Available online at www.sciencedirect.com
SCIENCE DIRECT®
Building and Environment 39 (2004) 989–999
BUILDING AND ENVIRONMENT
www.elsevier.com/locate/buildenv

A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization

Michael Wetter[a,*,1], Jonathan Wright[b,2]

[a] *Lawrence Berkeley National Laboratory, Simulation Research Group, Building Technologies Department, Environmental Energy Technologies Division, Berkeley, CA 94720, USA*
[b] *Department of Civil and Building Engineering, Loughborough University, Loughborough, Leicestershire, LE11 3TU, UK*

Abstract

In solving optimization problems for building design and control, the cost function is often evaluated using a detailed building simulation program. These programs contain code features that cause the cost function to be discontinuous. Optimization algorithms that require smoothness can fail on such problems. Evaluating the cost function is often so time-consuming that stochastic optimization algorithms are run using only a few simulations, which decreases the probability of getting close to a minimum. To show how applicable direct search, stochastic, and gradient-based optimization algorithms are for solving such optimization problems, we compare the performance of these algorithms in minimizing cost functions with different smoothness. We also explain what causes the large discontinuities in the cost functions.

Keywords: Optimization; Direct search; Hooke–Jeeves; Coordinate search; Genetic algorithm; Particle swarm optimization

1. Introduction

Detailed building simulation programs, such as EnergyPlus [1] and TRNSYS [2], are increasingly being used to evaluate the cost function in optimization problems. Annual simulations with these programs are typically computationally expensive. Also, these simulation programs contain code features—such as adaptive integration meshes, iterative solvers that iterate until a convergence criterion is met (e.g., Newton solvers or bisection algorithms) and `if–then–else` logic—that can cause optimization algorithms that require smoothness of the cost function to fail, possibly far from a solution. In many of these building simulation programs the solvers are implemented in a way that does not allow controlling the numerical error of the approximations to the state variables, and the solver tolerances are fixed at compile time, in some cases at coarse precision settings [3,4]. Thus, such computer code defines a numerical approximation to the cost function that is discontinuous with respect to the design parameter, and the discontinuities can be large.

It is, however, generally accepted in the simulation-based optimization community that the tolerances of such adaptive solvers must be tight if used in conjunction with optimization algorithms that require the cost function to be smooth [5–7]. If nonlinear programming algorithms are used to solve such optimization problems, then convergence to a stationary point can be established if the approximate cost functions, defined on the numerical approximations to the state variables, converge to a smooth function as the precision of the simulation is increased [7,8], or if the approximation error goes to zero sufficiently fast as the optimization algorithm approaches a solution [9,10]. However, many building simulation codes do not satisfy these requirements and it has been observed [3,4] that the solver tolerances are so coarse that optimization algorithms that require smoothness of the cost function can indeed fail far from a minimum.

Probabilistic optimization algorithms that do not require smoothness have frequently been used to solve building optimization problems with a small number of simulations (see, for example, [11,12,4]). However, these algorithms

* Corresponding author. Tel.: +1-510-486-7538; fax: +1-510-486-4089.
E-mail addresses: mwetter@lbl.gov (M. Wetter), j.a.wright@lboro.ac.uk (J. Wright).
[1] Supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of the Building Technologies Program of the U.S. Department of Energy, under Contract No. DE-AC03-76SF00098.
[2] Supported by a Foresight Award from the UK Royal Academy of Engineering.

0360-1323/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.buildenv.2004.01.022

are stochastic in nature, and to achieve convergence with a high confidence, a large number of simulations is required [13–15], which is impractical if the computation time required to evaluate the cost function is large.

Hence, it is not clear whether optimization algorithms that require smoothness of the cost function or stochastic algorithms, used with a low number of cost function evaluations, perform better on building optimization problems in which the design parameter is in $\mathbb{R}^n$ and has box-constraints. This is the question that we address in this paper.

We compare the performance of nine optimization algorithms using numerical experiments. We compare direct search algorithms (the coordinate search, the Hooke–Jeeves, and two versions of the Nelder–Mead simplex algorithm), stochastic population-based algorithms (a simple genetic algorithm (GA) and two particle swarm optimization (PSO) algorithms), a hybrid particle swarm Hooke–Jeeves algorithm and a gradient-based algorithm (the discrete Armijo gradient algorithm). Other promising methods that have successfully been used in simulation-based optimization, such as methods that use surrogate models [16], are beyond the scope of this paper.

In the numerical experiments, we solved six optimization problems using the EnergyPlus [1] whole building energy analysis program to evaluate the cost function. We used two simulation models, each with three different weather data. One simulation model is such that the cost function is rather smooth and the other is such that the cost function has discontinuities in the order of 2% of the cost function value. By selecting cost functions with different smoothness and identifying what code features can cause such large discontinuities, we believe that our conclusions will also be applicable if other simulation programs are used to evaluate the cost function.

In the first section, we give a formal definition of the optimization problem, which is used to identify the terms that cause difficulties in solving the optimization problems. Next, we discuss the two simulation models. Then we discuss the main features of the optimization algorithms. Finally, we compare the performance of all optimization algorithms and discuss the causes of the observed discontinuities in the cost functions.

2. Minimization problem

We consider problems of the form

$$\min_{x\in\mathbf{X}} f(x), \tag{1a}$$

where $x\in\mathbf{X}$ is the vector of independent variables, $f:\mathbf{X}\to\mathbb{R}$ is the cost function, and $\mathbf{X}\subset\mathbb{R}^n$ is the constraint set, defined as

$$\mathbf{X} \triangleq \{x\in\mathbb{R}^n \mid l^i \leqslant x^i \leqslant u^i, i\in\{1,\dots,n\}\} \tag{1b}$$

with $-\infty \leqslant l^i < u^i \leqslant \infty$, for all $i\in\{1,\dots,n\}$. The cost function $f(\cdot)$ is defined as

$$f(x) \triangleq F(z(x,1)), \tag{2}$$

where $F:\mathbb{R}^m\to\mathbb{R}$ is once continuously differentiable but defined on the solution of a coupled system of differential algebraic equations of the form

$$\frac{\mathrm{d}z(x,t)}{\mathrm{d}t} = h(x,\mu,p(x)), \quad t\in[0,\ 1], \tag{3a}$$

$$z(x,0) = z_0(x), \tag{3b}$$

$$\gamma(x,z(x,t),\mu) = 0, \tag{3c}$$

where $h:\mathbb{R}^n\times\mathbb{R}^l\times\mathbb{R}^q\to\mathbb{R}^m$, $z_0:\mathbb{R}^n\to\mathbb{R}^m$ and $\gamma:\mathbb{R}^n\times\mathbb{R}^m\times\mathbb{R}^l\to\mathbb{R}^l$ are smooth in all arguments and the matrix with partial derivatives $\gamma_\mu(\cdot,\cdot,\cdot)$ is nonsingular. The function $h(\cdot,\cdot,\cdot)$ describes the system dynamics, $z(\cdot,\cdot)$ is the vector of state variables whose components are the room air and construction temperatures (after the spatial discretization of the heat equation) and the heating, cooling and lighting power. The algebraic variable μ is the solution of (3c) and its components are state variables whose thermal capacities are assumed to be negligible, such as the window glass temperatures. The elements of the vector $p(\cdot)\in\mathbb{R}^q$ are the size of the cooling coil, heating coil, and supply and return fan. [3] Thus, system (3) is a mathematical model of a thermal building energy calculation. Under appropriate assumptions, one can show that (3) has a unique once continuously differentiable solution [7,17], and several optimization algorithms that use approximate solutions of (3), and progressively decrease the approximation error as the optimization approaches a solution, exist to solve (1) (see, for example, [7,8]).

However, we are interested in the situation where EnergyPlus is used to compute an approximate numerical solution of (3). EnergyPlus contains several adaptive spatial and temporal grid generators, `if–then–else` logic and iterative solvers that iterate until a convergence criterion is met. Due to these code features a change in the independent variable x can cause a change in the sequence of code executions, which causes the approximate numerical solution of (3) to be discontinuous in x. [4] It is generally accepted in the simulation-based optimization community [5–7] that in situations where (1) is solved using an optimization algorithm that requires the cost function to be smooth, one needs to compute high-precision approximate solutions of (3). However, in EnergyPlus the numerical solvers and grid generators are spread throughout the code and most solver

[3] Clearly, the dependence of $h(\cdot,\cdot,\cdot)$ on $p(\cdot)$ can be eliminated by defining $h(x,\mu,p(x)) \triangleq \tilde{h}(x,\mu)$, but we find it convenient for our discussion to show explicitly the dependence on $p(\cdot)$.

[4] Because $z(\cdot,1)$ and hence $f(\cdot)$ is discontinuous, $f(\cdot)$ may only have an infimum (i.e., a greatest lower bound) but no minimum even if $\mathbf{X}$ is compact. Thus, to be correct, (1a) should be replaced by $\inf_{x\in\mathbf{X}} f(x)$. For simplicity, we will not make this distinction.

tolerance settings are fixed at compile time, in some cases at coarse precision. The implementation of the solvers is such that it does not seem possible to control the numerical error. Thus, optimization algorithms that require smoothness may fail far from a solution or may at best converge much slower. Consequently, we are interested in how they perform compared to probabilistic population-based optimization algorithms on rather smooth cost functions and on cost functions with large discontinuities.

3. Simulation models

We will use two simulation models. The first is a simple simulation model that has four independent variables and no HVAC system simulation (the zone's heating and cooling loads are assumed to be met at each time step). The second is a detailed simulation model that has 13 independent variables and a detailed HVAC system simulation. To determine the size of the HVAC system of the detailed simulation model, EnergyPlus executes a code that contains iterations. Thus, in the simple simulation model the component size $p(\cdot)$ in (3a) is a constant, but in the detailed simulation model $p(\cdot)$ is a discontinuous function of x.

In all optimization problems $f(x)$ is the annual primary energy consumption for lighting, fan, cooling and heating of a mid-story office floor. The exterior walls have a U-value of 0.25 W/(m^2 K) and consist of (listed from outside to inside) 1 cm wood siding, 10 cm insulation and 20 cm concrete. The ceiling and floor consist of carpet, 5 cm concrete, insulation and 18 cm concrete. Interior walls are 12 cm brick. The windows are low-emissivity double pane windows with Krypton gas fill and exterior shading device. We use TMY2 weather data for Houston Intercontinental (TX), Chicago O'Hare (IL), and Seattle Tacoma (WA). Fig. 1 shows the office buildings.

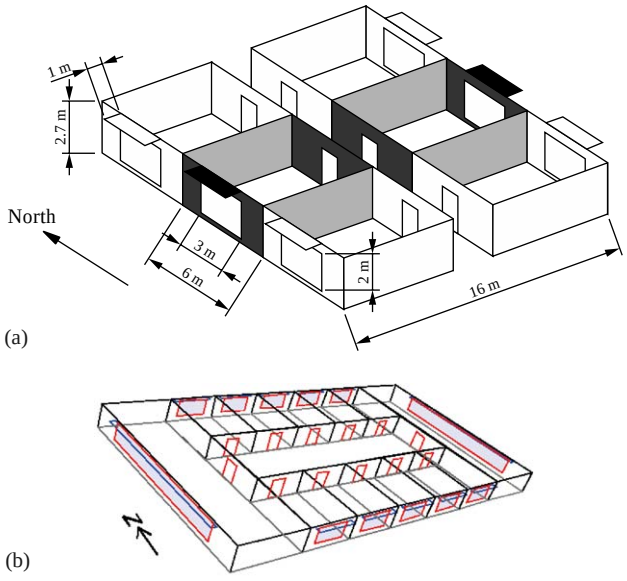

Fig. 1. Buildings used in the numerical experiments: (a) simple office building, (b) detailed office building.

3.1. Simple simulation model

The energy consumption of the gray shaded thermal zone in Fig. 1 is assumed to be representative for the energy consumption of an elongated office building. Both windows have an external shading device that is activated only during summer when the total solar irradiation on the window exceeds 200 W/m^2. Both windows have a fixed overhang that projects out 1 m. The zone has daylighting controls with an illuminance setpoint of 500 lx at a point 3 m from each window.

The annual source energy consumption is

$$f(x) \triangleq \frac{Q_h(x)}{\eta_h} + \frac{Q_c(x)}{\eta_c} + 3\,E_l(x), \tag{4}$$

where $Q_h(\cdot)$ and $Q_c(\cdot)$ are the zone's annual heating and cooling load, respectively, $E_l(\cdot)$ is the zone's electricity consumption for lighting, and the efficiencies $\eta_h = 0.44$ and $\eta_c = 0.77$ are typical plant efficiencies that relate the zone load to the primary energy consumption for heating and cooling generation, including electricity consumption for fans and pumps [18]. The electricity consumption is multiplied by 3.0 to convert site electricity to source fuel energy consumption.

Table 1 lists the independent variables, which are the building azimuth α, the width of the west and east windows w_w and w_e, respectively, and the shading device transmittance τ.[5] The column with header x_b shows the values of the independent variables for the base design, l and u are the lower and upper bounds, and s is the step size of the independent variables. (The step size will be used in the optimization algorithms.)

3.2. Detailed simulation model

We minimize the annual primary energy consumption for lighting, fan, cooling and heating for the mid-story office floor shown in Fig. 1. Lighting and fan electricity are multiplied by 3.0 and then added to the cooling and heating energy of the cooling and heating coil. All exterior zones have daylighting control. The simulated HVAC system is a VAV system with DX coil and outside-air economizer. The heating and cooling coil capacities and the air flow rates are auto-sized by EnergyPlus. Table 1 lists the independent variables. The variable w_i, $i \in \{N, W, E, S\}$, linearly scales the window width and height. The subscripts indicate north, west, east, and south, respectively. (The location and shape of the windows are used in the daylighting calculations.) For the north and south windows a value of 0 corresponds

[5] If $\alpha = 90°$, then the window that was initially facing west is facing north.

Table 1
Variable symbols, initial value x_b, lower bound l, upper bound u and step size s of the independent variable

Variable symbols	x_b	l	u	s	Best iterate x^* Houston, TX	Best iterate x^* Chicago, IL	Best iterate x^* Seattle, WA
Optimization problem with simple simulation model							
α	0	−180	180	10	92.81	86.25	87.50
w_W	3	0.1	5.9	0.2	5.203	4.200	5.900
w_E	3	0.1	5.9	0.2	3.565	5.900	5.900
τ	0.5	0.2	0.8	0.1	0.7964	0.5875	0.5375
$f(x_b)$ in kWh/(m^2 a)					208.2	185.1	164.9
$f(x^*)$ in kWh/(m^2 a)					190.3	155.8	138.0
Maximum obtained reduction in %					8.58	15.82	16.32
Optimization problem with detailed simulation model							
w_N	0.5	0	1	0.05	0.9969	1.000	1.000
w_W	0.5	0	1	0.05	0.1813	0.4000	0.5688
o_W	0.5	0	1	0.05	1.000	0.3500	0.6688
w_E	0.5	0	1	0.05	0.2125	0.3000	0.8813
o_E	0.5	0	1	0.05	1.000	0.4500	0.9656
w_S	0.5	0	1	0.05	0.6406	0.9500	1.000
o_S	0.5	0	1	0.05	1.000	0.1000	0.1438
s_W	200	100	600	25	398.4	400.0	312.5
s_E	200	100	600	25	406.3	450.0	200.0
s_S	200	100	600	25	375.0	575.0	600.0
T_u	22	20	25	0.25	24.61	24.00	24.00
T_i	22	20	25	0.25	22.98	24.75	24.95
T_d	15	12	18	0.25	12.00	12.00	12.00
$f(x_b)$ in kW h/(m^2 a)					165.4	130.1	114.6
$f(x^*)$ in kW h/(m^2 a)					141.5	115.7	95.82
Maximum obtained reduction in %					14.45	11.02	16.41

The variable symbols are explained in the text. The last three columns show the best obtained iterates x^*. The bottom rows show the corresponding cost function values and the obtained cost reductions for each optimization problem.

to a window that covers 13.6% of the facade area and 1 corresponds to 64.8%. For the west and east windows a value of 0 corresponds to a window that covers 20.4% of the facade area and 1 corresponds to 71.3%. The variable o_i, $i \in \{W, E, S\}$, scales the depth of the window overhangs. A value of 0 corresponds to a window overhang depth of 0.05 m (measured from the facade) and 1 corresponds to 1.05 m. The variable s_i, $i \in \{W, E, S\}$, is the setpoint for the shading device in W/m^2. If the total solar irradiation on the window exceeds s_i, then an external shading device with a transmittance of 0.5 is activated. The variable T_i, $i \in \{u, i\}$, is the setpoint for the zone air temperature for night cooling during summer and winter, respectively, in °C. The variable T_d is the cooling design supply air temperature that is used for the HVAC system sizing in °C.

4. Optimization algorithms

We compare the performance of nine optimization algorithms. In the following section, we briefly describe the main features of all algorithms. For a more detailed description we refer the reader to [4] for the simple GA and to the GenOpt manual [19] for all other algorithms, as well as to the references cited therein. Since the performance of the optimization algorithms depends on the algorithm parameters, we list all algorithm parameters, which may be used as initial choices for similar problems. We did not tune the algorithm parameters but used values which we believe will give good performance for the examined problems. A detailed explanation of all parameters is beyond the scope of this paper, and we refer the reader to [4,19] for details.

4.1. Algorithm descriptions

4.1.1. Coordinate search algorithm

The coordinate search algorithm searches along each coordinate direction for a decrease in $f(\cdot)$. Let $k \in \mathbb{N}$ be the iteration number, $x_k \in \mathbf{X}$ be the current iterate, $\Delta_k \in \mathbb{Q}_+$ be a scaling factor, called the *mesh size factor*, and $s \in \mathbb{R}^n$ be as in Table 1. Then, our coordinate search algorithm tests if $f(x') < f(x_k)$ for any $x' \in \mathscr{L}_k$, where

$$\mathscr{L}_k \triangleq \{x \in \mathbf{X} \,|\, x = x_k \pm \Delta_k s^i e_i, i \in \{1, \ldots, n\}\}. \quad (5)$$

If there exists an $x' \in \mathscr{L}_k$ that satisfies $f(x') < f(x_k)$, then the algorithm sets $x_{k+1} = x'$, $\Delta_{k+1} = \Delta_k$, and it replaces k by $k+1$. Otherwise, it sets $x_{k+1} = x_k$, decreases the mesh size factor by setting $\Delta_{k+1} = \Delta_k/2$, and it replaces k by $k+1$. If Δ_k is smaller than a user-specified limit, the search stops.

The coordinate search algorithm is a member of the family of generalized pattern search (GPS) algorithms. For $\mathbf{X} = \mathbb{R}^n$,

one can prove that any GPS method constructs a sequence of iterates with stationary accumulation points if $f(\cdot)$ is continuously differentiable and has bounded level sets [20,21]. For a more detailed description of GPS algorithms and an extension to constraint problems, see [21,8] and the review [22].

For the numerical experiments, we used a mesh size divider of 2, an initial mesh size exponent of 0, a mesh size exponent increment of 1 and 4 step reductions. Hence, $\Delta_0 = 1$ and, for the last iterations, $\Delta_k = \frac{1}{16}$. Thus, the best iterate x^* satisfies $f(x^*) \leqslant f(x')$, for all $x' \in \{x \in \mathbf{X} \mid x = x^* \pm \frac{1}{16} s^i e_i,\ i \in \{1, \ldots, n\}\}$.

4.1.2. Hooke–Jeeves algorithm

The Hooke–Jeeves algorithm is also a member of the family of GPS algorithms and has therefore the same convergence properties on smooth cost functions as the coordinate search algorithm. It adjusts the mesh size factor Δ_k using the same algorithm as the coordinate search algorithm but, in addition to the search on $\mathscr{L}_k$, it also makes progressively bigger steps in the direction that has reduced the cost in previous iterations. As in the coordinate search algorithm, the iterates of the Hooke–Jeeves algorithm belong to a mesh of the form

$$\mathbb{M}(x_0, \Delta_k, s) \triangleq \left\{ x_0 + \Delta_k \sum_{i=1}^{n} m^i s^i e_i \mid m \in \mathbb{Z}^n \right\}. \tag{6}$$

The introduction of $\mathbb{M}(\cdot,\cdot,\cdot)$ is convenient for the discussion of a modified PSO algorithm and a hybrid algorithm below.

We used the same algorithm parameters for the Hooke–Jeeves algorithm as for the coordinate search algorithm.

4.1.3. PSO algorithms

PSO algorithms are population-based probabilistic optimization algorithms first proposed by Eberhart and Kennedy [23,24]. At each iteration step, they compare the cost function value of a finite set of points, called *particles*. The change of each particle from one iteration to the next is modeled based on the social behavior of flocks of birds or schools of fish. Each particle attempts to change its location in $\mathbf{X}$ to a point where it had a lower cost function value at previous iterations, which models cognitive behavior, and in a direction where other particles had a lower cost function value, which models social behavior. Since our simulation model is computationally expensive, we run the PSO algorithms with a much lower number of simulations than the ones in [15,25,14], which makes convergence to a minimum less likely.

We used a PSO algorithm with inertia weight and a PSO algorithm with constriction coefficient. Both algorithms used the von Neumann topology, 16 particles, 20 generations, a seed of 0, a cognitive acceleration constant of 2.8, a social acceleration constant of 1.3 and velocity clamping with a maximum velocity gain of 0.5. For the PSO algorithm with inertia weight, we used an initial inertia of 1.2 and a final inertia of 0. For the PSO algorithm with constriction coefficient, we used a constriction gain of 0.5.

4.1.4. PSO algorithm that searches on a mesh

This is a modification of the above PSO algorithm with constriction coefficient which is introduced in [19]. In this algorithm, the cost function $f : \mathbb{R}^n \to \mathbb{R}$ is replaced by the function $\hat{f} : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{Q}_+ \times \mathbb{R}^n \to \mathbb{R}$, defined as

$$\hat{f}(x; x_0, \Delta, s) \triangleq f(\gamma(x)), \tag{7}$$

where $\gamma(x) \in \mathbb{M}(x_0, \Delta, s) \cap \mathbf{X}$ is the closest feasible mesh point and $\mathbb{M}(\cdot,\cdot,\cdot)$ is as in (6). Evaluating the cost function on the mesh reduces the number of simulations when the particles cluster.

We run this algorithm with two different settings for the algorithm parameters. In the first version, which we will call *PSO on mesh* (1), we used the same parameters as for the PSO algorithm with constriction coefficient and, in addition, a mesh size divider of 2 and an initial mesh size exponent of 1. Thus, $\Delta = \frac{1}{2}$ in (7). In the second version, which we will call *PSO on mesh* (2), we increased the number of particles from 16 to 36 and increased the constriction gain from 0.5 to 1. This causes the particles to cluster later in the search.

4.1.5. Hybrid particle swarm and Hooke–Jeeves algorithm

This hybrid global optimization algorithm does a PSO on a mesh for the first iterations, as described in the previous section. This is done for a user-specified number of generations. Afterwards, it starts the Hooke–Jeeves algorithm using for the initial iterate the mesh point that attained the lowest cost function value. Since the PSO algorithm evaluates $f(\cdot)$ only on a finite number of points in $\mathbb{M}(x_0, \Delta_0, s) \cap \mathbf{X}$, the PSO search can be considered to be a global search of a GPS algorithm. Hence, this hybrid algorithm is also a member of the family of GPS algorithms.

We run this algorithm with two different settings for the algorithm parameters. In the first version, which we will call *PSO and Hooke–Jeeves* (1), we used the same settings as for the algorithm *PSO on mesh* (1). In addition, we used, as for the Hooke–Jeeves and the coordinate search algorithms, a mesh size exponent increment of 1. Because $\Delta_0 = \frac{1}{2}$, we used 3 step reductions to obtain for the last iterations the same mesh size factor as for the Hooke–Jeeves and the coordinate search algorithms, namely $\Delta_k = \frac{1}{16}$. In the second version, which we will call *PSO and Hooke–Jeeves* (2), we increased the constriction gain from 0.5 to 1 to obtain a bigger spread in the particles for the late generations, but we kept the number of particles at 16.

4.1.6. Simple genetic algorithm

GA are algorithms that operate on a finite set of points, called a *population*. The different populations are called *generations*. They are derived on the principles of natural

selection and incorporate operators for (1) fitness assignment, (2) selection of points for recombination, (3) recombination of points, and (4) mutation of a point. Our GA is an implementation of the *simple GA* described by [26], but we use a Gray [27] rather than a pure binary encoding to represent the independent variables as a concatenated string of binary numbers.

The simple GA iterates either until a user-specified number of generations is exceeded, or until all iterates of the current generation have the same cost function value.

In the numerical experiments, we used a population size of 14, a maximum of 50 generations, 1 elite point and a probability for recombination and mutation of 1 and 0.02, respectively.

We selected a small population size because the number of independent variables is small and because we expected the cost function to have no significant local minima. The choice of a small population size was balanced by a high probability of recombination and mutation. Small population sizes have also been used successfully in the solution of other small scale building optimization problems [12].

4.1.7. Simplex algorithm of Nelder and Mead with the extension of O'Neill

The simplex algorithm of Nelder and Mead is a derivative free optimization algorithm. It constructs an n-dimensional simplex in the space of the independent variables. The cost function is evaluated at each of the $(n+1)$ simplex vertices. In each iteration step, the vertex with the highest cost function value is replaced by a new vertex. The new vertex is obtained either by reflecting the vertex with the highest cost function value at the centroid of the simplex, or by contracting and expanding the simplex. Despite the well-known fact that the simplex algorithm can fail to converge to a stationary point [10,28–32], both in practice and theory, particularly if the dimension of independent variables is large, say bigger than 10 [28], it is an often used algorithm. Several improvements to the simplex algorithm or algorithms that were motivated by the simplex algorithm exist, see for example [10,28,29,33]. However, here we used the original Nelder–Mead algorithm [34] with the extension of O'Neill [35] and, in some of the numerical experiments, a modification of the stopping criteria [19].

For the numerical experiments, we used an accuracy of 0.01 and a step-size factor of 0.1. In the experiments that are labeled *Nelder–Mead* (1), we modified the stopping criterion as described in [19] and prevented a new restart of the algorithm during the 10 iterations that followed a previous restart. In the experiments that are labeled *Nelder–Mead* (2), we did not modify the stopping criterion and did not prevent a restart of the algorithm. The second set of numerical experiments has been done because the first set showed poor performance. However, the change in algorithm parameters did not improve the performance.

4.1.8. Discrete Armijo gradient algorithm

We used the discrete Armijo gradient algorithm from [7], which can be used to minimize smooth functions. It approximates gradients by finite differences, with the difference increment reduced as the optimization progresses, and does line searches using the Armijo step-size rule. If $f(\cdot)$ is once continuously differentiable and bounded from below, then the discrete Armijo gradient algorithm constructs sequences with stationary accumulation points. However, the algorithm is sensitive to discontinuities in $f(\cdot)$, and hence, we recommend to not use this algorithm if the simulation program contains adaptive solvers with loose precision settings, such as EnergyPlus. However, since the simple simulation model defines a cost function that has only small discontinuities, we were interested in how this algorithm performs in solving the problems that use the simple simulation model. As we will shortly see, it failed far from a solution.

We used the following algorithm parameters: $\alpha = \frac{1}{2}$, $\beta = 0.8$, $\gamma = 0.1$, $k_0 = 0$, $k^* = -10$, $l_{\max} = 50$, $\kappa = 25$, $\varepsilon_m = 0.01$ and $\varepsilon_x = 0.05$.

5. Numerical experiments

The analysis presented here is in two parts. We will first compare the performance of the different optimization algorithms and then examine the cause of the discontinuities in the cost functions.

5.1. Comparison of the optimization results

For all optimizations, we used the optimization program GenOpt 2.0.0 [19] [6] and EnergyPlus 1.1.0. All computations were run on Linux computers with AMD processors. On a 2.2 GHz processor, one simulation of the simple model takes 14 s, and one simulation of the detailed model takes 2 min and 20 s. Thus, 300 simulations of the simple model (which is what most optimization algorithms used in our experiments) takes 1 h and 10 min, and 500 simulations of the detailed simulation model takes 19 h and 30 min. The overhead of the optimization algorithm and the file I/O is negligible.

We first need to define some measures that we will use to compare the optimization results. Let $x_b \in \mathbf{X}$ be the value of the independent variables for the base design, as listed in Table 1. For each optimization problem (i.e., for each simulation model with the corresponding weather data), we denote, for each optimization algorithm, by $x^* \in \mathbf{X}$ the iterate with the lowest cost function value, and we denote for each optimization problem by $\hat{x} \in \mathbf{X}$ the iterate with the lowest cost function value obtained by any of the tested optimization algorithms. Then, for each optimization problem, we

[6] Work for implementing a GA for GenOpt is in progress.

Table 2
Normalized cost reduction $r(x^*)$, distance to the maximum obtained cost reduction $d(r(x^*))$ and number of simulations m for all optimization problems

Algorithm	Houston, TX			Chicago, IL			Seattle, WA		
	$r(x^*)$ (%)	$d(r(x^*))$ (%)	m (—)	$r(x^*)$ (%)	$d(r(x^*))$ (%)	m (—)	$r(x^*)$ (%)	$d(r(x^*))$ (%)	m (—)
(a) *Optimization problems with simple simulation model and four independent variables.*									
PSOIW	8.44	0.14	316	15.42	0.41	316	16.11	0.21	314
PSOCC	8.27	0.31	313	14.49	1.34	314	14.77	1.55	315
PSOCC on a mesh (1)	8.27	0.31	169	14.57	1.25	160	14.89	1.43	146
PSOCC on a mesh (2)	8.47	0.12	672	15.75	0.07	673	16.23	0.09	652
Nelder–Mead (1)	8.58	0	259	15.71	0.11	672	16.18	0.14	1232
Nelder–Mead (2)	8.58	0	226	15.71	0.11	902	16.25	0.07	1451
PSO and Hooke–Jeeves (1)	8.58	0.01	242	15.82	0	237	16.30	0.02	215
PSO and Hooke–Jeeves (2)	8.56	0.02	326	15.74	0.08	371	16.32	0	358
Hooke–Jeeves	8.58	0.01	103	15.82	0	113	16.30	0.02	97
Coordinate search	8.58	0.01	105	15.82	0	119	16.30	0.02	116
Simple GA	8.53	0.05	194	15.52	0.31	185	16.23	0.09	176
Discrete Armijo gradient	7.93	0.66	315	13.08	2.75	364	14.95	1.37	216
(b) *Optimization problems with detailed simulation model and* 13 *independent variables.*									
PSOIW	13.91	0.54	317	10.63	0.39	317	15.39	1.03	318
PSOCC	11.97	2.49	313	9.66	1.37	314	14.18	2.23	317
PSOCC on a mesh (1)	12.17	2.28	195	9.68	1.34	209	14.20	2.22	242
PSOCC on a mesh (2)	13.49	0.96	710	10.39	0.63	712	15.84	0.57	707
Nelder–Mead (1)	14.09	0.36	2330	4.27	6.76	1228	15.59	0.82	5846
Nelder–Mead (2)	13.98	0.48	1578	—	—	—	—	—	—
PSO and Hooke–Jeeves (1)	14.16	0.29	653	10.94	0.09	755	16.18	0.23	843
PSO and Hooke–Jeeves (2)	14.45	0	740	10.96	0.06	669	16.41	0	889
Hooke–Jeeves	14.27	0.18	555	5.93	5.10	600	16.32	0.09	574
Coordinate search	9.60	4.86	430	4.74	6.29	552	13.04	3.37	501
Simple GA	14.06	0.40	586	11.02	0	592	16.35	0.07	583

define the normalized cost reduction as

$$r(x^*) \triangleq \frac{f(x_b) - f(x^*)}{f(x_b)} \tag{8}$$

and we define the distance to the maximum obtained reduction as

$$d(r(x^*)) \triangleq r(\hat{x}) - r(x^*) = \frac{f(x^*) - f(\hat{x})}{f(x_b)}. \tag{9}$$

Thus, $d(r(x^*))=0$ for the algorithm that achieves the biggest cost reduction.

Because of the discontinuities in the cost functions, we observed different behavior of the optimization algorithms on the problems that used the simple simulation model compared to the problems that used the detailed simulation model. The cost function, if evaluated by the simple simulation model, is rather smooth, but, if evaluated by the detailed simulation model, has discontinuities in the order of 2%, which makes optimization with descent algorithms difficult.

Table 2 shows the normalized cost reduction, the distance to the maximum obtained reduction and the number of simulations, and Fig. 2 shows a graphical representation of the distance to the maximum obtained cost reduction and the required number of simulations for each optimization problem.

We see that if the detailed simulation model is used, then the coordinate search algorithm tends to fail far from the minimum. On the same problems, the Hooke–Jeeves algorithm jammed less often compared to the coordinate search algorithm, which may be due to the larger steps that are taken in the global exploration.

All nonhybrid PSO algorithms come close to the minimum with a low number of simulations. By restricting the iterates of the PSO algorithm with constriction coefficient to the mesh $\mathbb{M}(x_0, \Delta, s)$, the number of simulations could be reduced by 50% for the problem with four independent variables and by 30% for the problem with 13 independent variables. Restricting the iterates to the mesh does not significantly affect the accuracy as we can see by comparing the results of the *PSOCC* and the *PSO on mesh* (1) algorithm, which both use the same algorithm parameters. However, in the PSO algorithm that searches on a mesh, increasing the number of particles from 16 to 36 and increasing the constriction gain from 0.5 to 1 prevented the particles to cluster early in the search and yielded larger cost reductions at the expense of three to four times more simulations.

The simple GA, however, got consistently closer to the minimum than the PSO algorithms with a comparable num-

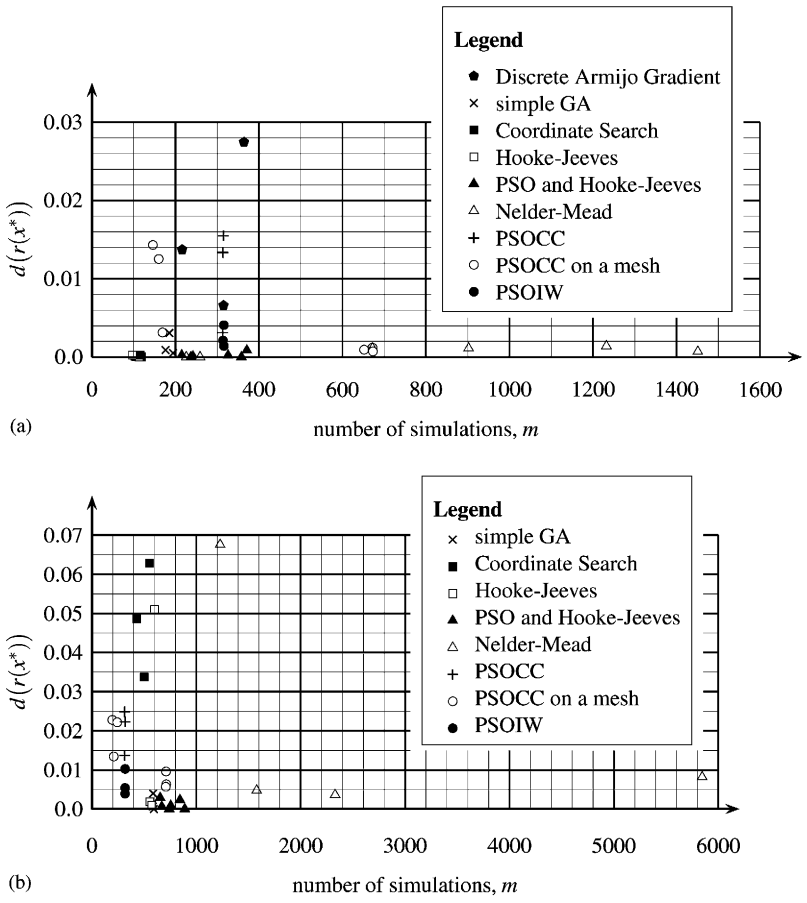

Fig. 2. Number of simulations vs. distance to the maximum obtained cost reduction, (a) Optimization problems with simple simulation model and four independent variables, (b) Optimization problems with detailed simulation model and 13 independent variables.

ber of simulations, except for one problem that used the simple simulation model. In this case, however, the difference in cost reduction is insignificant.

The overall best cost reductions has been achieved by the hybrid particle swarm and Hooke–Jeeves algorithm although with a higher number of simulations than the simple GA. For the hybrid algorithm, increasing the constriction gain from 0.5 to 1.0 did, in our experiments, only slightly change the results. We observed, however, that with a higher constriction gain the particles are more spread out in the early generations, which increases the chance to find a global minimum if the cost function has several minima.

The Nelder–Mead algorithm did not perform well on our test problems. It required a high number of simulations, and in one test case it failed far from the minimum. We believe that, in addition to the problems discussed in [10,28–32], some of the problems we observed when solving the optimization problems that used the detailed simulation model may have been caused by the stopping criterion. The stopping criterion used in [36,19] requires the variance of the function values at the simplex vertices to be smaller than a prescribed limit. However, if $f(\cdot)$ has large discontinuities, then this stopping criterion may never be satisfied.

The discrete Armijo gradient algorithm failed on the simple problem far from the minimum. This is not surprising because the algorithm is sensitive to discontinuities in the cost function. We recommend to not use this algorithm if EnergyPlus is used to evaluate the cost function.

In summary, the simple GA got close to a solution with a low number of simulations. The hybrid particle swarm and Hooke–Jeeves algorithm achieved the biggest cost reduction but required more simulations. Whether the increased number of simulations is justified depends on the savings due to a better solution and the expenses of a higher computation time. However, one advantage of the hybrid algorithm is that the global search of the PSO algorithm increases the chance to get close to the global minimum rather than only a local minimum, and the Hooke–Jeeves algorithm then refines the search locally.

If the discontinuities in the cost function are small, then the Hooke–Jeeves algorithm achieved a good reduction in cost and required only few iterations.

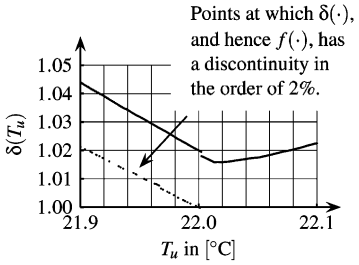

Fig. 3. Normalized change of the cost function value $\delta(T_u)$ as a function of the zone air setpoint temperature for night cooling during the summer months.

5.2. Discontinuities in the cost function

We observed that EnergyPlus computes for the detailed simulation model an energy consumption that has large discontinuities. Some of the discontinuities are caused by the adaptive grid generators—such as the ones used in computing the daylighting illuminance or in doing the variable time-step integration—some are caused by iterative solvers which fail to compute an accurate approximate solution and some are caused by programming errors.

Due to these discontinuities, for the optimization problem with the detailed simulation model and Chicago's weather data, the Hooke–Jeeves algorithm achieved only half of the cost reduction that was obtained by other algorithms. For this numerical experiment, we will now show the change in cost in a one-dimensional subspace of $\mathbf{X} \subset \mathbb{R}^{13}$. In particular, we will perturb one component of the independent variable and plot the change in cost function value. We will first introduce some notation. Let $x^*_{\mathrm{HJ}} \in \mathbf{X} \subset \mathbb{R}^{13}$ denote the iterate with the lowest cost function value of the Hooke–Jeeves algorithm, let $T_u \in \mathbb{R}$ denote the room setpoint temperature for night cooling during summer, and let $e_{T_u} \in \mathbb{R}^{13}$ denote the coordinate vector along T_u. We define $T^*_{u,\mathrm{HJ}} \triangleq \langle x^*_{\mathrm{HJ}}, e_{T_u} \rangle$ and we define the normalized change in cost with respect to T_u as

$$\delta(T_u) \triangleq \frac{f(x^*_{\mathrm{HJ}} + (T_u - T^*_{u,\mathrm{HJ}})e_{T_u})}{f(x^*_{\mathrm{HJ}})}. \tag{10}$$

In Fig. 3, we show $\delta(T_u)$ for $T_u \in [21.9,\ 22.1]$ using 1201 equidistant support points. For $T_u \leqslant 22.001°\mathrm{C}$, $\delta(\cdot)$ has discontinuities in the order of 2%. At $T_u = 22°\mathrm{C}$ is the discontinuity at which the Hooke–Jeeves algorithm got stuck, which is far from the minimum of $f(\cdot)$. The point-wise discontinuities in Fig. 3 are caused by round-off errors: depending on the system's minimum air-flow fraction, which depends on the system sizing $p(\cdot)$, a different branch of an `if–then–else` statement is executed to determine the part load air-flow fraction, and the two branches of the `if–then–else` statement are, due to a programming error, such that they introduce a discontinuity in the part load air-flow fraction. [7]

[7] This will be fixed in future EnergyPlus versions.

We will now show that the cost function of the detailed simulation model also has large discontinuities that are of a different structure than those in Fig. 3. Let $x^*_{\mathrm{CS}} \in \mathbb{R}^n$ denote the iterate with the lowest cost function value obtained by the coordinate search algorithm for Chicago, and let x^*_{GA} denote the iterate with the lowest cost function value obtained by the simple GA for Chicago. In Fig. 4, we show the normalized energy consumption for total primary energy, fan, cooling, heating and lighting energy. The normalized energy consumption is shown along part of the line $x(\lambda) \triangleq x^*_{\mathrm{CS}} + \lambda(x^*_{\mathrm{GA}} - x^*_{\mathrm{CS}})$. That is, $x(0) = x^*_{\mathrm{CS}}$ and $x(1) = x^*_{\mathrm{GA}}$. The graph shows discontinuities of the total primary energy consumption in the order of 1%. Clearly, such large discontinuities can cause optimization algorithms that require smoothness of the cost function to fail far from a minimum.

The biggest discontinuities seem to be caused by the fan sizing. At $\lambda = 0.827$, the fan energy changes by 4% and consequently the cooling and heating energy is also discontinuous at this point. However, at $\lambda = 0.931$, the cooling energy changes by 1.2% while the fan energy is smooth around this point. While such a discontinuity is small if one is only interested in the cooling sizing in a simulation study, it can cause problems in optimization and sensitivity studies, particularly if the cooling energy contributes much to the cost function. The discontinuities in lighting energy are small (the lighting energy does not depend on the fan, cooling or heating energy). We believe that the discontinuities in the lighting energy are caused by a change in the spatial discretization that is used in computing the daylight illuminance. We further believe that the large discontinuities in the fan, heating and cooling energy are caused by the system auto-sizing, which we expect to compute a low-precision approximate solution to $p(x)$, which will then be used in (3) for the whole interval of time integration. In fact, the system sizing is done by first repetitively simulating a so-called warm-up day until some state variables do not change more than a tolerance which is fixed at compile time, and then one day is simulated to determine $p(\cdot)$. This is done for a winter and a summer day. Thus, a change in x can cause a discrete change in the number of warm-up days, and hence in the initial state $z_0(x)$ and consequently in the system size $p(x)$.

6. Conclusions

We observed that in the optimization problems that used the detailed simulation model with auto-sizing of the HVAC components, the cost function has discontinuities in the order of 2%. On such problems optimization algorithms that require smoothness of the cost function are likely to fail far from a solution, which is what we indeed observed in our numerical experiments. Such discontinuities make optimization difficult and can lead to limited economic gains. This can be prevented if the solvers are implemented such that the approximation error can be controlled, and if the simulation program is written such that the approximate solutions

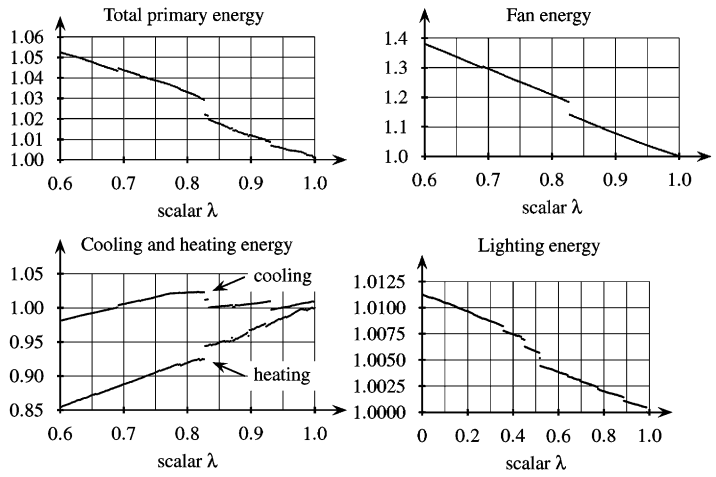

Fig. 4. Normalized cost function value $f(x(\lambda))/f(x(1))$ and primary energy consumption for fan, cooling, heating and lighting, normalized by dividing it by the value at the minimum point of the simple GA. The functions are evaluated on the line between the minimum point obtained by the coordinate search algorithm (at $\lambda = 0$) and the minimum point obtained by the simple GA (at $\lambda = 1$) for Chicago, IL.

of the differential algebraic equations converge to a smooth function as precision is increased. One of the authors is currently developing such a building energy analysis program.

The biggest cost reduction has been obtained with the hybrid particle swarm and Hooke–Jeeves algorithm. If a user is willing to accept a slight decrease in accuracy at the benefit of fewer simulations, then the simple GA is a good choice. However, due to the stochastic operators, the PSO and the simple GA can occasionally fail to get close to a solution, particularly if the number of simulations is small. In such situations, the second search in a hybrid algorithm can further decrease the cost. However, with our limited number of numerical experiments, we could not determine how big the risk of failing is.

For neither of the problems that we examined do we recommend using either the Nelder–Mead or the discrete Armijo gradient algorithm.

7. Nomenclature

7.1. Conventions

(1) Elements of a set or a sequence are denoted by subscripts.
(2) Vectors are always column vectors, and their elements are denoted by superscripts.
(3) The inner product in $\mathbb{R}^n$ is denoted by $\langle \cdot, \cdot \rangle$ and for $x, y \in \mathbb{R}^n$ defined by $\langle x, y \rangle \triangleq \sum_{i=1}^{n} x^i y^i$.
(4) $f(\cdot)$ denotes a function where $(\cdot)$ stands for the undesignated variables. $f(x)$ denotes the value of $f(\cdot)$ for the argument x. $f : A \to B$ indicates that the domain of $f(\cdot)$ is in the space A, and that the image of $f(\cdot)$ is in the space B.
(5) We say that a function $f : \mathbb{R}^n \to \mathbb{R}$ is once continuously differentiable on a set $\mathbf{S} \subset \mathbb{R}^n$ with respect to $x \in \mathbf{S}$ if $f(\cdot)$ is defined on $\mathbf{S}$, and if $f(\cdot)$ has a continuous derivative on $\mathbf{S}$.
(6) For $x^* \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$ continuously differentiable, we say that x^* is stationary if $\nabla f(x^*) = 0$.

7.2. Symbols

$f(\cdot)$	cost function
l	lower bound of the independent variable
n	dimension of the independent variable
t	time
u	upper bound of the independent variable
x	independent variable
$a \in A$	a is an element of A
$A \subset B$	A is a subset of B
$A \cap B$	intersection of the sets A and B
$\mathbf{X}$	feasible set of the independent variable
$\mathbb{N}$	$\{0, 1, 2, \ldots\}$
$\mathbb{Q}$	set of rational numbers
$\mathbb{Q}_+$	$\{q \in \mathbb{Q} \mid q > 0\}$
$\mathbb{R}$	set of real numbers
$\mathbb{Z}$	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
Δ	mesh size factor
$\triangleq$	equal by definition
e_i	unit vector along the ith coordinate direction

References

[1] Crawley DB, Lawrie LK, Winkelmann FC, Buhl WF, Huang YJ, Pedersen CO, Strand RK, Liesen RJ, Fisher DE, Witte MJ, Glazer J. EnergyPlus: creating a new-generation building energy simulation program. Energy and Buildings 2001;33(4):443–57.

[2] Klein SA, Duffie JA, Beckman WA. TRNSYS—a transient simulation program. ASHRAE Transactions 1976;82(1):623–33.
[3] Wetter M, Polak E. A convergent optimization method using pattern search algorithms with adaptive precision simulation. In: Augenbroe G, Hensen J, editors. Proceedings of the Eighth IBPSA Conference, vol. III. NL: Eindhoven; 2003. p. 1393–1400.
[4] Wetter M, Wright J. Comparison of a generalized pattern search and a genetic algorithm optimization method. In: Augenbroe G, Hensen J, editors. Proceedings of the Eighth IBPSA Conference, vol. III. NL: Eindhoven; 2003. p. 1401–8.
[5] Bertsekas DP. Nonlinear programming, 2nd ed. Athena Scientific, Belmont, MA, USA; 1999.
[6] Gill PE, Murray W, Wright MH. Practical optimization. London: Academic Press Inc.; 1981. ISBN 0-12-283950-1.
[7] Polak E. Optimization, algorithms and consistent approximations. Applied Mathematical Sciences, vol. 124. Berlin: Springer; 1997.
[8] Polak E, Wetter M. Generalized pattern search algorithms with adaptive precision function evaluations. Technical report LBNL-52629, Lawrence Berkeley National Laboratory, Berkeley, CA; 2003.
[9] Choi TD, Kelley CT. Superlinear convergence and implicit filtering. SIAM Journal on Optimization 2000;10(4):1149–62.
[10] Kelley CT. Iterative methods for optimization. Frontiers in Applied Mathematics. Philadelphia, PA: SIAM; 1999.
[11] Wright J, Farmani R. The simultaneous optimization of building fabric construction, HVAC system size, and the plant control strategy. In: Lamberts R, Negrão COR, Hensen J, editors. Proceedings of the Seventh IBPSA Conference, vol. I. Rio de Janeiro, Brazil; 2001. p. 865–72.
[12] Caldas LG, Norford LK. A design optimization tool based on a genetic algorithm. Automation in Construction 2002;11(2):173–84.
[13] Greenhalgh D, Marshall S. Convergence criteria for genetic algorithms. SIAM Journal for Computation 2000;30(1):269–82.
[14] Parsopoulos KE, Vrahatis MN. Recent approaches to global optimization problems through particle swarm optimization. Natural Computing 2002;1:235–306.
[15] van den Bergh F, Engelbrecht A. Effects of swarm size on cooperative particle swarm optimisers. In: GECCO. San Francisco, CA; 2001.
[16] Booker AJ, Dennis Jr. JE, Frank PD, Serafini DB, Torczon V, Trosset MW. A rigorous framework for optimization of expensive functions by surrogates. Structural Optimization 1999;17(1):1–13.
[17] Coddington EA, Levinson N. Theory of ordinary differential equations. New York, Toronto, London: McGraw-Hill Book Company, Inc.; 1955.
[18] Huang J, Franconi E. Commercial heating and cooling loads component analysis. Technical report LBL-37208, Lawrence Berkeley National Laboratory; 1999.
[19] Wetter M. GenOpt, generic optimization program, user manual, version 2.0.0. Technical report LBNL-54199, Lawrence Berkeley National Laboratory; 2004.
[20] Torczon V. On the convergence of pattern search algorithms. SIAM Journal on Optimization 1997;7(1):1–25.
[21] Audet C, Dennis Jr. JE. Analysis of generalized pattern searches. SIAM Journal on Optimization 2003;13(3):889–903.
[22] Tamara G, Kolda RML, Torczon V. Optimization by direct search: new perspectives on some classical and modern methods. SIAM Review 2003;45(3):385–482.
[23] Eberhart RC, Kennedy J. A new optimizer using particle swarm theory. In: Sixth International Symposium on Micro Machine and Human Science. Nagoya, Japan: IEEE; 1995. p. 39–43.
[24] Kennedy J, Eberhart RC. Particle swarm optimization. In: IEEE International Conference on Neural Networks, vol. IV. Perth, Australia; 1995. p. 1942–48.
[25] Kennedy J, Eberhart RC, Shi Y. Swarm intelligence. Los Altos, CA: Morgan Kaufmann Publishers; 2001.
[26] Goldberg D. Genetic algorithm in search, optimization, and machine learning. Reading, MA: Addison-Wesley; 1989.
[27] Press WH, Flannery BP, Tuekolsky SA, Vetterling WT. Numerical recipes in C: the art of scientific computing. Cambridge: Cambridge University Press; 1993.
[28] Torczon V. Multi-directional search: a direct search algorithm for parallel machines. PhD thesis, Rice University, Houston, TX, 1989.
[29] Kelley CT. Detection and remediation of stagnation in the Nelder–Mead algorithm using a sufficient decrease condition. SIAM Journal on Optimization 1999;10(1):43–55.
[30] Wright MH. Direct search methods: once scorned, now respectable. In: Griffiths DF, Watson GA, editors. Numerical analysis 1995. Harlow: Addison-Wesley Longman; 1996. p. 191–208.
[31] McKinnon KIM. Convergence of the Nelder–Mead simplex method to a nonstationary point. SIAM Journal on Optimization 1998;9(1):148–58.
[32] Lagarias JC, Reeds JA, Wright MH, Wright PE. Convergence properties of the Nelder–Mead simplex method in low dimensions. SIAM Journal on Optimization 1998;9(1):112–47.
[33] Tseng P. Fortified-descent simplicial search method: a general approach. SIAM Journal on Optimization 1999;10(1):269–88.
[34] Nelder JA, Mead R. A simplex method for function minimization. The Computer Journal 1965;7(4):308–13.
[35] O'Neill R. Algorithm AS 47—function minimization using a simplex procedure. Applied Statistics 1971;20:338–45.
[36] Nelder JA, Mead R. Simplex method for function minimization. The Computer Journal 1965;7(4):308–13.