Central Composite Discrepancy-Based Uniform Designs for Irregular Experimental Regions

Ray-Bing Chen, Yen-Wen Hsu, Ying Hung, and Weichung Wang

November 1, 2012
Central Composite Discrepancy-Based Uniform Designs for Irregular Experimental Regions

Ray-Bing Chen\textsuperscript{a}, Yen-Wen Hsu\textsuperscript{b}, Ying Hung\textsuperscript{c}, Weichung Wang\textsuperscript{b,*}

\textsuperscript{a}Department of Statistics, National Cheng Kung University, Tainan 701, Taiwan
\textsuperscript{b}Department of Mathematics, National Taiwan University, Taipei 10617, Taiwan
\textsuperscript{c}Department of Statistics and Biostatistics, Rutgers University, NJ 08854, USA

Abstract

Central composite discrepancy (CCD) has been proposed to measure the uniformity of a design over irregular experimental region. However, how CCD-based optimal uniform designs can be efficiently computed remains a challenge. Focusing on this issue, we proposed a set of particle swarm optimization-based algorithms to efficiently find optimal uniform designs with respect to the CCD criterion. Parallel computation techniques based on state-of-the-art graphic processing unit (GPU) are employed to accelerate the computations. Several two- to five-dimensional benchmark problems are used to illustrate the advantages of the proposed algorithms. By solving a real application in data center thermal management, we further demonstrate that the proposed algorithm can be extended to incorporate the non-collapsing property.

Keywords: Uniform Design, Central Composite Discrepancy, Discrete Particle Swarm Optimization, Graphic Processing Unit, Parallel Computing, Non-collapsing

1. Introduction

Efficient experimental design is crucial in the study of scientific experiments. These experiments may have high dimensional inputs, often prohibits straightforward approaches for running the experiment over a dense grid of input configurations. In particular, uniform design is one of the most widely used approaches in the literature (Fang, 1980; Fang et al., 2000, 2002, 2003; Ma and Fang, 2004). It possesses a desirable space-filling property in which the design points are placed over the entire experimental region. Further description of space-filling designs can be found in Santner et al. (2003).

The concept of uniform design is attractive. However, such designs are developed mainly for regular experimental regions, such as rectangular or hypercubic regions. In practice, the assumption of the regularity of experimental regions is often violated especially in computer experiments (Ranjan et al., 2008; Hung et al., 2010; Hung, 2011). Although a few design approaches have been...
proposed for irregular regions, such as Stinstra et al. (2003), Auffray et al. (2012) and Draguljić et al. (2012), they were not developed for uniform designs. Recently, Chuang and Hung (2010) proposed central composite discrepancy (CCD) as a new uniformity criterion for a large class of irregular regions. Lin et al. (2010) proposed a threshold accepting algorithm to generate the U-type designs used for the irregular regions based on CCD.

In this article, we focus on the CCD criterion for optimal uniform designs on arbitrary irregular experimental regions. Although CCD appears to be a reasonable extension of irregular regions, its computational challenges in optimal design search and uniformity evaluations remain unsolved, especially for larger numbers of factors and run sizes. This computational issue is the main focus of the current paper. The contributions of this paper are outlined as follows:

- A new particle swarm optimization method (PSO) for searching optimal CCD designs is proposed. The algorithm is a stochastic population-based heuristic that inherits the efficiency and capability of PSO for solving high-dimensional optimization problems with multiple optima.
- The complexity of the discrete version of the CCD criterion is analyzed and the result suggests that the computational cost increases rapidly. Parallel computing techniques based on the latest graphic processing unit (GPU) are thus applied to significantly accelerate the CCD function evaluations.
- The proposed algorithms are implemented and numerical results are compared with existing methods for two and three-dimensional irregular experimental regions. Results on higher dimension regions are also provided to demonstrate the capability of the proposed algorithms.
- The algorithms are used to solve a computer data center thermal management problem (Hung et al., 2010; Hung, 2011). In this real world application, we also show how the proposed algorithms can be easily extended to handle disconnected experimental regions and the non-collapsing property.

This paper is organized as follows. In Section 2, we introduce the central composite discrepancy criterion and state our optimization goal. The complexity of the CCD criterion evaluation is also analyzed. In Section 3, we propose a GPU-accelerated discrete particle swarm optimization (DPSO) algorithm to find the optimal designs over irregular experimental regions in terms of the CCD criterion. In Section 4, we apply the proposed algorithm to solve the flexible region benchmark examples and compare its performance to an existing method. In Section 5, we demonstrate how the algorithm can be extended to disconnected experimental regions and the reduction of collapse. Final conclusions are discussed in Section 6.

2. Central Composite Discrepancy and the Complexity Analysis

After introducing the definitions of uniform design and CCD, we study the computational complexity of the central composite discrepancy in this section. First, the definition of the uniform
design is given as follows. To spread out the design points uniformly over an experimental region \( D \), where \( D \subseteq R^K \) and \( K \) is the number of variables (or factors), Fang (1980) proposed the following uniform design:

**Definition 1 (Uniform design).** Suppose \( D \) is the experimental region in \( R^K \). Let \( P = \{p_1, \cdots , p_n | p_i \in D, i = 1, \cdots , n \} \) be an \( n \)-point design in \( D \) and let \( Z(n) \) be the set of all possible \( n \)-points designs in \( D \). Give a measure of uniformity, \( MU \). Then, a design \( P^* \in Z(n) \) is called a uniform design if

\[
MU(P^*) = \min_{P \in Z(n)} MU(P).
\]

For the uniformity measurements of the regular experimental region (i.e., hypercube), the \( L_p \)-star discrepancy (Hua and Wang, 1981) is widely used. Some generalizations of the \( L_p \)-star discrepancy are centered \( L_p \)-discrepancy and wrap-around \( L_p \)-discrepancy (Hickernell, 1998). However, these discrepancies cannot be used in instances where the experimental region is not a hypercube shape, such as the convex polygon in Chuang and Hung (2010). For an irregular experimental region, Chuang and Hung (2010) proposed the central composite discrepancy which can be treated as an extension of the centered \( L_2 \)-discrepancy. The definition of the CCD is given as follows.

**Definition 2 (Central composite discrepancy).** Give an \( n \)-points design \( P = \{p_1, \cdots , p_n \} \) on \( D \subseteq R^K \). The central composite discrepancy value of this design \( P \) is defined as

\[
CCD_P = \left\{ \frac{1}{v(D)} \int_D \left[ \frac{1}{MK} \sum_{k=1}^{MK} \frac{N(D_k(x), P)}{n} - \frac{v(D_k(x))}{v(D)} \right]^p dx \right\}^{1/p}.
\]

Here, \( v(D_k(x)) \) and \( v(D) \) denote the volume of \( D_k(x) \) and \( D \), respectively, \( p > 0 \), and \( N(D_k(x), P) = \sum_{i=1}^n I\{p_i \in D_k(x)\} \). The experimental region \( D \) is partitioned into \( MK \) sub-domains \( D_k(x) \). In particular, letting \( x^{(i)} = \{x : x+a_i < r \leq x+a_{i+1}, i = 0, \cdots , M-1\} \), where \( a_0 \equiv -\infty, a_M \equiv \infty, a_1 < a_2 < \cdots < a_{M-1} \) and some \( a_j = 0 \) for \( 1 \leq j \leq M-1 \), and \( D_k(x) = \{x^{(i)}, \cdots , x^{(j)}_{\{k\}} \} \cap D \).

The idea behind CCD is to first partition the experimental region \( D \) into several sub-regions and then compute both the ratio of the number of design points in each sub-region to the total number of design points and the ratio of the sub-region volume to the overall experimental region volume. In a uniform design, we expect these two ratios to be close, so the difference of these two ratios can be a criterion of uniformity. As the CCD criterion chooses every point in the region as the center to partition and take the integral over the whole region, it can be applied to all kind of experimental regions. More details about the CCD can be found in Chuang and Hung (2010) and Lin et al. (2010).

As mentioned in Chuang and Hung (2010), searching for the optimal design over a continuous experimental region based on the CCD criterion is an NP-hard problem. Consequently, it is common (Chuang and Hung, 2010; Lin et al., 2010) to consider a discrete version of CCD and then search for the discrete uniform design (or near uniform design) for practical considerations. In particular, we consider the following discrete version of CCD. Instead of choosing design points arbitrarily in the experimental domain by Definition 1, the candidate design points are limited to a set of the grid points in the experimental region. We apply the idea of “grid level” proposed
in Fang et al. (2000) to define the following discrete uniform design (denoted as $\overline{UD}$) and the corresponding discrete CCD (denoted as $\overline{CCD}$).

Next, we define the discrete uniform design and discrete CCD. Without loss of generality, we assume that the experimental region can be transformed to a region within a unit hypercube in the definition. Based on these definitions, Theorem 5 provides a complexity analysis for computing the discrete CCD over the candidate grid points, which is helpful for efficiently designing the search approach.

**Definition 3 (Discrete uniform design).** Define $\bar{D}(q^K) = \{x = (x_1, \ldots, x_K) | x_i = \frac{2l_i - 1}{2q}, l_i = 1, \ldots, q, i = 1, \ldots, n\} \cap D$, where $K$ is dimension, $q$ is grid level, and $D$ is the experimental region. Let $\bar{P} = \{p_1, \ldots, p_n\}$ be a discrete design if $n$ design points $p_i \in \bar{D}(q^K), i = 1, \ldots, n$. Let $\bar{Z}(n, q^K)$ be the set of all possible discrete designs with $n$ design points. A design $\bar{P}^* \in \bar{Z}(n, q^K)$ is called a uniform discrete design of $n$ design points under $q$ grid level if

$$MU(\bar{P}^*) = \min_{\bar{P} \in \bar{Z}(n, q^K)} MU(\bar{P}).$$

(3)

The corresponding discrete CCD value is defined below. For simplicity, $\bar{D}(q^K)$ is denoted as $\bar{D}$. In addition, we count the number of grid points falling in the $\bar{D}_k(x)$ and $\bar{D}$ as $|\bar{D}_k(x)|$ and $|\bar{D}|$ to approximate the volume of $v(\bar{D}_k(x))$ and $v(\bar{D})$, respectively. The detail of how to compute $\overline{CCD}_p(\bar{P})$ are presented in Appendix A.1.

**Definition 4 (Discrete CCD).** We define the discrete CCD value

$$\overline{CCD}_p(\bar{P}) = \left\{ \frac{1}{v(\bar{D})} \sum_{x \in \bar{D}} \frac{1}{M^K} \sum_{k=1}^{M^K} \frac{1}{n} \left| \frac{N(\bar{D}_k(x), \bar{P})}{N(\bar{D}, \bar{P})} - \frac{v(\bar{D}_k(x))}{v(\bar{D})} \right|^{1/p} \right\}^{1/p}$$

$$= \left\{ \frac{1}{|\bar{D}|} \sum_{x \in \bar{D}} \frac{1}{M^K} \sum_{k=1}^{M^K} \frac{1}{n} \left| \frac{N(\bar{D}_k(x), \bar{P})}{|\bar{D}|} - \frac{|\bar{D}_k(x)|}{|\bar{D}|} \right|^{1/p} \right\}^{1/p}. \quad (4)$$

Here, $\bar{D}_k(x) = \{x | x \in D_k(x) \cap \bar{D}\}$. $N(\bar{D}_k(x), \bar{P}) = \sum_{i=1}^{n} I\{p_i \in \bar{D}_k(x)\}$, $v(\bar{D}_k(x)) = |\bar{D}_k(x)|/q^K$, and $v(\bar{D}) = |\bar{D}|/q^K$.

The following theorem quantifies the computation complexity of $\overline{CCD}_p(\bar{P})$. It also suggests that the computational cost for evaluating $\overline{CCD}_p(\bar{P})$ increases rapidly in terms of the factor numbers and the number of candidate design points.

**Theorem 5 (Complexity analysis of the discrete CCD).** Let $n$ be the design point number, $K$ be dimensionality, and $|\bar{D}|$ be the number of grid points in $D$. Then, for a $n$-points discrete design with design points $\bar{P} = \{p_1, \ldots, p_n\}$, the computation complexity of $\overline{CCD}_p(\bar{P})$ on Eq. (4) is $O(|\bar{D}|KMN^K(n + |\bar{D}|))$. Furthermore, if we choose $|\bar{D}| = n^K$ (in the hypercubic domain we get this by setting grid level $q = n$ ), the complexity becomes $O(KMN^K n^{2K})$.

**Proof.** The main cost for computing $\overline{CCD}_p(\bar{P})$ is the computation of $N(\bar{D}_k(x), \bar{P})$ and $v(\bar{D}_k(x))$. Here, $1 \leq k \leq M^K$ and $x \in \bar{D}$. For computing $N(\bar{D}_k(x), \bar{P})$, we must consider whether each design
point is in the corresponding region. To compute for one point requires $O(K)$, and thus requires $O(nK)$ in total. Computation for $v(\bar{D}_k(x))$ is similar. Therefore, the computation complexity of the former and the latter is $O(nK)$ and $O(|D|K)$, respectively. The two summation entails that we must compute these two values $O(KM^K(n + |D|))$ times; thus, overall it needs $O(|\bar{D}|KM^K(n + |D|))$ to compute a single $\bar{CD}_p(\mathcal{P})$.

3. Computational Methods for Optimal Discrete CCD Uniform Design

The discussions in Section 2 result in the following discrete optimization problem for the discrete uniform design:

$$\min_{\mathcal{P} \in \mathbb{Z}(n,q)} \bar{CD}_p(\mathcal{P}).$$

(5)

The fundamental difficulty in solving this discrete optimization problem is the large number of candidate designs. For example, if finding the optimal uniform design with 11 points from a square domain with $31 \times 31 = 961$ grid points (i.e., $K = 2$, $n = 11$, and $q = 31$), the number of possible designs is $|\mathbb{Z}(11, 31^2)| \approx 1.53 \times 10^{25}$. It is thus essential to develop an efficient algorithm to solve the discrete optimization problem defined in (5).

Particle swarm optimization is a population-based stochastic optimization technique. Many studies have shown its ability to provide better results in a faster and cheaper way, especially for multiple extremes and high-dimensional search regions. PSO has also been used in statistical problems. For example, Toala et al. (2011) studied the kriging parameter estimation in a computer experiment via a modified PSO. In this section, we propose a discrete variant of PSO to solve the optimization problem defined in (5). In addition, to accelerate the computations of the discrete CCD values for each design (Theorem 5 has shown its fast growing complexity), the GPU is employed. The power of the GPU acceleration is demonstrated in Section 4.

3.1. Discrete particle swarm optimization

The PSO was first introduced by Kennedy and Eberhart (1995; 2002). The idea originated from simulations of large-scale social behavior, such as in bird flocks or fish schools. At the beginning of PSO, a group of particles is randomly initialized on the solution space. Then, the particles move around the solution space to search for the optimal solutions. These particles keep their own cognitive memories on the best values visited. This “personal best” location is denoted as $y_i(t)$ for the $i$th particle and the $t$th iteration. The particles also exchange social information to obtain the best value obtained by the swarm. This “group best” location is denoted as $\hat{y}(t)$. At each iteration, the position of each particle is updated by the formula

$$x_i(t + 1) = x_i(t) + v_i(t + 1).$$

(6)

In particular, the $j$th component of the location update $v_i(t + 1)$ is defined as

$$v_{ij}(t + 1) = v_{ij}(t) + c_1 r_{1ij}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2ij}(t)[\hat{y}_j(t) - x_{ij}(t)],$$

(7)
where $c_1$ and $c_2$ are positive acceleration constants used to scale the contribution of the cognitive and social components, respectively, and $r_{1,i}(t)$ and $r_{2,i}(t)$ are independently generated from $U[0, 1]$.

The aforementioned PSO is designed for continuous domains and cannot be used directly for solving the discrete optimization problem (5). Another obstacle arises from the irregular domain. It is difficult to develop a general scheme for keeping the particles moving within various irregular domains such that the PSO will not spend unnecessary objective function evaluations that are outside the irregular experimental regions.

To overcome these difficulties, we propose the following discrete particle swarm optimization algorithm (DPSO) to directly solve the problem (5). The DPSO algorithm differs from the original PSO algorithm in how the particles are updated. DPSO ensures that the particles are moving within the irregular experimental domain. Unlike the continuous-based update formula (7), DPSO updates the particle positions by the following two steps.

**TowardBestMove:** We represent an $n$-point uniform design with $K$ variables $\bar{P}$ as a $n \times K$ matrix. In particular, $[p'_1, \ldots, p'_n]'$, where $p_i$ is a $K \times 1$ vector to denote the position of the $i$th run. For each row of the current particle design matrix $\bar{P}$ (i.e., a point in design), we assume it has a probability $probG$ ($probP$) to be replaced by the corresponding groups
RandMove on x-dim element of current particle

RandMove on y-dim element of current particle

Figure 2: An example for RandMove.

(personal) best particle.

We illustrate this idea regarding how we move the particles toward the group or personal best points by the example shown in Figure 1. This example is a 3-point design for 2 variables, and the designs are represented by $3 \times 2$ matrices. Suppose the groups best particle is 
\[
\begin{bmatrix}
2 & 4 \\
1 & 1 \\
4 & 2
\end{bmatrix}
\]
the personal best particle is 
\[
\begin{bmatrix}
3 & 0 \\
0 & 2 \\
5 & 3
\end{bmatrix}
\]
and the current particle is 
\[
\begin{bmatrix}
1 & 2 \\
0 & 1 \\
5 & 2
\end{bmatrix}
\]. After TowardBestMove proceeds, the current particle design matrix becomes 
\[
\begin{bmatrix}
2 & 4 \\
0 & 2 \\
5 & 2
\end{bmatrix}
\]. That is, the first row is replaced by group best, the second row is replaced by personal best, and the third row is unchanged.

RandMove: The TowardBestMove step may lead the search process to a local optimal point. Therefore, we insert an additional step to allow the particle to depart from the neighborhood of a local optimal point by the following means. Each element of the current particle design matrix has a probability $\text{prob}R$ to be replaced by another element which is uniformly randomly generated from feasible grid levels in that dimension.

For example, in Figure 2(a), the current particle design matrix is 
\[
\begin{bmatrix}
2 & 4 \\
1 & 1 \\
4 & 2
\end{bmatrix}
\]. After a RandMove step, the element in the second row and first column of the design matrix is replaced by a number uniformly chosen from the set \{0, 1, 2, 3, 4, 5\}. Figure 2(b) shows an example in which the element in the third row and the second column of the design matrix is replaced by a number uniformly chosen from the set \{0, 1, 2, 3\} due to the limitation of the experimental region.

We conclude this section with the following three remarks.

1. Both TowardBestMove and RandMove steps ensure that the movement of particles remains within the irregular experimental region.
2. The DPSO algorithm is outlined in Algorithm 1.
3. For the tuning parameters $\text{prob}G$, $\text{prob}P$, and $\text{prob}R$ in the algorithm, we suggest choosing $\text{prob}G = \text{prob}P = 0.4$ and $\text{prob}R = 0.05$ or 0.1. We also use these settings in the numerical experiments.
4. Because the RandMove step leads to the possibility that all particles would explore the whole
design space $\bar{Z}(n, qK)$, and because this design space contains finite number of candidate
designs, we make the following conjecture.

**Conjecture 6 (Convergence for DPSO).** Let $\bar{D}(qK) = \{x = (x_1, \cdots, x_K) | x_i = \frac{2l_i-1}{2q}, l_i = 1, \cdots, q, i = 1, \cdots, n. \} \cap D$, where $K$ is dimension, $q$ is grid level, and $D$ is the experimental region. Let $\bar{P} = \{p_1, \cdots, p_n \}$ be a discrete design with $n$ design points $p_i \in \bar{D}(qK)$, and $i = 1, \cdots, n$. Let $\bar{Z}(n,qK)$ be the set of all possible discrete designs with $n$ design points, and $\bar{P}^* \in \bar{Z}(n,qK)$ is the uniform discrete design. Then, the solution design found by DPSO converges to $\bar{P}^*$ as the iteration number $I \to \infty$.

---

**Algorithm 1** The Discrete Particle Swarm Optimization (DPSO) Algorithm.

**Require:** probG, probP, probR, initial $n_s$-dimensional swarm $S = \{x_1, \cdots, x_{n_s} \} \subset \Omega$, $\Omega$ is the experimental region.

**Ensure:** Approximate global best position $\hat{y}$

1: while stopping criterion is not satisfied do
2:   for each particle $i = 1, \cdots, n_s$ do
3:     if $f(x_i) < f(y_i)$ then
4:       $y_i = x_i$
5:     end if
6:     if $f(y_i) < f(\hat{y})$ then
7:       $\hat{y} = y_i$
8:     end if
9:   end for
10:  for each particle $i = 1, \cdots, n_s$ do
11:     update the position via TowardBestMove
12:   update position via RandMove
13: end for
14: end while

---

3.2. Accelerating function evaluations via a GPU

The GPU was originally designed for rendering graphics. Driven by the insatiable market
demand for real-time, high-definition 3D graphics, the programmable GPUs have evolved into a
highly parallel, multithreaded, many-core processors with tremendous computational horsepower
and very high memory bandwidth. GPUs are well suitable for solving data-parallel computations
with high arithmetic intensity where the ratio of arithmetic operations to memory operations is
large. In this article, we use an NVIDIA GPU with the CUDA (NVIDIA, 2010) development
environment to accelerate the function evaluations.

From the complex analysis for computing CCD in Theorem 5, we know that the cost for
computing the CCD value for a given design increases quickly as design point number $n$ and/or
dimension $K$ increases. For example, where $K = 5$ and $n = 7$ in a hypercubic domain, $|D| = 16,807$
and the partitioned sub-region number on every grid point is $2^K = 2^5 = 32$ (see Table 1). Therefore,
to compute a $\overline{CCD}$ defined in Eq. (4), we need to
Step 1. Compute $32 \times 16,807 = 537,824$ of $N(\bar{D}_k(x), \mathcal{P})$ and $v(\bar{D}_k(x))$.

Step 2. Compute $537,824$ of ${N(\bar{D}_k(x), \mathcal{P})}$.

Step 3. Sum up $537,824$ of ${\left\{\frac{N(\bar{D}_k(x), \mathcal{P})}{n} - \frac{|\bar{D}_k(x)|}{|D|}\right\}}^2$.

Observing these three-step computations, we see that, in Steps 1 and 2, the computation of $N(\bar{D}_k(x), \mathcal{P})$, $v(\bar{D}_k(x))$, and $N(\bar{D}_k(x), \mathcal{P}) - v(\bar{D}_k(x))$ are independent between different $x_i, x_j \in \bar{D}$. Therefore, we can easily parallelize the computations on each $x \in \bar{D}$. In Step 3, we can also use multiple threads to share the workload to find the sum of these numbers. In particular, our first version of parallelization on the GPU, named DPSO-GPU-1, is illustrated as follows.

Step 1. To compute $2^K |D|$ of $N(\bar{D}_k(x), \mathcal{P})$ and $v(\bar{D}_k(x))$, we initialize $|D|$ threads in the GPU. Each thread computes $N(\bar{D}_k(x), \mathcal{P})$ and $v(\bar{D}_k(x))$, $1 \leq k \leq 2^K$ for some $x \in |D|$.

Step 2. To compute $2^K |D|$ of $N(\bar{D}_k(x), \mathcal{P}) - v(\bar{D}_k(x))^2$, we simply initialize $2^K |D|$ threads in the GPU. Each thread computes $\left\{\frac{N(\bar{D}_k(x), \mathcal{P})}{n} - \frac{|\bar{D}_k(x)|}{|D|}\right\}^2$ for some $x \in |D|$ and some $1 \leq k \leq 2^K$.

Step 3. To sum up $2^K |D|$ of $N(\bar{D}_k(x), \mathcal{P}) - v(\bar{D}_k(x))^2$, we initialize 512 threads within a single block with index $i$ for $1 \leq i \leq 512$. We treat $2^K |D|$ of $\left\{\frac{N(\bar{D}_k(x), \mathcal{P})}{n} - \frac{|\bar{D}_k(x)|}{|D|}\right\}^2$ as an array of length $2^K |D|$, whose elements are indexed by $j$ for $1 \leq j \leq 2^K |D|$. We first let the thread with index $i$ sum up array elements whose index mod 512 is $i$, and then we sum these 512 elements.

Further improvement to the efficiency of GPU can be realized by the following observations. These observations lead to another version of the GPU parallelization called DPSO-GPU-2.

- As $2^K |D|$ of $v(\bar{D}_k(x))$ only depends on grid level $q$ and experimental region $D$, the value of these $2^K |D|$ elements will not be altered in the whole DPSO algorithm. Therefore, we compute them only once in the initialization of DPSO and then store them as an array in the global memory of GPU to accelerate the speed of array element reading. In addition, we bind the array to the texture cache so that data are stored in the read-only texture memory, which leads to fast data access. Note that we also bind the data of $\bar{D}$ to texture memory.

- To $2^K |D|$ of $N(\bar{D}_k(x), \mathcal{P})$, we need to read the data of design $\mathcal{P}$ in the GPU. While we can map the memory that stores $\mathcal{P}$ in the CPU to the GPU, a more efficient way is to accelerate the data reading by first copying the data into the shared memory within each block of GPU. Like the texture memory, the shared memory also has special means for saving and loading data that are nearly as fast as the register.

- If $2^K |D|$ is too large, we need to modify the previous parallel scheme in Step 3. We partition those elements into groups, and each group contains $4 \times 512 = 2,048$ elements. We initialize multiple 512-thread blocks and let the block number be the same as the group number. First, we add up each group using one block with previous scheme Step 3 and then add the numbers obtained by each block.
Figure 3: Shapes of the flexible region for a different parameter $m$ for dimensions $K=2$ and 3.

Table 1: Chosen grid level and feasible grid point number $|D|$ for a different dimension $K$ and flexible parameter $m$.

|        | (a) Grid level | (b) Feasible grid point number $|D|$ |
|--------|----------------|-----------------------------------|
|        | $K$ | $m=9,999$ | $m=2$ | $m=1$ | $m=0.5$ | $m=0.3$ |
|        | $|D|$ | $|D|$ | $|D|$ | $|D|$ | $|D|$ |
| 2      | 11  | 13  | 17  | 27  | 49  |
| 3      | 11  | 15  | 21  | 45  | 117 |
| 4      | 11  | 15  | 25  | 71  | -   |
| 5      | 11  | 15  | 29  | 99  | -   |
| 2      | 121 | 137 | 145 | 137 | 129 |
| 3      | 1,331 | 1,791 | 1,561 | 1,385 | 1,377 |
| 4      | 14,641 | 15,649 | 16,641 | 15,609 | -   |
| 5      | 161,051 | 126,767 | 177,045 | 166,363 | -   |

We have proposed the DPSO algorithm and its GPU-based acceleration. To evaluate the performance of DPSO in searching uniform designs via discrete CCD, we consider in the next sections a set of $K$-dimension flexible region problems (Lin et al., 2010; Draper and Guttman, 1986) and a real-world application (Hung et al., 2010). This application concerns a sensor placement problem for a data center located in the IBM T. J. Watson Research Center.

4. Numerical Results for the Flexible Region Problems

Our first set of benchmark problems is the flexible region problems. As defined in Lin et al. (2010), a flexible region can be written as

$$R_m = \{ x = (x_1, \cdots, x_K) \in [0, 1]^K \mid 2(x_1 - 1/2)^m + \cdots + 2(x_K - 1/2)^m \leq 1, \}$$
and the corresponding discrete grid point set with q levels is

$$\bar{R}_m(q) = \{ x = (x_1, \cdots, x_K) | x_i = \frac{2l_i - 1}{2q}, l_i = 1, \cdots, q, i = 1, \cdots, n \} \cap R_m.$$ 

Figure 3 illustrates some examples of such flexible regions for $K = 2, 3, 4, 5$ and various m’s. Our choices of grid level $q$ for a different flexible parameter $m$ and dimension $k$ are listed in Table 1. For the same dimension, we appropriately choose $q$ such that feasible grid point numbers are similar for different flexible parameters $m$ (which correspond to different shapes of the experimental region).

All of the numerical experiments in this and the next sections were performed on a Linux workstation with Intel Xeon CPU E5520 at 2.27GHz, 24 GB main memory, and an NVIDIA TESLA C1060 GPU.

4.1. Time saving due to GPU

To illustrate how the GPU can reduce the execution time of the DPSO algorithm, we compare three versions of DPSO implementations: the CPU version, the DPSO-GPU-1, and the DPSO-GPU-2. The only difference between the CPU and GPU versions is the evaluation of the CCD function.

We chose grid level $q=11$ on the hypercubic experimental region and executed the three versions of the codes with 50 particles and 200 iterations in the DPSO algorithm to find the optimal 11-point design. In each execution of the DPSO algorithm, we evaluated the CCD function 10,000 times. We compared execution time between different versions for dimensions $K = 2, 3, 4, 5$. As
shown in Figure 4, the computational load grows in terms of the dimension, and the time saving due to the GPU also grows. DPSO-GPU-2 achieves a 55X time saving for $K = 5$.

4.2. Numerical results and comparisons for $K = 2$ and 3

Algorithm 2 The Threshold Accepting (TA) Algorithm used in Lin et al. (2010).

Require: $n_{\text{Rounds}}$, $n_{\text{Steps}}$, initial solution $x^c \in \Omega$, $\Omega$ is the solution space.

Ensure: Optimal solution $x^c \in \Omega$

Initialize threshold sequence $\{\tau_r\}$, $r = 1, \ldots, n_{\text{Rounds}}$

for $r = 1$ to $n_{\text{Rounds}}$ do
  for $i = 1$ to $n_{\text{Steps}}$ do
    randomly move to neighborhood solution $x^n \in \Omega$
    if $f(x^n) - f(x^c) < \tau_r$ then
      $x^c = x^n$
    end if
  end for
end for

In this section, the nearly uniform design for several different flexible regions is studied based on pre-specified grid point sets. Lin et al. (2010) has used the Threshold Accepting (TA) algorithm to find the $U$-type design over the flexible regions based on the CCD criterion. Here we also compare DPSO (Algorithm 1 with DPSO-GPU-2 implementation) with the TA. Note that the TA is a local search method proposed by Dueck and Scheuer (1990), and the details of the TA algorithm are shown in Algorithm 2.

According to Lin et al. (2010), the flexible parameters are chosen from $\{9999, 2, 1, 0.5, 0.3\}$. Lin et al. (2010) has shown the search results for $K = 2$ and 3. In addition to these two cases, we further consider the cases that $K = 4$ and 5. The numbers of design points are set to be 5 and 11 similar to Lin et al. (2010). In DPSO, we set $\text{prob}G = \text{prob}P = 0.4$ for all cases. For the “RandMove” step in DPSO, $\text{prob}R = 0.05$ for $m = 9999, 2, 1, 0.5$, and $\text{prob}R = 0.1$ for $m = 0.3$. To make fair comparisons, both the DPSO and TA algorithms are set to have the same number of function evaluations, which is equal to $10^5$ or $10^6$. To evaluate the algorithm stability, we independently repeat the experiment 100 times for each case of the flexible regions. In particular, we consider the following settings:

TA: 1,000 inner iterations, 100 outer iterations, total $10^5$ function evaluations.

DPSO: 200 particles, 500 iterations, total $10^5$ function evaluations.

TA+: 1,000 inner iterations, 1,000 outer iterations, total $10^6$ function evaluations.

DPSO+: 500 particles, 2,000 iterations, total $10^6$ function evaluations.

First, we consider the case in which $K = 2, 3$. The comparison results are summarized via box-plots shown in Figures 5 to 7. To remove the scale effects for all cases, we draw the box-plot...
Figure 5: Box-plot comparisons of the TA and DPSO algorithms on dimension $K = 2, 3$ with different point number $n$ and flexible parameter $m$. 

(a) $K = 2, n = 5, m = 9999$

(b) $K = 2, n = 11, m = 9999$

(c) $K = 2, n = 5, m = 2$

(d) $K = 2, n = 11, m = 2$

(e) $K = 2, n = 5, m = 1$

(f) $K = 2, n = 11, m = 1$

(g) $K = 2, n = 5, m = 0.5$

(h) $K = 2, n = 11, m = 0.5$
Figure 6: Box-plot comparisons of the TA and DPSO algorithms on dimension $K = 2, 3$ with different point number $n$ and flexible parameter $m$. 

(a) $K = 2, n = 5, m = 0.3$
(b) $K = 2, n = 11, m = 0.3$
(c) $K = 3, n = 5, m = 9999$
(d) $K = 3, n = 11, m = 9999$
(e) $K = 3, n = 5, m = 2$
(f) $K = 3, n = 11, m = 2$
(g) $K = 3, n = 5, m = 1$
(h) $K = 3, n = 11, m = 1$
Figure 7: Comparison of the TA and DPSO algorithms on dimension $K = 2, 3$ with different point number $n$ and flexible parameter $m$. 

(a) $K = 3, n = 5, m = 0.5$
(b) $K = 3, n = 11, m = 0.5$
(c) $K = 3, n = 5, m = 0.3$
(d) $K = 3, n = 11, m = 0.3$
Table 2: The $\overline{\text{CCD}}$ values (scaled by multiplying $10^3$) found by DPSO with function evaluation $I = 10^5, 10^6$, $K = 4$, various point numbers $n$, and flexible parameter $m$. Average timing results are shown in seconds.

(a) $I = 10^5$

<table>
<thead>
<tr>
<th>$n$</th>
<th>Best</th>
<th>Median</th>
<th>Std</th>
<th>Time$^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = 2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>66.738</td>
<td>66.955</td>
<td>0.201</td>
<td>56.3</td>
</tr>
<tr>
<td>11</td>
<td>38.323</td>
<td>38.950</td>
<td>0.285</td>
<td>58.8</td>
</tr>
<tr>
<td></td>
<td>$m = 1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>66.425</td>
<td>66.653</td>
<td>0.258</td>
<td>58.3</td>
</tr>
<tr>
<td>11</td>
<td>37.111</td>
<td>37.897</td>
<td>0.451</td>
<td>71.8</td>
</tr>
<tr>
<td></td>
<td>$m = 0.5$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>65.520</td>
<td>65.718</td>
<td>0.284</td>
<td>59.5</td>
</tr>
<tr>
<td>11</td>
<td>37.014</td>
<td>37.257</td>
<td>0.232</td>
<td>74.3</td>
</tr>
</tbody>
</table>

(b) $I = 10^6$

<table>
<thead>
<tr>
<th>$n$</th>
<th>Best</th>
<th>Median</th>
<th>Std</th>
<th>Time$^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = 2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>66.738</td>
<td>66.738</td>
<td>0.175</td>
<td>634.9</td>
</tr>
<tr>
<td>11</td>
<td>37.478</td>
<td>37.758</td>
<td>0.319</td>
<td>686.3</td>
</tr>
<tr>
<td></td>
<td>$m = 1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>66.343</td>
<td>66.494</td>
<td>0.159</td>
<td>580.8</td>
</tr>
<tr>
<td>11</td>
<td>36.930</td>
<td>37.283</td>
<td>0.196</td>
<td>719.0</td>
</tr>
<tr>
<td></td>
<td>$m = 0.5$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>65.305</td>
<td>65.640</td>
<td>0.216</td>
<td>592.7</td>
</tr>
<tr>
<td>11</td>
<td>36.444</td>
<td>36.827</td>
<td>0.259</td>
<td>735.3</td>
</tr>
</tbody>
</table>

Table 3: $\overline{\text{CCD}}$ values (scaled by multiplying $10^3$) for optimized designs via the DPSO algorithm with function evaluation $I = 10^5, 10^6$, a number of factors $K = 5$, different point number $n$, and flexible parameter $m$. Average timing results are shown in seconds.

(a) $I = 10^5$

<table>
<thead>
<tr>
<th>$n$</th>
<th>Best</th>
<th>Median</th>
<th>Std</th>
<th>Time$^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = 2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>55.598</td>
<td>55.843</td>
<td>0.139</td>
<td>362.9</td>
</tr>
<tr>
<td>11</td>
<td>32.990</td>
<td>33.533</td>
<td>0.345</td>
<td>550.3</td>
</tr>
<tr>
<td></td>
<td>$m = 1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>55.806</td>
<td>55.925</td>
<td>0.227</td>
<td>342.3</td>
</tr>
<tr>
<td>11</td>
<td>33.081</td>
<td>33.234</td>
<td>0.265</td>
<td>488.8</td>
</tr>
<tr>
<td></td>
<td>$m = 0.5$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>55.318</td>
<td>55.503</td>
<td>0.167</td>
<td>397.5</td>
</tr>
<tr>
<td>11</td>
<td>32.673</td>
<td>33.107</td>
<td>0.345</td>
<td>599.6</td>
</tr>
</tbody>
</table>

(b) $I = 10^6$

<table>
<thead>
<tr>
<th>$n$</th>
<th>Best</th>
<th>Median</th>
<th>Std</th>
<th>Time$^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = 2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>55.727</td>
<td>55.760</td>
<td>0.080</td>
<td>361.3</td>
</tr>
<tr>
<td>11</td>
<td>32.552</td>
<td>32.811</td>
<td>0.133</td>
<td>538.7</td>
</tr>
<tr>
<td></td>
<td>$m = 1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>55.699</td>
<td>55.838</td>
<td>0.137</td>
<td>336.8</td>
</tr>
<tr>
<td>11</td>
<td>32.100</td>
<td>32.512</td>
<td>0.226</td>
<td>654.1</td>
</tr>
<tr>
<td></td>
<td>$m = 0.5$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>55.318</td>
<td>55.437</td>
<td>0.133</td>
<td>388.9</td>
</tr>
<tr>
<td>11</td>
<td>31.957</td>
<td>32.302</td>
<td>0.213</td>
<td>758.0</td>
</tr>
</tbody>
</table>

as follows. In each case, the median of the $\overline{\text{CCD}}$ value obtained by TA is used as a principle value. Then, the box-plots are plotted based on the ratios of the $\overline{\text{CCD}}$ values and the median $\overline{\text{CCD}}$ values by TA. These figures suggest the following observations:

- Under the same number of function evaluations, DPSO (DPSO+) outperforms TA (TA+) in terms of the minimal values, the range, and the inter-quartile range of the $\overline{\text{CCD}}$ values.
- DPSO (with $10^5$ function evaluations) performs better than TA+ (with $10^6$ function evaluations) in most cases.
- DPSO and DPSO+ usually obtain smaller $\overline{\text{CCD}}$ values. In some small case problems, such as $K = 2, n = 5$, almost all of the 100 repetitions in DPSO converge to the smallest value we can ever find.
• In most cases, the DPSO algorithm leads to smaller ranges and inter-quartile ranges than those of the TA algorithms. In other words, DPSO is more stable among the 100 replications.

• For the cases that $K = 3$, $n = 11$, and $m = 9999, 2, 1$, the performance differences between the two algorithms decrease. This could be because the sizes of these problems are too large. Consequently, both DPSO and TA algorithms search only a small portion of the solution space.

4.3. Numerical results for $K = 4$ and $5$

In addition to the cases of $K = 2$ and $3$, we also illustrate the results for $K = 4, 5$ obtained by DPSO in Tables 2 and 3. The results suggest that DPSO is a stable method capable of solving the large problems efficiently. Using more iteration numbers, i.e., $I = 10^6$, leads to better results in terms of the minimal values and the standard deviation of the $CCD$ values, especially for $n = 11$. However, computational times for $I = 10^6$ are usually 10 times more than the cost for $I = 10^5$.

5. A Real Application with Slid-Rectangular Region

The proposed method is next applied to a data center thermal management problem introduced by Hung et al. (2010); Hung (2011). A data center is an integrated facility housing multiple-unit servers, providing application services or management for data processing. The objective
of this portion of the study is to obtain an optimal sensor placement plan for monitoring the thermal distribution in the data center, which is a crucial step in designing a data center with an efficient heat-removal mechanism. The sensors should be attached to facility surfaces to measure temperature in a data center, but the facilities are often stored irregularly due to usage limitations. Therefore, this study leads to an experimental design problem for an irregular region shown in Figure 8. Hung et al. (2011) defined such an irregular region as slid-rectangular.

In this example, we plan to select \( n = 24 \) positions among 106 feasible ones over the slid-rectangular region to place sensors. This example is intended to demonstrate the performance of DPSO in extending uniform design searches to disconnected regions. According to the definition of discrete CCD shown in Definition 4, the CCD criterion works whether the experimental region is connected or not. By using DPSO with the discrete CCD as the selection criterion, we can generate a uniform design over the slid-rectangular region as illustrated in Figure 9(a). The corresponding \( \text{CCD}^2 \) value is 0.0141439.

By the definition of the uniform design (1), the non-collapsing property might not hold. Based on the optimal uniform design shown in Figure 9(a), projection of the design points onto the \( x \)-axis leads to some replicates at \( x = 5, 7, 9, 14 \), which violates the non-collapsing property. To further ensure this non-collapsing property, we suggest modifying \( \text{CCD} \) by adding a penalty term for replicates. For this example where replicates occur on the \( x \)-axis, we modify the CCD criterion by

\[
\widehat{\text{CCD}}_2(P) = \text{CCD}_2(P) \times \left( \sqrt{\text{number of collapsing on } x\text{-axis in } P + 1} \right)
\]

The DPSO algorithm remains applicable to optimize \( \widehat{\text{CCD}}_2 \). Generated by DPSO, the optimal
design is illustrated in Figure 9(b). The corresponding $\overline{\text{CCD}}_2$ value is 0.0346129 (i.e., transformation from $\overline{\text{CCD}}_2$ to $\overline{\text{CCD}}_2$).

For the same example, we note that the probability-based Latin Hypercubic design (PLHD) reported in Hung (2011) has a $\overline{\text{CCD}}_2$ value 0.0384592, which is more than 10% higher than the optimal $\overline{\text{CCD}}_2$ design we found by using DPSO. This result indicates that the DPSO algorithm can obtain designs with better uniformity than PLHDs in terms of $\overline{\text{CCD}}_2$ measurement.

6. Conclusion

We have studied how optimal CCD-based uniform designs for irregular experimental regions can be determined efficiently. In the theoretical aspect, we have conducted the complexity analysis for the discrete CCD. In the computational aspect, we have developed the DPSO to find optimal designs with respect to the discrete CCD criterion. Our implementation of DPSO is also accelerated by using the state-of-the-art GPU parallelism. Numerical results on two- and three-dimensional flexible regions show that the DPSO algorithm outperforms the TA algorithm in both efficiency and stability. Efficiency of DPSO is analyzed numerically for four- and five-dimensional flexible regions. By solving a real world application, we further demonstrate how DPSO can be extended to handle disconnected irregular experimental regions and avoid collapse.

Several topics warrant further investigation. To generate non-collapsing uniform design, we have suggested modifying the uniformity measure by adding a penalty for collapsing designs. One example is shown in Section 5. However, designing this penalty term may be related to the experimental regions and is an interesting research problem. Another possible point of future research is to extend the DPSO algorithm to generate a nested spacing-filling design (Qian et al., 2009). Rennen et al. (2010) have proposed a numerical method for constructing the nested spacing-filling design based on a specific optimal criterion. It would be interesting to see how DPSO works for such nested design problems, which can be formulated as a constrained optimization problem. Finally, it is possible to further improve the computational efficiency by using multiple GPU. Some techniques can be found in Hung and Wang (2012).

Acknowledgments

The research of Hung is supported by National Science Foundation grants DMS 0905753 and CMMI 0927572. This specific work was supported in part by the National Science Council under grant NSC 99-2118-M-006-006-MY2 (Chen) and NSC 100-2628-M-002-011-MY4 (Wang), the Taïda Institute of Mathematical Sciences, the Center for Advanced Study in Theoretical Sciences, the National Center for Theoretical Sciences (Taipei Office) and the Mathematics Division of the National Center for Theoretical Sciences (South) in Taiwan.


**Appendix A. Appendix**

**Appendix A.1. Computational details of \( \overline{CCD}_p \)**

We illustrate the calculation detail of \( \overline{CCD}_2(P) \) in the following example. The example design is \( P = \{p_1 = (1/6, 1/2), p_2 = (1/2, 1/2)\} \) contains two design points \( (n = 2) \) on domain \( \overline{D}(3) \) (see Definition 3). Here, \( D = \{x = (x_1, x_2) | |2(x_1 - 0.5)| + |2(x_2 - 0.5)| \leq 1\} \). As \( K = 2 \) and \( m = 2 \), the inner summation index is \( k = 1, 2, 3, 4 \). For every point \( x \in \overline{D} \), we divide the region \( D \) into four sub-regions: \( D_1(x) = \{r = (r_1, r_2) \in D | r_1 \leq x_1, r_2 \leq x_2\} \), \( D_2(x) = \{r = (r_1, r_2) \in D | r_1 > x_1, r_2 \leq x_2\} \), \( D_3(x) = \{r = (r_1, r_2) \in D | r_1 \leq x_1, r_2 > x_2\} \), \( D_4(x) = \{r = (r_1, r_2) \in D | r_1 > x_1, r_2 > x_2\} \).

In this example, \( \overline{D}(3) = \{x_1 = (1/6, 1/2), x_2 = (1/2, 5/6), x_3 = (1/2, 1/2), x_4 = (1/2, 1/6), x_5 = (5/6, 1/2)\} \) which contains five grid points. That is, \( |\overline{D}| = 5 \).

From Eq. (4), we can see that we must compute \( N(\overline{D}_k(x), P), |\overline{D}_k(x)| \) at these five grid points. For example, at \( x_1 \), as shown in Figure 10(b), \( N_1(x_1) = 0, N_2(x_1) = 0, N_3(x_1) = 1, N_4(x_1) = 0, |\overline{D}_1(x_1)| = 1, |\overline{D}_2(x_1)| = 0, |\overline{D}_3(x_1)| = 3, \) and \( |\overline{D}_4(x_1)| = 1 \). At \( x_2 \), as shown in Figure 10(c),
Figure A.10: Examples for calculating approximate CCD.

Figure A.11: A trivial example: 1-point design \( P = \{(1/2, 1/2)\} \) on domain \( D = [0, 1]^2 \).

Figure A.11: A trivial example: 1-point design \( P = \{(1/2, 1/2)\} \) on domain \( D = [0, 1]^2 \).
\[ N_1(x_2) = 1, N_2(x_2) = 1, N_3(x_2) = 0, N_4(x_2) = 0, |\bar{D}_1(x_2)| = 1, |\bar{D}_2(x_2)| = 3, |\bar{D}_3(x_2)| = 0, \text{ and } |\bar{D}_4(x_2)| = 1. \] In short, we have

\[
\text{CCD}_2(P)^2 = \frac{1}{5} \left[ \frac{1}{4} \left( \frac{0}{2} - \frac{1}{5} \right)^2 + \frac{0}{2} - \frac{0}{7} \right] + \frac{1}{4} \left( \frac{0}{2} - \frac{1}{5} \right)^2 + \frac{0}{2} - \frac{0}{7} \right] + \frac{1}{4} \left( \frac{0}{2} - \frac{1}{5} \right)^2 + \frac{0}{2} - \frac{0}{7} \right] + \frac{1}{4} \left( \frac{0}{2} - \frac{1}{5} \right)^2 + \frac{0}{2} - \frac{0}{7} \right] + \frac{1}{4} \left( \frac{0}{2} - \frac{1}{5} \right)^2 + \frac{0}{2} - \frac{0}{7} \right] \]

and therefore \( \text{CCD}_2(P) = 0.197484176581315 \)

In addition, we compute \( \text{CCD}_2 \) and \( \text{CCD}_2 \) for different grid levels \( q \) of a trivial example shown in Figure A.11. In this example, we have the explicit form of the CCD integral. Therefore, we can compute the exact value of \( \text{CCD}_2(P)^2 \).

\[
\text{CCD}_2(P)^2 = \int_D \frac{1}{4} \sum_{k=1}^{4} \left| \frac{N(x)}{1} - \frac{v(x)}{1} \right|^2 dA \text{ (by symmetry)}
\]

\[
= \int_{0}^{1/2} \int_{0}^{1/2} \frac{1}{4} \left( (x^2 - x)^2 + [(1-x)^2 + [x(1-y)^2 + [1-(1-x)(1-y)]^2] \right) dA
\]

\[
= \int_{0}^{1/2} \int_{0}^{1/2} \frac{3}{6} \left( (x^2 - x)^2 + \frac{3}{6} x^2 (1-y)^2 \right)
\]

\[
= \int_{0}^{1/2} \frac{1}{24} y^3 + \frac{7}{72} \left( 1-y \right)^3 + \left[ \frac{1}{2} - \frac{3}{4} (1-y) + \frac{7}{24} (1-y)^2 \right] dy
\]

\[
= \frac{23}{288}
\]

That is, \( \text{CCD}_2(P) = (23/288)^{1/2} \approx 0.282597083 \). We also compute \( \text{CCD}_2(P) \) for grid level = 11, 51, and 101 and obtain the following results. For \( q = 11 \), \( \text{CCD}_2(P) \approx 0.2814774653 \). For \( q = 51 \), \( \text{CCD}_2(P) \approx 0.2825451145 \). For \( q = 101 \), \( \text{CCD}_2(P) \approx 0.2825838355 \).