

UCLA

Research Reports

Title

Optimizing Two-Level Supersaturated Designs Using Swarm Intelligence Techniques

Permalink

<https://escholarship.org/uc/item/3qh2c1jp>

Journal

Technometrics, 58(1)

ISSN

0040-1706 1537-2723

Authors

Phoa, Frederick Kin Hing

Chen, Ray-Bing

Wang, Weichung

et al.

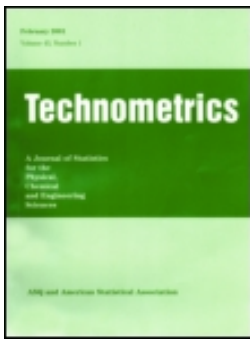
Publication Date

2016-02-22

DOI

10.1080/00401706.2014.981346

Peer reviewed




Optimizing Two-Level Supersaturated Designs Using Swarm Intelligence Techniques

Frederick Kin Hing Phoa, Ray-Bing Chen, Weichung Wang & Weng Kee Wong

To cite this article: Frederick Kin Hing Phoa, Ray-Bing Chen, Weichung Wang & Weng Kee Wong (2016) Optimizing Two-Level Supersaturated Designs Using Swarm Intelligence Techniques, Technometrics, 58:1, 43-49, DOI: [10.1080/00401706.2014.981346](https://doi.org/10.1080/00401706.2014.981346)



To link to this article: <http://dx.doi.org/10.1080/00401706.2014.981346>

 View supplementary material 

 Accepted author version posted online: 11 Mar 2015.
Published online: 22 Feb 2016.

 Submit your article to this journal 

 Article views: 66

 View related articles 

 View Crossmark data 

Optimizing Two-Level Supersaturated Designs Using Swarm Intelligence Techniques

Frederick Kin Hing PHOA

Institute of Statistical Science
Academia Sinica
Taipei 11529, Taiwan
(fredphoa@stat.sinica.edu.tw)

Ray-Bing CHEN

Department of Statistics
National Cheng Kung University
Tainan, Taiwan
(rbchen@stat.ncku.edu.tw)

Weichung WANG

Institute of Applied Mathematical Sciences
National Taiwan University
Taipei 10617, Taiwan
(wwang@math.ntu.edu.tw)

Weng Kee WONG

Department of Biostatistics
University of California, Los Angeles
Los Angeles, CA 90095
(wkwong@ucla.edu)

Supersaturated designs (SSDs) are often used to reduce the number of experimental runs in screening experiments with a large number of factors. As more factors are used in the study, the search for an optimal SSD becomes increasingly challenging because of the large number of feasible selection of factor level settings. This article tackles this discrete optimization problem via an algorithm based on swarm intelligence. Using the commonly used $E(s^2)$ criterion as an illustrative example, we propose an algorithm to find $E(s^2)$ -optimal SSDs by showing that they attain the theoretical lower bounds found in previous literature. We show that our algorithm consistently produces SSDs that are at least as efficient as those from the traditional CP exchange method in terms of computational effort, frequency of finding the $E(s^2)$ -optimal SSD, and also has good potential for finding D_3 -, D_4 -, and D_5 -optimal SSDs. Supplementary materials for this article are available online.

KEY WORDS: Balanced design; Columnwise-pairwise algorithm; D_f -criterion; $E(s^2)$ -criterion.

1. INTRODUCTION

Researchers are increasingly interested to investigate large-scale systems with a large number of potentially relevant factors. The cost of such a study can be prohibitive in terms of time, labor, and monetary resources. To address these challenges, research in experimental design has lately focused on supersaturated designs (SSDs) for their run-size economy.

A two-level SSD with N runs and m factors is represented by an $N \times m$ matrix X with every entry equal to 1 or -1 . It is assumed that $N < m$ for run-size economy and no two columns of X are identical. An SSD is called balanced if $+1$ and -1 levels of each factor appear the same number of times in the design. The balance property ensures that main effects and interactions are orthogonal so that the effect of each factor can be estimated and tested as if it were the only one under consideration and there is very little loss in efficiency in the presence of other factors (Wu and Hamada 2000). In what is to follow, we focus on balanced designs, which means that N is an even number. The effect sparsity principle (Box and Meyer 1986) suggests that only a few factors are active in practice and an SSD is cost saving for factor screening purposes. This has led to many refined analyses for SSDs in recent years, see Georgiou (2014) and Phoa (2014).

There are several ways to construct SSDs. Satterthwaite (1959) was an early proponent of constructing SSDs using random balanced designs. Lin (1993) proposed a class of special SSDs that can be easily constructed using half-fractions of the Hadamard matrices. We recall that a k -

dimensional Hadamard matrix H is a $k \times k$ square matrix with entries ± 1 and satisfies $HH^T = kI$, and I is the $k \times k$ identity matrix. A library of Hadamard matrices is available at <http://neilsloane.com/hadamard/> and reader unfamiliar with such matrices may refer to Seberry and Yamada (1992) for further discussion. Booth and Cox (1962) proposed a formal approach by minimizing the average nonorthogonality between all pairs of columns in the design matrix as an optimality criterion and called this the $E(s^2)$ criterion. Specifically, the $E(s^2)$ value of an SSD is

$$E(s^2) = \sum_{i < j} s_{ij}^2 / \binom{m}{2},$$

where s_{ij} is the dot product between the i th and j th columns of X . Both Nguyen (1996) and Tang and Wu (1997) independently derived a useful lower bound for the design criterion of a balanced SSD with m -factors and N -runs:

$$E(s^2) \geq \frac{m - N + 1}{(m - 1)(N - 1)} N^2.$$

This lower bound is called the Nguyen-Tang-Wu bound. When m is a multiple of $N - 1$, Cheng (1997) showed that an SSD that achieves the Nguyen-Tang-Wu bound is equivalent to

a balanced incomplete block design (BIBD) with $N - 1$ treatments and m blocks of size $N/2 - 1$. Bulutoglu and Cheng (2004) provided a BIBD-based theoretical method for constructing some SSDs that achieve the Nguyen-Tang-Wu bound. Bulutoglu (2007) gave a theoretical method for finding an N run $E(s^2)$ -optimal SSD that attained the Nguyen-Tang-Wu bound when there are m factors and $N = 0 \pmod{4}$ or when there are $2m$ factors and $N = 2 \pmod{4}$. A comprehensive list of work in the construction of optimal SSDs when there are many factors in the study is available in Liu and Liu (2011).

The design criterion $E(s^2)$ is a measure of nonorthogonality between any two active factors out of the m factors, and it is desirable to minimize it. Wu (1993) considered a more general setup when there are f -active-factors. For a user-specified positive integer f , let $X_{(f)}$ be an $N \times f$ submatrix of the full design matrix X and let

$$D_f = \sum_{X_{(f)}} \left| \frac{1}{N} X_{(f)}^T X_{(f)} \right|^{1/f} / \binom{m}{f},$$

where the summation is over all submatrices $X_{(f)}$. A D_f -optimal design maximizes the D_f value over all possible designs and when $f = 2$, Li and Wu (1997) showed that $E(s^2)$ and D_2 -criteria are closely related. In this article, we propose an algorithm based on swarm intelligence to find balanced $E(s^2)$ -optimal SSDs and has good potential for finding D_f -optimal designs as well.

Section 2 gives a brief discussion of swarm intelligence and shows how we develop an algorithm to find SSDs using ideas from swarm intelligence. In Section 3, we implement our algorithm to optimize SSDs under the $E(s^2)$ criterion and compare results with those from Bulutoglu and Cheng (2004), Bulutoglu (2007), and Lin (1993). Additionally, we study properties of the SSDs found using our algorithm. Section 4 discusses potential applications of our algorithm to find other types of optimal SSDs and Section 5 summarizes our work with some future directions of the work.

2. SWARM INTELLIGENCE FOR FINDING $E(s^2)$ -OPTIMAL SSDS

Swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence. The expression was introduced by Beni and Wang (1993) in the context of cellular robotic systems. A coffee table book by Fisher (2009) describes swarm intelligence in layman language and its broad applications to solve different types of real problems in different disciplines. Kennedy et al. (2001) provided a monograph with a good introduction to the topic. It also has some technical details and applications of swarm intelligence to solve real problems in various fields. Recent advances on swarm intelligence are available in the chapters of Tan et al. (2011) and Zhang et al. (2014).

According to Corne et al. (2012), SI systems consist typically of a population of many individual agents that share the following properties. (1) The individual agents are largely homogeneous. (2) The individual agents act asynchronously in parallel. (3) There is little or no centralized control. (4) Communication between agents is largely effected by some form of stigmergy.

(5) The “useful behavior” is relatively simple (finding a good place for food, or building a nest—not writing a symphony, or surviving for many years in a dynamic environment). In other words, SI is about the cooperation among these individual agents to achieve a particular goal. Examples of algorithms based on swarm intelligence include ant, bee, bat, cuckoo, and several others.

One of the most popular swarm intelligence-based algorithms is particle swarm optimization (PSO) proposed by Kennedy and Eberhart (1995) with follow-up work by Shi and Eberhart (1998) and Eberhart and Shi (2001), among many others. It is a good illustrative member of this class of algorithms. The optimization technique was originally aimed at studying social network structure and now it is widely used in computational intelligence, industrial optimization, computer science, and engineering research. PSO is also used in training neural networks (Braendler and Hendtlass 2002), solving dynamic economic dispatch problems (Chakrabarti et al. 2006), pole shape optimization problems (Brandstatter and Baumgartner 2002), quadratic assignment problems (Gong and Tuson 2008), and many other high-dimensional optimization problems in the real world with multiple optima.

PSO is a population-space-based optimization tool that begins with a population of particles and the whole population is traced during the PSO procedures. The iterative procedure of PSO relies on heuristics and stochastic and it has been shown in many applied disciplines that PSO usually converges or nearly converges to the global optimum. According to Engelbrecht (2005), Pathak et al. (2009), Panduro et al. (2009), and many others, the main advantages of PSO are that (i) time required to find the optimum is usually short compared with other methods, (ii) the algorithm is readily amenable to find the optimum for another model or another criterion, (iii) there are few tuning parameters in PSO and they are relatively easy to work with compared to other algorithms, such as genetic algorithm, and (iv) it is computationally inexpensive because the algorithm requires only primitive mathematical operations and minimal memory space. More interestingly, PSO does not require any assumption on the objective function to be optimized. This is in contrast to other methods of optimization, where, for example, linear programming requires the objective function to be linear and the Newton–Raphson’s method requires that the objective function be differentiable.

2.1 The SIBSSD Procedure

In this section, we used ideas from swarm intelligence to design an algorithm for finding optimal SSDs. We call it the swarm intelligence-based SSD algorithm and refer it as SIBSSD for short. Table 1 shows the main steps where each particle (SSD) is first mixed with selected SSDs and then moved or replaced by another SSD by the MOVE operation discussed below.

The initialization step can be viewed as the zeroth iteration of SIBSSD. Users have to first input a stopping criterion and the swarm size, which is the number of randomly generated $N \times m$ SSDs to be used in the search. Each of these SSDs has m two-level columns of length N with equal numbers of -1 and 1 levels. Here similar to PSO, at any particular time, each particle

Table 1. The SIBSSD algorithm

	Randomly generate a set of balanced $N \times m$ SSDs as initial particles
1:	Evaluate objective function value of each SSD
2:	Initialize the LB for all SSDs
3:	Initialize the GB
4:	while not converge do
5:	For each SSD, perform the MIX operation
6:	For each SSD, perform the MOVE operation
7:	Evaluate objective function value of each SSD
8:	Update the LB for all SSDs
9:	Update the GB
10:	end while
11:	

(SSD) has its own perceived location of the optimum in the search space and with each iteration, it moves toward it and then updates its perceived optimum location. This perceived optimum location of the particle (SSD) is its smallest criterion value attained by the particle to date and this SSD is often called the “local best” (LB). Throughout, the particles (SSDs) share information by comparing its LB with other LBs and collectively arrived at the perceived overall optimum location; this position is called “global best” (GB). Thus at initialization step, users may also input optional parameters for the number of columns from each SSD to be exchanged with the LB and GB SSDs. We denote these numbers by q_{LB} and q_{GB} , respectively, and obviously, both numbers must be smaller than m , the total number of factors in the SSDs. In the supplementary material, we recommend default values for q_{LB} and q_{GB} . The values of the objective function for all the SSDs, or simply design values, are then evaluated. Since there are no prior objective function values to compare with at the initial step, we take these SSDs to be the LB SSDs. The SSD that provides the optimal value for the objective function among all SSDs is the GB SSD. For our $E(s^2)$ -optimality criterion, the optimal value is the smallest value among all the SSDs.

Following the initialization step, the iteration starts from the while-loop in Table 1. First, SIBSSD uses the MIX operation to update all SSDs by exchanging a fixed number of their columns with columns from another. In the next step, call the MOVE step, SIBSSD moves the current design closer to the best SSD. We now describe the MIX and MOVE operations.

The MIX operation is a column exchange procedure somewhat similar to the CP exchange algorithm between the two designs. It consists of deleting and adding columns from another design. For each of the generated candidate SSD $X = (x_1, \dots, x_m)$, the idea is to replace q columns in X by q columns from another SSD $Y = (y_1, \dots, y_m)$ so that the replaced design has a smaller criterion value. In practice, the SSD Y is either the LB SSD or the GB SSD. If Y is the LB SSD, the number of columns to be exchanged is $q = q_{LB}$ and if Y is the GB SSD, the number of columns to be exchanged is $q = q_{GB}$. The values for q_{LB} and q_{GB} are arbitrary but our experience is that setting $q_{LB} > q_{GB}$ generally avoids premature convergence to an SSD with relatively small criterion value without adequate exploration of the search space. Our numerical studies in part 1 of the supplementary material suggest that setting $q_{LB} = [m/3]$ or

$[m/4]$ and $q_{GB} = [m/6]$, respectively, are reasonable choices. Here $[x]$ refers to the rounded positive integer at least as large as x .

The column deletion step for a design X is a recursive procedure that determines which column is the best column to remove from X . To do this, we first construct all m reduced designs X_{-1}, \dots, X_{-m} , where X_{-i} consists of all columns of X except the i th column ($1 \leq i \leq m$) and assume that among all these designs now with $m - 1$ factors, X_{-d} has the largest objective function value. We delete column d from the current design and treat it as the full design X in the next step. This deletion procedure continues sequentially until q columns are deleted from the original design X . We denote the reduced design with N rows and $m - q$ columns by R . The column addition step is the reverse of the column deletion step, where q columns from Y are sequentially added in a similar way to the reduced design R . We do this by first constructing extension matrices $R_i = [R; y_i]$, where R_i contains all columns in R and y_i is the i th column from Y , $i = 1, \dots, m$. The selected extended matrix is $[R; y_d]$, if the objective function value of R_d is minimal among all columns of Y . The process continues until q columns are filled into the reduced matrix R . When this mixing process for the candidate design X is complete, we have a new design called *mixwGB* if Y is the GB SSD and a new design called *mixwLB* if Y is the LB SSD. These two resulting designs incorporate the properties of the LB and GB SSDs in the search for the new SSD in the next iteration while retaining properties from the current SSD. This completes the MIX operation process for the particle X .

The MOVE operation comes after each X has completed the MIX procedure and it applies to each X as follows. Movement of a particle is accomplished by replacing each current SSD with possibly another SSD altogether, depending on its current value. If the value of *mixwGB* is smaller than the values of *mixwLB* and X , we replace the current SSD X by *mixwGB*. If, on the other hand, *mixwLB* has the smallest value among all the three designs, X is replaced by *mixwLB*; otherwise some columns of X are randomly chosen to be replaced by some random balanced columns. We recommend the number of such columns to be replaced is q_{LB} . The rationale for the random replacement is that it helps to ensure that the search space is sufficiently explored and at the same time, also reduces the possibility that X may be trapped in a local optimum. We denote the updated SSD design X in this way by *mixwRC*.

After each particle has completed the MOVE operation, the objective function values of all SSDs are evaluated again and compared to the values of the individual LB SSDs. If the updated SSD has a smaller value than its LB SSD value, the current SSD will be considered as the new LB SSD. Otherwise no change is made to the LB SSD. Next, the value of the best LB SSD among all LB SSDs is compared to that of the GB SSD, and this design is updated only if the best LB SSD has a smaller value. After all current SSDs, LB SSDs, and the GB SSD are updated, the search continues into its next iteration in the same way until the prespecified maximum number of iterations is reached or the GB SSD reaches its optimal value if known.

The MOVE operation has some similarities with the crossover and mutation operators in genetic algorithm. However, the significant differences between the two operations are (i) the order

in the MOVE operation is to perform a crossover before selection, but it is the opposite in genetic algorithm, and (ii) each SSD is mixed with LB or GB SSD in the MOVE operation, but some genes are crossed over with some better genes in genetic algorithm that are not necessarily the best.

2.2 Differences Between the SIBSSD and CP Exchange Algorithms

Li and Wu (1997) proposed the CP exchange algorithm for obtaining SSDs with good design properties under the D_f criterion. The MIX operation of SIBSSD has similar ideas but is based on swarm intelligence. First, the CP exchange algorithm selects a fixed number of columns to be exchanged at the beginning of each iteration and then updates them individually. In contrast, SIBSSD deletes and adds columns sequentially via the recursive procedure described in the MOVE operation. Second, the MIX operation in SIBSSD generates the candidate SSDs, and then an acceptance-rejection rule is used to make the decision. The LB and GB SSDs in SIBSSD are two candidate sets that are generated from our MIX operation, but the candidate column set in the CP exchange algorithm is obtained by switching the signs in the column. Third, SIBSSD allows a third candidate to mix with the current SSD by randomly exchanging columns to reduce the probability of a design being trapped in a local optimum. In contrast, the CP exchange algorithm always stops when there is no further improvement to be made.

3. RESULTS

In this section, we present results and characteristics of balanced SSDs found from the SIBSSD algorithm using $E(s^2)$ as the design criterion. Recall that we focused on balanced SSDs because a lower bound for $E(s^2)$ is available for such designs in Bulutoglu and Cheng (2004) and one may use these lower bounds as proxies to the best possible values, which in many cases, are the optimal values for $E(s^2)$. Accordingly, we propose to compare our SIBSSD-generated designs using the measure called E_r -efficiency defined by

$$E_r = \frac{E(s^2)_{\text{optimal}}}{E(s^2)_X} \times 100\%.$$

Here $E(s^2)_{\text{optimal}}$ is the best lower bound found in the literature for $E(s^2)$ and $E(s^2)_X$ is the $E(s^2)$ of the SSD X . Obviously, the larger the value of E_r , the better the SSD X is. If $E_r = 100\%$, X is $E(s^2)$ -optimal.

Table 2 lists the SSDs with the best E_r that we have found via SIBSSD for selected values of N and m . The first and second columns are, respectively, the number of rows and columns of SSDs. The third column is the lower bound for $E(s^2)$ obtained in Theorem 3.1 of Bulutoglu and Cheng (2004). The fourth column is the E_r -efficiency between our best and optimal SSDs. The fifth to eighth columns have the same headers as the first four columns but with different values of N and m .

When $N = 10$ and 12 , the table shows all but one of the SIBSSD-generated SSDs attain the values of lower bound for $E(s^2)$, implying that they are $E(s^2)$ -optimal. An exception is when the SSD has 10 rows and 13 columns, where the lower bound is not attained but its E_r -efficiency is about 96%. When

Table 2. E_r -efficiencies of SIBSSD-generated SSDs as measured by the lower bounds given in Bulutoglu and Cheng (2004)

N	m	Lower bound	E_r -efficiency	N	m	Lower bound	E_r -efficiency
10	10	4.00000	100.0%	14	18	5.67320	100.0%
	11	4.00000	100.0%		19	6.05848	100.0%
	12	4.00000	100.0%		20	6.35790	100.0%
	13	4.61538	95.75%		21	6.66667	98.87%
	14	5.05495	100.0%		22	6.90909	96.15%
	15	5.52381	100.0%		23	7.41502	100.0%
	16	5.86667	100.0%		24	7.82609	100.0%
	17	5.88235	100.0%		25	7.84000	97.35%
	18	5.88235	100.0%		26	7.84000	96.37%
12	12	2.18182	100.0%	27	8.38746	98.00%	
	13	3.69231	100.0%		28	8.80423	99.76%
	14	4.21978	100.0%		29	8.82758	98.46%
	15	4.57143	100.0%		30	8.82758	95.71%
	16	5.20000	100.0%	16	16	2.13333	100.0%
	17	5.64706	100.0%		17	3.76471	100.0%
	18	5.96078	100.0%		18	4.18300	95.24%
	19	6.45614	100.0%		19	4.49122	87.27%
	20	6.82105	100.0%		20	5.38947	91.43%
	21	6.85714	100.0%		21	6.09523	97.56%
	22	6.85714	100.0%		22	6.64935	96.97%
14	14	4.00000	100.0%	23	7.08300	97.39%	
	15	4.00000	100.0%		24	7.42029	96.97%
	16	4.00000	88.24%		25	7.68000	96.00%
	17	4.94118	100.0%		26	7.87692	95.24%

$N = 14$ and $m \leq 24$, the SSDs are still $E(s^2)$ -optimal except when we have $m = 16, 21$, and 22 factors. When the number of columns is larger than 24 , the SIBSSD-generated SSDs did not attain the values of the lower bound for $E(s^2)$ but all have consistently high E_r values averaging above 97%. When $N = 16$ and $m \leq 26$, we observe that the E_r values of the SIBSSD-generated designs are all above 91% except for the SSD when $m = 19$, in which case the E_r -efficiency is 85.7%.

Readers who are interested in the exact design structure can refer to the three tables in the supplementary materials. These tables provide designs with larger ranges of m than Table 2 does. In specific, in Table S3, $10 \leq m \leq 30$ for $N = 10$ and $12 \leq m \leq 36$ for $N = 12$; in Table S4, $14 \leq m \leq 42$ for $N = 14$; and in Table S5, $16 \leq m \leq 48$ for $N = 16$.

The above observations suggest that for some cases, SIBSSD has difficulties producing $E(s^2)$ -optimal SSDs when we use 100 or fewer particles to conduct the search. In general, we expect having more particles in the search will require fewer number of iterations to arrive at an SSD that attains the lower bound $E(s^2)$ because having more particles is likely to result in a more complete exploration of the search space. A possible downside is that with more particles, a larger number of computational steps and criterion evaluations are expected at each iteration. In particular, the random columns mixed in the MOVE operation may keep the particles from exploring the entire search space when the GB SSD does not change after a certain number of iterations. We illustrate our claim in the following figure.

Figure 1 illustrates an exemplary case where the $E(s^2)$ values of the 14×23 SSD listed in Table 2 converge to its lower bound after about 141 iterations. Each point in the Figure 1 represents the $E(s^2)$ value of the 14×23 GB SSD at selected

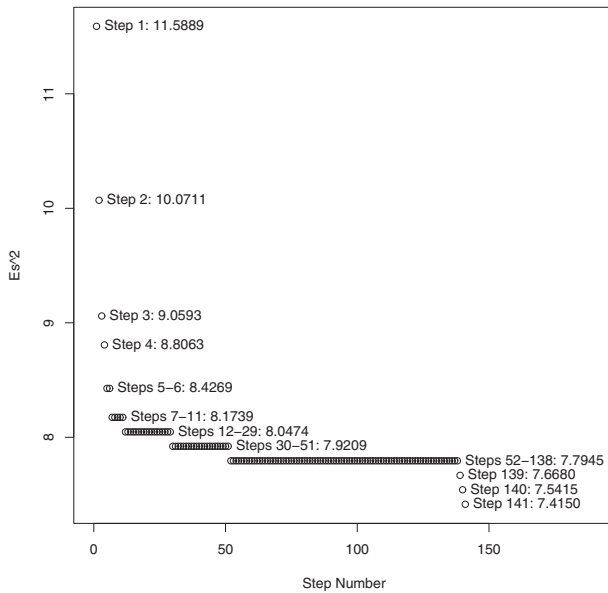


Figure 1. Convergence of the $E(s^2)$ values of an SIBSSD-generated 14×23 SSD to its optimal lower bound value of 7.4150.

iteration numbers. We observe that the $E(s^2)$ values decrease as the step number increases, and the search ends when either the magnitude reaches its lower bound or the maximum number of iterations is attained. Figure 1 shows how quickly SIBSSD generates an efficient $E(s^2)$ -optimal SSD; after the 12th iteration, the generated GB SSD has $E_r > 90\%$ and after the 52th iteration, an SSD with $E_r > 95\%$ is found. This pattern applies for all SSDs shown in Table 3 where only a small number of iterations is required to produce an SSD with high E_r -efficiency. The exceptions are cases when the SSD sought is small. Details are omitted but interested reader can request the details by writing to the first author.

The dimension of the SSDs affects the number of iterations required to generate an $E(s^2)$ -optimal design. In general, a longer time is required by SIBSSD to generate the optimal design if there are more columns or rows in the SSD. As an illustration, we used 500 particles in 500 repeated trials to generate three higher-dimensional SSDs. The average number of iterations required to obtain the 12×12 $E(s^2)$ -optimal SSD was 14.38. This average increased to 16.16 and 35.42 when the dimension of the sought optimal SSD becomes 12×13 and 12×14 , respectively. Table 3 lists the E_r -efficiencies of higher-dimensional SIBSSD-generated SSDs using the lower bounds provided in Bulutoglu (2007). These SSDs have much larger dimensions than those in Bulutoglu and Cheng (2004) and the SIBSSD algorithm required larger number of particles to generate these designs. Interestingly, the algorithm still had difficulty attaining the lower bounds for a small number of cases, specifically when $N = 18, m = 34$ and $N = 20, m = 38$. The actual SSDs produced by the algorithm are shown in part 2 of the supplementary materials.

4. DISCUSSION

This section discusses tradeoff issues between efficiency requirements and computational effort, performance of the SIBSSD algorithm relative to other methods for finding optimal SSDs, and potential applications of our algorithm to find other types of optimal SSDs.

Figure 1, along with Table 2, shows the initial swarm require more than 141 steps to converge to the $E(s^2)$ -optimal SSD with 23 factors and 14 runs. A similar pattern can be observed for other optimal SSDs shown in Tables 2 and 3. In practice, a judgment has to be made on the tradeoff between attaining an SSD with high E_r -efficiency and computational efficiency. Our guidelines on this issue are from results in part 3 of the supplementary materials, where we conducted an experiment

Table 3. E_r -efficiencies of SIBSSD-generated SSDs in larger dimensions as measured by the lower bounds from Bulutoglu (2007)

N	m	Lower bound	Our $E(s^2)$	E_r	N	m	Lower bound	Our $E(s^2)$	E_r
10	18	5.8824	5.8824	100.0%	18	34	9.8182	10.5597	92.98%
	36	8.5714	8.5714	100.0%		68	14.5075	14.6760	98.85%
	72	9.8592	9.8592	100.0%		102	16.0396	16.1204	99.50%
12	22	6.8571	6.8571	100.0%	136	16.8000	16.8488	99.71%	
	55	10.6667	10.6667	100.0%	170	17.2544	17.2812	99.85%	
	66	11.0769	11.0769	100.0%	204	17.5567	17.5767	99.89%	
	110	11.8899	11.8899	100.0%	272	17.9336	17.9475	99.92%	
	132	12.0916	12.0916	100.0%	306	18.0590	18.0721	99.93%	
	220	12.4932	12.4965	99.97%	408	18.3100	18.3177	99.96%	
14	13	4.0000	4.0000	100.0%	544	18.4972	18.5011	99.98%	
	26	7.8400	8.2339	95.22%	20	38	10.8108	11.7440	92.05%
	52	11.5294	11.5294	100.0%		57	14.2857	14.6266	97.67%
	78	12.7273	12.7699	99.67%		76	16.0000	16.1853	98.86%
	104	13.3204	13.3323	99.91%		114	17.6991	17.7885	99.50%
	130	13.6744	13.6935	99.86%		171	18.8235	18.8621	99.80%
	156	13.9097	13.9203	99.92%		190	19.0476	19.0744	99.86%
	182	14.0774	14.0851	99.95%		228	19.3833	19.4068	99.88%
	208	14.2029	14.2133	99.93%		342	19.9414	19.9575	99.92%
	234	14.3004	14.3087	99.94%					
	312	14.4952	14.4998	99.97%					

using E_r -efficiencies to compare abilities of two 14×23 SSDs to identify active factor in a screening experiment.

We have two suggestions. First, if the same method of analysis and the same criterion are used to analyze an SSD, an SSD with a lower $E(s^2)$ value is likely to provide more accurate inference in identifying the active factors. Our SIBSSD algorithm can be an efficient tool for finding SSDs with small $E(s^2)$ values when SSDs are unavailable from catalogs or softwares. Second, two SSDs with similar $E(s^2)$ values typically produce the same results from the same method of analysis. For example, Figure 1 suggests that at step 52, the SIBSSD algorithm found a GB SSD with $E(s^2) = 7.9209$. This is the 14×23 SSD proposed by Lin (1993). At step 141, the SIBSSD algorithm found a GB SSD with $E(s^2) = 7.4150$, which is $E(s^2)$ -optimal SSD. These step numbers imply that we have to spend additional amount of time to bring about roughly a 6% improvement in the $E(s^2)$ value. Is it worth spending the extra amount of time to obtain the $E(s^2)$ -optimal SSD in light of the first finding? Our answer is generally in the negative and this explains why we are satisfied with most of our SSDs in Tables 2 and 3, which have E_r -efficiencies, larger than 90%.

We also compared performance of the SIBSSD algorithm with the commonly used CP algorithm for generating efficient SSDs. Our CP algorithm exchanged three columns in one iteration and terminated the search when there was no further improvement in the next 100 iterations. Among 100 trials, the CP algorithm found the $E(s^2)$ -optimal SSDs 31 times with $E(s^2) = 4.00$. For the rest of the 69 times, CP produced SSDs with $E(s^2) = 4.49, 4.97, \text{ and } 5.46$. The frequency distribution for these three values was 48 times, 17 times, and 4 times, respectively. In contrast, the SIBSSD algorithm with a swarm size of 500 and a stopping criterion of not more than 100 iterations found the optimal SSD in all 100 times. We do not expect such excellent results to hold for every dimension but our experience is that the SIBSSD algorithm does consistently provide more efficient SSDs than the CP algorithm does and finds the $E(s^2)$ -optimal designs more often than the CP algorithm does.

Our SIBSSD algorithm is also applicable to find other types of optimal SSDs. The $E(s^2)$ criterion aims to reduce the average correlation among pairs of columns and the D_f criterion proposed by Li and Wu (1997) generalized the criterion to the case when we want to minimize the average correlation among sets of f columns. We modified our SIBSSD algorithm and found D_f -optimal SSDs when $m = 12$ factors and $N = 16$ runs. This setup is the same as the example in Li and Wu (1997). Results from the SIBSSD algorithm were compared with those obtained from Wu's method proposed in Wu (1993), Tang and Wu's method proposed in Tang and Wu (1997), CP($D = (E(s^2), m_0 = 0.33)$) and CP($D = (E(s^2), n_0 = 36)$) proposed in Li and Wu (1997). The values of m_0 and n_0 in the latter two methods are user-specified and represent the maximum absolute correlation among pairs of columns and the number of nonorthogonal pairs, respectively. The first CP method is considered to have the worst-case performance and the second CP method is considered to have average performance in terms of finding the D_f -optimal designs. Table 4 lists the various $E(s^2)$, D_3 , D_4 , and D_5 values for the generated design using the five methods.

Table 4. Performance of different methods for generating a 12×16 SSD

Method	$E(s^2)$	D_3	D_4	D_5
Wu	6.00	0.9551	0.9259	0.9011
Tang and Wu	6.00	0.9545	0.9283	0.8992
CP($D = (E(s^2), m_0 = 0.33)$)	5.20	0.9609	0.9382	0.9128
CP($D = (E(s^2), n_0 = 36)$)	5.20	0.9601	0.9366	0.9100
SIBSSD	5.20	0.9609	0.9382	0.9128

For the above comparison, we used 500 randomly generated particles or SSDs in the SIBSSD algorithm. After fewer than 10 iterations, SIBSSD produced an SSD with the following values: $D_3 = 0.9609$, $D_4 = 0.9382$, and $D_5 = 0.9128$. These D_f values are the same as the known optimal values obtained from the CP($D = (E(s^2), m_0 = 0.33)$) method. This suggests the adaptability and good performance of the SIBSSD algorithm for finding other types of D_f -optimal SSDs.

5. SUMMARY

Finding an efficient SSD for a design problem with a large number of factors is generally a difficult problem because of the complexity of the optimization problem. This article proposes a new and efficient algorithm based on swarm intelligence to search for efficient $E(s^2)$ -SSDs. Our work suggests that the proposed algorithm performs as efficient as the CP algorithm under the $E(s^2)$ -optimality criterion.

We conclude by noting that there are a number of directions for future work. First, our results in Table 4 from a small simulation study suggests that the SIBSSD algorithm has good potential for finding an optimal SSD under other design criteria, such as the D_f optimality and $f > 2$. This class of criteria is useful when we are concerned with detecting higher-order interactions during the screening process (Li and Wu 1997). Second, one can explore modifying the SIBSSD algorithm to search for (i) unbalanced SSDs by incorporating other design properties into the MIX operation and (ii) multi-level and mixed-level SSDs. These searches will undoubtedly require more computing effort and parallel computing with Graphics Parallel Units (GPUs) may be helpful. Specifically, we will let one GPU thread take charge of one particle (or one SSD) and launch a separate Compute Unified Device Architecture (CUDA) kernel at each step to achieve the necessary synchronization. Even though it is known that the resulting speedup is case and parameter dependent, it is encouraging to note that Chen et al. (2013) had found that GPU implementation can save significant computational time when large number of particles and a lot of iterations are involved.

SUPPLEMENTARY MATERIAL

Our article has supplementary material and it includes: (i) information on our numerical studies for the choice of number of columns to be exchanged in the MIX operation; (ii) design tables of SIBSSD-generated supersaturated designs (SSDs); and (iii) a practical use of an SIBSSD-generated SSD in a screening experiment analyzed using the Stepwise Response Refinement Screener (SRRS). All SSD we found in this paper are available to be downloaded in an online catalog and the website is

<http://phoa.stat.sinica.edu.tw:10080/SSD>. This online catalog is simple to use via entering basic parameters, and it is maintained continuously by the first author to make sure the results are always new and certified.

ACKNOWLEDGMENTS

All authors are grateful to the whole editorial team for a careful review of the article and the many useful suggestions and helpful comments. Phoa's research is supported by the Career Development Award of Academia Sinica (Taiwan) grant number 103-CDA-M04 and Ministry of Science and Technology (Taiwan) grant numbers 102-2628-M-001-002-MY3 and 104-2118-M-001-016-MY2. Chen's research is partially supported by the National Science Council (Taiwan) grant number 101-2118-M-006-002-MY2 and the Mathematics Division of the National Center for Theoretical Sciences (South) in Taiwan. Wang's research is partially supported by the National Science Council (Taiwan) and the Taida Institute of Mathematical Sciences. Wong's research is supported by National Science Council, Mathematics Research Promotion Center grant number 102-51; the Ministry of Education (Taiwan) Top University Project to the National Cheng Kung University. The joint research of Chen, Wang, and Wong reported in this article is also partially supported by the National Institute of General Medical Sciences of the National Institutes of Health under Award Number R01GM107639. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

[Received June 2012. Revised September 2014.]

REFERENCES

- Beni, G., and Wang, J. (1993), "Swarm Intelligence in Cellular Robotic Systems," *Robots and Biological Systems: Towards a New Bionics*, 102, 703–712. [44]
- Booth, K. H. V., and Cox, D. R. (1962), "Some Systematic Supersaturated Designs," *Technometrics*, 4, 489–495. [43]
- Box, G. E. P., and Meyer, R. D. (1986), "An Analysis for Unreplicated Fractional Factorials," *Technometrics*, 28, 11–18. [43]
- Braendler, D., and Hendtlass, T. (2002), "The Suitability of Particle Swarm Optimisation for Training Neural Hardware," in *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert System*, pp. 190–199. [44]
- Brandstatter, B., and Baumgartner, U. (2002), "Particle Swarm Optimization—Mass-Spring System Analogon," *IEEE Transactions on Magnetics*, 38, 997–1000. [44]
- Bulutoglu, D. A. (2007), "Cyclically Constructed $E(s^2)$ -Optimal Supersaturated Designs," *Journal of Statistical Planning and Inference*, 137, 2413–2428. [44,47]
- Bulutoglu, D. A., and Cheng, C. S. (2004), "Construction of $E(s^2)$ -Optimal Supersaturated Designs," *The Annals of Statistics*, 32, 1662–1678. [44,46,47]
- Chakrabarti, R., Chattopadhyay, P. K., Basu, M., and Panigrahi, C. K. (2006), "Particle Swarm Optimization Technique for Dynamic Economic Dispatch," *Journal of the Institution of Engineers (India)*, 87, 48–54. [44]
- Chen, R. B., Hsieh, D. N., Hung, Y., and Wang, W. (2013), "Optimizing Latin Hypercube Designs by Particle Swarm," *Statistics and Computing*, 23, 663–676. [48]
- Cheng, C. S. (1997), " $E(s^2)$ -Optimal Supersaturated Designs," *Statistica Sinica*, 7, 929–939. [43]
- Corne, D. W., Reynolds, A. P., and Bonabeau, E. (2012), "Swarm Intelligence," in *Handbook of Natural Computing*, eds. Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, New York: Springer, pp. 1599–1622. [44]
- Eberhart, R., and Shi, Y. (2001), "Particle Swarm Optimization: Developments, Application and Resources," *IEEE Proceedings of the Congress on Evolutionary Computation*, 1, 81–85. [44]
- Engelbrecht, A. P. (2005), *Fundamentals of Computational Swarm Intelligence*, Chichester, UK: Wiley. [44]
- Fisher, L. (2009), *The Perfect Swarm: The Science of Complexity in Everyday Life*, New York: Basic Books. [44]
- Georgiou, S. D. (2014), "Supersaturated Designs: A Review of Their Construction and Analysis," *Journal of Statistical Planning and Inference*, 144, 92–109. [43]
- Gong, T., and Tuson, A. L. (2008), "Particle Swarm Optimization for Quadratic Assignment Problems - A Forma Analysis Approach," *International Journal of Computational Intelligence Research*, 4, 177–185. [44]
- Kennedy, J., and Eberhart, R. (1995), "Particle Swarm Optimization," *Proceedings of IEEE International Conference on Neural Networks*, 4, 1942–1948. [44]
- Kennedy, R. C., Eberhart, R., and Shi, Y. (2001), *Swarm Intelligence*, San Francisco, CA: Morgan Kaufmann Publishers. [44]
- Li, W. W., and Wu, C. F. J. (1997), "Columnwise-Pairwise Algorithms With Applications to the Construction of Supersaturated Designs," *Technometrics*, 39, 171–179. [44,46,48]
- Lin, D. K. J. (1993), "A New Class of Supersaturated Designs," *Technometrics*, 35, 28–31. [43,44,48]
- Liu, Y., and Liu, M. Q. (2011), "Construction of Optimal Supersaturated Design With Large Number of Levels," *Journal of Statistical Planning and Inference*, 141, 2035–2043. [44]
- Nguyen, N. K. (1996), "An Algorithmic Approach to Constructing Supersaturated Designs," *Technometrics*, 38, 69–73. [43]
- Panduro, M. A., Brizuela, C. A., Balderas, L. I., and Acosta, D. A. (2009), "A Comparison of Genetic Algorithms, Particle Swarm Optimization and the Differential Evolution Method for the Design of Scannable Circular Antenna Arrays," *Progress in Electromagnetics Research*, B, 13, 171–186. [44]
- Pathak, N., Mahanti, G. K., Singh, S. K., Mishra, J. K., and Chakraborty, A. (2009), "Synthesis of Thinned Planar Circular Array Antennas Using Modified Particle Swarm Optimization," *Progress in Electromagnetics Research Letters*, 12, 87–97. [44]
- Phoa, F. K. H. (2014), "The Stepwise Response Refinement Screener (SRRS)," *Statistica Sinica*, 24, 197–210. [43]
- Satterthwaite, F. E. (1959), "Random Balance Experimentation," *Technometrics*, 1, 111–137. [43]
- Seberry, J., and Yamada, M. (1992), "Hadamard Matrices, Sequences and Block Designs," in *Contemporary Design Theory: A Collection of Surveys*, eds. J. H. Dinitz and D. R. Stinson, New York: Wiley, pp. 431–560. [43]
- Shi, Y., and Eberhart, R. (1998), "A Modified Particle Swarm Optimizer," *Proceedings of the IEEE Congress on Evolutionary Computation*, 1998, 69–73. [44]
- Tan, Y., Shi, Y., Chai, Y., and Wang, G. (2011), *Lecture Notes in Computer Science: Advances on Swarm Intelligence* (Vol. 6728), Berlin: Springer. [44]
- Tang, B., and Wu, C. F. J. (1997), "A Method for Constructing Supersaturated Designs and Its $E(s^2)$ Optimality," *Canadian Journal of Statistics*, 25, 191–201. [43,48]
- Wu, C. F. J. (1993), "Construction of Supersaturated Designs Through Partially Aliased Interactions," *Biometrika*, 80, 661–669. [44,48]
- Wu, C. F. J., and Hamada, M. (2000), *Experiments: Planning, Analysis, and Parameter Design Optimization*, New York: Wiley. [43]
- Zhang, Y., Agarwal, P., Bhatnagar, V., Balochian, S., and Zhang, X. (2014), "Swarm Intelligence and Its Applications 2014," *The Scientific World Journal*, 2014, Article ID 204294. [44]