SIAM J. ALG. DISC. METH.
Vol. 8, No. 3, July 1987

A HILL-CLIMBING ALGORITHM FOR THE CONSTRUCTION OF ONE-FACTORIZATIONS AND ROOM SQUARES*

J. H. DINITZ† AND D. R. STINSON‡

Abstract. In this paper we describe and discuss hill-climbing algorithms for the construction of one-factorizations of complete graphs, and orthogonal one-factorizations of complete graphs (i.e., Room squares).

Key words. hill-climbing, algorithm, Room squares, one-factorizations

AMS(MOS) subject classifications. 05B15, 68E10

1. Introduction. In this paper, we study hill-climbing algorithms for certain types of combinatorial designs. In the past, combinatorial designs have usually been constructed using backtracking algorithms (see [2] and [6], for example). Recently, however, hill-climbing algorithms have enjoyed some success in certain cases.

First, we briefly describe hill-climbing algorithms in a general setting. Suppose that we have some particular combinatorial optimization problem for which we want to design an algorithm. For any problem instance I, there is a set of *feasible solutions* $F(I)$; each feasible solution X has a *cost* $c(X)$. The *optimal solution* is the feasible solution X having the minimum cost. (Alternatively, we could associate a profit with each feasible solution, and ask for the feasible solution with maximum profit.)

We define a hill-climbing algorithm for a combinatorial optimization problem in terms of one or more heuristics H. Each heuristic is based on a neighbourhood system, as follows. A *neighbourhood* of X is any collection of feasible solutions $N(X)$ such that $X \in N(X)$. If, for every feasible solution X, we define a neighbourhood $N(X)$ of X, then we obtain a *neighbourhood system.* Given a neighbourhood system and a feasible solution X, the *heuristic* H nondeterministically chooses any feasible solution $Y \in N(X)$ such that $c(Y) \leqq c(X)$. If there is no such Y, then the heuristic fails, and we say that X is a *local minimum* (with respect to the heuristic H). We can define several different types of neighbourhoods and associate a different heuristic with each.

Suppose we have defined a heuristic H. Suppose also that we have some method of generating an "initial" feasible solution X. Then, the hill-climbing algorithm proceeds as follows:

```
generate initial feasible solution X;
while X is not a local minimum do
  begin
    choose any Y ∈ N(X) such that c(Y) ≦ c(X);
    X:= Y
  end;
```

Our hope is that the final value of X is optimal, or close to optimal.

Depending on how we define neighbourhoods, it may take a lot of time to search the entire neighbourhood for a feasible solution Y such that $c(Y) \leqq c(X)$. It is often

* Received by the editors April 28, 1986; accepted for publication November 20, 1986.
† Department of Mathematics and Statistics, University of Vermont, Burlington, Vermont 05405.
‡ Department of Computer Science, University of Manitoba, Manitoba, Canada R3T 2N2.

easier to first choose $Y \in N(X)$ nondeterministically and then check if $c(Y) \leqq c(X)$. At any given stage of the algorithm, it may be necessary to choose many such Y's before finding one with $c(Y) \leqq c(X)$.

If we take this approach, we would have to specify how many attempts we allow at each stage before we abandon the search. We would define some integer "threshold function" $f(c, I)$, which is a function of the instance I and the cost c of a feasible solution, to accomplish this. Then, we obtain the following algorithm:

```
generate initial solution X;
count:= 0;
repeat
   count:= count + 1;
   choose any Y ∈ N(X);
   if c(Y) ≦ c(X) then
      begin
         if c(Y) < c(X) then
            count:= 0;
         X:= Y
      end
until count > f(c(X), I).
```

If this approach is used, it is important to choose a suitable threshold function.

Our interest is in constructing combinatorial designs using hill-climbing algorithms. In the past, hill-climbing algorithms have been employed to successfully construct Steiner triple systems, Latin squares and strong starters. We refer the interested reader to [3], [4], [13] and [14]. Hill-climbing has been less successful in investigating other problems (see, for example, [1], [11] and [16]).

In this paper, we present new hill-climbing algorithms for some other classes of designs, namely, one-factorizations of complete graphs and Room squares. Room squares are the most "complicated" type of design for which a practical hill-climbing algorithm has been found.

2. A hill-climbing algorithm for finding one-factorizations of complete graphs. The complete graph K_n is the graph on n vertices in which every pair of points is joined by an edge. A *one-factor* of K_n is a set of $n/2$ edges that partitions the vertex set (this requires than n be even). A *one-factorization* of K_n is a set of $n - 1$ one-factors that partitions the edge set. It is well known that K_n has a one-factorization if and only if n is even. Many constructions for one-factorizations are known; a good survey is presented in [8].

In order to use a hill-climbing approach, we formulate the problem as an optimization problem. A problem instance consists only of the (even) integer n for which we want to construct the one-factorization of K_n and the set of vertices V on which K_n is defined. We will represent a one-factorization of K_n as a set $\mathbf{F}$ of pairs, each having the form $(f_i, \{x, y\})$, where $1 \leqq i \leqq n - 1$, and x and y are distinct vertices of K_n. There will be $n(n - 1)/2$ such pairs and the following properties must be satisfied:

1) Every edge $\{x, y\}$ of K_n occurs in a unique pair $(f_i, \{x, y\})$;
2) For every one-factor f_i, and for every vertex x, there is a unique pair of the form $(f_i, \{x, y\})$.

Property 1) says that every edge occurs in a unique one-factor, and property 2) says that every one-factor consists of a perfect matching.

Now, we can describe feasible solutions as being *partial* one-factorizations: in the representation above, we have a set **F** of pairs, each of which has the form $(f_i, \{x, y\})$, which satisfies the properties:

1) Every edge $\{x, y\}$ of K_n occurs in at most one pair $(f_i, \{x, y\})$;
2) For every one-factor f_i, and for every vertex x, there is at most one pair of the form $(f_i, \{x, y\})$.

We define the cost $c(\mathbf{F})$ of a feasible solution **F** to be $n(n-1)/2 - |\mathbf{F}|$, where $|\mathbf{F}|$ denotes the number of pairs in **F**. Then, it is easy to see that **F** is a one-factorization if and only if $c(\mathbf{F}) = 0$.

We must now define a heuristic. First, we construct a graph which tells us what is missing from a partial one-factorization. Given a feasible solution **F**, we define $d(\mathbf{F})$, the *defect graph* of **F**, to be the graph having vertex set $V \cup \{f_i, 1 \leqq i \leqq n-1\}$, where V is the vertex set of K_n, and having the following edges:

1) For every edge $\{x, y\}$ of K_n which does *not* occur in a pair of **F**, $\{x, y\}$ is an edge of $d(\mathbf{F})$;
2) For every f_i, $1 \leqq i \leqq n-1$, $\{f_i, x\}$ is an edge of $d(\mathbf{F})$ if and only if there is *no* pair of the form $(f_i, \{x, y\})$.

In fact, we define two heuristics, $\mathbf{H_1}$ and $\mathbf{H_2}$, based on the defect graph. We say that a vertex or one-factor is *live* if it has positive degree in the defect graph. Note that, since n is odd, if a vertex or one-factor is live, then its degree must be at least two. The heuristic $\mathbf{H_1}$ is defined as follows:

Given a partial one-factorization **F** with defect graph $d(\mathbf{F})$, perform the following operations:

1) choose any live point x (nondeterministically)
2) choose any one-factor f_i such that $\{x, f_i\}$ is an edge of $d(\mathbf{F})$ (nondeterministically)
3) choose any point y such that $\{x, y\}$ is an edge of $d(\mathbf{F})$ (nondeterministically)
4) if $\{y, f_i\}$ is an edge of the defect graph, then
   replace **F** by $\mathbf{F} \cup (f_i, \{x, y\})$
   else
   there is a pair in F of the form $(f_i, \{z, y\})$ $(z \neq x)$
   replace **F** by $\mathbf{F} \cup (f_i, \{x, y\}) \backslash (f_i, \{z, y\})$.

If we apply the heuristic $\mathbf{H_1}$, then we obtain a new feasible solution in which the cost either remains the same, or is reduced by one. Also, observe that the heuristic never "fails," since steps 1), 2) and 3) can always be performed. The heuristic $\mathbf{H_2}$ is a slight variation:

Given a partial one-factorization **F** with defect graph $d(\mathbf{F})$, perform the following operations:

1) choose any one-factor f_i (nondeterministically)
2) choose any two live points x and y such that $\{x, f_i\}$ and $\{y, f_i\}$ are both edges of $d(\mathbf{F})$ (nondeterministically);
3) if $\{x, y\}$ is an edge of the defect graph, then
   replace **F** by $\mathbf{F} \cup (f_i, \{x, y\})$
   else
   there is a pair of **F** of the form $(f_j, \{x, y\})$ $(f_j \neq f_i)$
   replace **F** by $\mathbf{F} \cup (f_i, \{x, y\}) \backslash (f_j, \{x, y\})$.

As was the case with $\mathbf{H_1}$, the heuristic $\mathbf{H_2}$ can always be applied, and it yields a new feasible solution with either the same cost or a cost of one unit lower.

There is no guarantee that these two heuristics are sufficient to always enable us to construct a one-factorization. It seems possible that one could reach a local minimum where no application of $\mathbf{H_1}$ or $\mathbf{H_2}$ produces a feasible solution of lower cost. However, this does not seem to happen in practice (though we cannot prove that it will never happen). In over 1000000 trials, the desired one-factorizations were always constructed. Thus, as a threshold function we can define

$$f(c, I) = \infty \quad \text{if } c > 0, \qquad f(0, I) = 0.$$

We can compare this hill-climbing algorithm to the algorithm to construct Steiner triple systems described in [14]. The algorithm in [14] also appears very unlikely to "fail" in practice, though it can conceivably do so [10].

We also want to note that it can be implemented so that each iteration requires only constant time. The method is similar to the hill-climbing algorithm described in [14]; so we do not describe the details here.

3. A hill-climbing algorithm for constructing Room squares. Suppose we have two one-factorizations of K_n, say $\mathbf{F} = \{f_1, \cdots, f_{n-1}\}$ and $\mathbf{G} = \{g_1, \cdots, g_{n-1}\}$. We say that $\mathbf{F}$ and $\mathbf{G}$ are *orthogonal* if any f_i and any g_j $(1 \leqq i \leqq n-1,\ 1 \leqq j \leqq n-1)$ contain at most one edge in common. A *Room square* of side $n-1$ is defined to be a square array R of side $n-1$, in which every cell either is empty or contains an edge of K_n, such that the filled cells in every row and every column of R form a one-factor, and such that every edge of K_n occurs in exactly one cell of R. Clearly, the rows of R will induce a one-factorization of K_n, as will the columns. Also, these two one-factorizations are orthogonal (this is equivalent to saying that no cell of R contains more than one edge of K_n). Conversely, a pair of orthogonal one-factorizations $\mathbf{F}$ and $\mathbf{G}$ give rise to a Room square in a very natural way: index the rows of a square array R by the one-factors of $\mathbf{F}$, and index the columns by the one-factors of $\mathbf{G}$, and place every edge $\{x, y\}$ in the cell (f_i, g_j), where $\{x, y\} \in f_i \cap g_j$.

Room squares were introduced by T. G. Room in 1957, though examples can be found in the literature as early as 1851 [7]. Room squares were studied extensively, but the existence question was not solved until 1975, when it was shown that there is a Room square of side n if and only if n is odd and $n \neq 3, 5$. A condensed proof is presented in Mullin and Wallis [9]. However, some of the constructions for constructing Room squares are quite complicated, and it seems worthwhile to have an algorithm for producing (many) different Room squares.

We have already noted that a Room square of side n is equivalent to a pair of orthogonal one-factorizations of order $n+1$, and that we have a practical method for constructing one-factorizations. Our strategy now is to construct a one-factorization orthogonal to a given one-factorization, thereby producing a Room square. So, suppose we have a one-factorization $\mathbf{F}$, and we wish to construct $\mathbf{G}$ orthogonal to $\mathbf{F}$. (As we construct $\mathbf{G}$, $\mathbf{F}$ remains fixed.) In terms of the Room square, we have determined the rows (say), and we are attempting to "sort out" the columns.

Let us first consider how we should modify the hill-climbing algorithm to construct a $\mathbf{G}$ orthogonal to a given $\mathbf{F}$. We will maintain the array $\mathbf{R}$, in which the rows are indexed by the one-factors of $\mathbf{F}$ and the columns are indexed by the one-factors of $\mathbf{G}$, as we proceed. At any stage of the algorithm, $R(f_i, g_j) = \{x, y\}$ if $\{x, y\} \in f_i \cap g_j$, and $R(f_i, g_j) = \varnothing$, otherwise.

We use the same two heuristics $\mathbf{H_1}$ and $\mathbf{H_2}$, as before, except now they may possibly fail, if the added constraint of orthogonality is violated. Our modified heuristic $\mathbf{H_1}$ is as follows:

1) choose any live point x (nondeterministically)
2) choose any one-factor g_i such that $\{x, g_i\}$ is an edge of $d(\mathbf{G})$ (nondeterministically)
3) choose any point y such that $\{x, y\}$ is an edge of $d(\mathbf{G})$ (nondeterministically)
4) let f_j be the one-factor of $\mathbf{F}$ which contains the edge $\{x, y\}$
5) if $R(f_j, g_i)$ is not empty then
   H_1 fails
   else if $\{y, g_i\}$ is an edge of the defect graph, then
   replace $\mathbf{G}$ by $\mathbf{G} \cup (g_i, \{x, y\})$
   define $R(f_j, g_i) := \{x, y\}$
   else
   there is a pair in $\mathbf{G}$ of the form $(g_i, \{z, y\})$
   replace $\mathbf{G}$ by $\mathbf{G} \cup (g_i, \{x, y\}) \backslash (g_i, \{z, y\})$
   define $R(f_j, g_i) := \{x, y\}$.

The heuristic $\mathbf{H_2}$ becomes:

1) choose any one-factor g_i (nondeterministically)
2) choose any two live points x and y such that $\{x, g_i\}$ and $\{y, g_i\}$ are both edges of $d(\mathbf{G})$ (nondeterministically);
3) let f_j be the one-factor of $\mathbf{F}$ which contains the edge $\{x, y\}$
4) if $R(f_j, g_i)$ is not empty then
   H_2 fails
   else if $\{x, y\}$ is an edge of $d(\mathbf{G})$, then
   replace $\mathbf{G}$ by $\mathbf{G} \cup (g_i, \{x, y\})$
   define $R(f_j, g_i) := \{x, y\}$
   else
   there is a pair in $\mathbf{G}$ of the form $(g_k, \{x, y\})$
   replace $\mathbf{G}$ by $\mathbf{G} \cup (g_i, \{x, y\}) \backslash (g_k, \{x, y\})$
   define $R(f_j, g_i) := \{x, y\}$
   define $R(f_j, g_k) := \varnothing$.

When we try to construct a one-factorization $\mathbf{G}$ orthogonal to a given one-factorization $\mathbf{F}$, it often does happen that we reach dead ends. For example, consider the situation when we have a feasible solution $\mathbf{G}$ with $c(\mathbf{G}) = 1$. The defect graph $d(\mathbf{G})$ consists of a triangle, of the form $g_i\, x\, y$. No matter which heuristic we apply, we will attempt to add this triangle to $\mathbf{G}$. However, this may violate the orthogonality constraint, as there may already be an edge $\{u, v\} \in g_i \cap f_j$, where $\{x, y\} \in f_j$. Such a situation is a local minimum (with respect to $\mathbf{H_1}$ and $\mathbf{H_2}$). Many other types of local minima can also arise, so we must define a threshold function to allow for these eventualities.

After doing some experimenting, we chose the following threshold function:

$$f(c, I) = 100 \cdot n \quad \text{if } c > 0 \quad \text{(where the instance } I \text{ consists of the graph } K_n\text{)},$$

$$f(0, I) = 0.$$

Given this choice of threshold function, we were interested in determining the probability $p(n)$ of success of the algorithm, as a function of n (the size of the instance). This probability seems impossible to estimate theoretically, so we performed a large number of experimental runs, in order to obtain an empirical result. As n varied over several values between 12 and 102, the probability $p(n)$ varied between .083 and .143, in a random fashion. The average value of p appears to be between .10 and .11, and there is

no trend for $p(n)$ to increase or decrease as a function of n. We also calculated the average cost of the local minima generated. Our results are presented in Table 1.

4. Constructing Room squares with subsquares. In this section, we mention an application of our hill-climbing algorithm to an as yet unsolved problem, where we expect to be able to prove some new results. This problem concerns the existence of subsquares in Room squares. If R is a Room square of side $n - 1$, and we can find $m - 1$ rows and columns of R whose intersection, S, is a Room square in its own right (of side $m - 1$) then we say that S is a *subsquare* (of R) of side $m - 1$. Observe that we can "unplug" S and replace it by any other Room square of side $m - 1$ on the same set of vertices as S and obtain another Room square. If we unplug S from R, we refer to the resulting array as an $(n - 1, m - 1)$ *incomplete* Room square.

Observe that there can exist no Room square which contains a subsquare of side 3 or 5, but it is possible for $(s, 3)$ or $(s, 5)$ incomplete Room squares to exist. Of course, these cannot be completed.

We are interested in the following question: for what ordered pairs (s, t) does there exist an (s, t) incomplete Room square? The following necessary conditions are not difficult to prove; we refer the reader to [12] for details.

THEOREM 4.1. *If there exists an (s, t) incomplete Room square, where $t \geqq 0$, then s and t are odd positive integers, $s \geqq 3t + 2$, and $(s, t) \neq (5, 1)$.*

We suspect that these conditions are also sufficient, but this has not yet been proved. The best known results concerning this problem can be found in [12].

We want to modify our hill-climbing algorithm to construct (s, t) incomplete subsquares. To do this, we need to reformulate the definitions in terms of one-factorizations and modify our heuristics accordingly. This is quite straightforward.

Let $\mathbf{F} = \{f_1, \cdots, f_{n-1}\}$ be a one-factorization of K_n. Given any m vertices, Y, of the K_n, there is an induced subgraph of K_m of K_n. If there are $m - 1$ one-factors in $\mathbf{F}$

TABLE 1
Construction of Room squares.

n	# trials	# successes	Probability of success	Average time per trial*	Average cost
12	1000	126	.126	0.09	1.289
16	1000	118	.118	0.16	1.316
22	1000	97	.097	0.32	1.433
26	1000	101	.101	0.57	1.512
32	1000	99	.099	0.67	1.561
36	1000	83	.083	1.2	1.563
42	1000	103	.103	1.2	1.598
46	1000	108	.108	1.6	1.526
52	1000	98	.098	1.8	1.670
56	1000	120	.120	2.1	1.573
62	1000	89	.089	2.1	1.630
66	1000	90	.090	2.1	1.684
72	500	45	.090	3.7	1.670
76	500	50	.100	4.9	1.680
82	500	56	.112	6.2	1.680
86	500	55	.110	5.2	1.582
92	300	43	.143	5.8	1.550
96	300	35	.117	8.0	1.623
102	300	25	.083	7.3	1.707

* We implemented our algorithm in Pascal/VS and ran it on the University of Manitoba Amdahl 5850 computer.

whose intersection with Y produces a one-factorization of order m, then we say we have a *sub-one-factorization* of order m. If we remove the sub-one-factorization of order m, we obtain an *incomplete* one-factorization. We can formally define this concept as follows. We start with the graph $K_n - K_m$, where m and n are both even. A *short one-factor* is defined to be a set of $(n - m)/2$ edges that partitions the vertices not in the K_m. Then, we can define an *incomplete* (n, m) one-factorization to be a set of $n - m$ one-factors and $m - 1$ short one-factors of $K_n - K_m$, whose union contains every edge of $K_n - K_m$ exactly once.

Now, we can relate incomplete one-factorizations to incomplete Room squares. Suppose we have two incomplete (n, m) one-factorizations, say $\mathbf{F} = \{f_1, \cdots, f_{n-1}\}$ and $\mathbf{G} = \{g_1, \cdots, g_{n-1}\}$, where the short one-factors are $f_1, \cdots, f_{m-1}$ and $g_1, \cdots, g_{m-1}$. We say that $\mathbf{F}$ and $\mathbf{G}$ are *orthogonal* if any f_i and any g_j $(1 \leqq i \leqq n - 1, 1 \leqq j \leqq n - 1)$ contain at most one edge in common, and further, any f_i and any g_j $(1 \leqq i \leqq m - 1, 1 \leqq j \leqq m - 1)$ contain no edges in common. It is not difficult to see that a pair of orthogonal incomplete (n, m) one-factorizations are equivalent to an incomplete $(n - 1, m - 1)$ Room square.

Hence, it is necessary only to modify the hill-climbing algorithm for one-factorizations and Room squares to handle incomplete one-factorizations. This is very simple. When we nondeterministically generate a triple $(f_i, \{x, y\})$, say, we must first check that this triple is permissible as part of an incomplete one-factorization. That is, x and y cannot both be points in the K_m, and if f_i is a short one-factor, then neither x nor y can be in the K_m. If either of these two situations arises, then the relevant heuristic fails, and we must try again.

When constructing these incomplete designs, there is a much greater probability that a heuristic will fail, so we should adjust the threshold function accordingly, allowing more tries at each level before we give up. We have run some experiments to test how the probability of success changes with different threshold functions. For each (n, m)-incomplete Room square considered, we tried several different threshold functions, of the form $f(c, I) = K \cdot (n + 1)$, if $c > 0, f(0, I) = 0$. We tried $K = 100$, 500, 1000 and 2000, as indicated.

We obtained the following data, which we present in Table 2. Note that, for fixed n and m, the probability of success tends to decrease as the threshold is increased.

5. Applications. The main application of a hill-climbing algorithm, such as the one we describe, is to produce many different designs very quickly. In [13], a hill-climbing algorithm was used to construct 2117600 Steiner triple systems of order 19. These were then tested for isomorphism using invariants, and 2111276 of the designs were nonisomorphic.

We expect that a similar approach could successfully be used to construct large numbers of nonisomorphic one-factorizations and Room squares. Modifications of the invariants used in [13] can be used to test isomorphism in these cases, as well.

We should also mention that the time and memory requirements for these algorithms are modest enough so that they can be implemented very successfully on most microcomputers. The algorithms can very easily be animated, so an observer can watch the designs being constructed. This also makes it possible to detect when the algorithm is caught in a "vicious circle." In an interactive environment, the observer could determine when a particular run has reached a "dead end," thus obviating the need for an objective function.

The other main application of hill-climbing is to construct previously unknown designs. Since the subsquare problem for Room squares is unsolved, the hill-climbing algorithm will enable us to produce new examples of Room squares with subsquares. It

TABLE 2
Construction of (n, m) *– incomplete Room squares.*
(500 *trials of each example.*)

n	m	Threshold	# successes	Probability of success	Average cost
19	0	$K = 100$	43	0.086	1.64
19	0	$K = 500$	46	0.092	1.57
19	0	$K = 1000$	52	0.104	1.49
19	0	$K = 2000$	58	0.116	1.46
19	1	$K = 100$	41	0.082	1.75
19	1	$K = 500$	50	0.100	1.37
19	1	$K = 1000$	51	0.102	1.39
19	1	$K = 2000$	49	0.098	1.40
19	3	$K = 100$	2	0.004	3.89
19	3	$K = 500$	14	0.028	2.49
19	3	$K = 1000$	14	0.028	2.57
19	3	$K = 2000$	11	0.022	2.43
19	5	$K = 100$	0	0.000	4.57
19	5	$K = 500$	0	0.000	4.82
19	5	$K = 1000$	0	0.000	4.68
19	5	$K = 2000$	1	0.002	4.30
29	0	$K = 100$	35	0.070	1.55
29	0	$K = 500$	60	0.120	1.33
29	0	$K = 1000$	58	0.126	1.35
29	0	$K = 2000$	53	0.106	1.35
29	1	$K = 100$	26	0.052	1.93
29	1	$K = 500$	46	0.092	1.47
29	1	$K = 1000$	38	0.076	1.50
29	1	$K = 2000$	56	0.112	1.41
29	3	$K = 100$	1	0.002	4.15
29	3	$K = 500$	10	0.020	2.69
29	3	$K = 1000$	15	0.030	2.32
29	3	$K = 2000$	31	0.062	1.85
29	5	$K = 100$	0	0.000	5.85
29	5	$K = 500$	0	0.000	5.13
29	5	$K = 1000$	1	0.002	4.39
29	5	$K = 2000$	3	0.006	4.93
39	0	$K = 100$	33	0.066	1.67
39	0	$K = 500$	61	0.122	1.44
39	0	$K = 1000$	54	0.108	1.46
39	0	$K = 2000$	46	0.092	1.39
39	3	$K = 100$	3	0.006	3.95
39	3	$K = 500$	29	0.058	2.10
39	3	$K = 1000$	39	0.078	1.74
39	3	$K = 2000$	42	0.084	1.70

should not be difficult to find an example of any particular order. Hopefully, recursive techniques will then lead to a complete solution of this problem.

For other applications of hill-climbing algorithms in obtaining new results in design theory, we refer the reader to [3], [4], [14] and [15].

REFERENCES

[1] C. J. COLBOURN AND E. MENDELSOHN, *Kotzig factorizations: existence and computational results*, Ann. Discrete Math., 12 (1982), pp. 65–78.
[2] M. J. COLBOURN, *Algorithmic aspects of combinatorial designs: a survey*, Ann. Discrete Math., 26 (1985), pp. 67–136.
[3] J. H. DINITZ AND D. R. STINSON, *A note on Howell designs of odd side*, Utilitas Math., 18 (1980), pp. 207–216.
[4] ———, *A fast algorithm for finding strong starters*, this Journal, 2 (1981), pp. 50–56.
[5] ———, *The spectrum of Room cubes*, European J. Combin., 2 (1981), pp. 221–230.
[6] P. B. GIBBONS, *Computing techniques for the construction and analysis of block designs*, Ph.D. thesis, University of Toronto, Toronto, Ontario, Canada, 1976.
[7] T. P. KIRKMAN, *Note on an unanswered prize question*, Cambridge and Dublin Math. J., 5 (1850), pp. 255–262.
[8] E. MENDELSOHN AND A. ROSA, *One-factorizations of the complete graph—A survey*, J. Graph Theory, 9 (1985), pp. 43–65.
[9] R. C. MULLIN AND W. D. WALLIS, *The existence of Room squares*, Aequationes Math., 13 (1975), pp. 1–7.
[10] K. T. PHELPS, Private communication.
[11] D. P. SHAVER, *Construction of* (v, k, λ) *configurations using a non-enumerative search technique*, Ph.D. thesis, Syracuse University, Syracuse, NY, 1973.
[12] D. R. STINSON, *Room squares and subsquares*, Proc. Combinatorial Mathematics X, Adelaide, Australia, 1982, pp. 86–95.
[13] D. R. STINSON AND H. FERCH, 2000000 *Steiner triple systems of order* 19, Math. Comp., 44 (1985), pp. 533–535.
[14] D. R. STINSON, *Hill-climbing algorithms for the construction of combinatorial designs*, Ann. Discrete Math., 26 (1985), pp. 321–334.
[15] D. R. STINSON AND S. A. VANSTONE, *A few more balanced Room squares*, J. Austral. Math. Soc. Ser. A, 39 (1985), pp. 344–352.
[16] M. TOMPA, *Hill-climbing: a feasible search technique for the construction of combinatorial configurations*, M.Sc. thesis, University of Toronto, Toronto, Ontario, Canada, 1975.