Nemeth [9], Chong and Chan [3], and Gross and Leonard [6]. Indeed, strong starters are known to exist for all orders relatively prime to 3, except for order 5. However, no general method is known for producing strong starters of order $3p$ for p prime. All strong starters of these orders have been found on computer by back-tracking methods (see [4] and [13]). However, for orders exceeding 70, back-tracking becomes impractical due to the excessive computing time required.

Using the algorithm presented in this paper, the authors have recently proven the following theorem [5].

THEOREM 1.2. *If $n < 1000$ is odd, $t \neq n-1$ and $(n, 2t) \neq (5, 6)$, then there exists an $H(n, 2t)$.*

The purpose of this paper is to describe and analyze the algorithm used to find these strong starters.

We wish to point out that we cannot prove that the algorithm will produce a strong starter of any particular order. However, in practice, the algorithm has always succeeded.

In § 2, we describe the algorithm. In § 3, we estimate the time required to be $O(n^2)$ where n is the order n of the strong starter. This estimate agrees with empirical timing results. In § 4, we give a brief geometrical description of strong starters.

* Received by the editors January 30, 1980, and in revised form August 6, 1980.

† Department of Mathematics, University of Vermont, Burlington, Vermont 05405.

‡ Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada.

2. The algorithm. We now present the algorithm used to find a strong starter of order $n = 2t+1$ in the cyclic group $\mathbb{Z}_n$.

Define a *partial strong starter* to be a set $S' = \{\{x_1, y_1\}, \{x_2, y_2\}, \cdots, \{x_r, y_r\}\}$ satisfying the following conditions:

(i) the x_i's and y_i's are distinct nonzero elements of $\mathbb{Z}_n$;

(ii) $y_i - x_i \neq \pm(y_j - x_j)$ if $i \neq j$;

(iii) $x_i + y_i \neq x_j + y_j$ if $i \neq j$, and $x_i + y_i \neq 0$ if $1 \leqq i \leqq r$.

Define def $(S') = t - r$. We say that def (S') is the *deficiency* of S'. The deficiency of S' is the number of "missing pairs". We say that a partial strong starter S' is *maximal* if there exists no $\{u, v\} \subseteq \mathbb{Z}_n$ such that $S' \cup \{\{u, v\}\}$ is a partial strong starter.

In a back-tracking algorithm, when a maximal partial strong starter is reached, the "last" pair $\{x_r, y_r\}$ is deleted from the strong starter. This increases the deficiency of the partial strong starter. The basic feature of the algorithm we will present is that the deficiency is never increased.

Let $D = \{1, 2, \cdots, t\}$. We refer to members of D as *differences*. Then, without loss of generality, we may assume that $y_i - x_i = d_i \in D$, if $1 \leqq i \leqq r$. An element $z \in \mathbb{Z}_n - \{0\}$ is said to be *used* if $z \in \{x_i, y_i\}$ for some $\{x_i, y_i\} \in S'$, otherwise z is *unused*. Similarly, a difference $d \in D$ is said to be *used* or *unused* depending on whether or not $d = d_i$ for some i, $1 \leqq i \leqq r$. Finally, $e \in \mathbb{Z}_n - \{0\}$ is said to be a *used* or *unused sum* depending on whether or not $e = x_i + y_i$ for some i, $1 \leqq i \leqq r$.

We now define a *state* of the algorithm to be a partial strong starter S', together with two distinct unused elements u_1 and u_2, and an unused difference $d \in D$. Given a state of the algorithm, let $T_i = \{u_i - d, u_i + d\}$, $i = 1, 2$, and let $T = T_1 \cup T_2$. The following operations can be performed on a state.

(a) Matching u_i with an unused element:

If there exists $w \in T_i$ such that w is an unused element and $u_i + w$ is an unused sum (for the appropriate $i = 1$ or 2), then let $S'' = S' \cup \{\{u_i, w\}\}$. If def $(S'') \neq 0$, choose a new u_1, u_2, d.

(b) Switching a pair:

If $w \in T_i$ is a used element, and $u_i + w$ is an unused sum, then let $S'' = S' \backslash \{\{x_j, y_j\}\} \cup \{\{w, u_i\}\}$, where $w = x_j$ or y_j for some j, $1 \leqq j \leqq r$. Set

$$d = d_j, \qquad u_1 = u_{3-i},$$

and

$$u_2 = \begin{cases} y_j & \text{if } w = x_j, \\ x_j & \text{if } w = y_j. \end{cases}$$

(c) Back-tracking:

Revert to the previous state of the algorithm if (b) or (c) was the last operation performed.

(d) Switching a difference:

Replace d by some other unused difference d'. Leave u_1, u_2 unchanged.

(e) Switching a pair:

Suppose $u_i - u_{3-i} = d_1 \in D$ is a used difference, and suppose $u_1 + u_2$ is an unused sum. Then set $S'' = S' \backslash \{\{x_{d_1}, y_{d_1}\}\} \cup \{u_1, u_2\}$; set $u_1 = x_{d_1}$, $u_2 = y_{d_1}$, and leave d unchanged.

We may now use operations (a)–(e) to describe our algorithm.

(1) Initialization: Set def $= t$, $S = \varnothing$, choose any distinct $u_1, u_2 \in \mathbb{Z}_n - \{0\}$, $d \in D$.

(2) If operation (a) can be performed, do so and go to (8).
(3) If operation (b) can be performed, do so and go to (2).
(4) If operation (c) can be performed, do so and go to (3).
(5) If operation (d) can be performed, do so and go to (2).
(6) If operation (e) can be performed, do so and go to (2).
(7) Stop (algorithm fails).
(8) Set def = def − 1, choose any distinct unused u_1, u_2 and d. If def $\neq 0$ go to (2).
(9) Stop (algorithm succeeds).

A few comments regarding the algorithm are in order. First, no operation increases the deficiency, and operation (a) decreases the deficiency by 1. Also, operations (d) and (e) are rarely executed since it is unlikely that (a), (b) and (c) all fail (more details in § 3). Note that if def = 1, then operation (d) cannot occur, since (d) requires an unused difference other than d. Finally, note that there may be more than one way to perform an operation (b) on a given state. As a heuristic in the implementation of the algorithm, the following is done. If a state is reached, and more than one way to perform operation (b) is possible, then one way is picked at random. If the state is again reached, this time by back-tracking (operation (c)), then the first way to perform operation (b) is excluded and one of the remaining ways is chosen at random. As an example of this, see lines 9–12 in Table 1 below.

We construct a strong starter of order 11 using this algorithm. Table 1 below traces the execution of the algorithm. Note that no operations (d) or (e) were required.

TABLE 1

Partial strong starter					State			Operation to be performed
diff 1	2	3	4	5	u_1	u_2	d	
					9	5	3	a
		1, 9			2	3	5	a
		1, 9		7, 2	3	4	4	a
		1, 9	3, 10	7, 2	4	5	2	b
	4, 2	1, 9	3, 10		5	7	5	b
	4, 2		3, 10	1, 7	5	9	3	a
	4, 2	9, 6	3, 10	1, 7	5	8	1	b
5, 4		9, 6	3, 10	1, 7	8	2	2	b
5, 4	10, 8	9, 6		1, 7	2	3	4	b
5, 4	10, 8	9, 6	7, 3		2	1	5	c
5, 4	10, 8	9, 6		1, 7	2	3	4	c
5, 4		9, 6	3, 10	1, 7	8	2	2	b
5, 4	8, 6		3, 10	1, 7	2	9	3	b
5, 4	8, 6	1, 9	3, 10		2	7	5	c
5, 4	8, 6		3, 10	1, 7	2	9	3	b
	8, 6	5, 2	3, 10	1, 7	9	4	1	b
9, 8		5, 2	3, 10	1, 7	4	6	2	a
9, 8	6, 4	5, 2	3, 10	1, 7				

3. Analysis of the algorithm. In this section, we estimate the efficiency of this algorithm by some probabilistic considerations and present some empirical data.

In order to calculate this estimate, one major assumption is made. We assume that the probability that an operation succeeds on a given state is independent of the

previous state performed. Theoretically, this assumption is probably not even true. However, analysis of the efficiency of this algorithm using the assumption of independence strongly agrees with the empirical data (see Tables 2 and 3). It thus appears (as

TABLE 2

100 strong starters of each order n

n	Average of $N_a+N_b+N_c+N_d+N_e$	Average $\frac{}{n \log n}$	95% Confidence interval	Average of N_d+N_e	Number that failed	Time[1]
51	393	1.95	53	.39	17	6.59 sec
101	757	1.62	96	.89	13	9.29
201	1611	1.51	173	2.05	11	18.21
301	2491	1.45	272	3.31	12	29.32
401	3486	1.45	317	4.64	10	41.35
501	4029	1.29	374	5.29	10	51.02
601	4932	1.28	423	7.16	12	65.25

TABLE 3

2 strong starters of each order n

n	Average of $N_a+N_b+N_c+N_d+N_e$	Average $\frac{}{n \log n}$	Time	$\frac{\text{Time}^1}{n^2}\times 10^6$
3001	31190	1.29	14.4 sec	1.60
5001	63645	1.50	32.0	1.28
8001	91852	1.28	77.5	1.21
10001	117020	1.27	117.1	1.17

intuition would indicate) that the states are nearly independent, particularly for n large. Because of the independence assumption, the analysis which follows is merely an estimate of the actual efficiency of the algorithm and is not a proof of the existence of strong starters.

First we estimate the probability that operation (a) succeeds for a given state with deficiency k. The number of unused elements, other than u_1 or u_2, is $2k-2$. If operation (b) was just performed, then one element of T will be used. The other three elements of T each have probability $(2k-2)/n$ of being unused and distinct from u_1 and u_2. The probability that a given element of T is unused is less than the probability p that there is some unused element in T. Thus, for some element $e \in T$, distinct from u_1 and u_2, the probability that e is unused is $(2k-2)/(n-2)$. So a lower bound on p is $p \geqq (2k-2)/(n-2)$.

There is also the possibility that $u_1-u_2=\pm d$. This happens with probability $2/(n-1)$. Finally, the probability that a given sum is nonzero and unused is $((n-1)/2+$

[1] The algorithm was implemented in Fortran on The Ohio State University Amdahl 470 system.

$k)/n$. Thus, the probability of (a) succeeding when the deficiency is k is at least

$$p_k(\mathrm{a}) > \left[\binom{2k-2}{n-2} + \frac{2}{n-1}\right]\frac{n+2k-1}{2n}$$
$$> \binom{2k-2+2}{n-1}\binom{n+2k-1}{2n}$$
$$= \frac{k(n+2k-1)}{(n-1)n}.$$

Thus, the number N_a of times operation (a) is attempted in the course of the algorithm is approximately

$$\sum_{k=1}^{(n-1)/2} \frac{1}{p_k(\mathrm{a})} < (n-1)n \sum_{k=1}^{(n-1)/2} \frac{1}{k(n+2k-1)}$$
$$< n \sum_{k=1}^{(n-1)/2} \frac{1}{k}$$
$$< n\left(\log\left(\frac{n-1}{2}\right)+1\right).$$

Thus, it appears that $N_a = O(n \log n)$.

Now, if we suppose that (c) does not fail in the course of the algorithm, we can have that $N_a = N_b - N_{c^*} + (n-1)/2$. N_b and N_{c^*} denote the number of times operations (b) and (c*) are attempted, where an operation (c*) is a maximal sequence of consecutive operations (c).

We now compute the probability $p_k(\mathrm{b})$ of (b) succeeding if (a) or (b) was just performed. If (a) was just performed, then there are four possibilities for pairs to be switched, if (b), then three. The probability of at least one sum being unused is at least

$$1 - \left(\frac{(n-1)/2-k}{n}\right)^3 = 1 - \left(\frac{n-2k-1}{n}\right)^3 > \frac{7}{8}.$$

If w were an unused element, then (a) would be performed. Thus, w is used (perhaps zero). If w is nonzero then (b) can be performed. In order to simplify the arithmetic, we assume w is nonzero. This does not greatly affect our estimate. Thus we estimate $p_k(\mathrm{b}) > \frac{7}{8}$. Since (c) occurs only after (b) fails, we have $N_{c^*} < \frac{1}{8}N_b$.

Finally, the number of operations (c) in one operation (c*) must be estimated. Denote by $p_k(\mathrm{b}=1)$ the probability that there is exactly one permissible choice in a (b) operation (where a (b) or (a) was just performed). Then

$$1 - p_k(\mathrm{b}=1) = 1 - 3\left(\frac{n-2k+1}{2n}\right)^2\left(\frac{n+2k-1}{2n}\right)$$
$$\geqq 1 - 3\left(\frac{n-1}{2n}\right)^2\left(\frac{n+1}{2n}\right)$$
$$\geqq \frac{5}{8}.$$

Thus we estimate that, on the average, less than $\frac{8}{5}$ (c) operations make up each (c*) operation.

By the above, we have $N_b = N_a + N_{c^*} - (n-1)/2$. Thus $N_b < N_a + \frac{1}{8}N_b - (n-1)/2$, so $N_b < \frac{8}{7}(N_a - (n-1)/2)$. Therefore, $N_b = O(n \log n)$. Also, $N_c < \frac{8}{5}N_{c^*} < \frac{1}{5}N_b$, so $N_c = O(n \log n)$.

Thus, we estimate that the number of operations, $N_a + N_b + N_c$, executed in the algorithm is $O(n \log n)$. Also, the time required for an operation is at most $O(n)$. Choosing a new u_1 and u_2 is the only time it is necessary to search through an array. With more sophisticated list processing techniques, this time could be reduced, perhaps to $O(\log n)$. Each of the operations (a)–(e) require $O(1)$ time. Thus, we estimate that the time required for the algorithm is $O(n^2 \log n)$.

This estimate can be improved slightly. The $O(n)$ operation is executed only $(n-1)/2$ times in the course of the algorithm. Thus, an estimate of $O(n^2)$ is obtained.

To test this estimate, the program was run until 100 strong starters were produced for each of 7 different orders, (See Table 2). Also, a 95% confidence interval about the mean μ of $N_a + N_b + N_c + N_d + N_e$ was computed. To test the algorithm on large orders, we produced two strong starters each of orders 3001, 5001, 8001, and 10001 (Table 3).

Although no theoretical upper bound for the probability of failure has been computed, in practice this number appears to be about $\frac{1}{10}$. The algorithm usually fails when deficiency equals 1 and no operation can be performed. This happens only in the first state after the deficiency has become 1, since otherwise the program will be able to back-track when no (b) can be performed. There is also the chance that the states might form a loop and thus not produce a starter. In order to prevent an infinite loop, a timer was written into the program. If the search for a starter took too long, the search would be aborted and the program started over again with def $= (n-1)/2$. However, this occurred only once in over 700 trials.

4. A geometric interpretation. Strong starters in $\mathbb{Z}_n$ have an interesting geometrical interpretation. Label n equally spaced points on a circle by the elements of $\mathbb{Z}_n$ (cyclically). If $\{x, y\} \in S$, then join points x and y on the circle by a straight line. The $(n-1)/2$ lines thus formed will have the following properties:

(1) no two lines have the same length;
(2) no two lines are parallel;
(3) no two lines have a common endpoint.

Conversely, any such geometric configuration generates a strong starter in $\mathbb{Z}_n$.

A strong starter of order 129 is geometrically represented in Fig. 1 below.

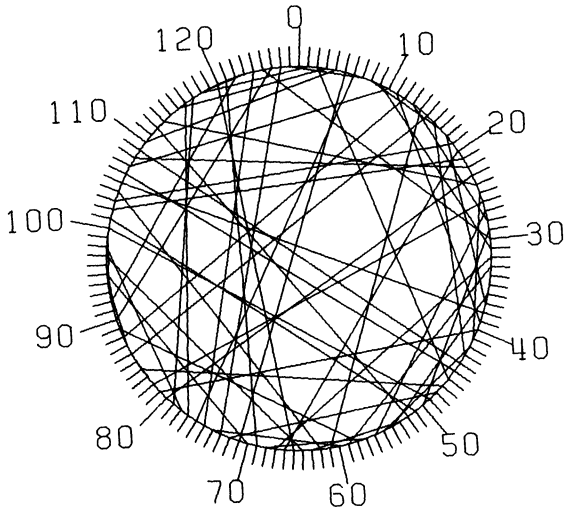

FIG. 1. *A strong starter of order* 129.

5. Conclusion. Thus, we have described an algorithm for finding strong starters. Using probabilistic arguments, we estimate that the algorithm should succeed in polynomial time (actually $O(n^2)$). In practice, this seems to be accurate.

Our algorithm is similar in some aspects to the algorithm of Posa [11] for finding Hamiltonian circuits in graphs. That is, at no time in the algorithm does one head "away" from the desired end results. In finding strong starters, the deficiency is never increased; in finding a Hamiltonian circuit, the length of a path is never decreased. Also, both algorithms involve a certain amount of randomness in making some choices. Finally, there is the possibility that the algorithm may fail. However, in practical applications both algorithms have a high rate of success.

Other probabilistic algorithms are described in [8] and [12].

It also appears that probabilistic algorithms based on this simple switching idea may be practical in other combinatorial applications such as constructing Steiner triple systems and finding transversals in Latin squares.

REFERENCES

[1] B. A. ANDERSON, *Howell designs from Room squares*, Proc. 2nd Caribbean Conf. on Comb. and Comp., Barbados, 1977, pp. 55–62.

[2] ———, *Starters, digraphs, and Howell designs*, Utilitas Math., 14 (1978), pp. 219–248.

[3] B. C. CHONG AND K. M. CHAN, *On the existence of normalized Room squares*, Nanta Math., 7 (1974), pp. 8–17.

[4] R. J. COLLENS AND R. C. MULLIN, *Some properties of Room squares – a computer search*, Proc. 1st Louisiana Conf. on Combinatorics, Graph Theory and Computing, Baton Rouge, 1970, pp. 87–111.

[5] J. H. DINITZ AND D. R. STINSON, *A note on Howell designs of odd side*, Utilitas Math., to appear.

[6] K. B. GROSS AND P. A. LEONARD, *The existence of strong starters in cyclic groups*, Utilitas Math., 7 (1975), pp. 187–195.

[7] J. D. HORTON, *Room designs and one-factorizations*, Aequationes Math., to appear.

[8] R. M. KARP, *The probabilistic analysis of some combinatorial search algorithms*, in Algorithms and Complexity, Academic Press, New York, 1976.

[9] R. C. MULLIN AND E. NEMETH, *An existence theorem for Room squares*, Canad. Math. Bull., 12 (1969), pp. 493–497.

[10] R. C. MULLIN AND W. D. WALLIS, *The existence of Room squares*, Aequationes Math., 13 (1975), pp. 1–7.

[11] L. POSA, *Hamilton circuits in random graphs*, Discrete Math., 14 (1976), pp. 359–364.

[12] M. O. RABIN, *Probabilistic Algorithms*, in Algorithms and Complexity, Academic Press, New York, 1976.

[13] R. G. STANTON AND R. C. MULLIN, *Construction of Room squares*, Ann. Math. Statist., 39 (1968), pp. 1540–1548.

[14] W. D. WALLIS, A. P. STREET AND J. S. WALLIS, *Combinatorics: Room Squares, Sum-free Sets, Hadamard Matrices*, Lecture Notes in Mathematics 292, Springer-Verlag, Berlin, 1972.