Global and local optimization algorithms in automated waveguide design

Mariusz Kamola [*] , Przemysław Miazga [**]
Warsaw University of Technology
Faculty of Electronics and Information Technology
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland

[*] Institute of Control and Computation Engineering, e-mail: mkamola@ia.pw.edu.pl
[**] Institute of Radioelectronics, e-mail: pjm@ire.pw.edu.pl

An automated CAD design problem is presented. The automation is achieved by embedding simulation software in two optimization routines of different nature. The paper presents first the optimization problem, the optimization algorithms and sample results obtained. The discussion on algorithms' accuracy and effectiveness is given. The example problem presented here, namely optimization of waveguide bend, is one of the highly important design tasks in the telecommunication networks.

1. Introduction

Let us consider a problem of automated CAD design. The system being designed has certain parameters that determine it completely. These are the decision variables. Having decided up to their values, the rest of system parameters can be calculated by dedicated simulation software. Design automation is accomplished by embedding the simulation software in an optimization routine and letting some overall performance function be subject to optimization with respect to the decision variables.

In the most of automated CAD processes the problems encountered are basically of two kinds:

- A simulation in which parameters of a newly designed structure are computed is a long-lasting procedure. Too much time spent on simulation may sometimes not satisfy solution promptness required on the market.
- There are many, sometimes contradictory, optimality criteria (e.g. price of materials, structure performance). Even if one manages finally to combine them in one scalar objective function, its shape is so complicated that involving standard optimization routines in the design phase does not suffice.

Therefore, the optimization procedure must benefit advantages of parallel computing in order to find the optimal design quickly. At the same time it should be able to manage objective function specifics (e.g. multimodality, undifferentiability). An example of such procedure application will be presented in this paper. In section 2 the problem of microwave circuit design is presented and the optimization problem is formulated. The section 3 contains description of the optimization routine that was in use formerly and description of the new optimization routine that conforms to the two postulates expressed above. The two routines have been tested in practice and a short discussion of results is given in section 4. Section 5 contains conclusions.

2. Objective function

Optimization subject is the design of microwave rectangular waveguide bend presented in fig.1. Waveguide is a kind of transmission line, which is frequently used in giga-hertz frequency band. The rectangular waveguide is in fact a metal pipe with rectangular cross-section.

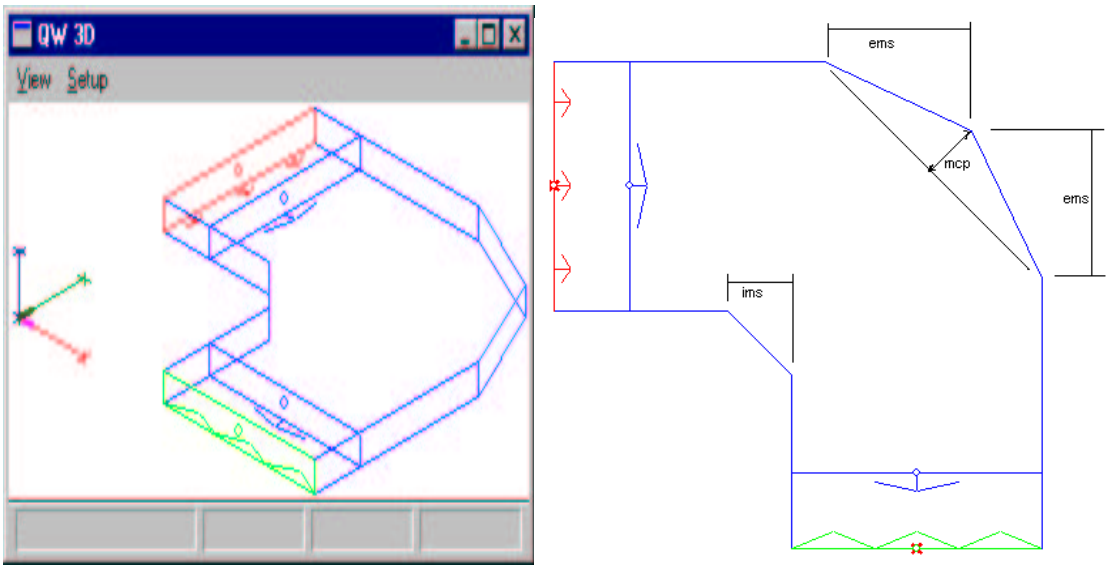

Fig.1 General layout of the considered waveguide bend (left) and top (right) view of the shape of the waveguide bend with its parametrization.

Air inside the pipe is a media, which guides the wave trough. Arrows on fig.1 show direction of wave propagation.

For optimization purposes, two of circuit dimensions are used as decision variables, namely: length of external edge cut - *ems* and its depth of *mcp* (see fig.1 right). Third parameter *ims* shown on fig. 1 defines a chamfer used for compensation of so-called fringing field effects [2] which occur on sharp edges of metal. Its value has been calculated from theoretical formulas and was not subject for optimization.

The aim is to make the bend frequency characteristics lie within specified bounds. Characteristics discrepancy is calculated using L^p norm and is the objective function $f(\cdot)$ being minimized:

$$\text{Find } \min_{\mathbf{x}\in D} f(\mathbf{x}), \quad D = \left[x_1^{\min}, x_1^{\max}\right] \times \left[x_2^{\min}, x_2^{\max}\right],$$

$$x_1 \equiv ems \text{ and } x_2 \equiv mcp\,.$$

Here, D is the optimization domain.

The surface plots of objective function are presented on fig.2.

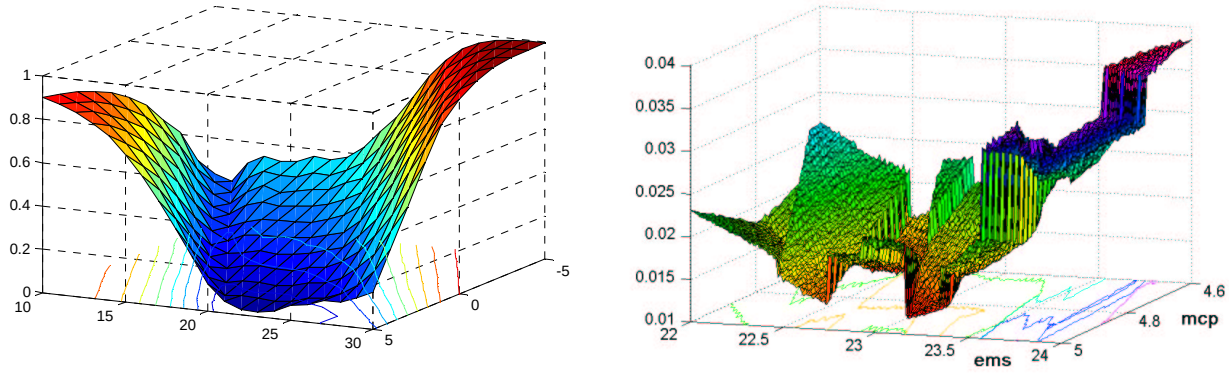

Fig.2 Objective function plots – general (left) and zoom of the optima neighborhood (right).

To evaluate objective function the simulator splits the circuit into cells and performs calculations for each cell separately. Each cell may be filled either fully by air, or partly by a combination of air and metal. Simulator uses different models for calculating an air and metal filled parts of the cell. Such internal model switching is inevitable and causes occasional sharp slopes on the objective function plot, as shown on fig. 2 and fig. 3. Steepness of the slopes depends considerably on the metric parameter c (namely cell size): the greater c the more accurate discrepancy measuring but also the more vertical canyon walls. Therefore, choosing the value of c is a matter of compromise between optimization simplicity and simulation accuracy.

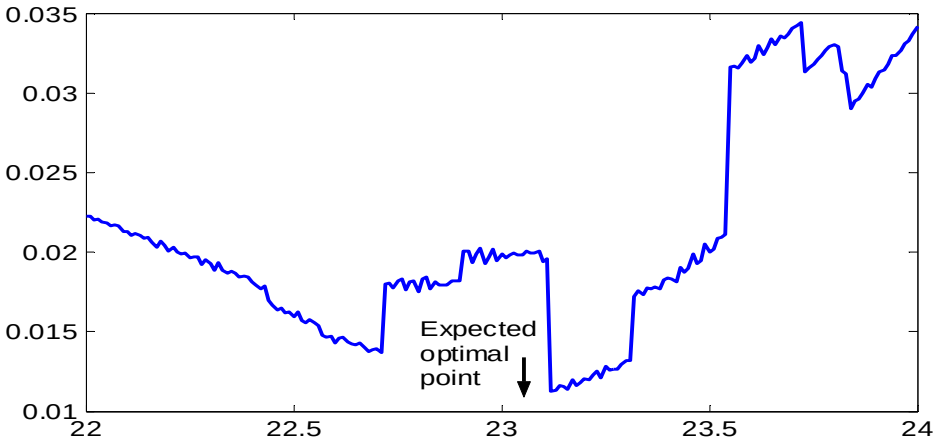

Fig.3 Cross-section trough objective function (see fig. 2) in the optima neighborhood. Chart presents reflection coefficient vs. *ems* for *mcp*=4.96.

Simulator errors disturb shape of the circuit response forming a canyon like landscape. We must notice that real (not affected by “canyon noise”) optima lies inside the hill.

3. Optimization methods

3.1 Powell algorithm

Powell algorithm follows the idea of subsequent directional minimizations in order to find optimization problem solution. Once the starting point $\mathbf{x}_s$ is chosen, there is always the dilemma how to generate directions for the line search subroutine. Iterating though a set of versors is mathematically correct as they span the optimization domain but can turn out to be strikingly ineffective if the objective function forms narrow curving valleys. The improvement would be to choose the next direction so that while optimizing along it the gradient stays perpendicular to the current direction. Such a pair of directions is called conjugate.

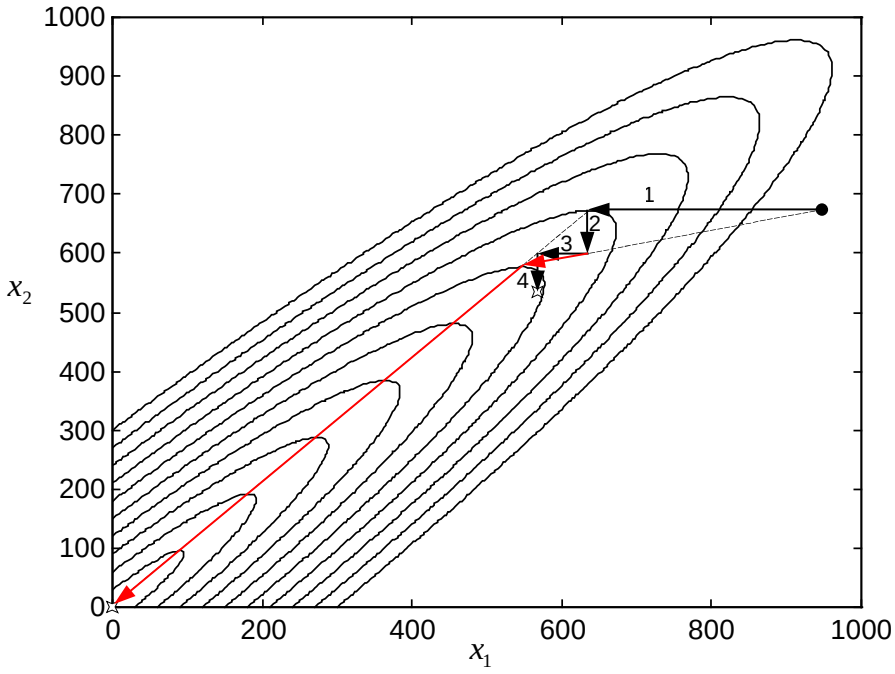

Fig. 4 Directional optimization along versors and conjugate directions

Directional optimization along versors vs. conjugate vectors is shown on figure 4. The first two steps (black arrows) are done in the direction of versors. The third and fourth can be done also along versors (black arrows) but it is much better to perform it using a conjugate vector (arrows nearby) that is created utilizing experience from the last two steps. The steps taken along conjugate vectors lead immediately to the function minimum.

Usually to make a new conjugate direction it is needed to have Hessian matrix (or its approximation) of the function being optimized. Powell suggested a routine in which to make next direction one utilizes solely the data form the last N line searches. The routine preserves algorithm's quadratic convergence rate but its drawback is that directions tend to be linearly dependent. This can be omitted in several ways: one of them is to give up the direction set periodically and to start over with a set of versors. Therefore, the algorithm shape (based on the description given in [7]) is like below:

- **Step 0.** Let $\mathbf{x}_0 = \mathbf{x}_s$, where $\mathbf{x}_s$ is the starting point chosen arbitrarily.
- **Step1.** Create the direction set $R = \{\mathbf{d}_1, ..., \mathbf{d}_n\}$. Initialize its components to search space versors: $\mathbf{d}_i = \mathbf{e}_i$.

- **Step 2.** Perform n line searches: for a single line search $i \in \{1,...,n\}$ start the optimization from the point $\mathbf{x}_{i-1}$ along $\mathbf{d}_i$ and call the result $\mathbf{x}_i$. Stop either if the maximum number of function evaluations has been reached or if $f(\mathbf{x}_i) > (1-\varepsilon) f(\mathbf{x}_{i-1})$, ε being the relative value improvement.
- **Step 3.** Remove $\mathbf{d}_1$ and shift the remaining directions ($\mathbf{d}_i := \mathbf{d}_{i+1}$). Complete the set with $\mathbf{d}_n = \mathbf{x}_n - \mathbf{x}_0$.
- **Step 4.** Perform additional minimization along $\mathbf{d}_n$ and call the result $\mathbf{x}_0$. Stop if the stop criterion (like in step 2) is satisfied; else return to step 1.

A single line search has two phases. The first phase is minimum bracketing routine: the algorithm tries to find a, b, c such that $a < b < c$, $f(\mathbf{x}_{i-1} + a\mathbf{d}_i) > f(\mathbf{x}_{i-1} + b\mathbf{d}_i)$ and $f(\mathbf{x}_{i-1} + c\mathbf{d}_i) > f(\mathbf{x}_{i-1} + b\mathbf{d}_i)$. The second phase is actual rough minimization. Golden partition and quadratic extrapolation/interpolation are utilized in both phases.

The domain boundedness is introduced via penalty functions: if during the line search the constraint is being crossed at, say, $\mathbf{x}_i + e_c\mathbf{d}_i$, then for $e > e_c$ the objective function changes its form to: $f(\mathbf{x}_i + c_c\mathbf{d}_i) + M(c - c_c)^2$, where M is a sufficiently large constant.

3.2 Parallel controlled random search algorithm

The idea of the Controlled Random Search (CRS) algorithm was developed by W. L. Price in [5]. The algorithm requires minimal preparation of data to operate, and can be applicable to constrained as well as to unconstrained optimization problems where objective function gradient is unavailable. In the preliminary stage a pool of randomly selected points is generated and the function values are calculated for each point. Then the main routine starts which computes subsequent trial points and updates the pool accordingly.

CRS, although simple, has several disadvantages in its initial form. Primarily, the convergence rate worsens while approaching neighborhood of the solution. Secondary, the unconditional rejection of a trial point that falls off the constraints makes it difficult to find optimal points that are located on the optimization domain boundaries. The first of the above drawbacks has not turned out to be dangerous for the specific problem. On the contrary, the second one did spoil convergence, and certain algorithm updates were necessary, which led to its final shape as presented below:

- **Step 0.** Let k be the step counter; set $k := 0$. Choose at random N points from D; the points constitute the initial pool $P_k = \{\mathbf{x}_1,...,\mathbf{x}_N\}$. The pool size should be sufficiently large, usually $N = 10(n+1)$. Calculate $f(\mathbf{x}_i)$, $i = 1..N$.
- **Step 1.** Find in the current pool P_k point $\mathbf{x}_{k,l}$ that has the lowest objective function value, and point $\mathbf{x}_{k,h}$ which has the highest objective function value.
- **Step 2.** Make $n+1$-dimensional subset of P_k, called simplex S_k. The simplex must contain $\mathbf{x}_{k,l}$. The remaining n points are chosen at random from P_k (without duplications; choosing $\mathbf{x}_{k,h}$ is allowed). Compute center of the simplex by the formula: $\mathbf{c}_k = \sum_{\mathbf{x}_i \in S_k} \mathbf{x}_i / (n+1)$.
- **Step 3.** Compute a trial point $\mathbf{x}_t$ by reflecting the current worst simplex element $\mathbf{x}_{k,h}$ by the simplex center $\mathbf{s}_k$: $\mathbf{x}_t = 2\mathbf{c}_k - \mathbf{x}_{k,h}$.

- **Step 4.** If any coordinate of $\mathbf{x}_t$ violates constraints, bounce the point back into the domain using the following scheme:

$$x_{t,i} := \begin{cases} 2x_i^{\min} - x_{t,i} & \text{if} \quad x_{t,i} < x_i^{\min} \\ 2x_i^{\max} - x_{t,i} & \text{if} \quad x_{t,i} > x_i^{\max} \\ x_{t,i} & \text{else} \end{cases}$$

- **Step 5.** If the test point is better than the worst point in the pool, i.e. if $f(\mathbf{x}_t) < f(\mathbf{x}_{k,h})$, then make new pool P_{k+1} that is P_k with $\mathbf{x}_{k,h}$ replaced by $\mathbf{x}_t$ and increase step counter. Else go to step 2 to try once more with a different simplex.
- **Step 6.** Stop if the stop criterion is satisfied, i.e. if the best objective function value has not decreased more than ε in a certain number of steps h:

$$f(\mathbf{x}_{k-h,l}) - f(\mathbf{x}_{k,l}) < \varepsilon, \quad f(\mathbf{x}_{k-h,l}) = f(\mathbf{x}_{0,l}) \text{ if } k - h < 0.$$

- **Step 7.** Go to step 1.

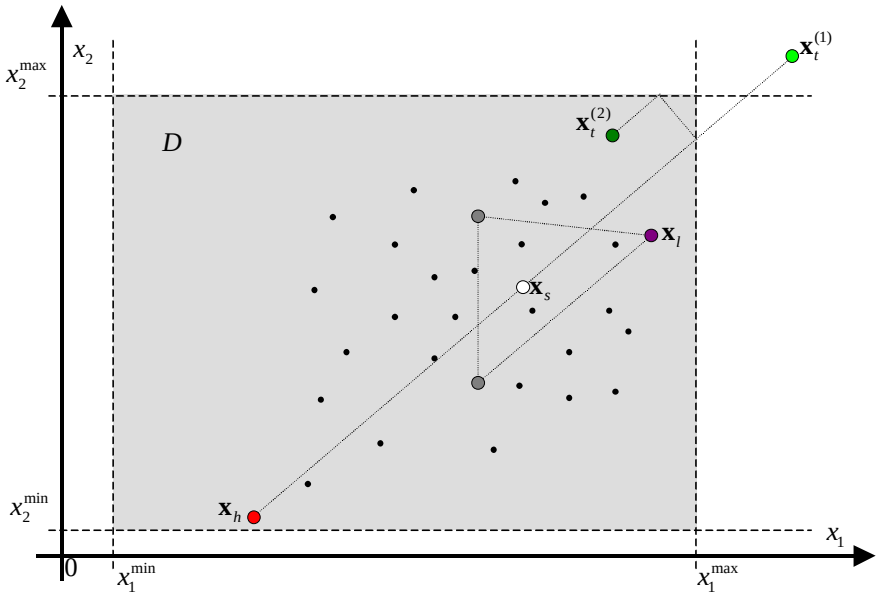

Fig. 5 Computing a new trial point in CRS

The procedure of choosing a simplex and computing a trial point is illustrated on figure 5. The simplex is a triangle in a two-dimensional case. Also, the trial point adjustment operation is shown there, initial and adjusted points being marked $\mathbf{x}_t^{(1)}$ and $\mathbf{x}_t^{(2)}$, respectively.

An additional stop criterion is the total number of function evaluations made. It has been introduced for two reasons: to have general control of costly (in terms of time) function evaluations, and to prevent the algorithm from falling accidentally into infinite loops.

Parallel implementation of CRS concerns two its parts: pool initialization and trial point computation. In the initialization phase, index of the first point in P_0 that has not been yet processed is available for all threads. A single thread starts to process the point (i.e. starts to compute objective function value) and increases the index by one. Therefore, the pool is not divided between threads in advance, but processed accordingly to a thread computation capability.

When the initialization is done, each thread enters its own trial point computation loop. There are several possibilities of interactions between loops in order to improve convergence. W. L. Price in [5] proposes that all threads operate on the same pool that is updated as soon as

a better point is found. In this scheme, all threads reflect the same point $\mathbf{x}_h$, but with regard to various simplex centers. For the specific problem another approach has been applied: a single thread operates on a subset P_k', which is P_k deprived of points that are being reflected by the other threads. All the other details of the routine are like in the basic algorithm: $\mathbf{x}_h'$ is found in P_k', then a reflection is made. After the replacement of $\mathbf{x}_h'$ by $\mathbf{x}_t'$ the algorithm makes $\mathbf{x}_t'$ available to use for the other threads; then the control goes back to step 1. This way, not only reflection centers vary but so do the points being reflected. Obviously, one has to balance the pool size against the number of threads so that sufficient number of points will remain in P_k'.

4. Results

4.1 Typical solution characteristics

Both methods have been used several times to evaluate the optimal solution from various starting points. A typical histogram of Powell and CRS operation is shown on fig.6.

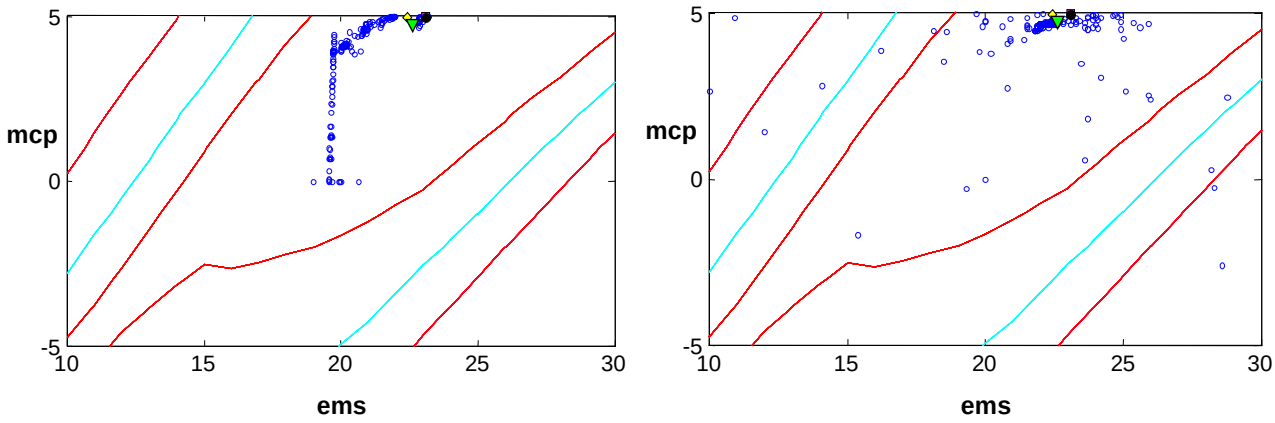

Fig. 6 Comparison of Powell (left) and CRS (right) histograms. Subsequent evaluation points are marked with circles. The true optimal point (i.e. not distorted by numerical errors in simulation) is marked with a triangle.

Both methods have been started from the domain center $[20,0]$ (for CRS, it was actually a centrally placed square of size 5×5). One can clearly see the first two line searches done by Powell (first, in the direction of *ems*, then along *mcp* parameter). Next steps, although taken in conjugate directions, do not lead straight to the solution. This is because of noise (cf. fig. 3) that deceives the method. Moreover, it must be stressed that not all Powell method runs lead safely to the optimum neighborhood. Here, a didactic example has been chosen just to present all optimization phases, but one must realize that a slight change of the starting point or algorithm parameters may cause dramatic performance decrease. This is why the algorithm can be operated successfully only by experts.

CRS histogram does not look strikingly more effective. In fact, the number of function evaluations made to achieve a solution comparable with Powell's is about only one third smaller. But the method is less prone to simulation noise and samples the domain more extensively than just along some directions. It results in method robustness, as it is shown in the next section.

4.2 Performance comparison

A series of optimizations have been performed using both algorithms in various starting conditions. The results are presented on fig. 7.

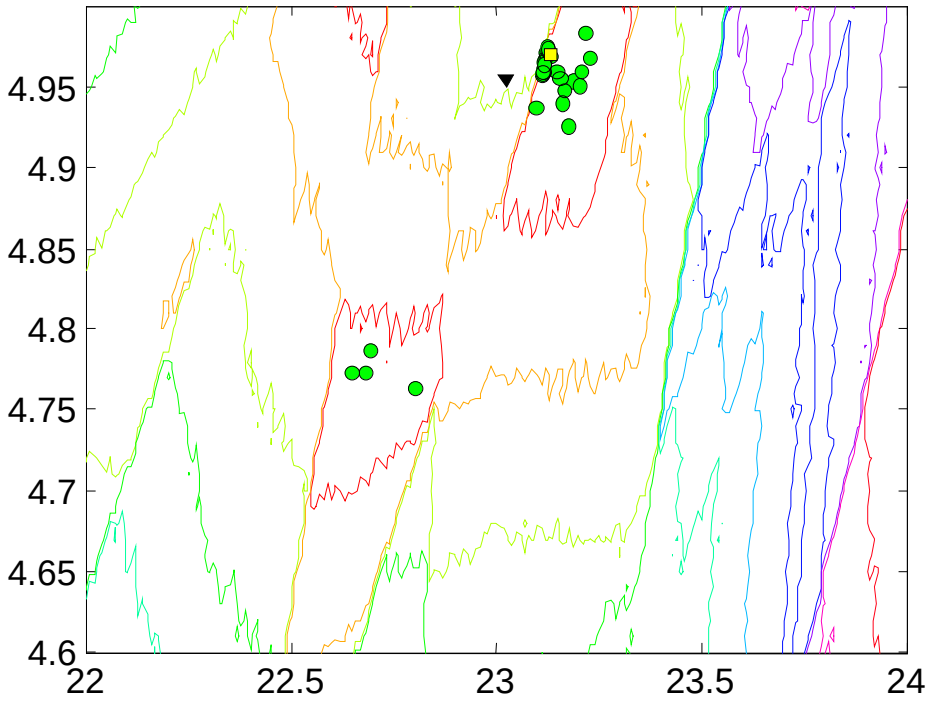

Fig.7. Results from several optimization runs. Circles and a rectangle represent CRS final points and the best Powell solutions, respectively. A triangle denotes a "real" (physical) optimal point.

The figure focuses only on the closest neighborhood of the "true" optimum. One can notice that level lines fold occasionally indicating canyon walls. The "true" minimum lies just below such a steep-sloped wedge and is, obviously, inaccessible. However great majority[*] of CRS solutions approach this solution (or another good one, in the figure center). Powell reaches it too, provided good starting point (i.e. located far enough from the optimum) and parameter values.

5. Conclusions

It is apparent from the above results that CRS surpasses Powell algorithm in accuracy. CRS solutions are better and less dependent on the area the initial pool is located in. Therefore, there is no need for a specialist that would look after the optimization process. Generally, even if the majority of CRS solutions are not globally optimal, they differ from the minimum negligibly and they are acceptable from the practical engineering point of view.

Another advantage of CRS method is its speed. Even on a single processor machine the average number of objective function calculations the method uses to find the solution is 30% less than that of Powell, not mentioning benefits from parallelization. The algorithm can be

[*] Results for 38 out of 40 simulation sessions were satisfactory. The typical simulation time oscillated between 25 and 60 min.

distributed easily and without significant synchronization or communication overhead onto $N-n$ processors, giving virtually linear computation speed-up.

Naturally, CRS has also its drawbacks. Its effectiveness drops rapidly as the problem dimension grows: it has been observed in [8] that for as few as 6 decision variables the number of necessary function evaluations that CRS requires reaches thousands, while Powell routine does well with as little as hundreds. Nevertheless, it is reasonable to employ CRS also in this case just in order to find a good starting point for Powell, which will do the rest of the work. Hopefully, as it was stated in [1], the typical number of decision variables for problems of this type does not exceed three.

The work has proved that while dealing with treachery, simulation-driven objective function it is always worthy to start off with a global optimization routine and to stick with it as long as improves the solution at acceptable rate; then to switch over to some local method in hope to extort anything better.

The problem of how to filter out wedges and canyons on function graph caused by simulator internal switching remains still a question to be solved.

6. Acknowledgments

This work was funded by The Program in Control, Information Technology and Automatization (CATID) of Warsaw University of Technology.

7. Bibliography

1. J.W.Bandler et al., *Electromagnetic optimization of 3D structures,* IEEE Trans on MTT, vol 45, May 1997, pp. 770—779
2. W.Gwarek, P.Miazga, *Improved design of coaxial impedance transformers using electromagnetic 2-D solver in an optimization loop*, MIKON 96, Warszawa, May 27-30, 1996, pp. 433—437
3. P.Miazga, W.Gwarek, *Improved Design of Passive Coaxial Components Using Electromagnetic 2-D Solver in an Optimization Loop*, IEEE Trans on MTT, vol 45, May, 1997, pp. 858—860
4. *QuickWave 3D electromagnetic simulator users manual*, „Qwed" sp. z o. o. Warszawa
5. W. L. Price, *Global Optimization Algorithms for a CAD Workstation*, Journal of Optimization Theory and Applications, 1987, Vol. 55, No. 1, pp. 133-146
6. W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical Recipes in C*, Cambridge University Press, 1992
7. W. Findeisen, J. Szymanowski, A. Wierzbicki, *Teoria i metody obliczeniowe optymalizacji*, PWN, Warszawa 1980
8. P. Miazga, *Optimization of the microstrip to waveguide transiton*, Application note, Institute of Radioelectronics, Warszawa 2001 (to be published).