An Algorithm to Compute Bounds for the Star Discrepancy

Eric Thiémard

Département de Mathématiques, Ecole Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland
E-mail: eric.thiemard@epfl.ch

Received November 3, 1999; accepted June 15, 2000; published online September 20, 2001

We propose an algorithm to compute upper and lower bounds for the star discrepancy of an arbitrary sequence of points in the \( s \)-dimensional unit cube. The method is based on a particular partition of the unit cube into subintervals and on a specialized procedure for orthogonal range counting. The cardinality of the partition depends on the dimension and on an accuracy parameter that has to be specified. We have implemented this method and here we present results of some computational experiments obtained with this implementation.

© 2001 Elsevier Science

1. INTRODUCTION

The star discrepancy is a measure for the irregularity of point sequences that is mainly used in quasi-Monte Carlo methods (see [15] for an overview). For a sequence of \( n \) points \( x = \{x_0, ..., x_{n-1}\} \) in the \( s \)-dimensional unit cube \( I^s = [0,1]^s \), \( s \geq 1 \), the star discrepancy \( D^*_s(x) \) is defined by

\[
D^*_s(x) = \sup_{P \in \mathcal{F}^*} \left| \frac{A(P, x)}{n} - \lambda(P) \right|,
\]

where \( \mathcal{F}^* \) is the family of all subintervals of \( I^s = [0,1]^s \) of the form \( \prod_{i=1}^s [0,p_i) \), \( \lambda(P) \) is the volume of a subinterval \( P \), and \( A(P, x) \) is the

850
number of points of $x$ belonging to $P$. Clearly we have $0 < D^*_s(x) \leq 1$. These notations are standard, except that we use $P$ for a subinterval, whereas many authors use it for point sets.

For $s \geq 2$, the computation of the star discrepancy is not an easy task. A discretization of (1) has been proposed by Niederreiter [13], but it still requires the evaluation of $O(n^s)$ terms, making this approach impractical in most cases. We also mention the existence of an $O(n^{1+s/2})$ time algorithm proposed by Dobkin and Eppstein [4]. We refer to [21] for computational experiments and a more detailed discussion.

The intractability of the computation of the star discrepancy is partially balanced by the existence of upper bounds for some particularly regular point sets called $(t, s)$-sequences and $(t, m, s)$-nets (see [18], [5], and [14]). However, the minimum sample size $n$ for which these upper bounds become meaningful grows exponentially with the dimension $s$ (see [12] and [21]).

Winker and Fang [23] proposed an algorithm inspired by simulated annealing to compute lower bounds for the star discrepancy. In [21], we proposed an algorithm to compute upper bounds for the star discrepancy of an arbitrary sequence in a multidimensional unit cube. This method uses considerable time and memory, but in many cases, provides better bounds than the classical ones stated in [14].

We carry on with this approach in the present paper and propose an improved algorithm to compute upper and lower bounds for the star discrepancy. We also give results of some numerical experiments and give bounds for the star discrepancy of some particular sequences. The method described in this paper has been implemented. The C code is available at http://rosowww.epfl.ch/papers/discrbound2/.

2. THEORETICAL FOUNDATION

With every subinterval of the form $P = \prod_{i=1}^{s} [a_i, b_i)$, we associate the pair of subintervals $P^- = \prod_{i=1}^{s} [0, a_i)$ and $P^+ = \prod_{i=1}^{s} [0, b_i)$. For any finite partition $\mathcal{P}$ of $I'$ into a family of subintervals of the form $\prod_{i=1}^{s} [a_i, b_i)$, Theorem 2.1 gives an upper bound for the star discrepancy in terms of $B(\mathcal{P}, x)$, where

$$B(\mathcal{P}, x) = \max_{P \in \mathcal{P}} \max \left\{ \frac{A(P^+, x)}{n} - \lambda(P^+), \frac{\lambda(P^-) - A(P^+, x)}{n} \right\}.$$
Theorem 2.1. For any finite partition $\mathcal{P}$ of $I^*$ into subintervals of the form $\prod_{i=1}^{s} [\alpha_i, \beta_i]$ and any sequence $x = \{x_0, \ldots, x_{n-1}\}$ in $I^*$, we have

$$D_n^*(x) \leq B(\mathcal{P}, x).$$

Proof.

$$D_n^*(x) = \sup_{P \in \mathcal{P}} \left| \frac{A(P, x)}{n} - \lambda(P) \right|$$

$$= \max_{P \in \mathcal{P}} \sup_{Q \subset P} \left\{ \left| \frac{A(Q, x)}{n} - \lambda(Q) \right| \right\}$$

$$\leq \max_{P \in \mathcal{P}} \left\{ \left| \frac{A(P^+, x)}{n} - \lambda(P^+) \right|, \left| \frac{A(P^-, x)}{n} - \lambda(P^-) \right| \right\}.$$ 

Let us define the weight $W(P)$ of a subinterval $P = \prod_{i=1}^{s} [\alpha_i, \beta_i] \in \mathcal{P}$ as the difference in volumes of $P^+ = \prod_{i=1}^{s} [0, \beta_i]$ and $P^- = \prod_{i=1}^{s} [0, \alpha_i]$:

$$W(P) = \lambda(P^+) - \lambda(P^-) = \prod_{i=1}^{s} \beta_i - \prod_{i=1}^{s} \alpha_i.$$

For any finite partition $\mathcal{P}$ of $I^*$ into subintervals, the following quantity is obviously a lower bound for the star discrepancy:

$$C(\mathcal{P}, x) = \max_{P \in \mathcal{P}} \left\{ \left| \frac{A(P^-, x)}{n} - \lambda(P^-) \right|, \left| \frac{A(P^+, x)}{n} - \lambda(P^+) \right| \right\}.$$ 

So we have the following lower and upper bounds:

$$C(\mathcal{P}, x) \leq D_n^*(x) \leq B(\mathcal{P}, x).$$

Of course, one would like this interval that bounds the star discrepancy to be small. The following result shows that it cannot be larger than $W(\mathcal{P})$, where

$$W(\mathcal{P}) = \max_{P \in \mathcal{P}} W(P).$$
Theorem 2.2. For any finite partition $\mathcal{P}$ of $I^*$ into subintervals of the form $\prod_{i=1}^{s} [\alpha_i, \beta_i]$ and any sequence $x = \{x_0, \ldots, x_{n-1}\}$ in $I^*$, we have

$$B(\mathcal{P}, x) - C(\mathcal{P}, x) \leq W(\mathcal{P}).$$

Proof. First we observe that for each $P \in \mathcal{P}$, we have

$$\frac{A(P^+, x)}{n} - \frac{\lambda(P^-)}{n} = \frac{A(P^+, x)}{n} - \frac{\lambda(P^+)}{n} + [\lambda(P^+) - \lambda(P^-)]$$

and

$$\frac{\lambda(P^-) - A(P^-, x)}{n} = [\lambda(P^+) - \lambda(P^-)] + \lambda(P^-) - \frac{A(P^-, x)}{n}$$

Thus, we obtain

$$B(\mathcal{P}, x) = \max_{P \in \mathcal{P}} \left\{ \frac{A(P^+, x)}{n} - \frac{\lambda(P^-)}{n}, \frac{\lambda(P^+) - A(P^-, x)}{n} \right\} + W(\mathcal{P})$$

$$\leq \max_{P \in \mathcal{P}} \left\{ \frac{A(P^+, x)}{n} - \frac{\lambda(P^-)}{n}, \frac{\lambda(P^+) - A(P^-, x)}{n} \right\} + W(\mathcal{P})$$

Since $W(\mathcal{P})$ is independent of the point set, one may choose an $\varepsilon \in (0, 1)$ and construct a partition $\mathcal{P}$ with $W(\mathcal{P}) = \varepsilon$ to guarantee an interval of width at most $\varepsilon$ containing the true discrepancy. Such a construction is proposed in Section 3.

Conversely, among partitions with a given cardinality, the best choice is the one that minimizes $W(\mathcal{P})$. In [21], we restricted the partition choice to grids and we tried to find, for a fixed integer $k \geq 2$, an optimal grid of cardinality $k^s$ by solving the following mathematical program:

$$M_k^* = \min_{a_1, \ldots, a_k \in \{1, \ldots, h\}^s} \left( \prod_{j=1}^{s} a_{n_j} - \prod_{j=1}^{s} a_{n-1} \right)$$

s.t. $0 = a_0 \leq \cdots \leq a_k = 1$
Basically, the larger $k$, the smaller $M^*_s$ and the better the upper bound $B(\mathcal{P}, x)$, but the more time and memory required for the computation. The problem with the restriction to grids is that it implies substantial computational effort. It appears that in an optimal solution of (2), among the $k^s$ subintervals, only very few have a weight close to $M^*_s$. For example, in dimension $s = 8$ with $k = 10$, less than 1% of the $10^8$ subintervals have a weight equal to $M^*_s \approx 0.2023$. On the other hand, with the new decomposition algorithm described in the present paper, $I^s$ can be partitioned into less than $4 \cdot 10^6$ subintervals of weight less than or equal to 0.2023 (among which 63% have a weight equal to 0.2023).

3. PARTITIONING $I^s$

For a sequence $x$ of $n$ points in dimension $s \geq 2$, we will choose an accuracy parameter $\epsilon \in (0, 1)$ and build a partition $\mathcal{P}$ of $I^s$ into subintervals such that

$$W(\mathcal{P}) = \epsilon. \tag{3}$$

Then, the corresponding interval $[C(\mathcal{P}, x), B(\mathcal{P}, x)]$ is of width at most $\epsilon$ and contains the value of the star discrepancy $D^*_s(x)$. From this point of view, the smaller $\epsilon$, the tighter the bounds. However, it is also clear that small values of $\epsilon$ imply large partitions and heavy computational costs. In practice, a compromise solution needs to be found.

We also suspect (though we could not prove it) that $B(\mathcal{P}, x) \geq W(\mathcal{P})$. This point has been observed in each experiment that we carried out. Then, if one suspects $D^*_s(x)$ to be close to some value $v$, it is a good idea to choose $\epsilon \leq v$.

We could not find a partition algorithm for which constraint (3) holds and cardinality $|\mathcal{P}|$ is minimal. However, in addition to generating a partition for which constraint (3) holds, our method has the following properties:

- $|\mathcal{P}|$ is finite;
- $W(P) = \epsilon$ for a large proportion of the subintervals $P \in \mathcal{P}$ (which might hopefully indicate that $|\mathcal{P}|$ is nearly minimal);
- the time required to generate the partition is optimal (relative to its cardinality), namely $O(s |\mathcal{P}|)$;
- the very nature of the partition $\mathcal{P}$ facilitates the computation of $B(\mathcal{P}, x)$ in the second phase of the algorithm. Specifically, the computational cost of counting $A(P^-, x)$ and $A(P^+, x)$ for each $P \in \mathcal{P}$ is sublinear in $n$. 

3.1. Decomposition Principle

Let \([\alpha, \beta] = \prod_{i=1}^{r} [\alpha_i, \beta_i] \subset I^r\) be a subinterval given by \(\alpha < \beta \in I^r\) where \(\alpha = (\alpha_1, \ldots, \alpha_r)\) and \(\beta = (\beta_1, \ldots, \beta_r)\). Then, for any \(d \in \{1, \ldots, s\}\) and any \(\gamma_d \in [\alpha_d, \beta_d]\), subinterval \([\alpha, \beta]\) can be partitioned into a pair of subintervals \([\alpha', \beta']\) and \([\alpha'', \beta'']\) where \(\alpha'\) and \(\beta'\) are defined by

\[
\alpha'_i = \begin{cases} 
\alpha_i & \forall i \neq d \\
\gamma_d & \text{if } i = d 
\end{cases} \quad \beta'_i = \begin{cases} 
\beta_i & \forall i \neq d \\
\gamma_d & \text{if } i = d 
\end{cases}
\]

In the following, this partition step will be called a decomposition of parameter \(\gamma_d\) in direction \(d\) (see Fig. 1). We will see that, starting with \(I^r\), such partition steps in different directions can be recursively applied until we obtain a partition that contains only subintervals of weight at most \(\varepsilon\).

We now define a partition step which combines at most \(s\) decompositions (in different directions) and partitions a subinterval \([\alpha, \beta] \subset I^r\) into at most \(s + 1\) subintervals. Given any \(\gamma \in [\alpha, \beta]\), we successively apply for each direction \(d = 1, \ldots, s\) a decomposition of parameter \(\gamma_d\) in direction \(d\) to one of the subintervals (the one with the preserved \(\beta\) “upper-corner”) resulting from the previous decomposition.

\[
\begin{align*}
P_0 & \leftarrow Q_1 \leftarrow P_1 \leftarrow Q_2 \leftarrow P_2 \leftarrow Q_3 \leftarrow P_3 \leftarrow \cdots \leftarrow Q_{s-1} \leftarrow P_{s-1} \leftarrow Q_s \leftarrow P_s
\end{align*}
\]
As illustrated in Fig. 2, starting with \( P = P_0 = [\alpha, \beta] \), for \( d = 1, \ldots, s \) we apply a decomposition of parameter \( \gamma_d \) in direction \( d \) to \( P_{d-1} \) and obtain two subintervals \( Q_d \) and \( P_d \) with

\[
P_{d-1} = Q_d \cup P_d \quad \text{for} \quad d = 1, \ldots, s.
\]

More explicitly, we have

\[
\begin{align*}
P_0 &= [(\alpha_1, \alpha_2, \ldots, \alpha_s), \beta), \\
Q_1 &= [(\alpha_1, \alpha_2, \ldots, \alpha_s), (\gamma_1, \beta_2, \ldots, \beta_s)), \\
P_1 &= [(\gamma_1, \alpha_2, \ldots, \alpha_s), \beta), \\
Q_2 &= [(\gamma_1, \alpha_2, \ldots, \alpha_s), (\beta_1, \gamma_2, \beta_3, \ldots, \beta_s)), \\
P_2 &= [(\gamma_1, \gamma_2, \alpha_3, \ldots, \alpha_s), \beta), \\
& \vdots \\
Q_s &= [(\gamma_1, \ldots, \gamma_{s-1}, \alpha_s), (\beta_1, \ldots, \beta_{s-1}, \gamma_s)), \\
P_s &= [(\gamma_1, \gamma_2, \ldots, \gamma_s), \beta) = [\gamma, \beta)
\end{align*}
\]

The resulting combined partition step will be called a decomposition of parameter \( \gamma \). Finally we can write

\[
[\alpha, \beta) = \left( \bigcup_{d=1}^s Q_d \right) \cup [\gamma, \beta).
\]

Depending on the choice of parameter \( \gamma \in [\alpha, \beta) \), subinterval \( [\alpha, \beta) \) can be partitioned into less than \( s+1 \) subintervals. A decomposition of parameter \( \gamma_d \) in direction \( d \) leads to an empty subinterval \( Q_d \) if \( \gamma_d = \alpha_d \):

\[
\gamma_d = \alpha_d \iff Q_d = \emptyset \iff P_d = P_{d-1}.
\]

In the following, we will need some formal notation for the above decomposition process. Given a subinterval \( P = [\alpha', \beta'] \) of \( I' \) and a point \( \gamma^P \in [\alpha', \beta'] \), the decomposition of \( P \) of parameter \( \gamma^P \) will be noted

\[
P = \left( \bigcup_{d=1}^t Q_d^P \right) \cup [\gamma^P, \beta^P),
\]
where \( Q^*_d = [\alpha^* Q^*, \beta^* Q^*] \) with

\[
\alpha^* Q^* = \begin{cases} 
\gamma_i^* & \text{for } i < d \\
\alpha_i^* & \text{for } i \geq d
\end{cases}
\]

\[
\beta^* Q^* = \begin{cases} 
\gamma_i^* & \text{for } i = d \\
\beta_i^* & \text{for } i \neq d.
\end{cases}
\]

We observe that

\[
a_i^* \leq \alpha^* Q^* \leq \gamma_i^* \leq \beta^* Q^* \leq b_i^* \quad \forall i, d \in \{1, ..., s\}.
\]

Of course, the decomposition principle proposed in this section leads to a very particular way to partition \( I' \). We tried various other approaches, but none had the nice structure discussed below and none led to partitions of substantially lower cardinality.

### 3.2. Choice of Parameter \( \gamma^p \)

Given a subinterval \( P = [\alpha^p, \beta^p] \subset I' \) with \( W(P) > \varepsilon \), there are many ways to choose \( \gamma^p \) in order to decompose \( P \) into subintervals of smaller weight (ultimately smaller than \( \varepsilon \)). Most of these strategies lead to infinite cardinality partitions, heavy computational cost or partitions with a structure that does not facilitate the evaluation of \( A(P^-, x) \) and \( A(P^+, x) \) in the computation of the upper bound \( B(P, x) \).

For example, it is possible to fix \( \gamma_1^p, ..., \gamma_s^p \) successively in order to build subintervals \( Q_1^p, ..., Q_s^p \) of weight \( \varepsilon \) (or empty subintervals if we reach the end of the process) and to start again with a decomposition of the remaining subinterval \( [\gamma^p, \beta^p] \) if \( W([\gamma^p, \beta^p]) > \varepsilon \). Unfortunately, when applied to the partition of \( I' \), this promising strategy can lead to infinite cardinality partitions (for example with \( s = 2 \) and \( \varepsilon = 0.15 \)).

It seems that a better strategy is to choose \( \gamma^p \) such that \( W([\gamma^p, \beta^p]) = \varepsilon \), but in a way which allows decomposition of the resulting nonempty subintervals \( Q_i^p \) of weight greater than \( \varepsilon \) into fewer and fewer subintervals in the course of their recursive decomposition. As will be shown later, this can be done efficiently with the following choice of \( \gamma^p \).

Given a subinterval \( P = [\alpha^p, \beta^p] \subset I' \) with \( W(P) > \varepsilon \), we define the decomposition parameter \( \gamma^p = \gamma^p(\delta^p) \in [\alpha^p, \beta^p] \) through \( s \) functions

\[
\gamma_i^p(\delta) = \begin{cases} 
\alpha_i^p & \text{if } \delta \beta_i^p \leq \alpha_i^p \\
\delta \beta_i^p & \text{otherwise}
\end{cases} \quad i \in \{1, ..., s\}
\]

and a factor \( \delta^p \in ]0, 1[ \) which is the unique solution of equation

\[
W([\gamma^p(\delta^p), \beta^p]) = \varepsilon.
\]
Intuitively, \( \gamma^p \) can be seen as a linear transformation of \( \beta^p \) that is, when necessary, truncated to \( \alpha^p \) in some components. For the directions that correspond to these truncations we have

\[
\delta^p \beta^p_d \leq \alpha^p_d \iff \gamma^p_d = \alpha^p_d \iff Q^p_d = \emptyset.
\]

We can now state the first version of our partition algorithm of \( I^p \):

**Algorithm 1** (*Input: s, e*  *Output: \( \mathcal{P}_s^e \)).

**Main**

\[ \mathcal{P}_s^e := \emptyset \]

\[ \text{decompose}(I^p) \]

**Procedure** decompose(\( P \))

compute \( \delta^p \) and \( \gamma^p \)

For \( d = 1, \ldots, s \)

If \( \gamma^p_d \neq \alpha^p_d \)

If \( W(Q^p_d) > e \)

\[ \text{decompose}(Q^p_d) \]

Else

\[ \mathcal{P}_s^e := \mathcal{P}_s^e \cup Q^p_d \]

\[ \mathcal{P}_s^e := \mathcal{P}_s^e \cup [\gamma^p, \beta^p] \]

This algorithm will be improved in the following section, but its basic structure will remain the same. We will also explain later how \( \delta^p \) and \( \gamma^p \) can be efficiently computed.

As an example, Table I and Fig. 3 illustrate the partition returned by the above algorithm for \( s = 3 \) and \( e = 0.6 \). We observe in Table I that both series (of \( \alpha \) and \( \beta \)) are generated in lexicographic order.

**TABLE I**

Decomposition of \( I^p \) for \( e = 0.6 \)

\[
\begin{align*}
Q_{1}^{0^p} &= [(0.000000, 0.000000, 0.000000), (0.420343, 1.000000, 1.000000)) \\
Q_{2}^{0^p} &= [(0.420343, 0.000000, 0.000000), (0.736806, 0.570494, 1.000000))] \\
Q_{3}^{0^p} &= [(0.420343, 0.570494, 0.000000), (0.736806, 1.000000, 0.570494)) \\
[\gamma^{0^p}, \beta^{0^p}] &= [(0.420343, 0.570494, 0.570494), (0.736806, 1.000000, 1.000000)) \\
Q_{1}^{1^p} &= [(0.736806, 0.000000, 0.000000), (1.000000, 0.369873, 1.000000))] \\
Q_{2}^{1^p} &= [(0.736806, 0.369873, 0.000000), (1.000000, 0.736806, 0.501995))] \\
Q_{3}^{1^p} &= [(0.736806, 0.369873, 0.501995), (1.000000, 0.736806, 1.000000))] \\
[\gamma^{1^p}, \beta^{1^p}] &= [(0.736806, 0.736806, 0.000000), (1.000000, 1.000000, 0.251999)) \\
Q_{1}^{2^p} &= [(0.736806, 0.736806, 0.000000), (1.000000, 1.000000, 0.736806))] \\
Q_{2}^{2^p} &= [(0.736806, 0.736806, 0.251999), (1.000000, 1.000000, 0.736806)) \\
Q_{3}^{2^p} &= [(0.736806, 0.736806, 0.736806), (1.000000, 1.000000, 1.000000))] \\
[\gamma^{2^p}, \beta^{2^p}] &= [(0.736806, 0.736806, 0.736806), (1.000000, 1.000000, 1.000000))
\end{align*}
\]
3.3. Structure of Partition $\mathcal{P}_s$

In this section we will actually show that both series \( \{x^P : P \in \mathcal{P}_s\} \) and \( \{y^P : P \in \mathcal{P}_s\} \) are generated by our algorithm in lexicographic order. This attractive property will be intensively exploited to speed up the computation of the upper bound $B(\mathcal{P}_s, x)$. Then, we will show how to compute decomposition parameters $\delta^P$ and $\gamma^P$ associated with a subinterval $P$ and we will formulate an efficient version of our partition algorithm.

**Lemma 3.1.** Let $P = [x^P, y^P) \subset I^r$ be a subinterval with weight $W(P) > \varepsilon$. Then, for any $j \in \{1, \ldots, s\}$ such that $Q_j^P \neq \emptyset$ and $W(Q_j^P) > \varepsilon$ we have
(a) \( \delta^\varphi < \delta^p \).
(b) \( \gamma^\varphi_i = \xi^\varphi_i = \gamma^p_i \quad \forall i < j \).

Proof.

(a) First we need to determine the value of \( \gamma^\varphi_i (\delta^p) \) for each \( i \in \{1, \ldots, s\} \). Rewriting definition (5) we obtain

\[
\gamma^\varphi_i (\delta^p) = \begin{cases} \xi^\varphi_i & \text{if } \delta^p \beta^\varphi_i \leq \xi^\varphi_i \\ \delta^p \gamma^p_i & \text{otherwise.} \end{cases}
\]  

(6)

Case 1: \( i < j \).

\[
i < j \implies \xi^\varphi_i = \gamma^p_i \quad \text{and} \quad \beta^\varphi_i = \beta^p_i
\]

\[
\implies \gamma^\varphi_i (\delta^p) = \begin{cases} \gamma^p_i & \text{if } \delta^p \beta^\varphi_i \leq \gamma^p_i \\ \delta^p \beta^p_i & \text{otherwise.} \end{cases}
\]  

(7)

Considering both cases of definition (5) we observe that

\[
\begin{cases}
\delta^p \beta^\varphi_i \leq \xi^\varphi_i \implies \gamma^p_i = \alpha^p_i \implies \gamma^\varphi_i (\delta^p) = \gamma^p_i \\
\delta^p \beta^\varphi_i > \xi^\varphi_i \implies \gamma^p_i = \delta^p \beta^p_i \implies \gamma^\varphi_i (\delta^p) = \gamma^p_i 
\end{cases}
\]

\[
\implies \gamma^\varphi_i (\delta^p) = \gamma^p_i \quad \forall i < j.
\]

Case 2: \( i = j \).

\[
i = j \implies \xi^\varphi_i = \gamma^p_i \quad \text{and} \quad \beta^\varphi_i = \gamma^p_i
\]

\[
\implies \gamma^\varphi_i (\delta^p) = \begin{cases} \xi^\varphi_i & \text{if } \delta^p \gamma^p_i \leq \xi^\varphi_i \\ \delta^p \gamma^p_i & \text{otherwise} \end{cases}
\]

\[
\implies \gamma^\varphi_i (\delta^p) = \delta^p \gamma^p_i.
\]

Case 3: \( i > j \).

\[
i > j \implies \xi^\varphi_i = \gamma^p_i \quad \text{and} \quad \beta^\varphi_i = \beta^p_i
\]

\[
\implies \gamma^\varphi_i (\delta^p) = \begin{cases} \xi^\varphi_i & \text{if } \delta^p \beta^p_i \leq \xi^\varphi_i \\ \delta^p \beta^p_i & \text{otherwise} \end{cases}
\]

\[
= \gamma^p_i.
\]
We can now bound the weight of subinterval \([γ^{QP}_j(δ^P), β^{QP}_j]\) by \(δ^P e\)

\[
W([γ^{QP}_j(δ^P), β^{QP}_j]) = \prod_{i=1}^s β_i^{QP_j} - \prod_{i=1}^s γ_i^{QP_j}(δ^P)
= \delta^P \prod_{i=1}^s β_i^P - \prod_{i=1}^s γ_i^{QP_j}(δ^P)
\leq \delta^P \left( \prod_{i=1}^s β_i^P - \prod_{i=1}^s γ_i^{QP_j}(δ^P) \right) = \delta^P e < e
\Rightarrow \delta^{Q_j} < \delta^P.
\]

(b) First we observe that \(α_i^{Q_j} = γ_i^P\ \forall i < j\) by definition (4). Next, from definition (5) we have

\[
gamma_i^{Q_j} = \begin{cases} α_i^{Q_j} & \text{if } \delta^Q β_i^{Q_j} \leq α_i^P \\ δ^{Q_j} β_i^{Q_j} & \text{otherwise.} \end{cases}
\]

\[
i < j \quad \Rightarrow (4) \quad γ_i^{Q_j} = \begin{cases} γ_i^P & \text{if } \delta^Q β_i^{Q_j} \leq γ_i^P \\ δ^{Q_j} β_i^{Q_j} & \text{otherwise.} \end{cases}
\]

Case 1: \(δ^P β_i^P \leq α_i^P\).

\[
\delta^P β_i^P \leq α_i^P \quad \Rightarrow (5) \quad γ_i^P = α_i^P
\]

\[
δ^{Q_j} < δ^P \quad \Rightarrow \quad δ^{Q_j} β_i^{Q_j} \leq δ^P β_i^P \leq α_i^P = γ_i^P \quad \Rightarrow \quad γ_i^{Q_j} = γ_i^P.
\]

Case 2: \(δ^P β_i^P > α_i^P\).

\[
δ^P β_i^P > α_i^P \quad \Rightarrow (5) \quad γ_i^P = δ^P β_i^P
\]

\[
δ^{Q_j} < δ^P \quad \Rightarrow \quad δ^{Q_j} β_i^{Q_j} < δ^P β_i^P = γ_i^P \quad \Rightarrow \quad γ_i^{Q_j} = γ_i^P.
\]

The following result is a direct consequence of Lemma 3.1b.

**Corollary 3.1.** Let \(P = [a^P, b^P] \subset I^s\) be a subinterval with \(W(P) > e\). Then, for any \(j \in \{1, \ldots, s\}\) such that \(Q^P_j \neq \emptyset\) and \(W(Q^P_j) > e\), we have

\[
Q^P_i = \emptyset \quad \forall i < j.
\]

This means that for each \(i < j\), the decomposition of \(Q^P_j\) (which is obtained from \(P\) after decompositions in directions \(1, \ldots, j\)) in direction \(i\)
leads to an empty subinterval and need not be considered. Thus, $Q^r_j$ is partitioned into at most $s-j+2$ subintervals, namely

$$Q^r_j = \left( \bigcup_{i=j}^{s} Q^r_i \right) \cup [y^r_i, \beta^r_i).$$

This leads to the following reformulation of the partition algorithm:

Algorithm 2 (Input: $s,e$  Output: $\mathcal{P}^r_e$).

**Main**

$\mathcal{P}^r_s := \emptyset$

$\text{decompose}(I^r,1)$

**Procedure** $\text{decompose}(P,j)$

$\gamma^r_i := \alpha^r_i$  $\forall i < j$

compute $\delta^r$ and $\gamma^r_j, \ldots, \gamma^r_s$

For $i = j, \ldots, s$

If $\gamma^r_i \neq \alpha^r_i$

If $W(Q^r_i) > e$

$\text{decompose}(Q^r_i, i)$

Else

$\mathcal{P}^r_i := \mathcal{P}^r_i \cup Q^r_i$

$\mathcal{P}^r_s := \mathcal{P}^r_s \cup [\gamma^r, \beta^r)$

**Theorem 3.1.** Both $\{\alpha^r: P \in \mathcal{P}^r_i\}$ and $\{\beta^r: P \in \mathcal{P}^r_i\}$ are generated by the partition algorithm in lexicographic order.

**Proof.** For a nonempty subinterval $Q^r_j$ arising in the course of the algorithm, we denote by $\mathcal{F}(Q^r_j) \subset \mathcal{P}^r_i$ the family of subintervals issued from the recursive decomposition of $Q^r_j$ into subintervals of weight smaller than or equal to $e$. By the recursion it is clear that, $\forall j < s$, the construction of $\mathcal{F}(Q^r_j)$ is entirely completed before the processing of $Q^r_{j+1}$ is started.

From definition (4) we see that the $s+1$ subintervals that can potentially arise from the decomposition of a subinterval $P$ are in lexicographic order:

$$\alpha^r = \alpha^r_1 < \ldots < \alpha^r_s < \gamma^r$$

$$\beta^r < \ldots < \beta^r_s < \beta^r.$$

In the following, we show this order is preserved between the subintervals in $\mathcal{F}(Q^r_j)$ and those in $\mathcal{F}(Q^r_{j+1})$ for all $j < s$, as well as between the subintervals in $\mathcal{F}(Q^r_j)$ and $[\gamma^r, \beta^r)$. 
Using definition (4), Lemma 3.1b, and its corollary, we have

\[ a_i^R = \gamma_i^R \quad \forall i < j, \forall R \in \mathcal{F}(Q_i^R). \]

For a nonempty \( Q_i^r \), we have \( a_i^Q = \alpha_i^r < \gamma_i^r = \beta_i^Q \).

\[ \Rightarrow \quad a_i^r \in [\alpha_i^r, \gamma_i^r) \quad \forall R \in \mathcal{F}(Q_i^r). \]

In particular, we have \( a_i^R < \gamma_i^R \forall R \in \mathcal{F}(Q_i^r) \) and we finally obtain

\[
\begin{cases}
\alpha^R = (\gamma_1^R, \ldots, \gamma_{j-1}^R, a_j^R, \ldots, a_{s}^R) < \alpha_{Q_i^r}^R & \text{if } j < s \\
\alpha^R = (\gamma_1^R, \ldots, \gamma_{j-1}^R, a_j^R) < \gamma^R & \text{if } j = s.
\end{cases}
\]

(b) \( \{ \beta^r: P \in \mathcal{P}^r \} \)

Using definition (4) and Corollary 3.1, we have

\[ \beta_i^R = \beta_i^r \quad \forall i < j, \forall R \in \mathcal{F}(Q_i^r). \]

By construction, \( \beta_i^R \leq \beta_j^Q = \gamma_j^r < \beta_i^r \forall R \in \mathcal{F}(Q_i^r) \)

\[ \beta_i^R \leq \gamma_i^r < \beta_i^r \quad \Rightarrow \quad \beta^R = (\beta_1^r, \ldots, \beta_{j-1}^r, \beta_j^r, \ldots, \beta_s^r) < \beta^r \quad \forall R \in \mathcal{F}(Q_i^r). \]

And finally, \( \forall j < s, \forall R \in \mathcal{F}(Q_j^r), \forall T \in \mathcal{F}(Q_{j+1}^r) \) we have

\[ \beta^R = (\beta_1^r, \ldots, \beta_{j-1}^r, \beta_j^r, \beta_{j+1}^r, \ldots, \beta_s^r) < (\beta_1^r, \ldots, \beta_j^r, \beta_{j+1}^r, \ldots, \beta_s^r) = \beta^T. \]

We now show how parameters \( \delta^r \) and \( \gamma^r \) can be determined and state the final version of our partition algorithm.

**Lemma 3.2.** For each call to \( \text{decompose}(P, j) \) during the decomposition process and each \( i \in \{j, \ldots, s\} \) with \( Q_i^r \neq \emptyset \) we have

\[ W(Q_i^r) = \delta^r W(P). \]

**Proof.** Examining definition (4), we observe that the \( a \) component of index \( s \) is preserved throughout the decomposition process:

\[ a_s^Q = \alpha_s^r = \cdots = 
\alpha_s^r = 0. \]
Then, we have
\[ W(Q^P_j) = \prod_{k=1}^{s} \beta^P_k - \prod_{k=1}^{s} \alpha^P_k = \prod_{k=1}^{s} \beta^P_k - \prod_{k=1}^{s} \beta^P_k = \delta^P \prod_{k=1}^{s} \beta^P_k = \delta^P W(P). \]

**Theorem 3.2.** For each call to \( \text{decompose}(P,j) \) during the decomposition process, we have
\[ \gamma^P_i = \begin{cases} \alpha^P_i & \text{if } i < j \\ \delta^P \beta^P_i & \text{if } i \geq j, \end{cases} \tag{8} \]
where
\[ \delta^P = \left( \frac{\prod_{i=1}^{j-1} \beta^P_i - \varepsilon}{\prod_{i=1}^{j-1} \alpha^P_i \prod_{i=j}^{s} \beta^P_i} \right)^{-1/j-1.} \tag{9} \]

**Proof.** By definition, \( \delta^P \in ]0,1[ \) is such that
\[ W( (\gamma^P, \beta^P) ) = \varepsilon \]
\[ \gamma^P_i = \begin{cases} \alpha^P_i & \text{if } \delta^P \beta^P_i \leq \alpha^P_i \\ \delta^P \beta^P_i & \text{otherwise} \end{cases} \]
\( i = 1, \ldots, s. \)

Considering the entire decomposition process, \( P \) is issued from \( I^s \) by a series of decompositions in directions of index smaller than or equal to \( j \) (Corollary 3.1). Then, from definition (4) we have
\[ \alpha^P_i = \cdots = \alpha^P_j = 0 \quad \forall i \geq j. \]

Thus, for any \( \delta \in ]0,1[ \), we have \( \delta \beta^P_i > \alpha^P_i = 0, \forall i \geq j. \) Moreover, using Lemma 3.1b, we have \( \gamma^P_i = \alpha^P_i, \forall i < j \). This establishes (8). We finally obtain (9) by solving equation \( W( (\gamma^P, \beta^P) ) = \varepsilon. \)

Combining these results, Corollary 3.1 can be strengthened:

**Corollary 3.2.** For each call to \( \text{decompose}(P,j) \) during the decomposition process of \( I^s \), \( P \) is partitioned into exactly \( s-j+2 \) subintervals, namely
- the \( s-j+1 \) subintervals \( Q^P_j, \ldots, Q^P_s \) of common weight \( \delta^P W(P) \);
- subinterval \( [\gamma^P, \beta^P] \) of weight \( \varepsilon. \)
We can now state the final version of our partition algorithm:

**Algorithm 3** (Input: $s, \varepsilon$  Output: $\mathcal{P}_s^\prime$).

**Main**

$\mathcal{P}_s^\prime := \emptyset$

decompose($I^s, 1, 1$)

**Procedure** decompose($P, j, v$)

compute $\delta^P$ according to (9)

compute $\gamma^P$ according to (8)

If $\delta^P v > \varepsilon$

For $i = j, \ldots, s$

decompose($Q_i^P, i, \delta^P v$)

Else

For $i = j, \ldots, s$

$\mathcal{P}_s^\prime := \mathcal{P}_s^\prime \cup Q_i^P$

$\mathcal{P}_s^\prime := \mathcal{P}_s^\prime \cup \omega^P$)

The overall running time of this algorithm is in $O(s |\mathcal{P}_s^\prime|)$.

### 3.4. Cardinality of Partition $\mathcal{P}_s^\prime$

For integers $w \geq 1$ and $h \geq 0$, the triangular tree $T_h^w$ (see Fig. 4 for an example) of height $h$ and width $w$ is recursively defined as follows:

- $T_0^w$ is a leaf;
- $T_h^w$ with $h > 0$ is a tree made up of a root with $w$ attached triangular trees $T_{h-1}^w, \ldots, T_h^w$.

**Theorem 3.3.** Triangular tree $T_h^w$ has $N_h^w = \binom{h+w-1}{w-1}$ leaves.

**Proof.** The claim is obvious for $h = 0$, $T_0^w$ is a leaf and $N_0^w = \binom{w-1}{w-1} = 1$. By induction, we suppose the result is proved for $h$ and we show it for $h+1$:

$$N_{h+1}^w = \sum_{i=1}^{w} N_h^i = \sum_{i=1}^{w} \binom{h+i-1}{i-1} = \sum_{i=0}^{w-1} \binom{h+w-i-1}{w-i-1}.$$

Applying identity $\binom{b+1}{a} = \binom{b}{a} + \binom{b}{a-1}$ recursively, we get

$$\binom{b+1}{a} = \sum_{i=0}^{a} \binom{b-i}{a-i}.$$
Using this identity with \( a = w - 1 \), \( b = h + w - 1 \), we obtain

\[
N^*_{h+1} = \binom{h+w}{w-1}.
\]

We now show partition \( \mathcal{P}_s^e \) is finite.

**Theorem 3.4.** For any dimension \( s \geq 2 \) and value \( e \in [0, 1[ \), we have

\[
|\mathcal{P}_s^e| \leq \binom{s+h}{s} \quad \text{with} \quad h = \left\lfloor \frac{s \log e}{\log (1-e)} \right\rfloor.
\]

**Proof.** For each call to \( \text{decompose}(P, j) \) during the decomposition process, subinterval \( P \) is partitioned into exactly \( s-j+2 \) subintervals \( Q^r_j, \ldots, Q^r_s \), and \( [y^r, \beta^r] \). Subinterval \( [y^r, \beta^r] \) has weight \( e \) and is inserted in the partition. As mentioned in Corollary 3.2, the weight of the remaining \( s-j+1 \) subintervals is reduced by a factor \( \delta^r \) (in relation to the weight of \( P \)). From Lemma 3.1a, we know that this factor \( \delta^r \) is smaller than the first one \( \delta^r = (1-e)^{1/s} \) occurring in the decomposition process.

Then, the whole decomposition process can be represented as a triangular tree of width \( s+1 \) (with numerous cut branches), whose leaves correspond to the subintervals of the partition. The height of this tree is bounded by the smallest integer \( h \) satisfying

\[
((1-e)^{1/s})^h \leq e.
\]

Thus we have

\[
h = \left\lfloor \frac{s \log e}{\log (1-e)} \right\rfloor \quad \text{and} \quad |\mathcal{P}_s^e| \leq N^*_{h+1} = \binom{s+h}{s}.
\]
### Table II

Cardinality of Partition $\mathcal{P}_s^e$ in Dimension $s = 5$ for Different Values of $e$

<table>
<thead>
<tr>
<th>$e$</th>
<th>0.3</th>
<th>0.2</th>
<th>0.1</th>
<th>0.05</th>
<th>0.03</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>\mathcal{P}_s^e</td>
<td>$</td>
<td>1’546</td>
<td>12’566</td>
<td>428’882</td>
</tr>
<tr>
<td>$(^{+}\mathcal{P}_s^e)$</td>
<td>26’334</td>
<td>850’668</td>
<td>153’476’148</td>
<td>18’934’442’604</td>
<td>542’256’910’321</td>
</tr>
<tr>
<td>$\frac{1}{s}$</td>
<td>2’057</td>
<td>15’625</td>
<td>50’000</td>
<td>160’000</td>
<td>205’761’317</td>
</tr>
</tbody>
</table>

**Note.** Note that the bound of Theorem 3.4 is loose and the estimate $s/e$ is fairly accurate.

Experimentally, the above bound for $|\mathcal{P}_s^e|$ is rather weak. On the other hand, as illustrated in Table II, the purely empirical estimate $\frac{1}{s}$ is usually fairly close to the exact cardinality. In any case, the “curse of dimension” still rules and $|\mathcal{P}_s^e|$ grows exponentially with $s$.

As mentioned in Section 2, the use of Theorem 2.1 when partition $\mathcal{P}$ is a grid implies substantial waste of computational effort. In Table III, we compare the cardinality of the thinnest optimal grid used in [21] with the cardinality of the corresponding partition $\mathcal{P}_s^e$ for certain dimensions $s \in \{2, ..., 20\}$. We observe that our new partitions $\mathcal{P}_s^e$ are more economical than the corresponding grids and that the advantage becomes clearer as dimension $s$ increases. Thus, for a given computational cost, we can expect to work with thinner partitions of $I^s$ and get better bounds with our new method.

### Table III

Comparison of the Cardinality of $\mathcal{P}_s^e$ with the Cardinality $k^e$ of the Thinnest Optimal Grid Considered in [21] for Different Dimensions $s \in \{2, ..., 20\}$

<table>
<thead>
<tr>
<th>$s$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e = M_s^e$</td>
<td>0.090263</td>
<td>0.10514</td>
<td>0.11374</td>
<td>0.12100</td>
<td>0.12737</td>
<td>0.13198</td>
</tr>
<tr>
<td>$k^e$</td>
<td>225</td>
<td>3’375</td>
<td>50’625</td>
<td>759’375</td>
<td>11’390’625</td>
<td>170’859’375</td>
</tr>
<tr>
<td>$</td>
<td>\mathcal{P}_s^e</td>
<td>$</td>
<td>176</td>
<td>1’763</td>
<td>17’600</td>
<td>163’171</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.18456</td>
<td>0.29999</td>
<td>0.42581</td>
<td>0.68599</td>
<td>0.89390</td>
<td></td>
</tr>
<tr>
<td>$214’358’881$</td>
<td>$282’475’249$</td>
<td>$244’140’625$</td>
<td>$43’046’721$</td>
<td>$1’048’576$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$8’419’312$</td>
<td>$2’761’801$</td>
<td>$533’026$</td>
<td>$4’734$</td>
<td>$21$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. COMPUTING BOUNDS FOR THE STAR DISCREPANCY

Now for given dimension $s \geq 2$ and accuracy parameter $\varepsilon \in (0, 1)$, we know how to construct a partition $\mathcal{P}_e$ of $I^s$ into subintervals such that

$$W(\mathcal{P}_e) = \varepsilon.$$ 

Then, for a given sequence of $n$ points $x = \{x_0, \ldots, x_{n-1}\}$ in $I^s$ we have to compute the upper bound $B(\mathcal{P}_e, x)$ for the star discrepancy (see Section 4.4 for the lower bound):

$$B(\mathcal{P}_e, x) = \max_{P \in \mathcal{P}_e} \max_n \left\{ \left( \frac{A(P^+, x)}{n - \lambda(P^+)} - \frac{A(P^-, x)}{n - \lambda(P^-)} \right) \right\}.$$

Thus, for each subinterval $P = \prod_{i=1}^{n} [a_i, b_i] \in \mathcal{P}_e$ we must determine the number of points of $x$ belonging to the subintervals $P^- = \prod_{i=1}^{n} (0, a_i)$ and $P^+ = \prod_{i=1}^{n} [0, b_i]$.

This counting can be done naively, by complete enumeration of the $n$ points for each of the $2|\mathcal{P}_e|$ subintervals to be considered. It leads to a simple $O(n|\mathcal{P}_e|)$ algorithm for the computation of $B(\mathcal{P}_e, x)$. However, we can do better than linear time in $n$. In the following, we propose a method based on orthogonal range counting that runs much faster.

To motivate our approach, we illustrate in advance this superiority with the results of Table IV in the case $s = 7$, $\varepsilon = 0.1$. We see that, with the naive enumeration counting algorithm, the computation time per point is almost constant (the slight upward trend probably being caused by cache memory effects), whereas with the orthogonal range counting algorithm presented below, this marginal time is smaller and decreases as $n$ increases.

Let us point out that partition $\mathcal{P}_e$ does not have to be stored. The numbers $A(P^-, x)$ and $A(P^+, x)$ can be computed on the fly, as subintervals of the partition are generated.

**TABLE IV**

<table>
<thead>
<tr>
<th>$n$</th>
<th>50</th>
<th>100</th>
<th>500</th>
<th>1000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerating counting</td>
<td>22.0</td>
<td>22.6</td>
<td>23.7</td>
<td>26.8</td>
<td>26.8</td>
</tr>
<tr>
<td>Orthogonal range counting</td>
<td>15.0</td>
<td>11.0</td>
<td>5.90</td>
<td>4.95</td>
<td>3.46</td>
</tr>
</tbody>
</table>
4.1. Orthogonal Range Counting

Let \( S \) be a set of \( n \) points in \( \mathbb{R}^s \) and let \( R \) be a family of subsets of \( \mathbb{R}^s \) called ranges. Enumerating the points of \( S \) which belong to a given subset of \( R \) is the basic range query problem called range searching. In the particular case where the set of ranges \( R \) is made of subintervals of \( \mathbb{R}^s \), we talk of orthogonal range searching. If we further simplify the problem from enumerating the points of \( S \) which belong to a given subinterval of \( \mathbb{R}^s \) to counting those points, we talk of orthogonal range counting. Most data structures for orthogonal range queries are based on range trees introduced by Luecker [10] and Bentley [2]. Surveys can be found in the books by Mehlhorn [11] and Preparata and Shamos [16].

In our situation, \( S \) is the set of \( n \) points \( x = \{x_0, ..., x_n-1\} \subset I^s \) and we want to compute \( A(P^-, x) \) and \( A(P^+, x) \) for every \( P = \prod_{i=1}^{s} [a_i, b_i] \in \mathcal{P}_s \).

This is clearly equivalent to solving an orthogonal range counting problem for every subinterval in \( \mathcal{R} = \mathcal{R}_a \cup \mathcal{R}_b \) where

\[
\mathcal{R}_a = \{P^- : P \in \mathcal{P}_s\} \quad \text{and} \quad \mathcal{R}_b = \{P^+ : P \in \mathcal{P}_s\}.
\]

We define an \( s \)-dimensional range tree on \( n \) points \( x = \{x_0, ..., x_n-1\} \subset I^s \) inductively as follows. In dimension \( s = 1 \), a range tree is a usual binary search tree. In dimension \( s > 1 \), a range tree is a binary search tree organized according to the first component of the points. In addition, each node \( x_i \) of this tree contains a pointer to an \((s-1)\)-dimensional range tree on the projections into dimensions \( 2, ..., s \) of the points in the left subtree rooted at \( x_i \) (including \( x_i \)). This construction is illustrated by an example in Fig. 5. Our definition slightly differs from the classical one found in the literature, but the underlying principles are exactly the same.

Then, the resolution of an orthogonal range counting problem for a given subinterval \( P = [0, p) \in \mathcal{R} \) proceeds as follows. We start with a binary search of \( p_1 \) from the root of the \( s \)-dimensional range tree. We observe that the length of the corresponding path is at most \( O(\log n) \). We denote by \( y(p, 1) \) the set of nodes in this path that are followed by a move to the right. In the example of Fig. 5, we have \( y(p, 1) = \{2, 4\} \). The subset of points of \( x \) with first component smaller than \( p_1 \) is simply the set of points in \( y(p, 1) \) and in their left subtrees.

Then, for \( d = 2, ..., s \), we search \( p_d \) in each of the \((s-d+1)\)-dimensional range trees pointed to by a node in \( y(p, d-1) \). We define \( y(p, d) \) as the set of nodes which belong to one of the \((s-d+1)\)-dimensional trees under consideration which are on the corresponding binary search path of \( p_d \) and are followed by a move to the right in this path.
FIG. 5. A portion of the 3-dimensional range tree on six points of the unit cube and the exploration induced by an orthogonal range query for a specified subinterval.

The subset of points of $x$ with their first $d$ components respectively smaller than $p_1, ..., p_d$ is the set of points in $y(p, d)$ and in their left subtrees. If we store in every node of every 1-dimensional tree the cardinality of its left subtree (including its root), then the solution of orthogonal range counting is obtained by summing these cardinalities for every node in $y(p, s)$.

Every path in this process being of length $O(\log n)$, the problem is solved in $O((\log n)^s)$ time. Moreover, exploiting the structure of $\mathcal{P}_s$, we will see that the exploration of the range tree simplifies drastically.

A range tree can be constructed in $O(n(\log n)^s)$ space and time, but with a constant term which grows exponentially with the dimension [10]. Hopefully, we will not have to build the whole range tree.

4.2. Exploiting the Structure of $\mathcal{P}_s$

Considering the structure exposed in the previous section, it is easy to see that $[y', \beta']$ is the last subinterval inserted into the partition. Then, for any subinterval $P \in \mathcal{P}_s \setminus \{[y', \beta']\}$, let $S(P)$ denote its successor in $\mathcal{P}_s$. In Theorem 3.1, we have shown that $a^P < a^{S(P)}$ and $b^P < b^{S(P)}$.

Let $\alpha^P$ (respectively $\beta^P$) denote the smallest index $i \in \{1, ..., s\}$ such that $a_{i, a}^{S(P)} > a_i^P$ (respectively $\beta_{i, a}^{S(P)} > \beta_i^P$). Then, we have

$$a_{i, a}^{S(P)} = a_i^P \quad \forall i < \omega^P \quad \text{and} \quad \beta_{i, a}^{S(P)} = \beta_i^P \quad \forall i < \omega^P.$$
These facts will prove to be very useful, but first let us show how $\omega^s$ and $\omega^p$ can be effectively determined.

**Theorem 4.1.** Let $P \in \mathcal{P}_s \setminus \{[\gamma^*, \beta^*]\}$. Then

$$\omega^s = \omega^p = \omega^r,$$

where

$$\omega^r = \begin{cases} j - 1 & \text{if } S(P) \text{ directly stems from a decomposition in direction of index } j^1 \\ s & \text{otherwise (that is, when } S(P) \text{ is a subinterval of the form } [\gamma^*, \beta^*]). \end{cases}$$

**Proof.** In the special case where $\delta^t \leq \varepsilon$, we have

$$\mathcal{P}_s = \{Q''_1, \ldots, Q''_s, [\gamma^*, \beta^*]\}.$$

Considering definition (4) we obtain

$$\omega^{s(t)} = \omega^{p(t)} = j \quad \text{for each } j \in \{1, \ldots, s\}.$$

Henceforth we assume that $\delta^t > \varepsilon$. It implies each subinterval $Q''_1, \ldots, Q''_s$ is further decomposed and there exists a subinterval $R$ from which $P$ stems after two decompositions (we mean $P$ has a grandfather $R$ as illustrated in Fig. 6). Basically, there are two possibilities:

1. The case $j = 1$ never occurs: the only subinterval that directly stems from a decomposition in direction 1 is the very first one of the partition. Thus $j \in \{2, \ldots, s\}$. 

![FIGURE 6](image_url)
• If $P = Q_i^R$ for some $i \leq j \leq s$, then

$$S(P) = \begin{cases} Q_{j+1}^R & \text{if } j < s \\ [\gamma_i^R, \beta_i^R] & \text{if } j = s. \end{cases}$$

Considering definition (4) in both cases we obtain

$$\omega^R = \omega^R = j.$$

• If $P = [\gamma_i^R, \beta_i^R]$ for some $i \leq s$, then for $d \leq i$ we have

$$\begin{align*}
\alpha_d^R &= \gamma_d^R \\
\beta_d^R &= \beta_d^R \\
\delta_d^R &= \gamma_d^R \\
\beta^R_d &= \beta_d^R \\
\gamma^R_d &= \gamma_d^R
\end{align*}$$

for $d < i$

for $d = i$,

for $d < i$

for $d = i$.

If $i = s$, then $S(P) = [\gamma_i^R, \beta_i^R]$, and we obtain

$$\omega^R = \omega^R = s.$$

Finally, if $i < s$, then $S(P)$ is either $Q_{i+1}^R$ (for we know $W(Q_{i+1}^R) = W(Q_i^R) > c$) or a subinterval of the form $Q_{i+1}^R$ obtained after a series of decompositions of $Q_{i+1}^R$ in direction $i+1$. In any case, we have

$$\begin{align*}
\alpha_d^R &= \gamma_d^R \\
\beta_d^R &= \beta_d^R \\
\delta_d^R &= \gamma_d^R \\
\beta^R_d &= \beta_d^R \\
\gamma^R_d &= \gamma_d^R
\end{align*}$$

for $d \leq i$

$$\Rightarrow \quad \omega^R = \omega^R = i.$$

Let $P \in P \setminus \{[\gamma_i^R, \beta_i^R]\}$. If $\omega^R > 1$, then for the orthogonal range counting problem associated with subinterval $[0, \alpha^R]$ we have

$$y(\alpha^R, i) = y(\alpha^R, i) \quad \forall i \in \{1, \ldots, \omega^R - 1\}.$$

Thus, considering the sequence of resolutions of orthogonal range counting problems, a part of the work done for $[0, \alpha^R]$ can be reused for $[0, \alpha^R]$. Namely, for $[0, \alpha^R]$, the exploration of the range tree can be started with the search of $\alpha^R$ in the trees pointed to by the nodes in $y(\alpha^R, \omega^R - 1)$. In practice, as suggested by Corollary 3.2 and Theorem 4.1, $\omega^R$ is frequently equal to $s$ or at least close to it. This fact drastically shortens the resolution of our orthogonal range counting problems.
Moreover, considering the fact that \( x_{s,f} > x_{s,f} \), we can even reuse a part of \( y(x^f, \omega^f) \) in the resolution of the problem associated with \([0, x_{s,f}^s])\). If, for a given tree pointed to by a node in \( y(x^s, \omega^s - 1) \), the beginning of the search path of \( x_{s,f}^s \) is composed only of moves to the right, then this initial path fragment will be the same for the search of \( x_{s,f}^s \) and need not be recomputed. For example, in Fig. 5, if we have \( \omega^f = 1 \), then we can be sure that \( \{2\} \subset y(S(p), 1) \). Using the same argument when \( \omega^s = s \), the partial summations of the cardinalities stored in the nodes along such an initial path fragment can be stored and do not have to be recomputed.

Of course, the above development for the requests in \( R_a \) can be reformulated for those in \( R_b \). In our implementation of the method, the entire range tree is not precomputed. The required parts of the range tree are built on the fly, as subintervals in \( P^s \) are generated.

Moreover, we can also save memory by discarding in the course of the algorithm the parts of the range tree which we know will never be used again. We will not go into detail, but mention that this can be done (in the suitable left part of the range tree) each time \( \omega^f = 1 \) and the length of the initial path fragment (as above) in the \( s \)-dimensional range tree is longer for the search of \( x_{s,f}^s \) than for \( x_{s,f}^f \).

### 4.3. Unbalancing the Trees

In our definition of range trees, we did not specify what kind of binary search trees were to be used. The optimal way, for uniformly distributed requests, is to use balanced binary search trees (for any node of such trees, the cardinalities of the left and right subtrees differ by at most one). However, our request set \( R \) is far from being uniformly distributed and requests appear to be concentrated in the neighborhood of 1 (particularly in high dimension). We can reduce our average request time by using unbalanced binary search trees.

For example, the results of Table IV were obtained with binary search trees characterized by an unbalance parameter of \( \frac{4}{5} \). This implies that, for any node of those trees, there are four times more points in the left subtree than in the right one. This fact considerably shortens the length of search paths for the numerous requests that are in the neighborhood of 1 (and lengthens it for the few that are not). The price paid for the resulting speedup is higher storage requirement.

As long as for any node a constant fraction of the nodes are in each of the left and right subtrees, the height of the whole unbalanced tree remains \( O(\log n) \) (with a constant term that depends on this fraction). Then, the use of balanced or unbalanced trees does not affect the worst case complexity for orthogonal range counting, which remains \( O((\log n)^s) \) time.
Unbalancing can be seen as a tradeoff between space and time. In our implementation, the unbalance parameter is an optional argument (the default being $\frac{1}{2}$) which can be set depending on memory availability. Schematically, the higher this parameter, the faster the algorithm, but the more memory required. In practice, the choice of a suitable unbalance parameter on a given machine calls for some experimentation.

4.4. Lower Bounds for the Star Discrepancy

We can take advantage of the work done for the upper bound $B(\mathcal{P}^*, x)$ for the star discrepancy $D^*_g(x)$ to compute the corresponding lower bound $C(\mathcal{P}^*, x)$. Namely, the values of $A(P, x)$ already computed for every $P \in \mathcal{R}$ can be reused:

$$C(\mathcal{P}^*, x) = \max_{P \in \mathcal{R}} \left| \frac{A(P, x)}{n} - \bar{\lambda}(P) \right|.$$

Moreover, we can easily obtain a tighter lower bound than $C(\mathcal{P}^*, x)$. For any subinterval $P = \prod_{i=1}^s [0, p_i) \in \mathcal{R}$, each of its components $p_i$ ($i = 1, \ldots, s$) can be rounded up or down (according to the sign of $\bar{\lambda}(P) - A(P, x)/n$) to the closest corresponding component of a point of $x$. It is clear that this transformation yields subintervals of optimized volumes with identical values of $A(P, x)$, and consequently a tighter lower bound for the star discrepancy.

**Remark 4.1.** The $L^2$ star discrepancy $T^*_g(x)$

$$T^*_g(x) = \left[ \int_{P \in \mathcal{R}} \left( \frac{A(P, x)}{n} - \bar{\lambda}(P) \right)^2 dP \right]^{\frac{1}{2}}$$

is another lower bound for the star discrepancy $D^*_g(x)$. $T^*_g(x)$ is easy to compute [22] [12], but it does not give better lower bounds than ours.

5. NUMERICAL EXPERIMENTS

In this section, we illustrate the performance of our method with some numerical results. First, we give best-known upper and lower bounds for the star discrepancy of some $(0, m, s)$-nets [14]. Then, we compare the star discrepancy of five different sequences in dimension 7. Finally, we use our algorithm to gain insight into some theoretical questions.
5.1. Faure Nets

As mentioned in [21], the computation of the star discrepancy with the discretization proposed in [13] is only reasonable in low dimensions (say \( s \leq 6 \)) and for small sample sizes. That discretization method is intractable for the cases treated below.

In Table V, we give best-known bounds for the star discrepancy of some Faure [5] \((0, m, s)\)-nets in base \( b \) for \( s \in \{7, 10, 12, 15, 20, 100\} \). For each of these nets (except the one in dimension 2), the corresponding upper bound proposed in [14] is greater than 1. These results are also better than the corresponding ones obtained with the grid decomposition method [21].

Accuracy parameter \( e_{up} \) has been fixed so that each of these bounds required a few days of computation. However, with only slightly greater values, much less time is required. For example, the computation of the upper bound 0.416446 for the Faure \((0,3,15)\)-net with \( e = 0.35 \) took 112 hours on a 500 MHz Pentium III PC, whereas with \( e = 0.45 \) it only takes 2 hours to obtain an upper bound of 0.465755. This is related to the fact that the computation is approximately proportional to \(|\mathcal{P}_s|\), which grows roughly like \( s/e^s \).

### Table V

<table>
<thead>
<tr>
<th>( s )</th>
<th>( b )</th>
<th>( m )</th>
<th>( n = b^m )</th>
<th>( e_{low} )</th>
<th>( e_{up} )</th>
<th>( D^*_s(x) \in )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>20</td>
<td>1'048'576</td>
<td>0.000007</td>
<td>0.000007</td>
<td>[0.0000071, 0.0000136] *</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>2</td>
<td>49</td>
<td>0.05</td>
<td>0.05</td>
<td>[0.269011, 0.295125]</td>
</tr>
<tr>
<td>3</td>
<td>343</td>
<td>0.59</td>
<td>0.05</td>
<td>[0.129832, 0.168598]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2'401</td>
<td>0.6</td>
<td>0.05</td>
<td>[0.030518, 0.074176]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>2</td>
<td>121</td>
<td>0.6</td>
<td>0.125</td>
<td>[0.248508, 0.337404]</td>
</tr>
<tr>
<td>3</td>
<td>1'331</td>
<td>0.58</td>
<td>0.15</td>
<td>[0.093028, 0.220886]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>2</td>
<td>169</td>
<td>0.61</td>
<td>0.18</td>
<td>[0.265266, 0.396727]</td>
</tr>
<tr>
<td>3</td>
<td>2'197</td>
<td>0.58</td>
<td>0.22</td>
<td>[0.096713, 0.283217]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>2</td>
<td>289</td>
<td>0.59</td>
<td>0.26</td>
<td>[0.256021, 0.455008]</td>
</tr>
<tr>
<td>3</td>
<td>4'913</td>
<td>0.57</td>
<td>0.35</td>
<td>[0.085855, 0.416446]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>23</td>
<td>2</td>
<td>529</td>
<td>0.58</td>
<td>0.5</td>
<td>[0.259366, 0.722888]</td>
</tr>
<tr>
<td>3</td>
<td>12'167</td>
<td>0.55</td>
<td>0.5</td>
<td>[0.080737, 0.509607]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>101</td>
<td>1</td>
<td>101</td>
<td>0.99</td>
<td>0.96</td>
<td>[0.954159, 0.961973]</td>
</tr>
</tbody>
</table>

*Note.* The upper and lower bounds were respectively obtained with \( e_{up} \) and \( e_{low} \).

*In low dimensions, our algorithm can compute fairly accurate bounds for large sample sizes, although upper bounds in [14] are often better (0.0000105 in this case).
5.2. Comparison of Sequences

Star discrepancy is a popular measure for the irregularity of sequences. A low-discrepancy sequence $x$ in $I^s$ is a sequence for which we have

$$D^*_g(x) \leq C_x \frac{(\log n)^s}{n} + O\left(\frac{(\log n)^{s-1}}{n}\right) \quad \text{for all } n \geq 2. \quad (11)$$

It is widely believed, though not yet proved, that there exists no sequence with a better order of magnitude (the only thing we are sure of is that any sequence satisfies $D^*_g(x) \geq B_n n^{-s-1} (\log n)^{s/2}$ for infinitely many $n$, where the constant $B_n$ depends only on $s$ [17]). Thus, low-discrepancy sequences are often compared by the value of their best-known dominant constant term $C_x$ [6]. Such comparisons are presumably not meaningful for practical purposes, for such constants characterize a whole sequence, whereas a quasi-Monte Carlo approximation only uses a finite number of points. Moreover, these constants themselves being upper bounds, the result of the comparison may depend more on our ability to find good constants, than on the actual star discrepancy of the sequences. On the other hand, with our algorithm we can effectively compare the star discrepancy of finite point sets in some nontrivial dimensions. In Fig. 7 we give upper and lower bounds for the star discrepancy of samples from five different sequences in dimension 7:

- Rand(): the pseudo-random generator from the standard C library associated with the gcc compiler (presumably poor—it’s period is only $2^{15}$);
- MRG32k3a: a powerful combined multiple recursive pseudo-random generator proposed by L’Ecuyer [9];
  - the Halton [7] sequence in bases 2, 3, 5, 7, 11, 13, and 17;
  - a scrambled$^2$ Sobol sequence as implemented by Bratley and Fox [3];
  - a scrambled$^2$ Faure sequence as implemented by Thiémard [20].

First we observe that among these five sequences, the Sobol and Faure samples have the smallest star discrepancy and present similar performances for $n \leq 100$. From $n = 150$ points, the Halton samples seem to perform as well as those of Sobol and Faure. The samples from the generator of L’Ecuyer clearly have the highest star discrepancy and the most irregular downward trend among the five sequences. This comes as no surprise from a sequence designed to mimic a uniform random distribution.

$^2$ The Sobol and Faure sequences are scrambled with a ($b$-ary) Gray code (see [1, 19, 20]).
Such a sequence has both regularity and independence properties, whereas a low-discrepancy sequence only aims at regularity.

For Halton, Sobol, and Faure sequences in dimension 7, the best-known constants $C_x$ in upper bound (11) are, respectively, 17.3, 5.28, and 0.0041 [6]. Predictably, our results show that, at least in dimension 7 and for small sample sizes, these constants are not representative of the actual star discrepancy of the corresponding sequences.

5.3. An Intuitive Insight-Providing Tool

Let us denote by $n(s,d)$ the minimal number of points in $I^s$ with star discrepancy at most $d$. In a recent paper, Heinrich, Novak, Wasilkowski, and Woźniakowski [8] proved (in a non-constructive way) that $n(s,d)$ depends linearly on $s$ and at most quadratically on $d^{-1}$.

It would be interesting to know if corresponding minimal samples from classical families of low-discrepancy sequences present the same growth, or
if we should suspect that there exist intrinsically better sequences than those currently known. We cannot answer this, but our algorithm can be used to develop insight for this question.

For the special case of Faure sequences scrambled with a Gray code [19] [20], we used our algorithm to seek the minimal number of points with star discrepancy at most 0.45 in dimension $s = 4, \ldots, 12$. Since our algorithm only computes upper and lower bounds for the star discrepancy, we could not determine these minima but could only provide ranges for them. These sets are represented in Fig. 8.

We should be cautious here, since we do not know what happens in higher dimensions. However, it seems that for $s = 4, \ldots, 12$ the minimal number of points from a scrambled Faure sequence with star discrepancy at most 0.45 tends to grow faster than linearly with dimension. We will not try to further extrapolate from these results. Our purpose here is simply to show that our algorithm can also be used to gain some intuition for such questions.

![FIG. 8. Sets of possible values for the minimal number of points of a scrambled Faure sequence with star discrepancy at most 0.45 in dimension $s = 4, \ldots, 12$.](image)
6. CONCLUSION

The method proposed in this paper is a step towards better assessment of the star discrepancy. It can compute nontrivial upper and lower bounds in dimensions that previously seemed out of reach. However, its efficiency is limited, because the cardinalities of the partitions to consider grow very rapidly with dimension and accuracy requirements.

ACKNOWLEDGMENT

I thank the editors Fred Hickernell and Henryk Woźniakowski for suggesting Theorem 2.2 and for their useful comments on Section 2. I also thank Leslie Trotter for careful reading and for suggestions that improved the quality of this paper.

REFERENCES

8. S. Heinrich, E. Novak, G. W. Wasilkowski, and H. Woźniakowski, The star discrepancy depends linearly on the dimension, manuscript submitted for publication.


