A general software tool for constructing rank-1 lattice rules

Lattice Builder

David Munger & Pierre L'Ecuyer

Département d'informatique et de recherche opérationnelle
Université de Montréal

MCQMC 2012 / February 13–17, 2012

Plan

1 **Motivation**

2 **Features**
- Overview of the Current Features
- Basic Usage
- Advanced Usage
- Usage for Research

3 **Challenges and Solutions**

4 **Conclusion**

How to Choose a Generating Vector?

Ideal Approach

1. Consider all possible generating vectors
2. Minimize the error or variance for our problem

Practical Approach

1. Narrow down the search space for generating vectors
2. Minimize a figure of merit
   representative of:
   - square error (QMC)
   - variance (RQMC)
   - others?

That is the purpose of Lattice Builder!

What Was Needed

Find Good Lattice Rules

- Ordinary and embedded lattices
- Figures of merit and weights adapted to one's problem
- Any number of points and dimension, when needed
- Various construction methods
- Optional normalization and filters

Research Purposes

- Evaluate a figure of merit for a lattice
- Compare performance of algorithms
- Distribution of values of a figure of merit
- Correlation between two figures of merit

What Was Already Out There

Using

Using Lattice Rules

- SSJ simulation library (Java)
- John Burkardt (C++, Fortran, Matlab)
- Christiane Lemieux's QMC Library (C)
- Dirk Nuyens' McInt for lattice sequences (C++)

Tables of Generating Vectors

- Published papers
- Authors' websites (Kuo)

What Was Already Out There

Constructing

Dirk Nuyens' Matlab Code

- Fast CBC construction
- Product weights
- Order-dependent weights
- Ordinary and embedded lattices

Lattice Builder Features

Input Parameters

Lattice type	ordinary and embedded
Number of points	
Dimension	
Figures of merit	$\mathcal{P}_{2\alpha}$, spectral (sum or max)
Weights	product, order-dependent, projection-dependent, POD (planned)
Construction	exhaustive, random, (random) Korobov, (random) CBC, fast CBC
Normalizations	bounds on the average for $\mathcal{P}_{2\alpha}$
Filters	low-pass
Combining	for multiple embedded levels: sum or max

Command-Line Tool

Command-Line Example

```
latbuilder
    --lattice-size ordinary:1021
    --dimension 7
    --merit sum:spectral
    --weights product:0.1:1.0,0.7,0.3
    --construction CBC
```

Output

```
LatDef(1021, [1, 96, 496, 81, 298, 354, 37]): 6.15564
CPU time: 0.41784 seconds
```

Lattice Size and Dimension

Ordinary Lattices

```
--lattice-size ordinary:1021
--dimension 7
```

Embedded Lattices

```
--lattice-size embedded:2^8
--dimension 7
```

(Complete example later…)

Weighted Figure of Merit

Weighted Spectral Merit

```
--merit max:spectral
```

$$\max_{\emptyset \neq \mathfrak{u} \subseteq \{1,\ldots,s\}} \gamma_{\mathfrak{u}} \left[\bar{\mathcal{L}}_{\mathfrak{u}}(P_n)\right]$$

$\mathfrak{u}$ set of coordinates $\rightarrow$ projection

$\gamma_{\mathfrak{u}}$ projection-dependent weights

$\bar{\mathcal{L}}_{\mathfrak{u}}(P_n)$ normalized maximum distance between two successive parallel hyperplanes

Weighted Figure of Merit

Weighted $\mathcal{P}_{2\alpha}$ Discrepancy With $\alpha = 1$

```
--merit sum:P2
```

$$\sum_{\emptyset \neq \mathfrak{u} \subseteq \{1,\ldots,s\}} \gamma_{\mathfrak{u}} \left[\frac{1}{n} \sum_{i=0}^{n-1} \prod_{j \in \mathfrak{q}} 2\pi^2 B_2(u_{i,j}) \right]$$

B_2 Bernoulli polynomial of the second degree

$u_{i,j}$ j-th coordinate of the i-th point in P_n

Weights

Product Weights

```
--weights product:0.1:1.0,0.7,0.3
```

$$\gamma_{\mathfrak{u}} = \prod_{j \in \mathfrak{u}} \gamma_j$$

$$\gamma_1 = 1.0$$
$$\gamma_2 = 0.7$$
$$\gamma_3 = 0.3$$
$$\gamma_4 = \cdots = \gamma_7 = 0.1$$

Construction Method

Component-By-Component (CBC)

```
--construction CBC
```

- Given the first $s-1$ components of $\boldsymbol{a}$, minimize the figure of merit with respect to the s-th component in

$$U_n = \{i \in \{1, \ldots, n-1\} \ : \ \gcd(i, n) = 1\}.$$

- Do so for $s = 1, \ldots, s_{\max}$.

Specialized Computation of the Figure of Merit

Specialized Computation of $\mathcal{P}_{2\alpha}$

```
--merit sum:CS-P2
```

algorithm	add s-th component
generic	$\mathcal{O}(2^s sn)$
proj.-dep.	$\mathcal{O}(2^s n)$
order-dep.	$\mathcal{O}(sn)$
product	$\mathcal{O}(sn)$
POD	

For product and order-dependent weights:

Cools, Kuo & Nuyens (2006), Nuyens & Cools (2006)

(Takes advantage of the symmetries in $\mathcal{D}^2$.)

Filters

Selective Random CBC

Perform a random CBC construction with 10 random samples per component of the generating vector, such that the normalized weighted $\mathcal{P}_2$ value is smaller than unity. Minimizes the *filtered*, *normalized* merit values.

```
latbuilder
    --lattice-size ordinary:1021
    --dimension 7
    --merit sum:CS-P2
    --weights product:0.1:1.0,0.7,0.3
    --construction random-CBC:10
    --filters norm:P2-DPW08 low-pass:1.0
```

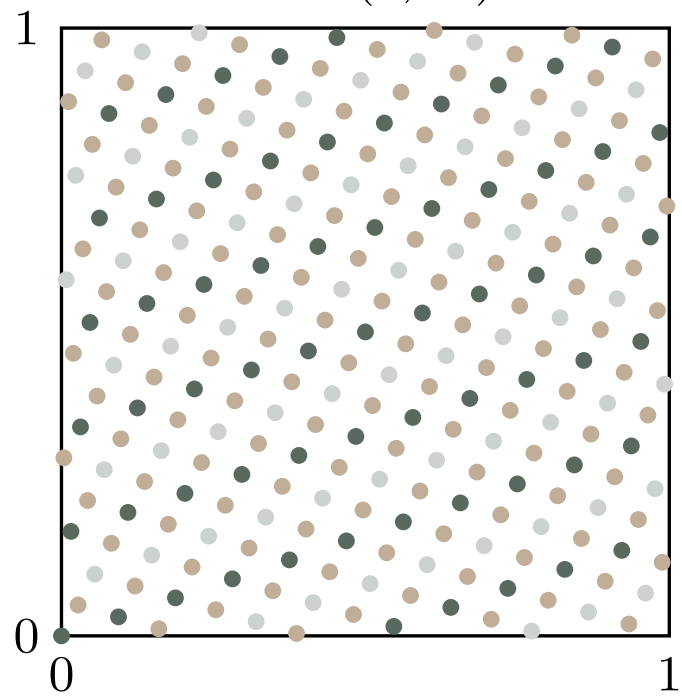
Normalization from Dick, Pillichshammer & Waterhouse (2008).

Embedded Lattices

Extending a Lattice

- Start with points for $n = 2^6 = 64$.
- Keep the same points; add $n = 2^6 = 64$ new points to obtain a new lattice rule with $n = 2^7 = 128$ points.
- Keep the old and the new points; add $n = 2^7 = 128$ new points to obtain a new lattice rule with $n = 2^8 = 256$ points.

$$n = 2^6 = 64,$$
$$n = 2^7 = 128,$$
$$n = 2^8 = 256,$$
$$\boldsymbol{a} = (1, 75)$$



Construction of Embedded Lattices

Fast CBC for Embedded Lattices

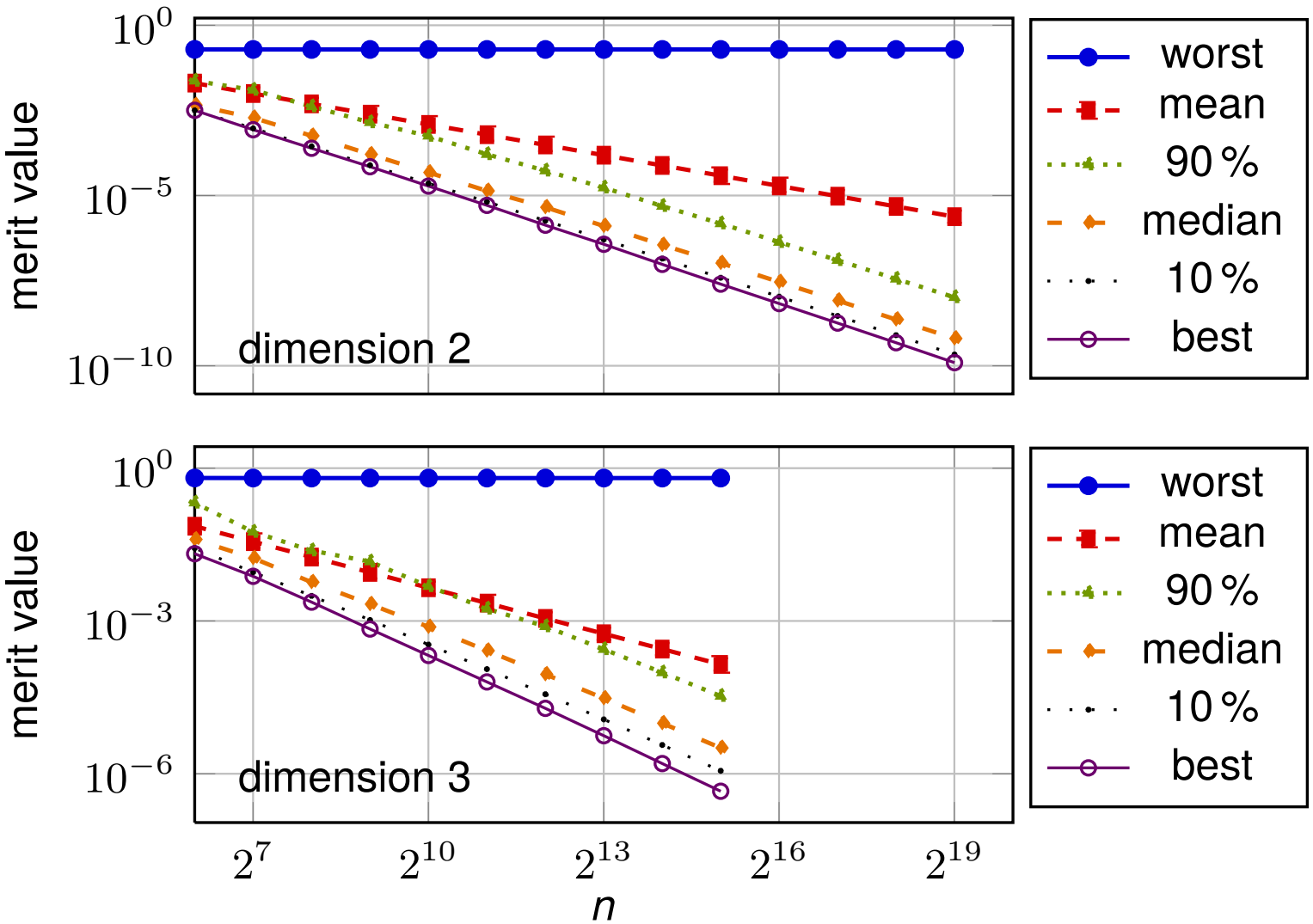
Perform a fast CBC construction for embedded lattices with a maximum of 2^{20} points, keeping only candidates for which the normalized weighted $\mathcal{P}_2$ value for each embedded level from $n = 2^8$ through 2^{20} is smaller than unity. A weighted sum of the normalized merit values for each level is minimized.

```
latbuilder
  --lattice-size embedded:2^20
  --dimension 7
  --merit sum:CS-P2
  --weights product:0.1:1.0,0.7,0.3
  --construction fast-CBC
  --filters norm:P2-DPW08:8:20 low-pass:1.0 combiner:sum
```

LatDef(2^20, [1, 444567, 292089, 375177, 58809, 241003, 36985]): 0.00347515

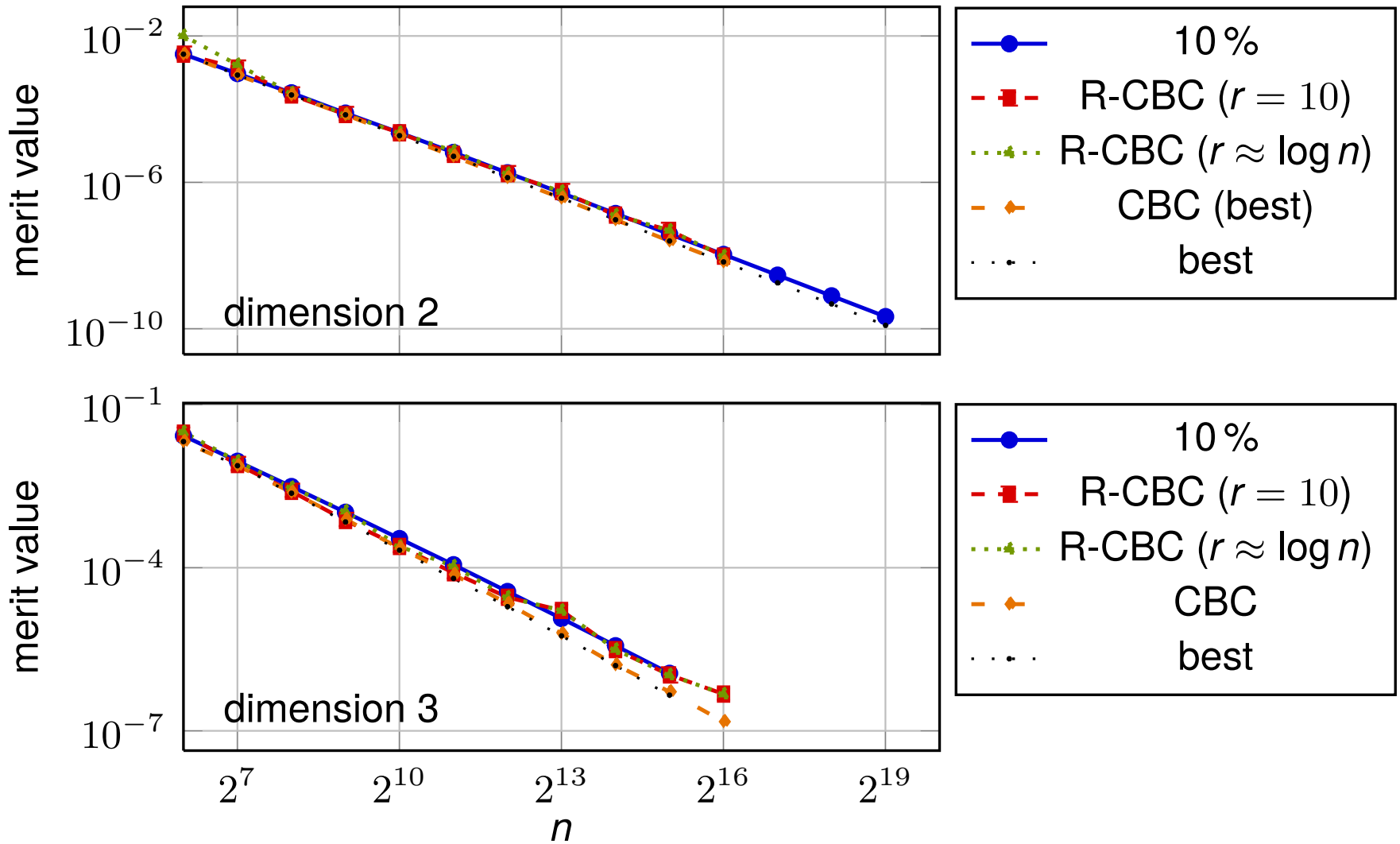
CPU time: 3.95632 seconds

Distribution of $\mathcal{P}_2$ Values



Quantiles for all lattices with components coprime with n.

How Good Is CBC?



CBC and random CBC.

Support Projection-Dependent Weights

Fast CBC

- Fast CBC
- Recursive evaluation of the figure of merit, e.g., Cools, Kuo & Nuyens (2006)
  - Reuse results from lower dimensions

Support Projection-Dependent Weights

Where recursive evaluation does not apply

- Sums over all projections $\rightarrow$ large number of terms
- Avoid complete evaluation of sums over all projections when possible

Flexibility with Performance

Challenges

- Support different:
  - Figures of merit
  - Types of weights
  - Construction methods
  - Lattice types
  - Normalizations
  - Filters
- Allow for extensions
- Maintain good performance
- Avoid duplication of code

Flexibility with Performance

Solutions

- Decoupled components:
  - Figures of merit
  - Types of weights
  - Construction methods
  - Lattice types
  - Normalizations
  - Filters
- Code generation through C++ templates:
  - Faster than polymorphism: displace work from runtime to compile time
  - Less restrictive than polymorphism, e.g. operations on scalar vs. vector data with the same syntax embedded)

Other Challenges

Fast CBC vs. Others

- **Fast CBC:** all merit values computed simultaneously
- **Others:** evaluate only when necessary
- **Solution:** hide computation in iterators to do only the necessary work

Other Challenges

- Enumerate integers coprime with n without repeating coprimality check
- Reduce search space when figure of merit symmetric under $a_j \mapsto n - a_j$
- Flexible assignment of the weights $\rightarrow$ parsers for weights
- Take advantage of CBC optimizations even with non-CBC

Conclusion

- Search for good lattice rules with specific input parameters for RQMC integration.
- Analyze and compare algorithms and figures of merit for RQMC research.
- What else are you interested in?