Graph based isomorph-free generation of two-level regular fractional factorial designs

Abhishek K. Shrivastava and Yu Ding
Department of Industrial & Systems Engineering, Texas A& M University, College Station, TX 77843-3131, USA

***Abstract*—We develop a new method for generating non-isomorphic 2-level fractional factorial designs. The design problem is modeled as a graph isomorphism problem so that the methods available in graph theory can be applied. Doing so provides an innovative isomorphism check and can substantially reduce computation time in generating non-isomorphic designs.**

I. Introduction

Fractional factorial designs are a popular choice in designing experiments for studying the effects of multiple factors simultaneously. They are especially preferable for screening experiments where it is commonly assumed that only a few effects are significant. Recently, fractional factorial designs with large run sizes have been reported for use in screening experiments, for example, with over 600 runs in Lin and Sitter [2008]. The first step in planning such an experiment is the selection of an appropriate design. This requires searching through a catalog of designs. This paper provides an efficient method of constructing this catalog in the case of 2-level regular fractional factorial designs.

Two fractional factorial designs are isomorphic or equivalent to each other if the design table of one design can be obtained from the other by some permutation (i.e., relabeling) of factor, run and level labels. For example, the two 2^{7-3} fractional factorial designs shown in Figs. 1(a) and 1(b) are isomorphic to each other, as shown in Fig. 1(c). Isomorphic fractional factorial designs have the same statistical properties and lead to the same ANOVA models in the subsequent analysis. Moreover, the number of (mutually) non-isomorphic designs is much smaller than the original set. For example, the total number of 2^{15-10} designs is 5,311,735, whereas the number of non-isomorphic designs of resolution ≥ 3 is only 144 [Chen et al., 1993]. Therefore, keeping only non-isomorphic designs considerably reduces the effort in searching the appropriate design.

Enumeration or cataloging of non-isomorphic fractional factorial designs involves removing isomorphs (equivalent designs) from the set of all possible designs. This is difficult for two reasons. Firstly, the total number of candidate designs, from which isomorphs are to be eliminated, increases exponentially with the number of independent factors ($= n - k$, in a 2^{n-k} design). In general, the total number of 2^{n-k} designs is $\binom{(2^{n-k}-1)-(n-k)}{k}$. Secondly, the problem of checking whether two given fractional factorial designs are isomorphic or not is combinatorial and becomes increasingly difficult with design size.

	A	B	C	D	E	F	G
1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	1
3	0	0	1	0	0	1	0
4	0	0	1	1	0	1	1
5	0	1	0	0	1	0	1
6	0	1	0	1	1	0	0
7	0	1	1	0	1	1	1
8	0	1	1	1	1	1	0
9	1	0	0	0	1	1	0
10	1	0	0	1	1	1	1
11	1	0	1	0	1	0	0
12	1	0	1	1	1	0	1
13	1	1	0	0	0	1	1
14	1	1	0	1	0	1	0
15	1	1	1	0	0	0	1
16	1	1	1	1	0	0	0

(a) $E = AB$, $F = AC$, $G = BD$

	A	B	C	D	E	F	G
1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	1
3	0	0	1	0	0	1	1
4	0	0	1	1	0	1	0
5	0	1	0	0	1	0	0
6	0	1	0	1	1	0	1
7	0	1	1	0	1	1	1
8	0	1	1	1	1	1	0
9	1	0	0	0	1	1	0
10	1	0	0	1	1	1	1
11	1	0	1	0	1	0	1
12	1	0	1	1	1	0	0
13	1	1	0	0	0	1	0
14	1	1	0	1	0	1	1
15	1	1	1	0	0	0	1
16	1	1	1	1	0	0	0

(b) $E = AB$, $F = AC$, $G = CD$

	A	C	B	D	F	E	G
1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	1
5	0	0	1	0	0	1	1
6	0	0	1	1	0	1	0
3	0	1	0	0	1	0	0
4	0	1	0	1	1	0	1
7	0	1	1	0	1	1	1
8	0	1	1	1	1	1	0
9	1	0	0	0	1	1	0
10	1	0	0	1	1	1	1
13	1	0	1	0	1	0	1
14	1	0	1	1	1	0	0
11	1	1	0	0	0	1	0
12	1	1	0	1	0	1	1
15	1	1	1	0	0	0	1
16	1	1	1	1	0	0	0

(c) Reordered matrix of (a). Relabeling $B \leftrightarrow C$, $E \leftrightarrow F$ and rows gives (b)

Fig. 1. Design matrices of two 2^{7-3} designs

Various necessary and sufficient checks for design isomorphism, i.e., testing two designs for isomorphism, have been proposed in literature. The reader may refer to Katsaounis and Dean [2008], Lin and Sitter [2008] for a review. The most efficient necessary and (conjectured) sufficient check for 2-level regular fractional factorial designs was given by Lin and Sitter [2008]. They construct a word-pattern matrix for each design and compute eigenvalues of certain submatrices. If two designs have distinct eigenvalues then they are non-isomorphic. They conjecture that two non-isomorphic designs will always

have distinct eigenvalues. They further provide an efficient algorithm for enumerating non-isomorphic designs. To the best of our knowledge, Lin and Sitter [2008]'s enumeration algorithm, along with conjectured eigenvalue check, is currently the most efficient procedure. Therefore, we use them as a benchmark for comparing our results.

We provide a new approach for testing the isomorphism of two 2-level regular fractional factorial designs by modeling them as simple bipartite graphs. The problem is now transformed into a graph isomorphism problem. We use an efficient graph isomorphism check algorithm [McKay, 1981] to provide a necessary and sufficient check for design isomorphism. For generating the complete set of non-isomorphic 2-level fractional factorial designs, we improve Lin and Sitter [2008]'s enumeration algorithm by using our isomorphism check and reducing the candidate designs from which isomorphs are to be eliminated.

The contributions of our paper are two-fold: (i) we provide a new necessary and sufficient check for design isomorphism; (ii) we provide an enumeration algorithm that can generate non-isomorphic designs much faster than any of the previous methods. The reason that our enumeration algorithm is faster can be understood as follows. Firstly, for removing isomorphs from a collection, we run the computationally expensive isomorphism check only once for each design and not once for each pair of designs. The only other existing method that does this is Lin and Sitter [2008]'s eigenvalue check. However, Lin and Sitter [2008]'s eigenvalue check is not guaranteed to correctly distinguish two isomorphic designs. Other methods, e.g., Clark and Dean [2001], compare pairs of designs to determine if they are isomorphic to each other or not. Secondly, because we model the problem as a graph isomorphism problem, we are able to reduce the number of candidate designs considered, when eliminating isomorphs. Not only does our algorithm generate non-isomorphic designs much faster, it is also able to generate designs with run sizes of 2048 and 4096 runs, which were not generated by any existing methods.

This paper is organized as follows. In Section II we give details on our proposed isomorphism check. Section III gives the algorithm for enumerating non-isomorphic designs. We provide computational results in Section IV and compare our enumeration algorithm's run times with those from our implementation of Lin and Sitter [2008]. Finally, in Section V we conclude the paper and discuss further extensions.

For the remainder of the paper, by design or fractional factorial design we will always mean 2-level regular fractional factorial design, unless stated otherwise.

II. The isomorphism check

A 2-level regular fractional factorial design, 2^{n-k}, has n factors, each with two levels, and consists of 2^a $(a = n - k)$ runs. Thus, it is the $\frac{1}{2^k}$th fraction of a 2^n full factorial design, where the fraction is determined by k *defining words*. Each *word* consists of letters (e.g., A, B, C, D) denoting factors. Each letter, denoting a factor, is an element of Galois field $GF(2)$. The defining words form an abelian group called the *defining contrast subgroup*. The defining words are a set of generators for this group. The group, therefore, consists of 2^k words, including the identity element which is usually denoted by I. A fractional factorial design is usually presented in a table, the experimental plan, listing the (ordered) runs and the factor levels at each run. A design table / matrix is similar to the experiment table but without any specific ordering of the runs. A 2-level regular fractional factorial design (matrix) is uniquely defined by the number of factors and its defining contrast subgroup (or equivalently, a set of defining words). Fig. 1(a) shows the design table of a 16-run 2^{7-3} designs that has defining words $\{ABE, ACF, BDG\}$. It must be noted that the defining words are not unique for a given design, i.e., two different set of defining words may exist that generate the same defining contrast subgroup.

We represent a 2^{n-k} design by the tuple $\{n, S\}$, where S is the defining contrast subgroup. It is easy to see that this representation is immutable under row label permutations as it does not list any runs. Changes in levels also do not change the contrast subgroup. Permutation of level labels in a design table only leads to the same design table with permuted row labels. This gives us the result in Lemma 2.1.

Lemma 2.1: Two 2-level regular fractional designs, $d_1 \equiv \{n, S_1\}$ and $d_2 \equiv \{n, S_2\}$, are isomorphic to each other *if and only if* one of S_1 or S_2 can be obtained from the other by some permutation of factor labels and reordering of words.

II.1 *2-level fractional factorial designs as graphs*

A simple undirected *graph* $G(V, E)$ consists of disjoint finite sets V of *vertices*, and E of *edges*. Each edge is a pair of distinct vertices, and no two edges repeat in the edge set E. We provide here a new bipartite graph representation of a 2-level regular fractional factorial design. A bipartite graph $G(V_a, V_b, E)$ is a graph in which the vertex set V can be partitioned into disjoint subsets V_a and V_b such that each edge has one vertex in V_a and one in V_b.

Algorithm 1: Construction of bipartite graph $G(V_a, V_b, E)$ for design $d \equiv \{n, S\}$
Input: design $d \equiv \{n, S\}$

1) Start with an empty graph with no vertices, i.e., $V_a = \phi$ and $V_b = \phi$ (and hence, no edges, i.e.,

$E = \phi$).

2) For each factor in the design d, add a vertex in V_a, i.e., add vertices $v_{a1}, \ldots, v_{an}$ in V_a.
3) For each word, in the defining contrast subgroup S, add a vertex in V_b, i.e., add vertices $v_{b1}, \ldots, v_{b|S|}$ in V_b, where $|S|$ denotes the cardinality of set S.
4) For each word in S, add edges between the vertex (in V_b), corresponding to the word, and the vertices (in V_a), corresponding to the factors in the word.

The vertex sets V_a and V_b form the two partitions of the graph. From Algorithm 1, it is clear that there exists a bipartite graph for each 2-level regular fractional factorial design. Fig. 2 shows the graph representation of the 2^{7-3} design given in Fig. 1(a), with contrast subgroup $\{I,\ ABE,\ ACF,\ BDG,\ ADEG,\ BCEF,\ CDEFG,\ ABCDFG\}$.

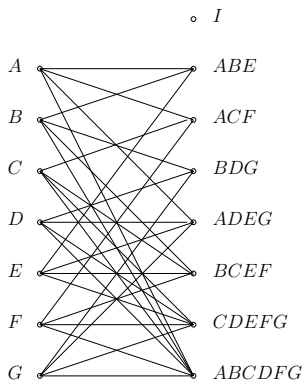

Fig. 2. Bipartite graph for the 2^{7-3} design in Fig. 1(a). Vertices on the left, set V_a, correspond to factors, and vertices on the right, set V_b, correspond to words in the contrast subgroup.

II.2 *Graph isomorphism and fractional factorial design isomorphism*

The graph isomorphism problem is to check, given two graphs, if there exists a permutation of vertex labels of one graph that makes it identical to the other. The fractional factorial design isomorphism problem can be transformed to the problem of checking isomorphism between the corresponding graph representations of the designs. The permutations of factor labels and reordering of words, of a design, then correspond to the permutations of vertex labels within V_a and V_b partitions, respectively.

Theorem 2.2: Two 2-level regular fractional factorial designs, $d_1 \equiv \{n, S_1\}$ and $d_2 \equiv \{n, S_2\}$, with graph representations $G_1(V_{a1}, V_{b1}, E_1)$ and $G_2(V_{a2}, V_{b2}, E_2)$, respectively, are isomorphic to each other *if and only if* G_1 and G_2 are isomorphic to each other.

Theorem 2.2 gives a *necessary and sufficient condition* for checking the isomorphism between two fractional factorial designs by solving the graph isomorphism problem. The solution to the graph isomorphism problem is discussed in Section II.3. We skip the proof of Theorem 2.2 here due to space limitations. The theorem holds primarily because Algorithm 1 gives a bijection from the set of designs to the set of bipartite graphs constructed from these designs. So, for each design there is a unique graph, and for each graph (in the set of graphs constructed from these designs) there is a unique design.

II.3 *Solving the graph isomorphism problem*

The graph isomorphism problem has been extensively studied in mathematics and computer science. For a review on the history of the problem and algorithmic developments towards solving this problem, please see Read and Corneil [1977], Fortin [1996].

We use the so called *canonical labeling* approach for solving the graph isomorphism problem. In this approach, a canonical graph is obtained for each graph. The canonical graph is computed in such a way that if two graphs are isomorphic to each other, then the canonical graphs computed for the two graphs are identical. The most efficient canonical labeling algorithm is implemented in a C package *nauty* [McKay, 2004] based on McKay [1981]. This package is available freely for research purposes from the developer's website [McKay, 2004]. The algorithm is known to take exponential running time, in the number of vertices, in the worst case [Kocay, 1996]. Thus, in the worst case, the design isomorphism problem can be solved in exponential time in the number of words in the defining contrast subgroup (since Algorithm 1 requires $O(n \cdot |S|)$ running time to transform a design to a graph). But, in practice *nauty* has been found to be extremely efficient for most graphs and out-performs all other graph-isomorphism algorithms [Kocay, 1996]. Therefore, we expect our isomorphism check to also be very efficient for most designs.

III. Generating non-isomorphic 2-level regular fractional factorial designs

We enumerate the set of non-isomorphic 2^{n-k} designs in a recursive manner. We start with the only $2^a (a = n-k)$ full factorial design, generate all the non-isomorphic $2^{(a+1)-1}$ designs, then all non-isomorphic $2^{(a+2)-2}$ designs, $\ldots$, and finally all non-isomorphic 2^{n-k} designs. Each intermediate step involves constructing larger (child) designs by adding a new factor and defining word to smaller (parent) designs.

Fig. 3 gives the algorithm at the intermediate step, for generating the set of non-isomorphic $2^{(n+1)-(k+1)}$ designs from the set of 2^{n-k} designs. This algorithm differs from Lin and Sitter [2008] at the two steps highlighted with dashed boxes in Fig. 3. The Lin and Sitter [2008] algorithm does not have our candidate word

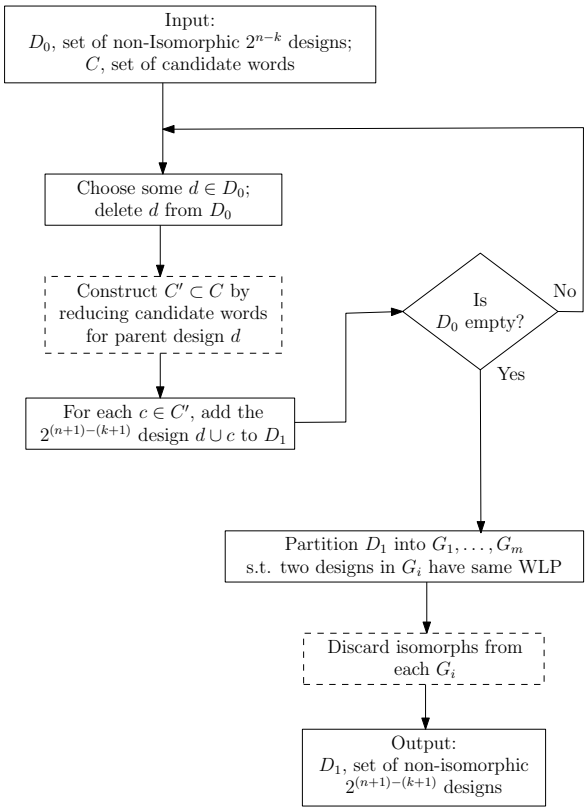

Fig. 3. Algorithm for generating the complete set of non-isomorphic $2^{(n+1)-(k+1)}$ designs from the set of non-isomorphic 2^{n-k} designs. Dashed boxes highlight steps different from Lin and Sitter [2008]'s algorithm.

reduction method and uses a different isomorphism check for discarding the isomorphs in the penultimate step.

The algorithm in Fig. 3 proceeds by first constructing the set of candidate defining words, C. It is the set of all possible ways the new factor can be defined with the $(n-k)$ independent factors. For example, for generating the 2^{7-3} designs from 2^{6-2} designs, the candidate words are AG, BG, $\ldots$, $BCDG$, $ABCDG$. We then reduce this set of candidate words, as described in Section III.1. Next, the set D_1 of candidate $2^{(n+1)-(k+1)}$ designs is constructed by adding each of these candidate words as a defining word to each of the non-isomorphic 2^{n-k} (parent) designs. We then partition D_1 into sets $G_1, \ldots, G_m$ containing designs with the same word length pattern (WLP). Word length pattern (WLP) of a design is a vector, with its jth element equal to the number words of length j in the defining contrast subgroup of the design. It is known that word length pattern is a necessary check for distinguishing isomorphic designs [Draper and Mitchell, 1967], i.e, two designs with distinct word length patterns are non-isomorphic. We then discard isomorphs from each set G_i as described in Section III.2.

III.1 *Reducing candidate words for each parent design*

Reducing the candidate words for each parent design leads to fewer designs in D_1. Thus, fewer designs need to be checked for isomorphism, a computationally expensive procedure.

Suppose the 2^{6-2} design, d_1, with defining contrast subgroup $\{ABE, ACF, BCEF\}$ is currently the parent design. We want to construct 2^{7-3} designs from this design. Consider the two defining words BDG and CDG in the set C of all candidate words. Adding BDG to d_1 gives the design in Fig. 1(a) and adding CDG to d_1 gives the design in Fig. 1(b). These two designs are isomorphic under the factor label permutations ($B \leftrightarrow C$, $E \leftrightarrow F$). We would ideally want to consider only one of these two candidate words. Applying this same permutation to the contrast subgroup of d_1 gives $\{ACF, ABE, CBFE\}$, which is the same as the original contrast subgroup of d_1. Therefore, a factor label permutation ($B \leftrightarrow C$, $E \leftrightarrow F$), which does not alter the parent design, has detected candidate words that lead to isomorphic designs.

In general, the candidate words that are non-isomorphic, under any permutation that does not alter the design, only need to be kept. These candidate words form the reduced set of candidate words for the design, C'. This idea of reducing the candidate words is borrowed from graph theory [McKay, 1998]. The candidate words, along with the parent design, can be represented as graphs (by Algorithm 1). The set of permutations that do not alter the graph form the automorphism group of the graph. The reduced candidate words are non-isomorphic under this automorphism group of the parent graph. This automorphism group of the graph is also computed using the C package *nauty*. Our graph representation of designs allows us to extend this graph theory result to designs.

III.2 *Discarding isomorphs*

For discarding the isomorphs in the penultimate step of the algorithm, we use the graph based isomorphism check described in Section II. This involves, for some partition G_i, constructing the graph for each design, computing the canonical graph (using *nauty*) and comparing the canonical graphs to discard any duplicates. Lin and Sitter [2008] use two different sufficiency checks in two versions of their enumeration algorithm. In the first one, they use the sufficiency check given by Clark and Dean [2001] that compares pairs of designs, within some partition G_i, to check whether they are isomorphic or not. In the second one, for each design in the partition G_i, they construct a word pattern matrix, compute eigenvalues of certain submatrices of this word pattern matrix and compare these eigenvalues to discard any duplicates. This eigenvalue check is conjectured to be sufficient. This second version of their algorithm outperforms their first version. Hence, we use this second version to benchmark

our algorithm. One of the primary reasons for this improvement in speed, in this second version, is from the fact that the second version requires the expensive isomorphism check to be computed only $|G_i|$ times, once for each design in the partition, compared to $\frac{|G_i|(|G_i|-1)}{2}$ isomorphism check calls, once for each pair of designs, in the first version. In Section IV, we will compare our results with this second version of the Lin and Sitter [2008] algorithm.

III.3 *Admissibility of the generation procedure*

Theorem 3.1: The algorithm in Fig. 3 generates the complete set of non-isomorphic $2^{(n+1)-(k+1)}$ designs.

Theorem 3.1 ensures that the algorithm in Fig. 3 gives the correct and desired result. We only sketch the proof here. Neglecting our candidate word reduction and replacing our isomorph discarding procedure with that of Clark and Dean [2001], the algorithm in Fig. 3 is identical to the that in Lin and Sitter [2008]. Their method gives the complete set of non-isomorphic designs. Since our isomorph discarding procedure uses a necessary and sufficient condition (of Section II), replacing Clark and Dean [2001]'s isomorphism check with our's will not alter the algorithm's output. Also, since the candidate word reduction keeps at least one design from each isomorphism class, the set D_1 before the isomorph discarding step, still contains at least one design from each isomorphism class. Therefore, the output from our algorithm is the complete set of non-isomorphic designs.

IV. Results

Using the algorithm in Fig. 3 we were able to generate all the designs generated by Lin and Sitter [2008]. Additionally, we could generate 1024-run (resolution≥ 6) designs up to 20 factors, and all of 2048-run (resolution≥ 7) and 4096-run (resolution≥ 8) designs. All the computations were done on a Windows Server 2003 R2 Standard x64 edition with an Intel Xeon 3GHz processor and 16 GB RAM. The programs were written in C++ and built as 32-bit applications on the Microsoft Visual C++ 8.0 compiler. Our implementation of Lin and Sitter [2008]'s eigenvalue condition uses LAPACK++ [Stimming, 2007], a C++ library for high performance linear algebra computations.

Table I shows the run times for Lin and Sitter [2008] (with the eigenvalue criteria as sufficient condition, EigVal), our graph-based algorithm without the candidate reduction (GBAnoR) and our graph-based algorithm with the candidate reduction (GBA). These run times include the time needed to generate a design through the recursive procedure starting from the full factorial design. Compared to EigVal run times, for $k \geq 4$, the run times for GBAnoR are 2% or less for 128 and 512-run designs, and 2 – 4% for 256-run designs. Since, the only difference between EigVal and GBAnoR is the isomorphism check used, these large differences indicate that our isomorphism check is significantly faster than the eigenvalue check in Lin and Sitter [2008]. Better yet, our check is proven to be necessary and sufficient whereas their's is only proven necessary. The incremental improvement in run times by including the candidate reduction method is much lesser. For $k \geq 4$, the run times for GBA are between 50 – 80% of the run times for GBAnoR. This gain is better evident for large designs that take longer time to generate. For example, for the 512-run designs with $k = 8$, the further reduction in run time due to candidate reduction method is about 2.5 minutes (150 seconds).

The improvement due to candidate reduction is better reflected in Table II. The number of designs left in D_1, from which isomorphs need to deleted, is about 4000 – 8000 lesser, for 128-run designs with $k \geq 7$, when candidate reduction is used. For larger designs, for which the calls to *nauty* could be more expensive, such reductions will lead to considerable reduction in computation times.

V. Conclusion and Discussion

We develop a new necessary and sufficient check for testing the isomorphism of two 2-level fractional factorial designs. This check is based on a graph representation of the design. This check differs from other (proven) sufficient checks [Chen et al., 1993, Clark and Dean, 2001] in that it does not directly compare two designs. Instead, the method generates a canonical representation of a design such that two isomorphic designs always have the same canonical representation. Thus, for comparing a collection of m designs, our method is more efficient as it needs to be run only m times, a smaller number compared to $\frac{m(m-1)}{2}$ pairs in the collection. This feature of our sufficiency check makes it a better choice in the non-isomorphic fractional factorial design generation algorithm.

Using results from the graph isomorphism problem we improve the existing design generation algorithm of Lin and Sitter [2008] by reducing the number of designs to be tested for isomorphism. We use this algorithm to generate 2-level designs for run sizes up to 4096. The computational results provide empirical evidence that the algorithm and our isomorphism check, both, are significantly faster than the current best.

The isomorphism check and the design generation algorithm can be extended to 2-level regular fractional factorial split plot designs. The graph constructed in this case will be a colored bipartite graph. The vertices corresponding to the factors will need to be partitioned into two sets – one corresponding to the whole-plot factors and the other corresponding to the sub-plot fac-

TABLE I

CUMULATIVE CPU TIME (IN SECONDS) FOR GENERATING DESIGNS: COMPARISON BETWEEN [LIN AND SITTER, 2008] (EIGVAL), OUR GRAPH-BASED ALGORITHM WITH NO CANDIDATE REDUCTION (GBANOR) AND OUR GRAPH-BASED ALGORITHM WITH CANDIDATE REDUCTION (GBA)

k	128 run ($R \geq 4$)			256 run ($R \geq 5$)			512 run ($R \geq 5$)		
	EigVal	GBAnoR	GBA	EigVal	GBAnoR	GBA	EigVal	GBAnoR	GBA
1	0.062	0.000	0.015	0.046	0.015	0.000	0.203	0.031	0.015
2	0.249	0.015	0.015	0.312	0.031	0.015	2.109	0.109	0.046
3	1.843	0.046	0.031	1.906	0.093	0.031	20.155	0.499	0.156
4	12.484	0.218	0.125	6.609	0.218	0.093	126.341	2.328	0.765
5	84.029	1.140	0.671	17.671	0.515	0.265	750.344	11.327	5.453
6	523.646	5.515	3.515	31.530	0.953	0.546	5119.450	58.592	38.484
7	3290.970	26.265	18.390	41.467	1.406	0.921	*	328.462	269.687
8	21401.300	132.997	101.062	42.858	1.703	1.187	*	1927.920	1771.250
9	*	774.532	629.093	43.061	1.890	1.265	*	*	*

* Problem size is too large. It prevents a valid run to complete.

TABLE II

NUMBER OF DESIGNS IN D_1 BEFORE DISCARDING ISOMORPHS: COMPARISON BETWEEN [LIN AND SITTER, 2008], OUR GRAPH-BASED ALGORITHM WITHOUT CANDIDATE REDUCTION (GBANOR) AND OUR GRAPH-BASED ALGORITHM WITH CANDIDATE REDUCTION (GBA)

k	128 run ($R \geq 4$)			256 run ($R \geq 5$)			512 run ($R \geq 5$)		
	EigVal	GBAnoR	GBA	EigVal	GBAnoR	GBA	EigVal	GBAnoR	GBA
1	98	98	98	162	162	162	381	381	381
2	185	185	62	227	227	68	703	703	166
3	495	495	177	409	409	146	2,063	2,063	496
4	1,273	1,273	703	480	480	206	4,739	4,739	1,497
5	3,346	3,346	2,026	453	453	267	11,077	11,077	5,731
6	7,560	7,560	4,952	205	205	137	25,913	25,913	18,444
7	15,336	15,336	11,110	51	51	42	*	60,545	52,917
8	28,766	28,766	22,572	2	2	2	*	132,909	128,292
9	*	49,708	41,421	0	0	0	*	*	*

* Problem size is too large. It prevents a valid run to complete.

tors. Each set of vertices will have a distinct color. The graph isomorphism, in this case, also needs to preserve the colors of the vertices. So, permutations are allowed only within vertices of the same color. The program *nauty* can still be used for finding the canonical graph representation. The extensions to regular multi-level and non-regular designs are non-trivial and constitute our future work.

REFERENCES

J. Chen, D. X. Sun, and C. F. J. Wu. A catalogue of two-level and three-level fractional factorial designs with small runs. *International Statistical Review*, 61 (1):131–145, 1993.

J. B. Clark and A. M. Dean. Equivalence of fractional factorial designs. *Statistica Sinica*, 11:537–547, 2001.

N. R. Draper and T. J. Mitchell. The construction of saturated 2_r^{k-p} designs. *The Annals of Mathematical Statistics*, 38(4):1110–1126, 1967.

S. Fortin. The graph isomorphism problem. Technical Report TR 96–20, Department of Computer Science, The University of Alberta, Edmonton, Alberta, Canada, 1996.

T.I. Katsaounis and A.M. Dean. A survey and evaluation of methods for determination of combinatorial equivalence of factorial designs. *Journal of Statistical Planning and Inference*, 138(1):245–258, 2008.

William Kocay. *Computational and Constructive Design Theory*, volume 368, chapter On writing isomorphism programs, pages 135–175. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

C. D. Lin and R. R. Sitter. An isomorphism check for two-level fractional factorial designs. *Journal of Statistical Planning and Inference*, 138(4):1085–1101, 2008.

B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30(1):45–87, 1981.

B. D. McKay. Isomorph-free exhaustive generation. *J. Algorithms*, 26(2):306–324, 1998.

B. D. McKay. *nauty Users Guide (Version 2.2)*. Computer Science Department, Australian National University, ACT 0200, Australia, Oct 2004. URL http://cs.anu.edu.au/people/Brendan.McKay/nauty/nug.pdf.

R. C. Read and D. G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1(1):339–363, 1977.

Christian Stimming. *Lapack++ 2.5.2*, July 2007. URL http://lapackpp.sourceforge.net/.