Winker, Peter; Fang, Kai-Tai

Working Paper

Application of threshold accepting to the evaluation of the discrepancy of a set of points

Diskussionsbeiträge - Serie II, No. 248

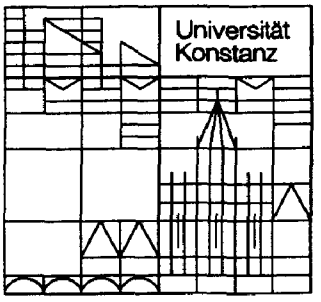
Provided in Cooperation with:
Department of Economics, University of Konstanz

Suggested Citation: Winker, Peter; Fang, Kai-Tai (1995) : Application of threshold accepting to the evaluation of the discrepancy of a set of points, Diskussionsbeiträge - Serie II, No. 248, Universität Konstanz, Sonderforschungsbereich 178 - Internationalisierung der Wirtschaft, Konstanz

This Version is available at:
https://hdl.handle.net/10419/101782

Sonderforschungsbereich 178
„Internationalisierung der Wirtschaft“

Diskussionsbeiträge



Juristische
Fakultät

Fakultät für Wirtschafts-
wissenschaften und Statistik

Peter Winker
Kai-Tai Fang

Application of Threshold Accepting to the Evaluation of the Discrepancy of a Set of Points

---

Postfach 5560
D-78434 Konstanz

Serie II — Nr. 248
Februar 1995

Application of Threshold Accepting to the Evaluation of the Discrepancy of a Set of Points

Peter Winker

Kai-Tai Fang

Serie II - Nr. 248

Februar 1995

Application of Threshold Accepting to the Evaluation of the Discrepancy of a Set of Points[1]

Peter Winker
Fakultät für Wirtschafts-
wissenschaften und Statistik
Universität Konstanz
Postfach 5560
78434 Konstanz
F.R.G.

Kai–Tai Fang
Department of Mathematics
Hong Kong Baptist University
224 Waterloo Road
Kowloon
Hong Kong

Efficient routines for multidimensional numerical integration are provided by quasi–Monte Carlo methods. These methods are based on evaluating the integrand at a set of representative points of the integration area. A set may be called representative if it shows a low discrepancy. However, in dimensions higher than two and for a large number of points the evaluation of discrepancy becomes infeasible. The use of the efficient multiple purpose heuristic Threshold Accepting offers a possibility to obtain at least good approximations to the discrepancy of a given set of points. This paper presents an implementation of Threshold Accepting, an assessment of its performance for some small examples and results for larger sets of points with unknown discrepancy.

KEYWORDS: Number–theoretic methods; discrepancy; numerical integration; threshold accepting.

1991 MATHEMATICS SUBJECT CLASSIFICATION: 65K10, 62K99.

[1]Research was supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 178, "Internationalisierung der Wirtschaft" at the University of Konstanz.

1 Introduction

The efficient evaluation of multidimensional integrals by numerical methods is required in various fields, such as in statistics, physics, economics and econometrics. Several numerical methods in economics and econometrics require efficient routines for multidimensional numerical integration. A straightforward generalization of classical integration rules for the one dimensional case, such as the trapezoidal or Simpson's rule, leads to error bounds of the order $O(n^{-2/d})$, where d denotes the dimensionality. Consequently, in order to guarantee a fixed level of accuracy the number of interpolation points would have to grow exponentially in d, the "curse of dimensionality".

Monte Carlo and quasi–Monte Carlo methods yield powerful tools to overcome these difficulties in multidimensional integration as well as in other domains such as optimization, experimental design, geometric probability and statistical inference.[2] In applying the standard Monte Carlo method (*MCM*), a set of "pseudo" random numbers has to be generated. It is known that the (probabilistic) convergence rate of MCM is $O(n^{-1/2})$.[3] More precisely, let $F(\mathbf{x})$ be the distribution function of the uniform distribution, $U(C^d)$, on a unit cube C^d and $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ be an independent sample from $U(C^d)$, i.e. $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ are random numbers. Let $F_n(\mathbf{x})$ be the empirical distribution function of $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$, i.e.

$$F_n(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} I(\mathbf{u}_i \leq \mathbf{x}), \tag{1}$$

where $I(.)$ is the indicator function. The discrepancy of $\mathcal{U}_n = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n\}$ is defined by[4]

$$D(\mathcal{U}_n) = \sup_{\mathbf{x} \in C^d} |F_n(\mathbf{x}) - F(\mathbf{x})|. \tag{2}$$

If f has bounded variation $V(f)$ on C^d in the sense of Hardy and Krause,[5] then the following inequality holds for the approximation of a multidimensional integral by MCM (Koksma–Hlawka inequality; cf. Niederreiter (1992b), p. 16 equation (2)):

$$\left| \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{u}_i) - \int_{C^d} f(\mathbf{x}) d\mathbf{x} \right| \leq V(f) D(\mathcal{U}_n) \tag{3}$$

It is well known by the central limit theorem that $D(\mathcal{U}_n) = O(n^{-1/2})$ in probability. This convergence rate is still slow and the use of random numbers often leads to *unacceptably large errors.*

[2] Cf. Fang and Wang (1993).

[3] Cf. Niederreiter (1992a), p. 3ff.

[4] This corresponds to the discrepancy at the origin D^* in Faure (1982) or star discrepancy in Weyl (1916) or Niederreiter (1992a).

[5] Cf. Hua and Wang (1981).

Quasi–Monte Carlo or number theoretical methods (*NTM*) overcome the problem of generating independent real random numbers[6] and lead to deterministic error bounds as a fixed set of points is used for the approximation of the multidimensional integral. Even more than that, for a good choice of nodes NTM give better error bounds than MCM.

NTM are a class of techniques by which representative points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of the uniform distribution on the unit cube $C^d = [0,1[^d$ can be generated deterministically.[7] As in the standard Monte Carlo method the multidimensional integral of a function f on C^d is approximated by

$$\int_{C^d} f(\mathbf{x})d\mathbf{x} \approx \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_i)\,. \tag{4}$$

However, $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are deterministic points. The aim of NTM is to generate sets of points with a lower order of magnitude of their discrepancy than that of equally sized sets of random numbers. Let $\mathcal{P}_n = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be a set of points on C^d and $G_n(\mathbf{x})$ be its empirical distribution function. Then, the (deterministic) convergence rate for the discrepancy of $\mathcal{P}_n$ can reach

$$D(\mathcal{P}_n) = \sup_{\mathbf{x}\in C^d} |G_n(\mathbf{x}) - F(\mathbf{x})| = O((\log(n))^d/n), \tag{5}$$

which is better than the rate $O(n^{-1/2})$ obtained by the use of MCM (cf. Niederreiter (1988) and Niederreiter (1992b), p. 19).

However, the convergence rate gives us just a general idea about the asymptotic performance of the procedures, but it is not enough to judge the quality of the procedures for a given problem. For practical applications we have to know the value of the discrepancy of a given set of points. For example, we have to calculate the discrepancy for choosing the uniform design (cf. Fang and Wang (1993), Chapter 5), for giving an upper error bound for numerical integration, and for comparing the discrepancy of two sets with the same number of points generated by *MCM* and *NTM* separately.

For $d = 1$ Niederreiter (1973) gave a convenient formula for calculating the discrepancy while Clerk (1986) proposed a method for the exact calculation of the star–discrepancy for $d = 2$. When $d > 2$ the calculation of the discrepancy is a really difficult task. Unfortunately, there was no publication approaching this problem until Bundschuh and Zhu (1993) gave an algorithm for the low–dimensional case.[8] When n and d are large, their algorithm – which essentially is an enumeration algorithm – uses a tremendous amount of computing time growing at the order

[6] In practice, all applications of MCM use pseudo random number generators.

[7] Recently, NTM have been extended to generate representative points for many useful multivariate distributions and have been systematically applied in statistics (cf. Fang and Wang (1993)).

[8] Dieter (1992), p. 4, states that "No methods are known for calculating the discrepancy in dimension greater than two".

$O(n^d)$. The results presented in section 4 might give an impression of the order of magnitude one has to expect.

2 Calculation of Discrepancy and Generation of Good Lattice Point Sets

In order to obtain the high convergence rate of equation (5) the empirical distribution of the chosen points has to be close to the uniform distribution on the unit cube. To put it differently, they have to be scattered "evenly" or "regularly" on C^d. Referring to inequality (3) the discrepancy of a set of points seems to be a useful concept to measure the irregularity of the distribution of its elements on the unit cube.

Consider a set $\mathcal{P} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of points in the unit cube C^d. Let $\mathbf{x}_k = (x_{k1}, \ldots, x_{kd})$ be the coordinates of the k–th element of $\mathcal{P}$ and denote by $\mathcal{X}_j = \{1, x_{kj} \mid k = 1, \ldots, n\}$, $j = 1, \ldots, d$ the set of j–th coordinates of the elements of $\mathcal{P}$. Furthermore, denote by $\mathcal{X}^* = \{(x_1, \ldots, x_d) \mid x_j \in \mathcal{X}_j, j = 1, \ldots, d\}$ the set of all elements of C^d composed by the coordinates of the elements of $\mathcal{P}$. The number of points in $\mathcal{X}^*$ is at most $(n+1)^d$. For each $\mathbf{x} = (x_1, \ldots, x_d) \in \mathcal{X}^*$ let $M(\mathbf{x})$ be the number of points in $\mathcal{P}$ satisfying $\mathbf{x}_k \leq \mathbf{x}$ and let $M_1(\mathbf{x})$ be the number of points in $\mathcal{P}$ satisfying $\mathbf{x}_k < \mathbf{x}$. Then the definition of the discrepancy (2) is equivalent to

$$D(\mathcal{P}) = \sup_{\mathbf{x} \in C^d} \left| \frac{M(\mathbf{x})}{n} - F(\mathbf{x}) \right|, \tag{6}$$

where $F(\mathbf{x}) = \mathbf{x}_1 \cdot \mathbf{x}_2 \cdots \mathbf{x}_d$ is the distribution function of $U(C^d)$. It is easy to see that

$$D(\mathcal{P}) = \max_{\mathbf{x} \in \mathcal{X}^*} \left\{ \left| \frac{M(\mathbf{x})}{n} - F(\mathbf{x}) \right|, \left| \frac{M_1(\mathbf{x})}{n} - F(\mathbf{x}) \right| \right\}. \tag{7}$$

We can take any kinds of sets of points generated by NTM, such as mentioned in Chapter 3 of Niederreiter (1992a) or generated by MC to test our algorithm. The examples we use in this paper for the demonstration of the algorithm fall under the heading of *good lattice point sets* (GLP), which are particularly suited for periodic integrands. Let $(n; h_1, \ldots, h_d)$ be a vector with integral components satisfying $0 < h_i < n$, $h_i \neq h_j$ $(i \neq j)$, and $d < n$. Let

$$x_{ki} = \left\{ \frac{2kh_i - 1}{2n} \right\}, \; i = 1, \ldots, d, \; k = 1, \ldots, n, \tag{8}$$

where $\{x\}$ stands for the fractional part of x. Then the set $\{\mathbf{x}_k = (x_{k1}, \cdots, x_{kd}), k = 1, \cdots, n\}$ is called the good lattice point set of the generating vector $(n; h_1, \ldots, h_d)$. It can be shown that for any $d \geq 2$ and $n \geq 2$ there exists a GLP–set with n points in C^d with discrepancy $O((\log(n))^d/n)$.[9]

[9] Cf. Niederreiter (1992a), p. 115.

As pointed out above there exists no efficient algorithm for calculating the discrepancy of a given set of points for $d \gg 2$. Hence, we want to apply a heuristic optimization algorithm for evaluating the discrepancy $D(\mathcal{P})$. The integer optimization problem becomes as follows:

$$\max_{\mathbf{x} \in \mathcal{X}^*} f(\mathbf{x}) \,, \tag{9}$$

where the domain $\mathcal{X}^*$ has at most $(n+1)^d$ points, and the objective function on $\mathcal{X}^*$ is given by

$$f(\mathbf{x}) = \max \left\{ \left| \frac{M(\mathbf{x})}{n} - F(\mathbf{x}) \right|, \left| \frac{M_1(\mathbf{x})}{n} - F(\mathbf{x}) \right| \right\} . \tag{10}$$

If the algorithm solves (9) to global optimality we obtain the exact value of the discrepancy of $\mathcal{P}$. Otherwise, the result of the algorithm will be a lower bound to the discrepancy.

3 Threshold Accepting Algorithm for Approximation of Discrepancy

The problem of evaluation of the discrepancy of a given set of points as exposed at the end of the previous section falls under the heading of large scale integer programming problems. As the set of possible solutions $\mathcal{X}^*$ is finite a simple enumeration algorithm would give the exact result. However, as the cardinality of $\mathcal{X}^*$ is approximately $(n+1)^d$ this algorithm is infeasible for high dimensional problems with many points. The same holds true for the somewhat refined enumeration algorithm proposed by Bundschuh and Zhu (1993). In the next section we will give some results and timings for this algorithm. As we do not know about any other exact algorithm requiring only a reasonable amount of computing resources for large d and n, the use of optimization heuristics seems appropriate. However, further research might give new insights as to the real computational complexity of the problem.

As integer optimization heuristic we use the Threshold Accepting algorithm (TA) introduced by Dueck and Scheuer (1990) which was successfully implemented for various problems including the NP–hard travelling salesman problem (Dueck and Scheuer (1990), Winker (1994b)), the NP–complete problem of optimal aggregation (Chipman and Winker (1992, 1994)), portfolio optimization (Dueck and Winker (1992)) and the identification of multivariate lag structures (Winker (1994a)). The TA algorithm may be described as a refined local search algorithm. Sometimes it is also referred to as an evolutionary algorithm.

The basic idea of the TA algorithm is quite simple and similar to the one used for the Simulated Annealing (SA) algorithm (Kirkpatrick et al. (1983)). TA starts with an element $\mathbf{x}_0 \in \mathcal{X}^*$ which might be randomly chosen. Then, a (high) number of iterations is performed. In each iteration step the algorithm tries to replace its current solution $\mathbf{x}^c$ with a new one. The new candidate $\mathbf{x}^n$ is chosen (randomly) as

a small perturbation of the current solution, or — speaking in mathematical terms — in a given neighbourhood of the current solution $\mathbf{x}^c$. The value of the objective function is calculated for the new candidate (in general, this might be done by updating the value for $\mathbf{x}^c$) and the results are compared: $\Delta f = f(\mathbf{x}^n) - f(\mathbf{x}^c)$.

The decision rule in a standard or trivial local search algorithm is to accept $\mathbf{x}^n$ as the new current solution if and only if $\Delta f \geq 0$. If the number of iterations is large enough this algorithm will end up in a local maximum with certainty. However, in general the quality of the local maximum will be low, i.e. the difference to the global maximum will be large. Applications of the trivial local search algorithm to travelling salesman problems show differences in the order of magnitude of 10 percent. A further increase of the number of iterations will not improve the quality of the results. In contrast, both the SA and the TA algorithm do converge to the global optimum with the number of iterations tending to infinity (cf. Aarts and Korst (1989) and Althöfer and Koschnick (1991), respectively).

The refined local search algorithms overcome the problem of getting stuck in a bad local maximum by admitting a temporary worsening of the objective function during the iteration process. In the case of TA the new element $\mathbf{x}^n$ is accepted as the current solution if and only if $\Delta f \geq T$ for a given threshold value T. As this threshold value is non–zero, i.e. negative during most of the iteration steps, bad local maxima which might have been reached can be left again. The thresholds are changed during the running of the algorithm in order to end up at zero at the very end. Thus, the TA algorithm will also end up in a local maximum though of higher quality with good chances of being the global maximum at least for small problem sizes.

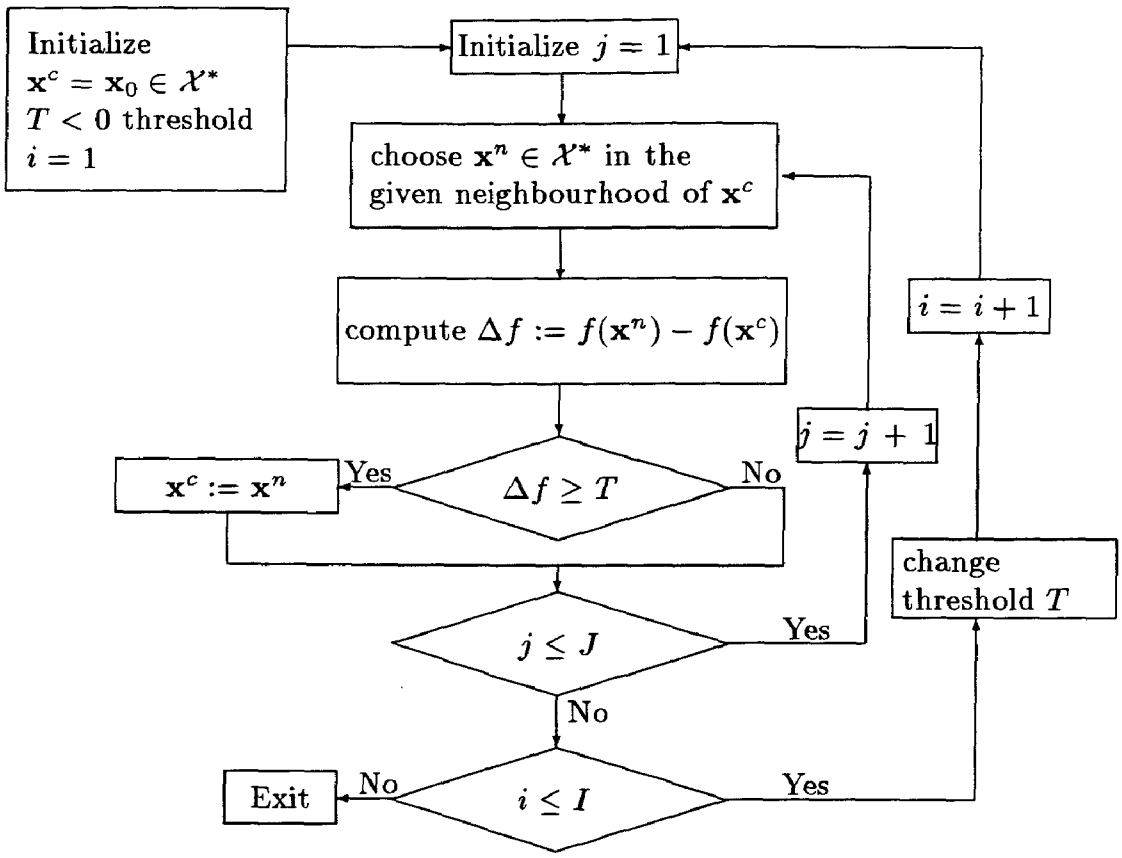
Before turning to a discussion of the choice of local structure by imposing a set of neighbourhoods on $\mathcal{X}^*$ and the choice of the threshold values, figure 1 presents a flow chart of the TA implementation for maximizing the given objective function f, i.e. for approximating the discrepancy for a given set of points.

In analogy to Simulated Annealing the sequence of threshold values T might be interpreted as a cooling schedule. Consequently, for a given value of T a number of J iterations has to be performed in order to bring the system in a stable state with regard to this threshold parameter. Afterwards, the threshold value is increased (decreased in absolute terms) until it reaches zero after I steps of the outer loop.

Although the structure of the algorithm is quite simple its implementation to the approximation of the discrepancy for a given set of points has to take into consideration several aspects. We will concentrate on two central aspects, namely the definition of local neighbourhoods and the generation of a threshold sequence, and sketch two associated coding methods used to reduce the time and memory requirements of our implementation.

Let us start with the introduction of some local structure on the domain $\mathcal{X}^*$. As $\mathcal{X}^*$ is a (finite) subset of the d- dimensional unit cube C^d the use of the projections of some ε–spheres with regard to the Euclidian metric in $\mathbf{R}^d$ seems to be natural. However, this choice would generate two problems. Firstly, the cardinalities of

Figure 1: Threshold Accepting Algorithm for Approximating Discrepancy



the neighbourhoods can be very different. In particular, for large d and $n \ll \infty$ one would have to choose large ε (> 0.5) to make sure that each neighbourhood contains at least two elements. Otherwise, the algorithm would risk getting stuck in the random start vector $\mathbf{x}_0$. The second problem arises only for large d and n and is connected to calculating and storing the resulting neighbourhoods. Either this has to be done once and for all points in $\mathcal{X}^*$ resulting in tremendous memory requirements, or the neighbourhood has to be calculated in each iteration step for the current solution $\mathbf{x}^c$ generating a very high time consumption.

Due to these limitations of projected ε–spheres we use a different concept corresponding to a maximum order norm. In order to define the neighbourhoods we have to introduce some notations. For $1 \leq j \leq d$ $\mathcal{X}_j$ is the set of j–th coordinates of $\mathcal{X}^*$.[10] Let $m_j := \#\mathcal{X}_j$ be the cardinality of $\mathcal{X}_j$ and

$$\rho_j : \{1, \ldots, m_j\} \longrightarrow \mathcal{X}_j \tag{11}$$

[10] Cf. page 2.

an ordering of $\mathcal{X}_j$, i.e.

$$\rho_j(i_1) \geq \rho_j(i_2) \Longleftrightarrow i_1 \geq i_2\,. \tag{12}$$

Now, for $\tilde{x} = (\tilde{x_1}, \ldots, \tilde{x_j}, \ldots, \tilde{x_d}) \in \mathcal{X}^*$ and $k = 2l + 1, l \in \mathbb{N}$ we can define the set of k next neighbours for the j–th coordinate by

$$\mathcal{U}_j^k(\tilde{x}) := \{\mathbf{x} \in \mathcal{X}^* \mid \rho_j^{-1}(x_j) \in [\max\{0, \rho_j^{-1}(\tilde{x}_j) - l\}, \min\{m_j, \rho_j^{-1}(\tilde{x}_j) + l\}]\}\,, \tag{13}$$

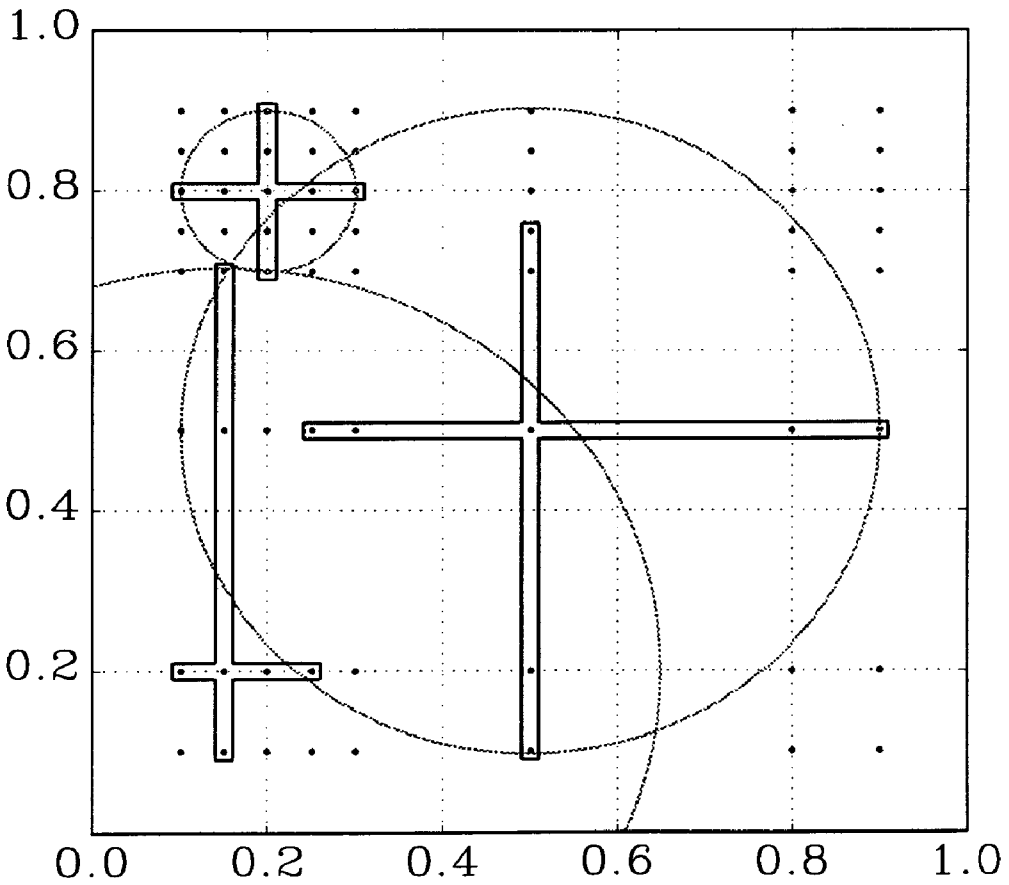
and finally

$$\mathcal{U}^k(\tilde{x}) := \bigcap_{j=1}^{d} \mathcal{U}_j^k(\tilde{x}) \tag{14}$$

is the set of all elements of $\mathcal{X}^*$ with all coordinates being one of the k next neighbours to the corresponding coordinate of $\tilde{x}$ with regard to the orderings ρ.

The following figure 2 might illustrate the concept for $d = 2$. Except for points near to the borders of the unit cube all neighbourhoods have the same number of elements for a given k though they differ distinctly with regard to the also plotted smallest projected ε–spheres of the Euclidian metric including the complete neighbourhood. Of course, this effect becomes even more marked for larger dimensions.

Figure 2: Neighbourhoods for $d = 2$ and $k = 5$



A second advantage of using this definition of local structure is related to the coding of the algorithm. In order to choose a new element in the neighbourhood of the current solution $\mathbf{x}^n \in \mathcal{U}^k(\mathbf{x}^c)$ we only need the orderings $\rho_1, \ldots, \rho_d$ which can be stored using memory growing only at rate $n \cdot d$ in the dimensions of the problem. Finally, the right hand side plots in figure 3 indicate that the objective function f has a reasonable local behavior with regard to this neighbourhood definition, i.e. a strong concentration of only small deviations within a neighbourhood.

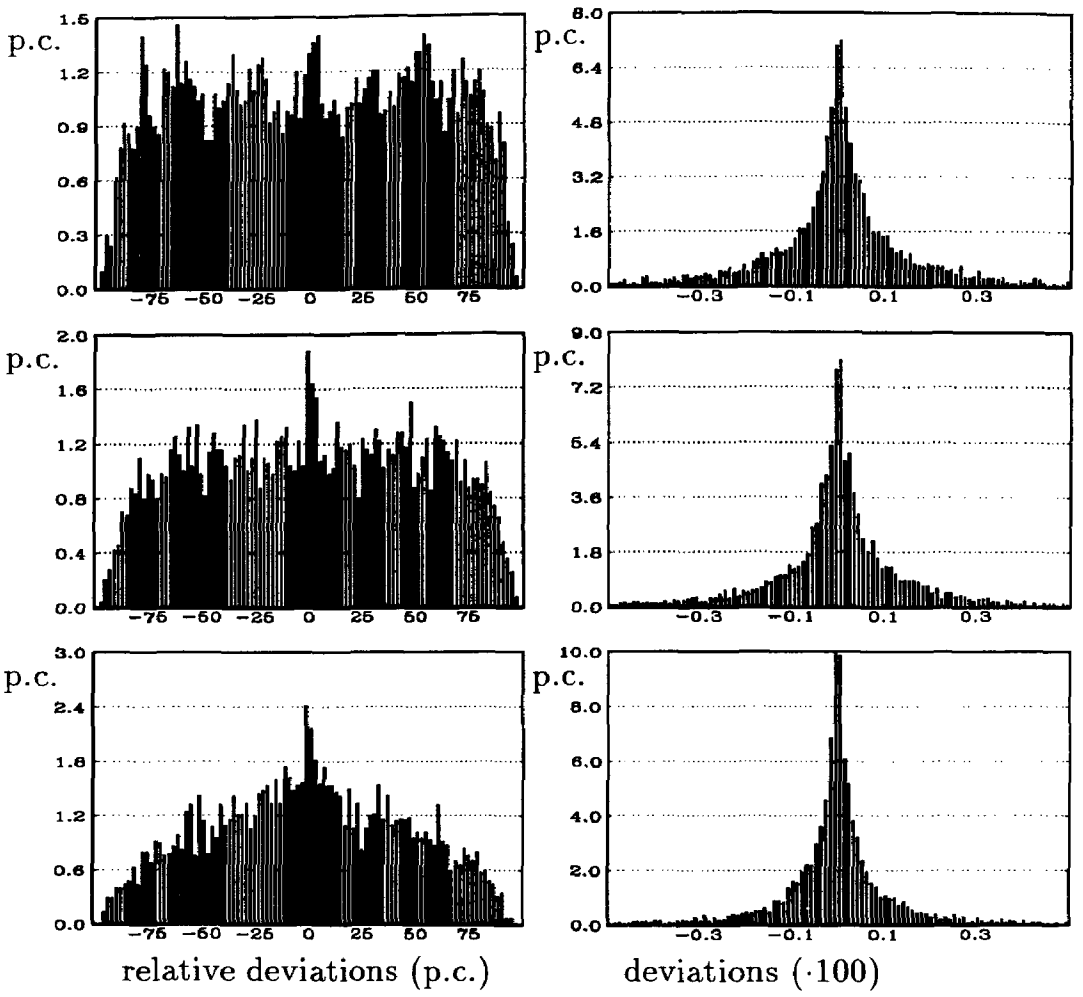
The plots in figures 3 and 4 have been obtained by simulation. The simulations were performed for the glp–set $GLP_{6,2129}$ (cf. table 3 in section 4 for the generating vector). We randomly chose 5.000 points $\mathbf{x}_0 \in \mathcal{X}^*$ and for each $\mathbf{x}_0$ a neighbour $\mathbf{x}_1 \in \mathcal{U}^k(\mathbf{x}_0)$. Afterwards, for each pair $(\mathbf{x}_0, \mathbf{x}_1)$ the objective function f was evaluated at $\mathbf{x}_0$ and $\mathbf{x}_1$. The left hand side plots in figure 3 show histograms of the resulting relative deviations $(f(\mathbf{x}_1) - f(\mathbf{x}_0))/\max\{f(\mathbf{x}_0), f(\mathbf{x}_1)\}$, whereas the right hand side plots in figure 3 and all the plots in figure 4 show histograms of the deviations $f(\mathbf{x}_1) - f(\mathbf{x}_0)$.

The plots in figure 3 were obtained for $k = 101$. The uppermost histograms correspond exactly to the previously given neighbourhood definition. For the lower plots a further restriction was made by allowing only 4 or 2 of the coordinates to change to one of the k next neighbours. This reduction of the neighbourhood seems to introduce also some local structure for the relative deviations. However, due to the fact that the objective function f is restricted to take values only in the interval $[0, 1]$ the choice of relative deviations seems inappropriate in this context. Moreover, the distributions of deviations appear to be symmetric with rather flat tails, i.e. most of the deviations between two elements of a neighbourhood are small. About the same shape of the distribution can be found for other integer optimization problems with unlimited objective functions using relative deviations. For these applications the threshold parameter T is chosen as a mark up factor on the current solution. However, for approximating the discrepancy of a set of points the choice of an additive threshold parameter is superior.

Of course, the choice of local neighbourhoods $\mathcal{U}^k$ still leaves some degrees of freedom for the generation of $\mathbf{x}^n$ in a neighbourhood of $\mathbf{x}^c$. Firstly, k can be varied. Figure 4 shows the influence of changes in k. It plots the distribution of deviations for $k = 11, 101, 1001$. The influence of k on the distribution is not very pronounced. Naturally the distribution becomes flatter for larger k going from top to bottom in the plots in figure 4. However, the distribution shape tells only part of the story. Small values for k mean small cardinality of the neighbourhoods. Consequently, the risk of getting stuck in a local maximum is larger, whereas the number of iterations needed to achieve a stable state is smaller. We find a trade–off of small neighbourhoods with low computational burden and larger neighbourhoods with higher result quality. However, this relation is not monotonic over the whole range of possible k values. We might come back to this point in the next section.

Secondly, the neighbourhoods might be restricted to changes in only a few coordinates as demonstrated in figure 3. Finally, the drawing of $\mathbf{x}^n$ out of a given neigh-

Figure 3: Histograms of Local Deviations[a)]

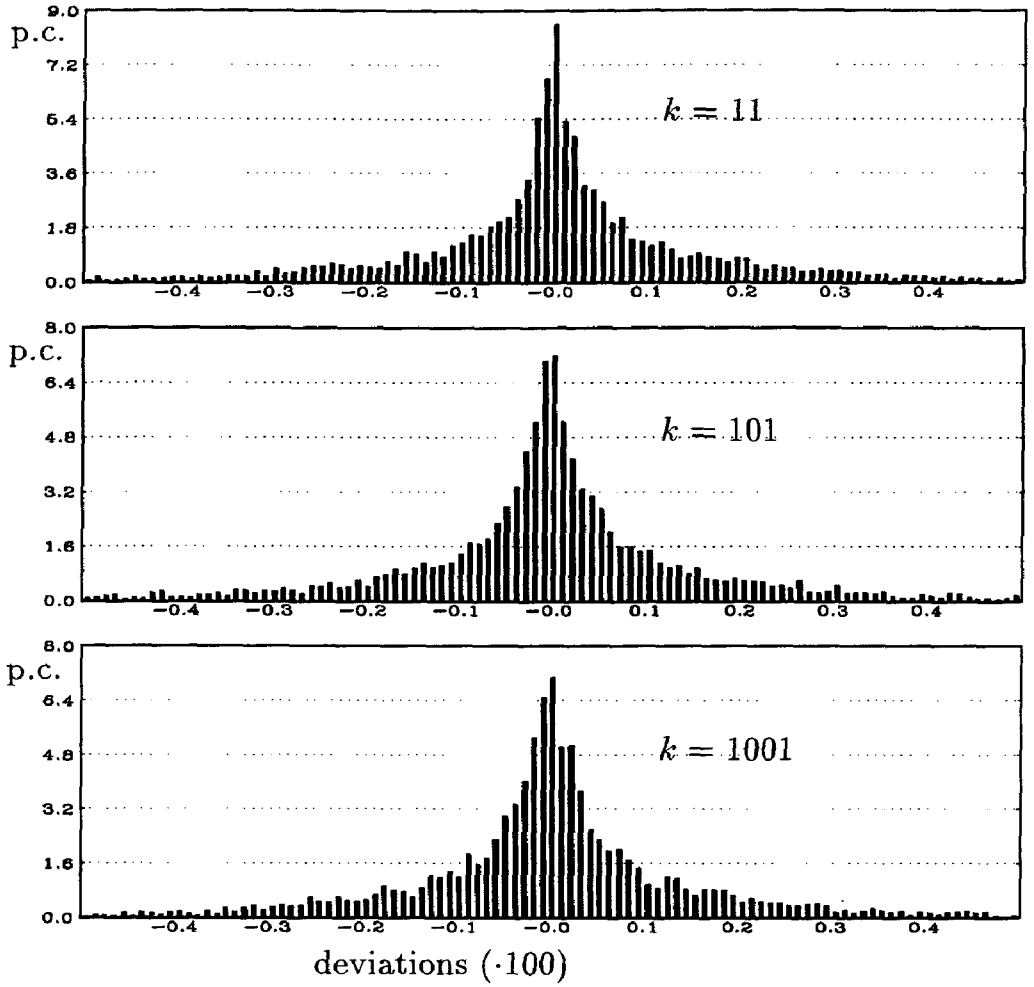


a) See text for details.

bourhood is subject to some weighting. All the simulations presented in figures 3 and 4 were performed using uniform weights. Using a curtailed normal distribution leads to quite similar results.

The exercise of simulating the distribution of local deviations contributes not only some insights to the local structure of the discrete optimization problem under study, but enables us to generate a threshold sequence endogenously. Instead of fixing the threshold sequence exogenously as done in Dueck and Scheuer (1990) or choosing a geometric sequence as in Nissen and Paul (1994) we generate our threshold sequences from an empirical distribution. As for the simulations presented above a (large) number $\tilde{I}$ of pairs $(\mathbf{x}_0, \mathbf{x}_1)$ is randomly generated and the negative absolute deviations $- \mid f(\mathbf{x}_0) - f(\mathbf{x}_1) \mid$ are calculated and sorted in increasing order. The performance of the algorithm can be improved using only a fraction α of this sequence, i.e. $I := \alpha\tilde{I}$ and we use the values closest to zero. As the steps between two

Figure 4: Histograms of Deviations[a)]



a) See text for details.

consecutive thresholds T in this sequence become smaller as the sequence approaches zero, we can use a fixed number of iterations J for each step. Nevertheless, the total number of iterations for a given interval of thresholds will increase due to the shape of the distribution of deviations.

4 Some Results

In order to assess the performance of our TA implementation for approximating the discrepancy of a given set of points we used randomly generated small GLP–sets. For each $d \in \{4, 5, 6\}$ ten GLP–sets with minimal 50 (25 for $d = 6$) and maximal 500 (250 for $d = 5$, 100 for $d = 6$) points were produced along the following lines. Using the uniform random number generator of GAUSS 3.2 n was chosen in the

admissible range. Then, d different elements $h_1, \ldots, h_d$ of $\{1, \ldots, n-1\}$ were drawn and the GLP–set generated as described in equation (8).

For each of the 30 generated GLP–sets the exact value of the discrepancy was calculated using a C–implementation of the algorithm by Bundschuh and Zhu (1993) (we are indebted to Mr. Vincent Chin and Mr. J.X. Pan for leaving as the C–code for our calculations). Table 1 presents the GLP–sets by the number of elements n and the generating elements $h_1, \ldots, h_d$ together with the calculated discrepancy and the time consumption of the algorithm on an IBM RS6000/3AT workstation (SPECfp: 187.2).

To each GLP–set the TA implementation introduced in the previous section was applied. We used 5 ad hoc chosen different parameter constellations ($\tilde{I} = \sqrt{\text{iterations}} \cdot 1.0$ and $\alpha = 0.9 \ldots 1.0$). For each constellation 20 trials were performed.[11] For each parameter constellation the program was run with the total number of iterations varying from only 10.000 up to 100.000. Each trial with 10.000 iterations took about 0.39 seconds of CPU–time and each trial with 100.000 iterations 3.84 seconds. Hence, the time consumptions of our TA implementation is some orders of magnitude smaller than for the (deterministic) algorithm of Bundschuh and Zhu (1993). As the number of iterations was chosen independent of the problem size this effect becomes more marked for larger instances.

Table 2 shows only the results for the runs with 10.000 and 100.000 iterations. As could be expected the quality of the results will increase for a larger number of iterations. It should be noted that we did not intensively tune the algorithm so far neither with regard to the parameters $\tilde{I}$ and α nor with regard to the choice of neighbourhoods. Table 2 presents the ad hoc choice of neighbourhood parameters (maximum number of coordinates to be changed in one step mc and number of next neighbours to be considered k) together with performance results.

The results show that the TA implementation gives a reasonable mean approximation to the (exact) discrepancy of a given set of points. Without formal tuning at least one out of 100 trials gave the global optimum with only 10.000 iterations in about a quarter second with the only exception being the instance 4.451. The percentage of correct results reaches 100 per cent for some instances as the number of iterations goes to 100.000. However, there are some instances for which the mean performance and the percentage of correct results is not yet satisfying. Some more tuning of the optimization parameters probably could increase the quality of the results for these instances.

Of course, we cannot expect this good performance to hold for larger instances. Nevertheless, increasing the number of iterations we can hope to achieve good approximations to the discrepancy for large instances when an exact calculation using the C–code of the algorithm of Bundschuh and Zhu (1993) becomes completely infeasible. We will discuss the "fine tuning" of our TA implementation for a GLP–set

[11] The results in Nissen and Paul (1994) indicate that a number of 5 to 10 trials might be enough to obtain a good estimator of the mean performance. However, as we are interested not only in the mean performance we chose to perform a larger number of trials.

Table 1: Discrepancy for randomly generated GLP–sets

name	d	n	$h_1, \ldots, h_d$						Discrepancy	time
4.145	4	145	38	74	80	143			0.073097	107,59s
4.255	4	255	5	70	110	131			0.109302	1.534,67s
4.312	4	255	138	143	274	285			0.061847	4.010,91s
4.376	4	376	91	218	279	300			0.075314	9.934,17s
4.388	4	388	13	321	362	367			0.129728	11.514,78s
4.442	4	442	64	368	370	431			0.061960	21.606,05s
4.448	4	448	206	264	279	433			0.054795	23.079,21s
4.451	4	451	225	233	343	348			0.027050	23.794,33s
4.471	4	471	1	82	113	401			0.028638	29.380,69s
4.487[a)]	4	487	95	248	251	273			0.041270	34.758,52s
5.102	5	102	13	32	45	54	81		0.121584	422,14s
5.122	5	122	16	35	51	75	95		0.086018	1.226,64s
5.147	5	147	22	69	83	89	120		0.145597	3.130,62s
5.153	5	153	4	52	65	72	99		0.107473	3.876,93s
5.169	5	169	9	82	84	106	157		0.075502	6.824,10s
5.170	5	170	31	49	128	132	154		0.086021	7.050,87s
5.195	5	195	58	155	159	168	177		0.157370	15.281,83s
5.203	5	203	69	86	91	104	174		0.167494	18.939,71s
5.235	5	235	79	111	133	169	224		0.078614	43.869,17s
5.236	5	236	33	61	65	176	189		0.058171	44.947,22s
6.28	6	28	1	4	6	14	15	21	0.536033	3,31s
6.29[a)]	6	29	1	8	12	14	19	22	0.253197	4,20s
6.35	6	35	7	12	17	23	28	30	0.343061	11,63s
6.50	6	50	10	13	18	30	35	41	0.314829	91,24s
6.61[a)]	6	61	15	20	50	51	54	56	0.193738	301,82s
6.73[a)]	6	73	40	49	51	54	56	63	0.148542	899,98s
6.81	6	81	2	11	38	60	70	79	0.250000	1.687,32s
6.88	6	88	6	10	25	27	40	82	0.265817	2.819,58s
6.90	6	90	15	28	33	52	68	74	0.199153	3.267,27s
6.92	6	92	2	8	18	60	65	89	0.163515	3.738,54s

a) Prime number of elements.

Table 2: Performance of TA for Discrepancy Evaluation

name of set	neighbourhood		mean (10.000 it.)	best value	percentage best	
	mc	k			10.000 it.	100.000 it.
4.145	2	41	0.07151	0.07310	33.0	79.0
4.255	2	41	0.10426	0.10930	47.0	89.0
4.312	2	41	0.06162	0.06168	97.0	100.0
4.376	2	41	0.07418	0.07531	44.0	90.0
4.388	2	41	0.12939	0.12973	15.0	86.0
4.442	2	41	0.05602	0.06196	39.0	87.0
4.448	2	41	0.05469	0.05480	60.0	96.0
4.451	2	41	0.02462	0.02705	0.0	2.0
4.471	2	41	0.02323	0.02864	14.0	52.0
4.487	2	41	0.03893	0.04127	7.0	30.0
5.102	3	41	0.11528	0.12158	8.0	16.0
5.122	3	41	0.08394	0.08602	6.0	56.0
5.147	3	41	0.14168	0.14560	23.0	80.0
5.153	3	41	0.10493	0.10747	16.0	26.0
5.169	3	41	0.07038	0.07550	1.0	11.0
5.170	3	41	0.08235	0.08602	31.0	92.0
5.195	3	41	0.15495	0.15737	18.0	82.0
5.203	3	41	0.16747	0.16749	99.0	100.0
5.235	3	41	0.07444	0.07861	15.0	69.0
5.236	3	41	0.05504	0.05817	3.0	13.0
6.28	3	11	0.53603	0.53603	100.0	100.0
6.29	3	11	0.24930	0.25320	16.0	42.0
6.35	3	11	0.30663	0.34306	34.0	92.0
6.50	3	11	0.31472	0.31483	99.0	100.0
6.61	3	21	0.18862	0.19374	42.0	88.0
6.73	3	21	0.14567	0.14854	14.0	81.0
6.81	3	21	0.24999	0.25000	96.0	100.0
6.88	3	21	0.26582	0.26582	100.0	100.0
6.90	3	21	0.19915	0.19915	100.0	100.0
6.92	3	21	0.16248	0.16452	70.0	94.0

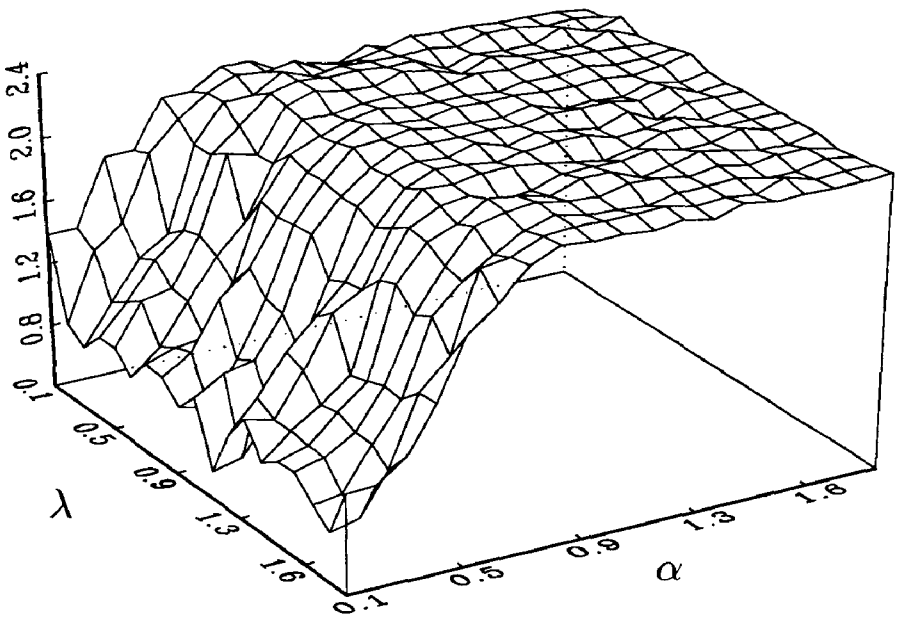
Table 3: Generating Vectors for "large" GLP–Sets

name	n	h_1, h_2, ...										
6.2129	2129	1	41	279	578	793	1681					
7.3997	3997	1	375	1417	2311	3034	3564	3888				
8.3997	3997	1	375	1417	2311	3034	3211	3564	3888			
9.3997	3997	1	375	1417	1962	2311	3034	3211	3564	3888		
10.4661	4661	1	715	1702	2570	3122	3304	3889	4289	4315	4574	
11.4661	4661	1	715	1702	1879	2570	3122	3304	3889	4289	4315	4574

with 2129 points and also present results for the other "large" GLP–sets given by the generating vectors in table 3.

In a first tuning step we analyzed the effect of varying $\tilde{I}$ and α on the mean outcome of 20 optimization runs with 10.000 iterations each. Figure 5 shows the resulting surface plot with 100 times the mean approximation of the discrepancy out of the 20 runs. The numbers on the x–axes show multiples γ of $\sqrt{\text{iterations}}$ used to determine $\tilde{I}$, whereas the y–axes represents fractions α of the empricial distribution used for the threshold sequence. Values of α greater than 1 stand for using the complete empirical jump distribution as threshold sequence muptliplying each value by α.

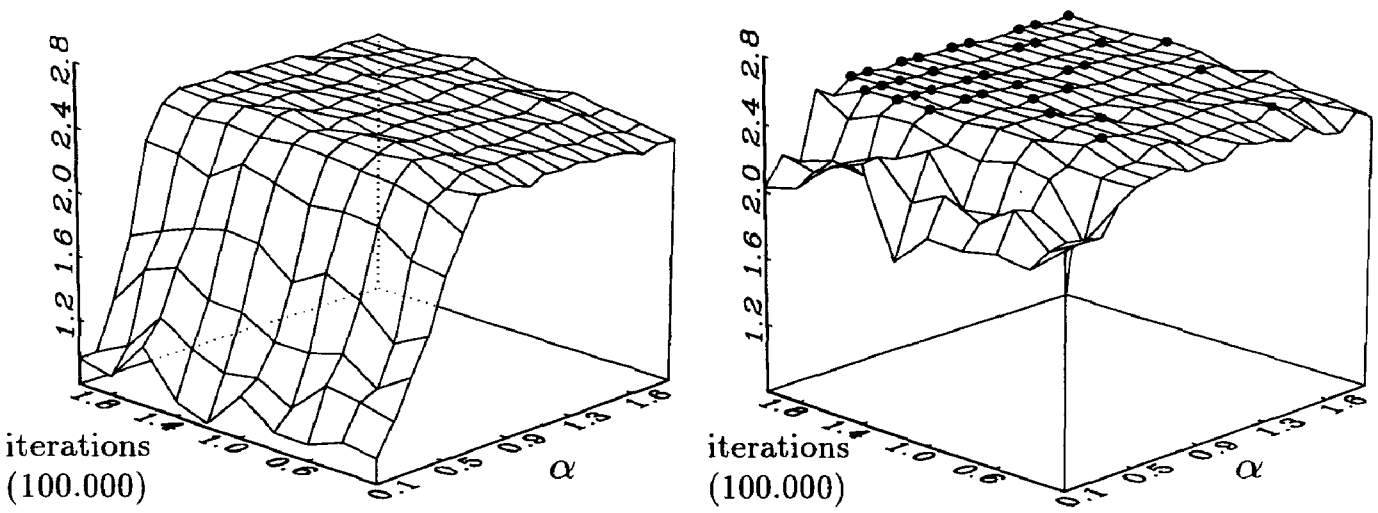
Figure 5: Mean Discrepancy for Different Parameter Choices



As can be seen, the best mean results were achieved choosing α close to one. Furthermore, the quality of the results seems to be quite independent of the choice of $\tilde{I}$ as long as it is choosen large enough. The fact that α has to be choosen close to one instead of the 0.1 to 0.4 optimal for other applications might be attributed to the fact that in this application we use an additive threshold instead of a multiplicative one. However, we again find the feature that a too large choice of α is less harmful than a too small one.

Consequently, for the next tuning step we fixed $\tilde{I}$ to be equal to $\sqrt{\text{iterations}}$ and had a closer look at the effect of the number of iterations performed. Again, for each value of α ranging from 0.05 to 1.95 and each number of iterations ranging from 20.000 to 200.000 twenty replications were performed. Figure 6 plots the resulting surfaces for the mean of these replications on the left side and for the maximum on the right. Given a value of α in the range found optimal by the previous simulation experiment, i.e. between 0.9 and 1.0, the mean quality of the results increases with the total number of iterations though at a diminishing rate.

Figure 6: Mean and Maximum Discrepancy for Different Parameter Choices



Looking at the right hand side of the plot, the marked points represent paramter constellations for which at least once the overall maximal discrepancy was achieved. As can be seen, this global optimum is reached already for quite small number of iterations, but the probability to reach the global optimum increases with the number of iterations leading to the improved mean performance represented by the left hand side part of the figure.

Using the parameters in the range found optimal by the simulations presented

Table 4: Approximation of Discrepancy for "large" GLP–Sets

	neighb.hood		discrepancy				
name	mc	k	α	maximal	mean	std. deviation	CPU–time
6.2129	3	301	0.90	0.025423	0.024821	0.000445	3.287,20s
7.3997	3	301	1.05	0.025382	0.022764	0.001645	3.145,20s
8.3997	3	301	0.95	0.025382	0.021807	0.002131	3.248,51s
9.3997	3	301	0.80	0.038732	0.025850	0.007729	2.627.50s
10.4661	3	301	1.15	0.027182	0.020067	0.004459	3.475.11s
11.4661	4	501	1.15	0.027238	0.021467	0.004625	2.873.80s

for our GLP–set with 2129 points[12]we approximated the discrepancy for the other large GLP–sets given in table 3. The results are presented in the following table 4. The CPU–time in this table gives the total time used on our RS 6000/360 workstation, which is slower than the 3AT by a factor of about 2.6, for 20 replications with 200.000 iterations each for the optimal parameter constellation leading to the maximal discrepancy presented in the fifth column. Hence, a time of about 3.200 seconds would correspond to only 20 minutes on a RS 6000/3AT workstation.

We have no possibility to check whether these results represent already the global optimum, i.e. the actual discrepancy for the given set of points, as the algorithm of Bundschuh and Zhu (1993) would require about 10^27 years of CPU–time on our fastest workstation for the smallest instance 6.2129. Nevertheless, the comparison for the smaller instances certified the high quality of the approximation by the TA algorithm. Thus, we might as least assume that the given values are good lower bounds. Furthermore, all attempts to increase the obtained lower bounds by changing the paramter constellation or increasing the total number of iterations up to 1.000.000 only yielded a minor improvement for the largest instance 11.4661 to 0.02830.

5 Conclusion and Outlook

A central goal of our further research will be to extend the TA implementation to allow for searching "good" point sets, i.e. not only approximating the discrepancy for a given set of points, but for given n and d finding n points, $\mathbf{x}_1{}^*, \ldots, \mathbf{x}_n{}^*$, in C^d such that

$$D(\mathbf{x}_1{}^*, \ldots, \mathbf{x}_n{}^*) = \min_{\mathbf{x}_1, \ldots, \mathbf{x}_n \in C^d} D(\mathbf{x}_1, \ldots, \mathbf{x}_n)\,. \tag{15}$$

Of course, this will introduce a further degree of complexity to the problem. Nevertheless, we are optimistic that TA allows this kind of double–loop optimization.

[12] $\tilde{I}$ was always set equal to $\sqrt{\text{iterations}}$ and α was allowed to vary in the range of 0.8 to 1.2.

References

[1] Aarts, E., and J. Korst (1989). *Simulated Annealing and Boltzmann Machines.* Chichester: John Wiley & Sons.

[2] Althöfer, I., and K.–U. Koschnick (1991). *On the Convergence of 'Threshold Accepting'.* Applied Mathematics and Optimization, 24, 183–195.

[3] Bundschuh, P., and Y. C. Zhu (1993). *A method for exact calculation of the discrepancy of low–dimensional finite point set (I).* Abhandlungen aus dem Math. Seminar der Univ. Hambung, Bd. 63.

[4] Chipman, J. S., P. Winker (1992). *Optimal Aggregation by Threshold Accepting.* Working Paper No. 180, Sonderforschungsbereich 178, University of Konstanz.

[5] Clerk, L. D. (1986). *A method for exact calculation of the star–discrepancy of plane sets applied the sequences of Hammersley.* Mh. Math., 101, 261–278.

[6] U. Dieter (1992). *Principles for Generating Non–Uniform Random Numbers.* In: Jöckel, K.–H., Rothe, G., and W. Sendler (eds.). *Bootstrapping and Related Techniques.* Lecture Notes in Economics and Mathematical Systems 376, Springer–Verlag, Berlin.

[7] Dueck, G., and T. Scheuer (1991). *Threshold Accepting: A General Purpose Algorithm Appearing Superior to Simulated Annealing.* Journal of Computational Physics, 90, 161–175.

[8] Dueck, G., and P. Winker (1992). *New Concepts and Algorithms for Portfolio Choice.* Applied Stochastic Models and Data Analysis, 8, 159–178.

[9] Fang, K. T., and Y. Wang (1993). *Applications of Number Theoretic Methods in Statistics.* Chapman and Hall, London.

[10] H. Faure (1982). *Discrépance de suites associées à un système de numération (en dimension s).* Acta Arithmetica 41, 337–351.

[11] Hua, L. K. and Y. Wang (1981). *Number Theory to Numerical Analysis.* Springer–Verlag and Science Press, Berlin and Beijing.

[12] Kirkpatrick, S., C. Gelatt, and M. Vecchi (1983). *Optimization by Simulated Annealing.* Science, 220, 671–680.

[13] Niederreiter, H. (1973). *Application of diophantine approximations to numerical integration.* In: C. F. Osgood (ed.). *Diophantine Approximation and Its Applications.* Academia Press, New York, 129–199.

[14] Niederreiter, H. (1988). *Low–Discrepancy and Low–Dispersion Sequences.* Journal of Number Theory 30, 51–70.

[15] Niederreiter, H. (1992a). *Random Number Generation and Quasi–Monte Carlo Methods.* SIAM CBMS–NSF Regional Conference Series in Applied Mathematics, Philadelphia.

[16] Niederreiter, H. (1992b). *Lattice Rules for Multiple Integration.* In: Marti, K. (ed.). *Stochastic Optimization.* Lecture Notes in Economics and Mathematical Systems 379, Springer–Verlag, Berlin.

[17] Nissen, V., and H. Paul (1994). *A Modification of Threshold Accepting and its Application to the Quadratic Assignment Problem.* OR–Spektrum, special issue on Local Search, to appear.

[18] Weyl, H. (1916). *Über die Gleichverteilung der Zahlen mod Eins.* Math. Ann., 77, 313–357.

[19] Winker, P. (1994a). *Identification of Multivariate AR–Models by Threshold Accepting.* Computational Statistics & Data Analysis, forthcoming.

[20] Winker, P. (1994b). *The Tuning of the Threshold Accepting Heuristic for Travelling Salesman Problems.* Paper presented at the OR 94, Berlin.