Numerical Algorithms **26:** 93–109, 2001.
© 2001 *Kluwer Academic Publishers. Printed in the Netherlands.*

Fast calculation of coefficients in the Smolyak algorithm

Knut Petras [*]
Institut für Angewandte Mathematik, Technische Universität Braunschweig, Pockelsstr. 14, D-38106 Braunschweig, Germany
E-mail: k.petras@tu-bs.de

Received 6 April 2000; revised 12 July 2000
Communicated by H. Woźniakowski

For many numerical problems involving smooth multivariate functions on d-cubes, the so-called Smolyak algorithm (or Boolean method, sparse grid method, etc.) has proved to be very useful. The final form of the algorithm (see equation (12) below) requires functional evaluation as well as the computation of coefficients. The latter can be done in different ways that may have considerable influence on the total cost of the algorithm. In this paper, we try to diminish this influence as far as possible. For example, we present an algorithm for the integration problem that reduces the time for the calculation and exposition of the coefficients in such a way that for increasing dimension, this time is small compared to dn, where n is the number of involved function values.

1. Introduction

Suppose, we have basic sequences $\big(Q_{L_k}^{(i)}\big)_{i\in\mathbb{N}}$ of approximation formulae (respectively functionals) for continuous linear functionals L_k on $C(S_k)$, $S_k \subset \mathbb{R}$. Important functionals are, e.g., given by

$$L_k[f] = \Phi_{x_0}[f] = f(x_0) \quad \text{or} \quad L_k[f] = I[f] = \int_0^1 f(x)\,\mathrm{d}x. \tag{1}$$

Then, the Smolyak algorithm [13] defines an approximation formula (cf. [15])

$$Q_L(q,d) = \sum_{q-d+1\leqslant|\mathbf{i}|\leqslant q} (-1)^{q-|\mathbf{i}|}\binom{d-1}{q-|\mathbf{i}|} Q_{L_1}^{(i_1)}\otimes\cdots\otimes Q_{L_d}^{(i_d)}, \quad \text{where } |\mathbf{i}| = \sum_{\nu=1}^{d} i_\nu, \tag{2}$$

for the tensor product

$$L = L_1\otimes\cdots\otimes L_d : C(S_1\times\cdots\times S_d) \to \mathbb{R}, \quad \text{where } L_k : C(S_k)\to\mathbb{R}. \tag{3}$$

This tensor product is the unique continuous linear functional satisfying

$$(L_1\otimes\cdots\otimes L_d)[f_1\cdot\,\cdots\,\cdot f_d] = L_1[f_1]\cdot\,\cdots\,\cdot L_d[f_d] \quad \text{for all } f_k \in C(S_k). \tag{4}$$

[*] The author is supported by a Heisenberg scholarship of the Deutsche Forschungsgemeinschaft.

Important corresponding examples are now

$$(\Phi_{x_1} \otimes \cdots \otimes \Phi_{x_d})[f] = f(x_1, \ldots, x_d) \tag{5}$$

or

$$I^d[f] = \int_{[0,1]^d} f(\mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{6}$$

Setting $Q_L^{(i)} := 0$ for $i \leqslant 0$ and $\Delta_L^{(i)} := Q_L^{(i)} - Q_L^{(i-1)}$ for all i, we have the alternative representation

$$Q_L(q, d) = \sum_{|\mathbf{i}| \leqslant q} \Delta_{L_1}^{(i_1)} \otimes \cdots \otimes \Delta_{L_d}^{(i_d)} \tag{7}$$

(cf. [15]). Continuous linear approximation functionals are of the form

$$Q_{L_k}^{(i)}[f] = \sum_{\mu=1}^{n_{i,k}} a_{\mu,k}^{(i)} l_{\mu,k}^{(i)}[f], \tag{8}$$

where the $l_{\mu,k}^{(i)}$ are continuous linear functionals on $C(S_k)$. In principle, we could omit $a_{\mu,k}^{(i)}$ and normalize $l_{\mu,k}^{(i)}$ accordingly. On the other hand, thinking of quadrature formulae, for example,

$$Q^{(i)}[f] = \sum_{\mu=1}^{n_i} a_\mu^{(i)} f\big(x_\mu^{(i)}\big), \tag{9}$$

which implies

$$\big(l_{\mu_1,1}^{(i_1)} \otimes \cdots \otimes l_{\mu_d,d}^{(i_d)}\big)[f] = f\big(x_{\mu_1,1}^{(i_1)}, \ldots, x_{\mu_d,d}^{(i_d)}\big), \tag{10}$$

notation (8) makes sense and is crucial for the following. Beside that, the $l_{\mu,k}^{(i)}$ often appear in several basic formulae $Q^{(i)}$ but with different factors. Let us, furthermore, write

$$\Delta^{(i)}[f] = \sum_{\nu=1}^{m_{i,k}} d_{\mu,k}^{(i)} \tilde{l}_{\mu,k}^{(i)}[f]. \tag{11}$$

The $\tilde{l}_{\mu,k}$ run through the union of functionals used by $Q^{(i)}$ and $Q^{(i-1)}$. We have $m_{i,k} = n_{i,k}$ and $\tilde{l}_{\mu,k}^{(i)} = l_{\mu,k}^{(i)}$ if the sequence of basic functionals is nested, i.e., if the functionals used by $Q^{(i-1)}$ are also used by $Q^{(i)}$. Nestedness usually reduces the costs considerably because we can reuse already calculated funcionals. We will, therefore, always assume that the basic sequence is nested (otherwise we could formally introduce zero coefficients in $Q^{(i)}$ for those functionals used by $Q^{(i-1)}$ but not by $Q^{(i)}$).

The Smolyak formulae may now be written as

$$Q_L(q, d)[f] = \sum_{\nu=1}^{n} a_\nu \cdot \Big(l_{\mu_1(\nu),1}^{(i_{\nu,1})} \otimes \cdots \otimes l_{\mu_d(\nu),d}^{(i_{\nu,d})}\Big)[f], \tag{12}$$

where a_ν is composed from all tensor products in (2), respectively (7), involving the functional $l^{(i_{\nu,1})}_{\mu_1(\nu),1} \otimes \cdots \otimes l^{(i_{\nu,d})}_{\mu_d(\nu),d}$. Here and in the following, let $\mathbf{i}_\nu$ be the minimal multi-index such that the corresponding tensor product formula contributes to a_ν, i.e., let $l^{(i_{\nu,1})}_{\mu_1(\nu),1} \otimes \cdots \otimes l^{(i_{\nu,d})}_{\mu_d(\nu),d}$ only appear in tensor products $Q^{(\tilde{i}_1)}_{L_1} \otimes \cdots \otimes Q^{(\tilde{i}_d)}_{L_d}$ with $\tilde{i}_k \geqslant i_k$ for all k. For $j_{\nu,k} \geqslant i_{\nu,k}$, let, furthermore, denote by $\tilde{\mu}_k(\nu)$ the index in $Q^{(j_{\nu,k})}$ (respectively in $\Delta^{(j_{\nu,k})}$) that corresponds to $l^{(i_{\nu,k})}_{\mu_k(\nu),k}$. Then, the corresponding coefficient is given by

$$a_\nu = \sum_{\substack{j_{\nu,k} \geqslant i_{\nu,k} \\ q-d+1 \leqslant |\mathbf{j}| \leqslant q}} (-1)^{q-|\mathbf{j}|} \binom{d-1}{q-|j|} a^{(j_{\nu,1})}_{\tilde{\mu}_1(\nu),1} \cdots a^{(j_{\nu,d})}_{\tilde{\mu}_d(\nu),d} \tag{13a}$$

$$= \sum_{\substack{j_{\nu,k} \geqslant i_{\nu,k} \\ |\mathbf{j}| \leqslant q}} d^{(j_{\nu,1})}_{\tilde{\mu}_1(\nu),1} \cdots d^{(j_{\nu,d})}_{\tilde{\mu}_d(\nu),d}. \tag{13b}$$

Of course, there is no way to avoid the evaluation of the functionals $l^{(i_{\nu,1})}_{\mu_1(\nu),1} \otimes \cdots \otimes l^{(i_{\nu,d})}_{\mu_d(\nu),d}$. However, there are different ways to provide the coefficients a_ν. The first one is to store $a_1, \ldots, a_n$ but in many (in particular higher dimensional) applications, n is very large, such that storing and reading is costly. Furthermore, q and d may vary from application to application, which almost forbids storing. The second possibility is to calculate a_ν via equations (13a,b). This takes a considerable time (see the numerical examples below) and existing bounds for realistic basic sequences $(Q^{(i)}_L)_{i\in\mathbb{N}}$ allow the costs to be notably larger than dn (see [7]), which is the total number of univariate functional evaluations (where each factor of a tensor product functional counts, since usually, all d variables are involved in the computation of a product functional if the function itself depends on all variables). We therefore rewrite equations (13a,b) in section 2 and give cost bounds for the resulting algorithm. In section 3, we describe simplifications in the case that the L_j as well as the $Q^{(i)}_{L_j}$ are equal for all j. This yields a further reduction of the costs. Section 4 gives some numerical examples that compare the costs of the different algorithms.

2. Divide and conquer

A method that, if applicable, often reduces the computational cost is "divide and conquer" (or "divide et impera"), see, e.g., [4, p. 45]. We have two representations of the coefficients of $Q(q,d)$. For rewriting the first one (equation (13a)), we introduce the notation

$$a^{=}_\nu\big(r, s, \alpha, (i_{\nu,r}, \ldots, i_{\nu,s})\big) = \sum_{\substack{j_{\nu,k} \geqslant i_{\nu,k},\ k=r,\ldots,s \\ |(j_{\nu,r},\ldots,j_{\nu,s})|=\alpha}} a^{(j_{\nu,r})}_{\tilde{\mu}_r(\nu),r} \cdots a^{(j_{\nu,s})}_{\tilde{\mu}_s(\nu),s} \tag{14}$$

and

$$a_\nu^{\leqslant}\bigl(r, s, \alpha, (i_{\nu,r}, \dots, i_{\nu,s})\bigr) = \sum_{\substack{j_{\nu,k} \geqslant i_{\nu,k},\ k=r,\dots,s \\ \alpha-d+1 \leqslant |(j_{\nu,r},\dots,j_{\nu,s})| \leqslant \alpha}} (-1)^{\alpha-|(j_{\nu,r},\dots,j_{\nu,s})|} \times \binom{d-1}{\alpha - |(j_{\nu,r},\dots,j_{\nu,s})|} a^{(j_{\nu,r})}_{\tilde\mu_r(\nu),r} \cdots a^{(j_{\nu,s})}_{\tilde\mu_s(\nu),s}. \tag{15}$$

Then,

$$a_\nu = a_\nu^{\leqslant}(1, d, q, \mathbf{i}). \tag{16}$$

Analogously, to rewrite (13b), define

$$d_\nu^{=}\bigl(r, s, \alpha, (i_{\nu,r}, \dots, i_{\nu,s})\bigr) = \sum_{\substack{j_{\nu,k} \geqslant i_{\nu,k},\ k=r,\dots,s \\ |(j_{\nu,r},\dots,j_{\nu,s})| = \alpha}} d^{(j_{\nu,r})}_{\tilde\mu_r(\nu),r} \cdots d^{(j_{\nu,s})}_{\tilde\mu_s(\nu),s} \tag{17}$$

and

$$d_\nu^{\leqslant}\bigl(r, s, \alpha, (i_{\nu,r}, \dots, i_{\nu,s})\bigr) = \sum_{\substack{j_{\nu,k} \geqslant i_{\nu,k},\ k=r,\dots,s \\ |(j_{\nu,r},\dots,j_{\nu,s})| \leqslant \alpha}} d^{(j_{\nu,r})}_{\tilde\mu_r(\nu),r} \cdots d^{(j_{\nu,s})}_{\tilde\mu_s(\nu),s}. \tag{18}$$

Then,

$$a_\nu = d_\nu^{\leqslant}(1, d, q, \mathbf{i}). \tag{19}$$

For $s \leqslant t \leqslant r-1$, we obviously have the recursion formulae (here written for representation (19) – they look identically for representation (16))

$$\begin{aligned} &d_\nu^{\leqslant}\bigl(r, s, \alpha, (i_{\nu,r}, \dots, i_{\nu,s})\bigr) \\ &\quad= \sum_{\mu=0}^{\alpha-|(i_{\nu,r},\dots,i_{\nu,s})|} d_\nu^{=}\bigl(r, t, \bigl|(i_{\nu,r}, \dots, i_{\nu,t})\bigr| + \mu, (i_{\nu,r}, \dots, i_{\nu,t})\bigr) \\ &\qquad \times d_\nu^{\leqslant}\bigl(t+1, s, \alpha - \bigl|(i_{\nu,r}, \dots, i_{\nu,t})\bigr| - \mu, (i_{\nu,t+1}, \dots, i_{\nu,s})\bigr) \end{aligned} \tag{20}$$

as well as

$$\begin{aligned} &d_\nu^{=}\bigl(r, s, \alpha, (i_{\nu,r}, \dots, i_{\nu,s})\bigr) \\ &\quad= \sum_{\mu=0}^{\alpha-|(i_{\nu,r},\dots,i_{\nu,s})|} d_\nu^{=}\bigl(r, t, \bigl|(i_{\nu,r}, \dots, i_{\nu,t})\bigr| + \mu, (i_{\nu,r}, \dots, i_{\nu,t})\bigr) \\ &\qquad \times d_\nu^{=}\bigl(t+1, s, \alpha - \bigl|(i_{\nu,r}, \dots, i_{\nu,t})\bigr| - \mu, (i_{\nu,t+1}, \dots, i_{\nu,s})\bigr). \end{aligned} \tag{21}$$

We apply them with $t = \lceil (r+s)/2 \rceil$ until we have sums with one summation index respectively.

There is almost no difference in applying (16) or (19) together with the recursion. However, we prefer the slightly simpler representation (13b), i.e., (19). All theoretical results are also valid for the other one.

Lemma 1. Let $\mathrm{cost}(q, d, \mathbf{i})$ denote the number of floating point operations (multiplications and additions) that are necessary to compute the coefficient a_ν of

$$\Big(l^{(i_{\nu,1})}_{\mu_1(\nu),1} \otimes \cdots \otimes l^{(i_{\nu,d})}_{\mu_d(\nu),d}\Big)[f] \tag{22}$$

in representation (12). Let, furthermore, $p \in \mathbb{N}$ be so chosen that

$$2^{p-1} < d \leqslant 2^p \tag{23}$$

and suppose that all sums

$$d^{\leqslant}(s, s, j_{\nu,s}, i_{\nu,s}) = \sum_{\tau=i_{\nu,s}}^{j_{\nu,s}} d^{(j_{\tau,s})}_{\tilde{\mu}_s(\tau),s} = a^{(j_{\nu,s})}_{\tilde{\mu}_s(\nu),s} - a^{(i_{\nu,s})}_{\mu_s(\nu),s} \tag{24}$$

(the last equality holds since $d^{(i_{\nu,s})}_{\mu_s(\nu),s} = a^{(i_{\nu,s})}_{\mu_s(\nu),s}$) for $j_{\nu,s} > i_{\nu,s}$ are precomputed. Then, defining $\Theta := q - |\mathbf{i}|$, recursions (20) and (21) applied to representation (19) yield

$$\mathrm{cost}(q, d, \mathbf{i}) = \begin{cases} 2\Theta + 1 & \text{for } d = 2, \\ 2(\Theta+1)^2 & \text{for } d = 3, \\ 2\Theta^2 + 6\Theta + 3 & \text{for } d = 4 \quad \text{and} \end{cases}$$
$$\mathrm{cost}(q, d, \mathbf{i}) \leqslant \frac{2^p}{p!}\left(\Theta + \frac{p+1}{2}\right)^p - 2 \quad \text{for } d = 2^{p-1}+1, \dots, 2^p, \text{ where } p \geqslant 3. \tag{25}$$

For fixed Θ and increasing d, we have

$$\mathrm{cost}(q, d, \mathbf{i}) \leqslant 2^p\left[\binom{p}{\Theta} + \mathrm{O}\big(p^{\Theta-1}\big)\right]. \tag{26}$$

Proof. Since the cost is monotonically increasing with increasing dimension, we have to prove the Lemma only for $d = 2^p$ and $d = 3$. Furthermore, the cost depends only on d and Θ. Let, therefore,

$$c_p(\Theta) = \mathrm{cost}(q, d, \mathbf{i}), \quad \text{where } d = 2^p \text{ and } \Theta = q - |\mathbf{i}|. \tag{27}$$

By formulas (13b), and (19)–(21), we see that

$$c_1(\Theta) = 2\Theta + 1 \tag{28}$$

and

$$c_{p+1}(\Theta) = 2\Theta + 1 + \sum_{\nu=0}^{\Theta}\big(c_p(\nu) + c_p(\Theta-\nu)\big) = 2\Theta + 1 + 2\sum_{\nu=0}^{\Theta} c_p(\nu), \tag{29}$$

which immediately gives the case $p = 2$, i.e., $d = 4$ in the lemma. The explicit calculation of $\mathrm{cost}(q, 3, \mathbf{i})$ is similar. Furthermore, equation (29) yields

$$c_3(\Theta) = \frac{2^3}{3!}(\Theta + 2)^3 - 2 - \frac{4\Theta + 5}{3}. \tag{30}$$

The induction over $p \geqslant 3$ is done as follows,

$$\begin{aligned} c_{p+1}(\Theta) &\leqslant 2\Theta + 1 - 2(\Theta + 1) + 2 \cdot \frac{2^p}{p!} \sum_{\nu=0}^{\Theta} \left(\nu + \frac{p+1}{2}\right)^p \\ &\leqslant -1 + \frac{2^{p+1}}{p!} \int_{-1/2}^{\Theta+1/2} \left(x + \frac{p+1}{2}\right)^p \mathrm{d}x \\ &\leqslant -1 + \frac{2^{p+1}}{(p+1)!}\left(\Theta + \frac{p+2}{2}\right)^{p+1} - \underbrace{\frac{2^{p+1}}{(p+1)!}\left(\frac{p}{2}\right)^{p+1}}_{\geqslant 1}. \end{aligned} \tag{31}$$

In order to prove the estimates for fixed Θ, we consider the generating function

$$f_\Theta(x) = \sum_{p=0}^{\infty} c_p(\Theta) x^p. \tag{32}$$

Using $c_0(\Theta) = 0$, equation (29) gives

$$\begin{aligned} \frac{1}{x} f_\Theta(x) = \sum_{p=0}^{\infty} c_{p+1}(\Theta) x^p &= 2 \sum_{p=0}^{\infty} \sum_{\nu=0}^{\Theta} c_p(\nu) x^p + (2\Theta + 1) \sum_{p=0}^{\infty} x^p \\ &= 2 \sum_{\nu=0}^{\Theta} f_\nu(x) + \frac{2\Theta + 1}{1 - x}, \end{aligned} \tag{33}$$

i.e.,

$$f_\Theta(x) = \sum_{\nu=0}^{\Theta-1} \frac{2x f_\nu(x)}{1 - 2x} + \frac{(2\Theta + 1)x}{(1 - 2x)(1 - x)}. \tag{34}$$

We need $d - 1$ multiplications in the case $\Theta = 0$, i.e., $c_p(0) = 2^p - 1$ and therefore

$$f_0(x) = \frac{1}{1 - 2x} - \frac{1}{1 - x}. \tag{35}$$

Hence, equation (34) gives by induction the representation

$$f_\Theta(x) = \frac{(2x)^\Theta}{(1 - 2x)^{\Theta+1}} + \frac{P_\Theta(x)}{(1 - 2x)^\Theta (1 - x)}, \tag{36}$$

where P_Θ is a polynomial. The coefficient of x^p, $p \geqslant \Theta$, of the first summand is $2^p\binom{p}{\Theta}$, while the respective coefficient of the second summand is of order $\mathrm{O}(2^p p^{\Theta-1})$. $\square$

To determine the cost $\mathrm{cost}(q,d)$ of calculating all $n = n(q,d)$ coefficients a_ν, we have to count the number of coefficients corresponding to a certain Θ. This is obviously $n(q-\Theta,d) - n(q-\Theta-1,d)$.

For given i, let now the numbers $n_{i,k}$ be the same n_i for each dimension k, i.e., ith elements of the basic sequences in the different dimensions are based on the same number of functionals. This usually holds, if all dimensions are of the same importance. Then, we consider two different cases, large d and moderate $q-d$ and vice versa. Set $r = q-d$.

First, we fix r and consider $d \to \infty$. We have proved in [11] that, if $n_2 > n_1$, i.e., if $Q^{(2)}$ uses additional information compared with $Q^{(1)}$,

$$n(d+r,d) = n_1^d \sum_{j=0}^{r} c_{j,r} \binom{d}{j} \tag{37}$$

with certain positive constants $c_{j,r}$ being independent of d. Therefore,

$$n(d+r,d) - n(d+r-1,d) \sim n_1^d \cdot d^r, \tag{38}$$

(the notation $a_n \sim b_n$ means that there exist constants $c, C > 0$ such that $cb_n \leqslant a_n \leqslant Cb_n$) and thus, using the notation of the proof of lemma 1 (and $n(d-1,d) := 0$),

$$\begin{aligned} \mathrm{cost}(d+r,d) &\leqslant \sum_{\nu=0}^{r} c_p(r-\nu)\big[n(d+\nu,d) - n(d+\nu-1,d)\big] \\ &\leqslant c_p(0)n(d+r,d) + \mathrm{O}\Bigg(\sum_{\nu=0}^{r-1} c_p(r-\nu)d^\nu\Bigg) \\ &= (d-1)n(d+r,d) + \mathrm{O}\Bigg(\sum_{\nu=0}^{r-1} 2^p p^{r-\nu} d^\nu\Bigg). \end{aligned} \tag{39}$$

Since

$$p = \lceil \log_2 d \rceil \leqslant \log 2d + 1, \tag{40}$$

we can sumarize:

Corollary 1. For $n_{i,k} = n_i$, $k = 1,\dots,d$, and $n_2 > n_1$ as well as for fixed r and increasing d, we have

$$\mathrm{cost}(d+r,d) \leqslant d \cdot n(d+r,d) \cdot \big(1 + \mathrm{O}(\ln d/d)\big). \tag{41}$$

If each variable contributes to the calculation of each product functional in equation (12) even with only one elementary operation, the corollary means that the calculation of coefficients is asymptotically not more expensive than the remaining cost of the algorithm. The relative expenses for the calculation of coefficients of course decrease, if the function becomes more complicated, i.e., if the influence of the variables on the calculation time for a functional evaluation increases.

Let now d be fixed and r increasing. In numerical cubature, it seems that we have typically

$$n(d+r,d) \sim \alpha^r (r+1)^{d+\beta}, \tag{42}$$

where $\alpha \geqslant 1$ and β are constants (see [5,7]). Lemma 1 gives the cost bound

$$\operatorname{cost}(d+r,d,\mathbf{i}) = \mathrm{O}\bigl((\Theta+1)^p\bigr) \tag{43}$$

(see equation (25)) and therefore

$$\begin{aligned}\operatorname{cost}(d+r,d) &= \alpha^r r^{d+\beta}\mathrm{O}\left(1+\sum_{\nu=1}^{r}\frac{(\nu+1)^p}{\alpha^\nu}\left(1-\frac{\nu}{r+1}\right)^d\right)\\ &= n(d+r,d)\cdot\mathrm{O}\left(1+\int_0^r\frac{(x+1)^p}{\alpha^x}\left(1-\frac{x}{r+1}\right)^d \mathrm{d}x\right)\\ &= n(d+r,d)\cdot\mathrm{O}\left(1+\int_0^r\frac{1+x^p}{\alpha^x}\,\mathrm{d}x\right).\end{aligned} \tag{44}$$

Corollary 2. For $n_{i,k} = n_i$, $k = 1,\dots,d$, fixed d and increasing r, we have

$$\operatorname{cost}(d+r,d) \leqslant n(d+r,d)\cdot\begin{cases}\mathrm{O}(1) & \text{if } \alpha > 1\\ \mathrm{O}\bigl(r^{1+p}\bigr) & \text{if } \alpha = 1,\end{cases} \quad \text{where } p = \lceil \log_2 d\rceil. \tag{45}$$

We can say that there are situations, where the total cost is proportional to the number of one-dimensional functional evaluations (the application of a tensor product of d one-dimensional functionals counts d evaluations). This happens for fixed r, increasing dimension d and arbitrary basic sequences or for fixed d, increasing r if the number of functionals in the basic sequence increases exponentially (e.g., the Clenshaw–Curtis–Smolyak method described and analyzed in [6] or the Kronrod–Patterson–Smolyak method described and analyzed in [2,3,10]). On the other hand, we cannot prove proportionality for fixed d and increasing r, if the number of functionals in the basic sequence increase only polynomially (e.g., the Smolyak method in [7, section 6] or the delayed Smolyak method in [11]).

3. Power functionals – the coefficient tree

If the problem is a power of one functional on $C(S)$, e.g.,

$$L = I^d, \quad \text{i.e.,} \quad L[f] = \int_{[0,1]^d} f(\mathbf{x})\,\mathrm{d}\mathbf{x}, \tag{46}$$

then, we often use the same basic sequence in each dimension. This implies that many of the coefficients a_ν in (12) are the same. To see this, let $\{l_0,\dots,l_s\}$ be the set of all univariate functionals that occur on the right-hand side of (12) for any **i** and that contribute to $Q(q,d)$. Then, the coefficients a_ν in (12) corresponding to a product of functionals that involve

$$k_0 \times l_0, \quad k_1 \times l_1, \quad \ldots, \quad k_s \times l_s \tag{47}$$

with fixed $k_0, \ldots, k_s$ are equal. We have $\sum_{k_i} = d$. For example, if $l_i[g] = g(x_i)$, then the coefficient of $f(x_0, x_0, x_0, x_1, x_1, x_2)$ is, by symmetry of the dimensions, the same as that of $f(x_1, x_0, x_2, x_0, x_0, x_1)$ (in this example, $k_0 = 3$, $k_1 = 2$, $k_2 = 1$, $k_3 = k_4 = \cdots = 0$).

Remark 1. If we have to approximate a power of a symmetric functional, i.e., if there is a number c such that

$$L_j[f] = L_j[g] \tag{48}$$

whenever $f(x) = g(c-x)$ (respectively whenever $f(x) = -g(c-x)$), then, also the elements of the basic sequence of one-dimensional approximation formulae are often symmetric in the same sense. This implies that there is an ordering and normalization of the functionals of the sum (12) such that

$$l^{(i)}_{\mu,k}[f] = l^{(i)}_{n_i+1-\mu,k}[g] \quad \text{and} \quad a^{(i)}_{\mu,k} = a^{(i)}_{n_i+1-\mu,k} \quad \text{(respectively } a^{(i)}_{\mu,k} = -a^{(i)}_{n_i+1-\mu,k}\text{)}. \tag{49}$$

Examples are symmetric quadrature formulae (such as, e.g., midpoint, trapezoidal, Simpson, Clenshaw–Curtis, Gauss, Gauss–Kronrod, or Kronrod–Patterson formulae for symmetric weight functions). Then, by an appropriate ordering, we have to consider only the coefficients of functionals involving $\{l_0, \ldots, l_{\lfloor s/2 \rfloor}\}$. This leads to a further reduction. It seems that only this reduction and not that described above has already been used in former algorithms (see, e.g., [2,3,6]).

Now, the necessary coefficients can be stored in a tree. To avoid superfluous loops, this tree is not calculated at the beginning but generated dynamically during the calculation of the Smolyak formula. When a coefficient is needed, we try to find it in the tree. If the search path exists, then the coefficient had been calculated before. If not, the search path is built and the coefficient is calculated and stored at the correct position. We still have to describe the structure of the tree. We recommend two different ways to organize the tree depending on whether s is SMALL or LARGE. (E.g., for $Q(d+10, d)$, we have $s = 1025$ for Smolyak cubature if based on Clenshaw–Curtis formulae and $s = 15$ if based on the delay Smolyak formulae; see below.)

SMALL. In order to find a coefficient that corresponds to a tuple $(k_0, \ldots, k_s)$ as described in (46), we have to go the following way through the binary tree,

- k_0 steps to the left and one step to the right,

- then k_1 steps to the left and one step to the right,

  $\vdots$

- We can stop after d steps to the left. The current node contains the coefficient.

The 'left'-directions are organized using arrays of pointers. The k_ith entry in the array corresponds to k_i steps to the left. Then, we can do k_i steps at once. This procedure is relatively simple and also the use of it is easy. For each request for a coefficient, we have to calculate the k_i and then search in the tree. Each search costs at most $d + s$ steps through the tree as well as d operations for determining $(k_0, \dots, k_s)$. The cost $s + 2d$ is exactly the problem for large s.

LARGE. For each search, we have to know the subset $\{l_{\nu_1}, \dots l_{\nu_m}\}$ of $\{l_0, \dots, l_s\}$, ordered by $\nu_i < \nu_{i+1}$, whose elements are involved in the current summand. The corresponding coefficient can be found along the following path:

- ν_1 steps to the left and k_{ν_1} steps to the right,
- $\nu_2 - \nu_1$ steps to the left and k_{ν_2} steps to the right,

  $\vdots$

- $\nu_m - \nu_{m-1}$ steps to the left and k_{ν_m} steps to the right.

Now the steps in both directions are organized via arrays of pointers. After the first i items, the length of the allocated array for the left direction must have (at most) $s - (\nu_1 + \cdots + \nu_i)$ entries and the allocated array for the right direction $d - (k_{\nu_1} + \cdots + k_{\nu_m})$ entries. Then, we have to perform $2m \leqslant 2\min(r, d, s)$ steps in the tree to find a coefficient of $Q(d + r, d)$.

Remark 2. In particular, for high dimension it is important to have $n_1 = 1$. Otherwise, the cost of the Smolyak algorithm grows exponentially with d. Let the functional used by $Q^{(1)}$ be l_s. Then, it is obvious that $k_s = d - (k_0 + \cdots + k_{s-1})$ and we do not have to perform the last item in LARGE if $\nu_m = s$ but we can store the coefficient at the current node in the tree before the last step. If we have to calculate $Q(d + r, d)$ with $d \gg r$, we have always a large $k_s \geqslant d - r$. In this situation, we want to build the ordered set $\{l_{\nu_1}, \dots, l_{\nu_m}\}$ for the search of a coefficient during the selection of the corresponding node. We have to count each occurrence of a functional different from l_s. These are at most r operations. The calculation of k_s also needs at most r operations. Finally, we have to order the $m + 1 \leqslant r + 1$ occurring functionals, which requires $\mathrm{O}(r \ln r)$ operations. Calculation of coefficients is done for all combinations of $k_0, \dots, k_{s-1}$. Since the sum of these numbers is at most r, we have the same number of different coefficients for all $d > r$, i.e., for all $d > r$, we have the same number of required coefficient calculations for $Q(d + r, d)$ as for $Q(2r, r)$. Now, let cost_f be the cost of all functional evaluations

Table 1
Numbers of function respectively coefficient calculations.

r	$n^{\mathrm{CC}}(2r,r)$	$n_s^{\mathrm{CC}}(2r,r)$	$n_t^{\mathrm{CC}}(2r,r)$	$n^{\mathrm{del}}(2r,r)$	$n_s^{\mathrm{del}}(2r,r)$	$n_t^{\mathrm{del}}(2r,r)$
2	13	6	4	9	4	3
3	69	23	8	39	14	6
4	401	98	17	193	48	9
5	2433	437	36	903	142	12
6	15121	1995	79	4161	460	22
7	95441	9242	172	19855	1626	32
8	609025	43258	379	97153	5568	42
9	3918273	204053	832	477327	18770	64
10	25370753	968441	1831	2347809	64804	86

and cost_e the extra cost (coefficient calculation, memory allocation, summation in (12), etc.). We have for fixed r, increasing d and the algorithm LARGE

$$\begin{aligned}\mathrm{cost}_e(d+r,d) &\leqslant \mathrm{O}\big(\mathrm{cost}_e(2r,r)+(r\ln r)\cdot n(d+r,d)\big)\\ &= \mathrm{o}\big(d\cdot n(d+r,d)\big)=\mathrm{o}\big(\mathrm{cost}_f(d+r,d)\big).\end{aligned} \tag{50}$$

This relation means that the relative overhead by coefficient calculations tends to zero for increasing dimension.

According to remark 2 we want to illustrate in table 1 for an example from numerical cubature, how many coefficients have to be calculated in the formula $Q(2r,r)$. We introduce the notation

- $n_s(q,d)$: number of coefficient calculations if we use only the symmetries mentioned in remark 1;
- $n_t(q,d)$: (upper bound for the) number of different coefficients, i.e., number of coefficients that are calculated in the tree versions.

As one-dimensional basic sequences, we choose (as in the numerical examples below) the Clenshaw–Curtis sequence with exponentially increasing numbers of nodes (see [1,6]; we denote the corresponding numbers by the upper index CC) and the Kronrod–Patterson sequence [9], delayed and thereore with linearly increasing numbers of nodes (see [11]; denoted by the upper index del).

4. Numerical tests

In our tests, we compare the total computation time with the time that has been used without calculation of the coefficients. The different methods are USUAL (equations (13a,b)), DAC (Divide And Conquer, see equations (20) and (21)) and TREE (Coefficient Tree, see section 3). The test examples for the calculation of the integral over the d-dimensional unit cube are for $d=6, 12, 18, 24$ and 360.

As at the end of the previous section, we use one basic sequence with exponential increase of the numbers of nodes, namely the Clenshaw–Curtis sequence for dimensions 6, 12 and 18, and one with linear increase, the delayed Kronrod–Patterson sequence for dimensions 6, 12 and 24, where the corresponding Smolyak formulae are denoted by Q^{CC} and Q^{del}, respectively.

In dimensions $d \leqslant 24$, we take the sample function

$$f(t) = \exp\left(-\sum_{i=1}^{d} c_i^2 (t_i - w_i)^2\right) \tag{51}$$

with certain (randomly chosen) constants c_i and w_i. This function has been taken several times for test purposes (see [2,3,6,10–12] and the references therein).

In the following graphs, we illustrate for the different methods the ratio

$$\frac{\mathrm{tc}(d+r,d)}{\mathrm{pc}(d+r,d)}, \tag{52}$$

where tc denotes the total computation time, while pc denotes the partial computation time without calculation of the coefficients. For comparison, we additionally draw a line at ratio 1, respectively. Therefore, the computational overhead of the respective method is represented by the grey area above this line (compared with the white area under the line).

We see that divide and conquer does not help for low dimnesion $d = 6$ if the numbers of nodes of the basic sequence increase slow (cf. figures 1(d) and (e)) in all other examples, we see that TREE is faster than DAC, which is again itself faster than USUAL. The extreme case can be seen in figures 3(a) and (c), where for the computation of $Q^{\mathrm{CC}}(25, 18)$, TREE is more than 35 times faster than USUAL.

Example from finance, $d = 360$

The high dimensional example is from finance and is described, for example, in [14]. The Smolyak type method is defined in [11] and is similar to method proposed by Novak et al. [8]. Characteristic for the applied method $\widetilde{Q}^{\mathrm{del}}$ is that, due to the structure of the integrand, we use differently fast increasing sequences from the same set of quadrature formulae in different dimensions. As a basic sequence, we choose again the delayed Kronrod–Patterson sequence.

For $r = 7$, the program using TREE is more than 10 times faster then the USUAL program. On a 400 Mhz Pentium processor, TREE requires 2 min CPU time, while USUAL requires about 20 min. Comparing only the calculation of the coefficients in this case, TREE is more than 335 times faster than USUAL.

We see that the calculation of coefficients can be almost of no influence on the total computation time if it is done in an efficient way, but with the commonly used technique USUAL, the computer can spend almost all the time for the calculation of coefficients.

d=6

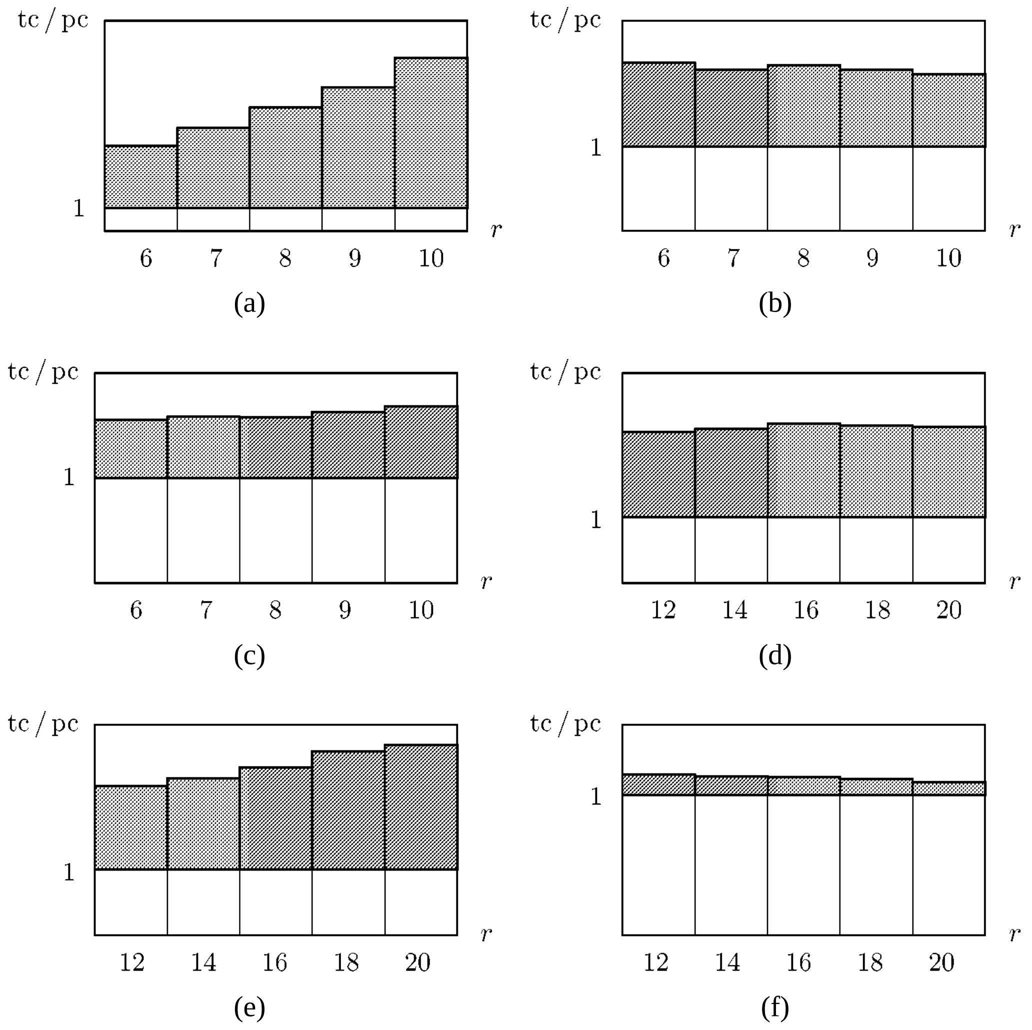

Figure 1. (a) $Q^{\mathrm{CC}}(6+r, 6)$, USUAL; (b) $Q^{\mathrm{CC}}(6+r, 6)$, DAC; (c) $Q^{\mathrm{CC}}(6+r, 6)$, TREE; (d) $Q^{\mathrm{del}}(6+r, 6)$, USUAL; (e) $Q^{\mathrm{del}}(6+r, 6)$, DAC; (f) $Q^{\mathrm{del}}(6+r, 6)$, TREE.

The programs are available via internet on the site `http://www-public.tu-bs.de:8080/~petras/software.html`

Acknowledgement

I would like to thank Erich Novak and Klaus Ritter for stimulating discussions during the preparation of this paper.

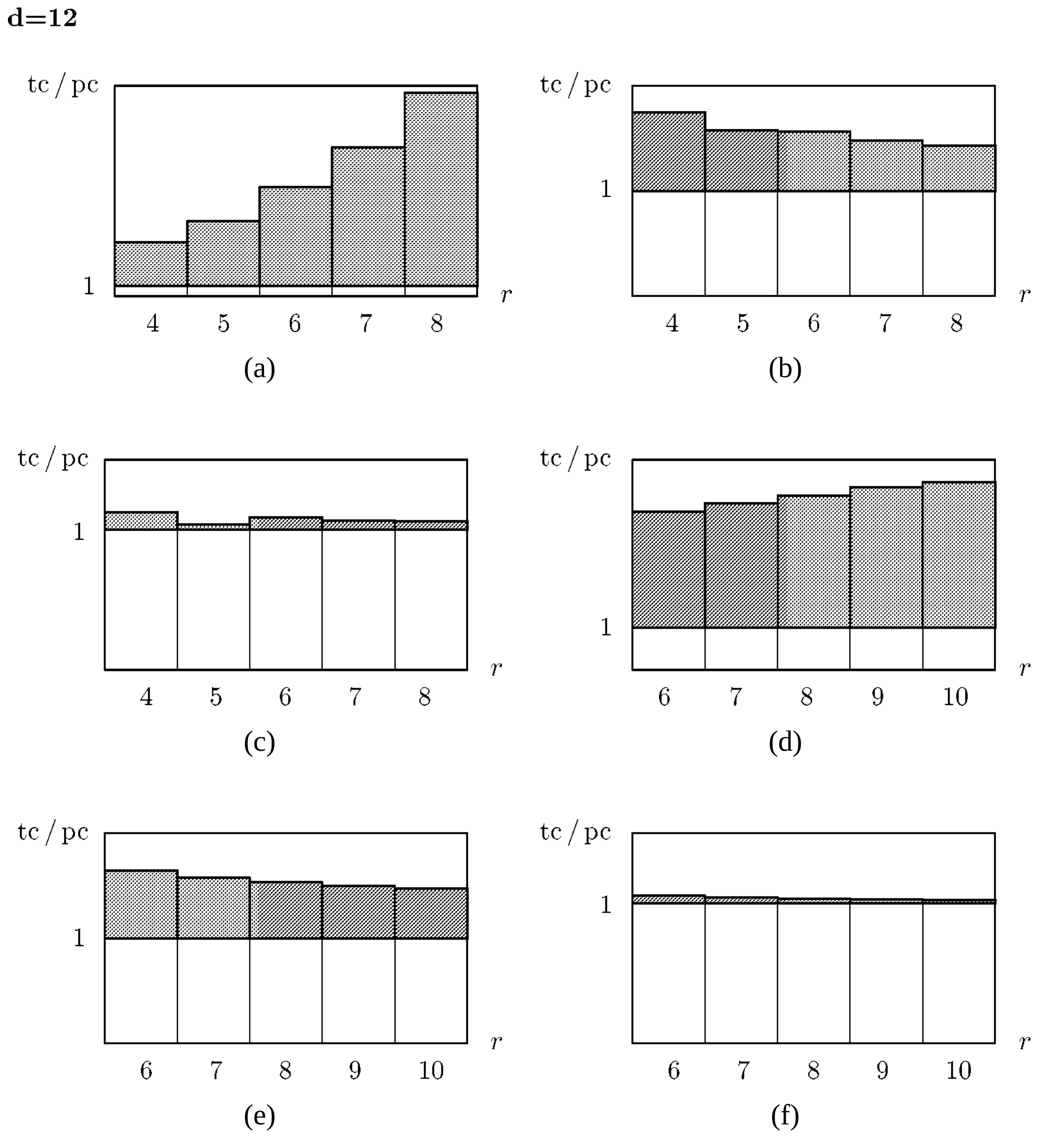

Figure 2. (a) $Q^{\text{CC}}(12 + r, 12)$, USUAL; (b) $Q^{\text{CC}}(12 + r, 12)$, DAC; (c) $Q^{\text{CC}}(12 + r, 12)$, TREE; (d) $Q^{\text{del}}(12 + r, 12)$, USUAL; (e) $Q^{\text{del}}(12 + r, 12)$, DAC; (f) $Q^{\text{del}}(12 + r, 12)$, TREE.

d=18, d=24

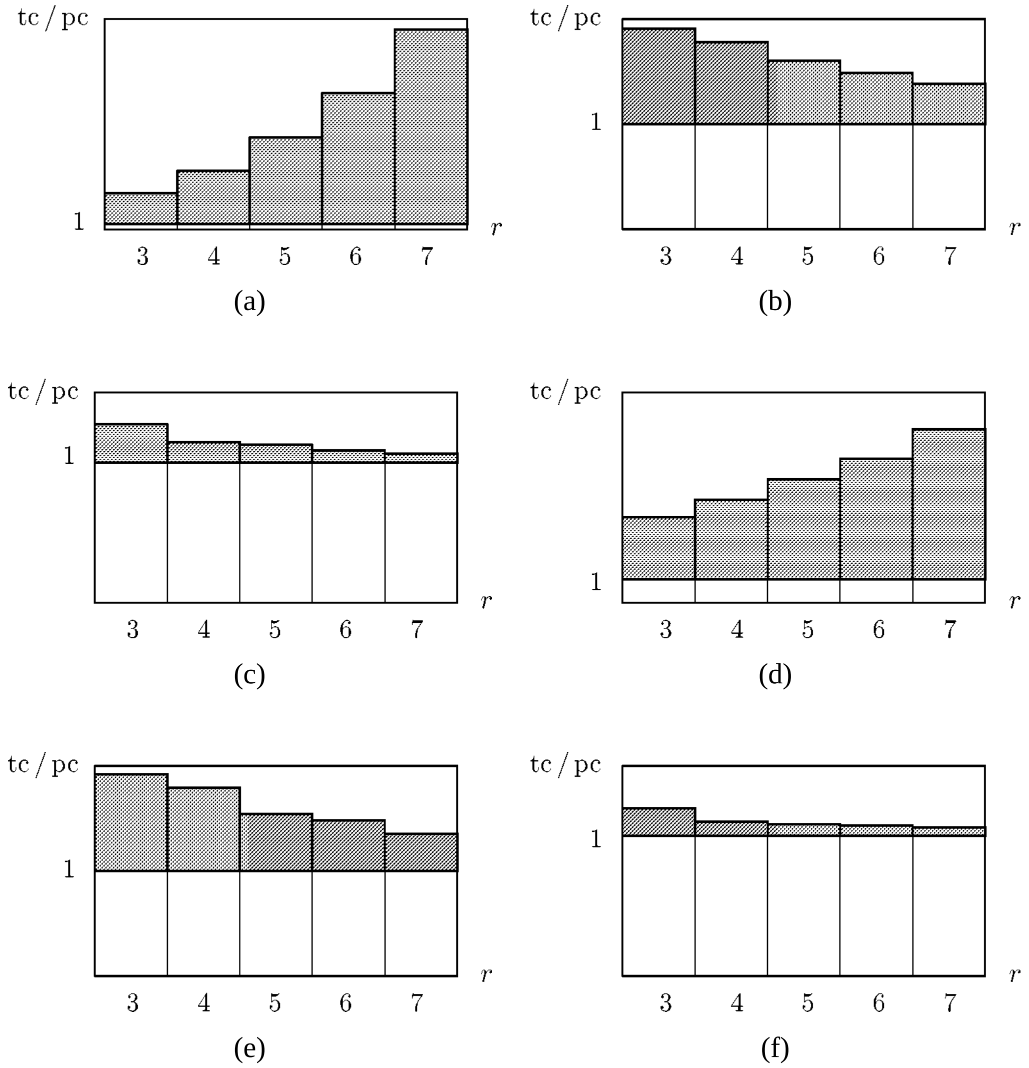

Figure 3. (a) $Q^{\mathrm{CC}}(18 + r, 18)$, USUAL; (b) $Q^{\mathrm{CC}}(18 + r, 18)$, DAC; (c) $Q^{\mathrm{CC}}(18 + r, 18)$, TREE; (d) $Q^{\mathrm{del}}(24 + r, 24)$, USUAL; (e) $Q^{\mathrm{del}}(24 + r, 24)$, DAC; (f) $Q^{\mathrm{del}}(24 + r, 24)$, TREE.

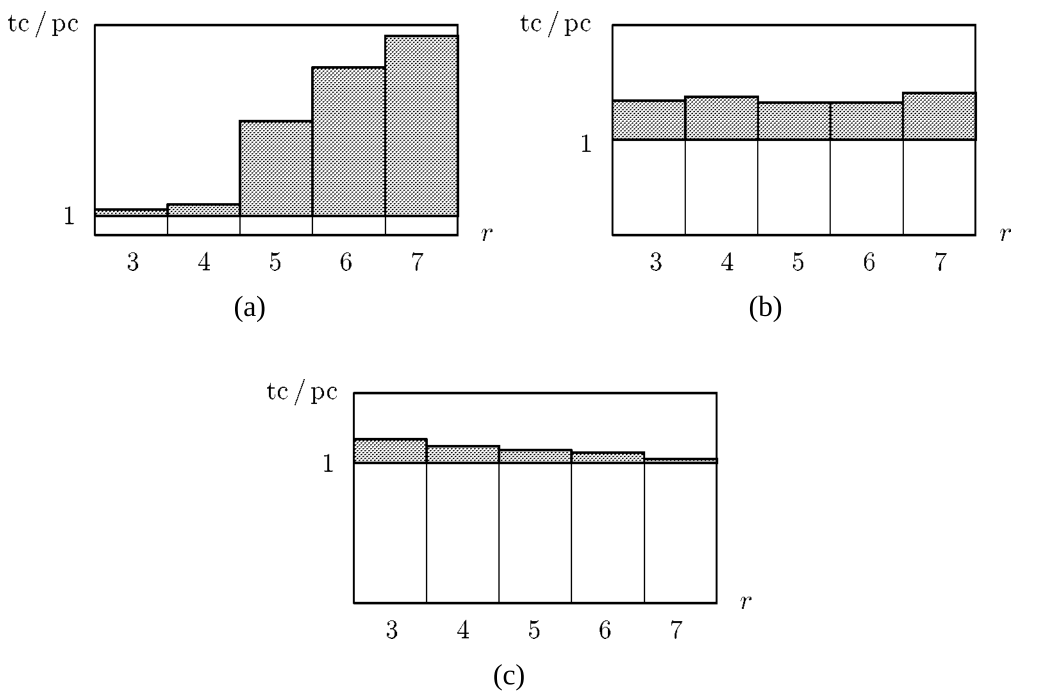

Figure 4. (a) $\widetilde{Q}^{\mathrm{del}}(360+r,360)$, USUAL; (b) $\widetilde{Q}^{\mathrm{del}}(360+r,360)$, DAC; (c) $\widetilde{Q}^{\mathrm{del}}(360+r,360)$, TREE.

References

[1] H. Brass, *Quadraturverfahren* (Vandenhoeck & Ruprecht, Göttingen, 1977).

[2] R. Cools and B. Maerten, Experiments with Smolyak's algorithm for integration over a hypercube, Internal report, Department of Computer Science, Katholieke Universiteit Leuven (1997).

[3] T. Gerstner and M. Griebel, Numerical integration using sparse grids, Numer. Algorithms 18 (1998) 209–232.

[4] G. Hämmerlin and K.-H. Hoffmann, *Numerische Mathematik* (Springer, Berlin, 1989).

[5] Th. Müller-Gronbach, Hyperbolic cross designs for approximatin of random fields, J. Statist. Plann. Inference 49 (1997) 371–385.

[6] E. Novak and K. Ritter, High-dimensional integration of smooth functions over cubes, Numer. Math. 75 (1996) 79–97.

[7] E. Novak and K. Ritter, Simple cubature formulas with high polynomial exactness, Constr. Approx. 15 (1999) 499–522.

[8] E. Novak, K. Ritter and A. Steinbauer, A multiscale method for the evaluation of Wiener integrals, in: *Approximation Theory IX,* Vol. 2, eds. C.K. Chui and L.L. Schumaker (Vanderbilt Univ. Press, Nashville, TN, 1998) pp. 251–258.

[9] T.N.L. Patterson, The optimum addition of points to quadrature formulae, Math. Comp. 22 (1968) 847–856.

[10] K. Petras, On the Smolyak cubature error for analytic functions, Adv. Comput. Math. 12 (2000) 71–93.

[11] K. Petras, Asymptotically minimal Smolyak cubature, preprint, available on `http://www-public.tu-bs.de:8080/~petras/software.html`.

[12] I.H. Sloan and S. Joe, *Lattice Methods for Multiple Integration* (Clarendon Press, Oxford, 1994).

[13] S.A. Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions, Soviet Math. Dokl. 4 (1963) 240–243.

[14] S. Tezuka, Financial applications of Monte Carlo and quasi-Monte Carlo methods. Random and quasi-random point sets, in: *Lecture Notes in Statistics*, Vol. 138 (Springer, New York) pp. 303–332.

[15] G.W. Wasilkowski and H. Woźniakowski, Explicit cost bounds of algorithms for multivariate tensor product problems, J. Complexity 11 (1995) 1–56.