Contents lists available at SciVerse ScienceDirect

Energy

journal homepage: www.elsevier.com/locate/energy

PEM fuel cell modeling using differential evolution

Uday K. Chakraborty [a,*], Travis E. Abbott [a], Sajal K. Das [b]

[a] Department of Mathematics and Computer Science, University of Missouri, St. Louis, MO 63121, USA
[b] Department of Computer Science and Engineering, University Texas at Arlington, Arlington, TX 76019, USA

ARTICLE INFO

Article history:
Received 30 June 2011
Received in revised form
16 December 2011
Accepted 19 January 2012
Available online 3 March 2012

Keywords:
Proton exchange membrane fuel cell
PEMFC stack
Differential evolution
Genetic algorithm

ABSTRACT

This paper develops a differential-evolution-based solution for the problem of proton exchange membrane fuel cell stack modeling. The problem is analytically intractable and computationally hard. The present paper produces results that outperform state-of-the-art approaches on three performance metrics: solution quality corresponding to a given cost, cost of finding a solution of a given quality, and frequency of producing an optimal or near-optimal solution.

1. Introduction

Fuel cells [13,10] convert chemical energy of fuels into electricity, and provide a practical method of reliable, clean and noise-free power generation. With natural sources of energy, such as petroleum and coal, being fast depleted, alternative energy sources such as renewable energy systems [15] are assuming increasingly important roles in today's power generation applications. The past few years have seen a tremendous growth in research activities on fuel cells and full cell-based power generation systems. The technical feasibility of these environment-friendly devices having already been established, the main goal of present-day research is to bring down the cost and increase the reach of this technology so that it can compete with traditional power generation methods.

Proton exchange membrane (PEM, or polymer electrolyte membrane) fuel cells find wide applications because of their low operating temperature, low emission, no harmful wastes (water is the by-product), high efficiency, and potential cost reduction (due to economies of scale). Unlike other distributed generation technologies, such as wind or photovoltaic systems, PEM fuel cells are suitable for placement at any site in a distribution system (e.g., in cars, as CO sensors, for integration on a silicon chip towards low-power wireless sensor network applications).

PEM fuel cells are emerging as viable energy sources for pervasive and ubiquitous sensing applications and for pervasively deployed computing and communication units (e.g., [6,17]). The future of pervasive and mobile computing is inextricably linked with that of smart energy, and PEMFCs are important for smart energy. Wireless sensor platforms are designed to run on batteries that, in general, have a very limited lifetime. If sensor networks are to become a ubiquitous part of our environment, alternative power sources must be employed.

For a PEMFC system to deliver the desired amount of power, individual fuel cells are combined in series (to increase the voltage), or in parallel (to increase the current), or both, creating a stack. An increase in the cell surface area results in more current from each cell.

Modeling the operation of PEM fuel cell stacks [9] poses a problem that is often analytically intractable and computationally complex. The present paper addresses the problem of finding values of seven parameters that best fit the empirically observed polarization characteristics of a given PEMFC stack. The use of computational algorithms [12] for producing optimized stack models is an emerging field of research [18,4,2,20,21]. The hitherto best-known method for solving this problem was published in Ref. [20]. This paper presents a new, differential evolution approach that outperforms the state-of-the-art approaches of Refs. [20,18]. Differential evolution (DE) [5,23] is an efficient scheme for global optimization over continuous spaces. The present paper applies an improved DE algorithm, called DEGL (and its two variants, DEGL/SAW/bin and DEGL/SAW/exp), to the PEMFC problem. Performance comparisons using polarization behavior prediction establish the superiority of the present method.

* Corresponding author. Tel.: +1 314 516 6339; fax: +1 314 516 5400.
E-mail address: chakrabortyu@umsl.edu (U.K. Chakraborty).

Nomenclature

PEMFC	
N_s	number of series cells in a stack
E_{Nernst}	Nernst (reversible) potential of a single cell, V
V_{cell}	output terminal voltage of a single PEMFC, V
p_{H_2}	partial pressure of hydrogen, atm
p_{O_2}	partial pressure of oxygen, atm
T	stack temperature, K
RH_a	relative humidity of vapor in anode
RH_c	relative humidity of vapor in cathode
$p^{\text{sat}}_{H_2O}$	saturation pressure of water vapor, atm
p_a	anode inlet pressure, atm
p_c	cathode inlet pressure, atm
A	cell area (effective electrode area), cm^2
i	cell current, A
R_C	resistance, Ω
R_M	equivalent resistance of membrane, Ω
ρ_M	specific resistivity of membrane for electron flow, Ω cm
ℓ	membrane thickness, cm
i_{den}	current density, A/cm^2
$i_{\text{limit,den}}$	limiting current density, A/cm^2
B	concentration loss constant, V
V_{stack}	output terminal voltage of the PEMFC stack, V
η_{act}	activation loss, V
η_{conc}	concentration loss, V
η_{ohm}	Ohmic loss, V
x_1,x_2,x_3,x_4	parametric coefficients of the cell
λ	parameter of the cell
Algorithms	
NP	DE population size
C_r	DE crossover probability
F,α,β	DE scale factors
w	DE scalar weight
k	DE neighborhood radius
p_c	GA crossover probability
p_m	GA mutation probability

The remainder of this paper is organized as follows. Section 2 describes the PEMFC electrochemical model and derives the main equations needed for solving the problem. Section 3 defines the fitness function, explains the encoding scheme, and describes the algorithm, including how the improved mutation operator in differential mutation helps the search process. Section 4 presents simulation results and statistical tests. Conclusions are drawn in Section 5.

2. The problem

The overall reaction in a PEM fuel cell is given by

$$H_2 + \frac{1}{2}O_2 = H_2O(\ell)$$

where the ℓ, standing for liquid water, is commensurate with 100% relative humidity assumed for the simulations (see Table 1 later in Section 4). The thermodynamic potential, E_{Nernst} (also referred to as the emf or the Nernst voltage or the reversible voltage), of a single PEM fuel cell is given by the Nernst equation [13]:

$$E_{\text{Nernst}} = 1.229 - 0.85 \times 0.001(T - 298.15) + 4.3085 \times 10^{-5} T \ln\left(p_{H_2}\sqrt{p_{O_2}}\right) \quad (1)$$

where T is the cell temperature (K), p_{H_2} is the partial pressure of hydrogen at the anode catalyst/gas interface (atm), and p_{O_2} is the partial pressure of oxygen at the cathode catalyst/gas interface (atm) [16].

The partial pressures change with cell current [7] and are given by [1]

$$p_{H_2} = 0.5 RH_a \times p^{\text{sat}}_{H_2O}\left(\frac{1}{\frac{RH_a p^{\text{sat}}_{H_2O}}{p_a}\exp\left(\frac{1.635(i/A)}{T^{1.334}}\right)} - 1\right), \quad (2)$$

$$p_{O_2} = RH_c \times p^{\text{sat}}_{H_2O}\left(\frac{1}{\frac{RH_c p^{\text{sat}}_{H_2O}}{p_c}\exp\left(\frac{4.192(i/A)}{T^{1.334}}\right)} - 1\right). \quad (3)$$

The saturation pressure of water vapor (atm) is given as a function of the cell temperature, T [19,18]:

$$\log_{10}\left(p^{\text{sat}}_{H_2O}\right) = 2.95 \times 10^{-2}(T - 273.15) - 9.19 \times 10^{-5}(T - 273.15)^2 + 1.44 \times 10^{-7}(T - 273.15)^3 - 2.18. \quad (4)$$

The following types of losses (or voltage drops, or irreversibilities) are generally considered in the literature (see, e.g., [13,2,3,18,20]):

- Activation loss, caused by the slowness of the electrochemical reactions taking place on the surface of the electrode [1,16]:

$$\eta_{\text{act}} = -\left[x_1 + x_2 T + x_3 T \ln(C_{O_2}) + x_4 T \ln(i)\right] \quad (5)$$

where x_1,x_2,x_3,x_4 are parametric coefficients for the PEMFC, and

$$C_{O_2} = \frac{p_{O_2}}{5.08 \times 10^6 \exp(-498/T)}. \quad (6)$$

- Concentration loss (or mass transport loss) that results from the change in concentration of the reactants at the surface of the electrodes as the fuel is used:

$$\eta_{\text{conc}} = -B \ln\left(1 - \frac{i_{\text{den}}}{i_{\text{limit,den}}}\right) \quad (7)$$

where B is a parametric coefficient (V), i_{den} is the current density ($i_{\text{den}} = i/A$), and $i_{\text{limit,den}}$ the limiting current density.

Table 1
PEMFC stack parameter values and operating conditions.

Parameter	Value
N_s	24
A	27 cm^2
ℓ (Nafion 115:5 mil)	127 μm
$i_{\text{limit,den}}$	860 mA/cm^2
RH_a	1
RH_c	1
T	353.15 K
p_a	3 bar
p_c	5 bar

- Ohmic loss due to the electrical resistance of the electrodes, the polymer membrane, and the conducting resistance between the membrane and the electrodes:

$$\eta_{\text{ohm}} = i(R_M + R_C) \tag{8}$$

where R_C is the resistance (assumed constant) to the transfer of protons through the membrane, and R_M the equivalent resistance of the membrane given by

$$R_M = \frac{\rho_M \ell}{A} \tag{9}$$

where ℓ is the membrane thickness, A is the cell active area, and ρ_M is the specific resistivity given by [16]

$$\rho_M = \frac{181.6\left\{1 + 0.03\left(\frac{i}{A}\right) + 0.062\left(\frac{T}{303}\right)\left(\frac{i}{A}\right)^{2.5}\right\}}{\left\{\lambda - 0.634 - 3\left(\frac{i}{A}\right)\right\}\exp\left\{4.18\left(\frac{T-303}{T}\right)\right\}} \tag{10}$$

Combining all these losses, the terminal voltage, $V_{\text{cell,}}$ of a single cell is given by

$$V_{\text{cell}} = E_{\text{Nernst}} - \eta_{\text{act}} - \eta_{\text{ohm}} - \eta_{\text{conc}}. \tag{11}$$

(The loss due to fuel crossover and internal currents [13] is not considered in this model.) Now, for N_s cells connected in series to form a stack, the terminal (load) voltage of the stack is given by

$$V_{\text{stack}} = N_s V_{\text{cell}}. \tag{12}$$

The problem that this paper addresses is how to find optimal values of the seven parameters, x_1,x_2,x_3,x_4,λ,R_C,B, such that a model that best fits a given PEMFC stack can be created.

3. Methodology

3.1. Solution encoding and initialization

A PEMFC model is encoded for differential evolution as a seven-component vector representing the seven parameters to be optimized. Initialization of candidate solutions is done with values chosen uniformly randomly from predetermined lower and upper bounds. Two different sets of bounds (Table 3), taken from Refs. [18,20], are used for this purpose.

3.2. Fitness function

The problem is framed as one of minimization. The fitness is obtained as the sum-squared error (SSE) between the experimental stack voltage and model stack voltage:

$$\sum_{j=1}^{J}\left(V_{\text{stack-actual},j} - V_{\text{stack-model},j}\right)^2. \tag{13}$$

$V_{\text{stack-model}}$ is computed from equation (12), using the DE-evolved values of the seven parameters. $V_{\text{stack-actual}}$ is assumed known or is obtained from experiments on a physical fuel cell stack (or on a simulated stack, as in this paper). In this paper, the word “experiment” means numerical experiment (simulation by a computer program). Clearly, the best possible fitness is zero.

3.3. The algorithm

The DE is a population-based stochastic search heuristic where a population of candidate solutions (trial chromosomes) are evolved with time, by the application of carefully-designed operators, hopefully culminating in the finding of a near-optimal or optimal solution. Let $P_G = [\vec{X}_{1,G}, \vec{X}_{2,G}, \ldots, \vec{X}_{NP,G}]$ represent a DE population where each $\vec{X}_{i,G}(i = 1,2,\ldots,NP)$, is a D-dimensional parameter vector. The vector indices are arranged randomly (as obtained during initialization) in order to preserve the diversity of each neighborhood. For each member of the population, $\vec{X}_{j,G}$, a donor vector $\vec{V}_{j,G}$ is created by employing the best (fittest) vector in the population and any two other vectors chosen from the population:

$$\vec{V}_{j,G} = \vec{X}_{\text{best},G} + F\left(\vec{X}_{r_1^j,G} - \vec{X}_{r_2^j,G}\right), \tag{14}$$

where $r_1^j, r_2^j \in [1, NP]$ with $r_1^j \neq r_2^j \neq j$. F is the scaling factor. The above strategy, referred to as DE/best/1, can be changed to DE/rand/1 by replacing $\vec{X}_{best,G}$ in the above equation with a randomly chosen one, $\vec{X}_{r,G}$, from the population.

The donor vector $\vec{V}_{j,G}$ is crossed (see the next sub-section) with the target vector $\vec{X}_{j,G}$, producing an offspring vector. The offspring replaces the target vector if the offspring is equal in fitness to, or better than, the target [5,23,8].

3.4. Crossover

To increase the potential diversity of the population, a crossover operation is used after the donor vector has been generated

Table 2
Parameters used to generate $V_{\text{stack-actual}}$.

x_1	x_2	x_3	x_4	λ	R_C (Ω)	B (V)
−0.944957	0.00301801	7.401×10^{-5}	-1.88×10^{-4}	23	0.0001	0.02914489

Table 3
Bounds of model parameters.

Parameter	x_1	x_2	x_3	x_4	λ	R_C (Ω)	B (V)
Ohenoja-Leiviska [20]							
Upper bound	−0.8532	0.005	9.8×10^{-5}	-9.54×10^{-5}	24	0.0008	0.5
Lower bound	−1.19969	0.001	3.6×10^{-5}	-2.60×10^{-4}	10	0.0001	0.0136
Mo et al. [18]							
Upper bound	−0.944	0.005	7.8×10^{-5}	-1.88×10^{-4}	23	0.0008	0.5
Lower bound	−0.952	0.001	7.4×10^{-5}	-1.98×10^{-4}	14	0.0001	0.016

Table 4
Experimental results: two DEGL/SAW variants (high noise, target fitness 1.578).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (1.578)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
DEGL/SAW/bin	$k = 6, \alpha = \beta = 0.7$, Cr = 0.7	1.57677	9.03362e-16	0	3970.83	587.526	0.04	0.00787839
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.8	1.57677	9.03362e-16	0	2797.73	431.758	0.0276667	0.00678911
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.9	1.57677	9.03362e-16	0	1841.43	297.342	0.0163333	0.00490133
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.7	1.57677	9.03362e-16	0	4434.07	836.822	0.0473333	0.0111211
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.8	1.57677	9.03362e-16	0	3177.57	841.349	0.0306667	0.0114269
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.9	1.57677	9.03362e-16	0	2067.23	509.835	0.0193333	0.00583292
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.7	1.57677	9.03362e-16	0	5954.80	1411.010	0.0633333	0.0156102
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.8	1.57677	9.03362e-16	0	4136.20	759.909	0.042	0.0109545
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.9	1.57677	9.03362e-16	0	2769.00	514.092	0.025	0.00731083
DEGL/SAW/exp	$k = 6, \alpha = \beta = 0.7$, Cr = 0.7	1.57677	9.03362e-16	0	3229.87	639.202	0.033	0.00876907
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.8	1.57677	9.03362e-16	0	3040.57	635.444	0.0286667	0.00819307
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.9	1.57677	9.03362e-16	0	3121.67	705.006	0.0306667	0.00907187
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.7	1.57677	9.03362e-16	0	3382.27	842.427	0.032	0.00996546
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.8	1.57677	9.03362e-16	0	3674.93	627.832	0.0353333	0.00819307
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.9	1.57677	9.03362e-16	0	3667.27	775.060	0.0373333	0.00944433
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.7	1.57677	9.03362e-16	0	4955.83	1097.370	0.0506667	0.0125762
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.8	1.57677	9.03362e-16	0	4675.50	996.573	0.0493333	0.0120153
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.9	1.57677	9.03362e-16	0	4513.50	1025.650	0.0453333	0.0133218

through mutation. The donor vector $\overrightarrow{V}_{j,G}$ exchanges its components with the target vector $\overrightarrow{X}_{j,G}$ to create what is known as the trial vector, $\overrightarrow{U}_{j,G}$. Binomial crossover is performed on each of the D components whenever a randomly picked number between 0 and 1 is less than or equal to the C_r (crossover rate) value. In this case, the number of components inherited from the donor has a (nearly) binomial distribution. A second type of crossover, exponential crossover, chooses a random starting point in the target vector and a random length for exchanging components with the donor vector. If the trial vector yields an equal or better value of the fitness

Table 5
Experimental results: traditional DE variants (high noise, target fitness 1.578).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (1.578)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
DE/rand/1/bin	$F = 0.7$, Cr = 0.7	1.57677	9.03362e-16	0	16378.7	2502.2	0.176	0.0287198
	$F = 0.7$, Cr = 0.8	1.57677	9.03362e-16	0	11347	1480.39	0.122	0.0180803
	$F = 0.7$, Cr = 0.9	1.57677	9.03362e-16	0	7162.9	1050.79	0.0753333	0.0127937
	$F = 0.8$, Cr = 0.7	1.57677	9.03362e-16	0	21110.4	3855.08	0.228	0.0423776
	$F = 0.8$, Cr = 0.8	1.57677	9.03362e-16	0	14652.6	1696.22	0.158667	0.0188887
	$F = 0.8$, Cr = 0.9	1.57677	9.03362e-16	0	9934.9	1432.37	0.105333	0.0159164
	$F = 0.9$, Cr = 0.7	1.57677	9.03362e-16	0	28780.2	4343.3	0.313333	0.0485893
	$F = 0.9$, Cr = 0.8	1.57677	9.03362e-16	0	18894.1	2596.04	0.204	0.0294314
	$F = 0.9$, Cr = 0.9	1.57677	9.03362e-16	0	13543.5	1520.19	0.143333	0.0170867
DE/rand/1/exp	$F = 0.7$, Cr = 0.7	1.57677	9.03362e-16	0	12788.2	1603.22	0.136	0.0165258
	$F = 0.7$, Cr = 0.8	1.57677	9.03362e-16	0	12832.4	2073.45	0.135667	0.0235889
	$F = 0.7$, Cr = 0.9	1.57677	9.03362e-16	0	13208.9	1651.3	0.141333	0.0179527
	$F = 0.8$, Cr = 0.7	1.57677	9.03362e-16	0	17517.1	2525.31	0.187	0.0286657
	$F = 0.8$, Cr = 0.8	1.57677	9.03362e-16	0	17826.1	2659.31	0.191	0.0296357
	$F = 0.8$, Cr = 0.9	1.57677	9.03362e-16	0	17182.9	2376.86	0.184667	0.026747
	$F = 0.9$, Cr = 0.7	1.57677	9.03362e-16	0	22376.3	3509.58	0.240333	0.0399554
	$F = 0.9$, Cr = 0.8	1.57677	9.03362e-16	0	22481	2909.93	0.242	0.0327372
	$F = 0.9$, Cr = 0.9	1.57677	9.03362e-16	0	22553.7	2724.23	0.243	0.0307549
DE/best/1/bin	$F = 0.7$, Cr = 0.7	1.62435	0.0877117	7	4704.35	1218.69	0.0452174	0.0123838
	$F = 0.7$, Cr = 0.8	1.59037	0.0517305	2	2961.64	869.778	0.0278571	0.0110075
	$F = 0.7$, Cr = 0.9	1.60396	0.0704976	4	2128.19	547.133	0.0176923	0.00651625
	$F = 0.8$, Cr = 0.7	1.59716	0.0622141	3	6872.96	1718.64	0.0718519	0.020388
	$F = 0.8$, Cr = 0.8	1.62435	0.0877143	7	4484.13	1150.1	0.0434783	0.0126522
	$F = 0.8$, Cr = 0.9	1.61755	0.0829519	6	3072.71	689.402	0.0295833	0.0080645
	$F = 0.9$, Cr = 0.7	1.61755	0.0829543	6	9637.92	2365.9	0.10125	0.0262616
	$F = 0.9$, Cr = 0.8	1.66019	0.104132	12	6751.5	1888.29	0.0694444	0.0204284
	$F = 0.9$, Cr = 0.9	1.65743	0.133988	10	4326.35	886.948	0.043	0.010311
DE/best/1/exp	$F = 0.7$, Cr = 0.7	1.57677	1.27802e-05	0	3309.47	1076.45	0.0296667	0.0109807
	$F = 0.7$, Cr = 0.8	1.63594	0.12441	7	3452.61	860.789	0.0321739	0.0116605
	$F = 0.7$, Cr = 0.9	1.60396	0.0704976	4	3456.27	1162.82	0.0296154	0.0121592
	$F = 0.8$, Cr = 0.7	1.63794	0.0950392	9	4996.38	1388.28	0.0485714	0.0127615
	$F = 0.8$, Cr = 0.8	1.60396	0.0704967	4	5651.12	1944.2	0.0565385	0.0218984
	$F = 0.8$, Cr = 0.9	1.60396	0.0704948	4	4914.31	1576.51	0.0488462	0.0172805
	$F = 0.9$, Cr = 0.7	1.68461	0.135596	14	7769.81	2421.9	0.0775	0.0274469
	$F = 0.9$, Cr = 0.8	1.67435	0.149956	12	8794.39	3493.37	0.0911111	0.0392412
	$F = 0.9$, Cr = 0.9	1.67872	0.103693	15	8508	2200.23	0.088	0.0239643

Table 6
Experimental results: genetic algorithm (high noise, target fitness 1.578).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (1.578)			CPU Seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
GA	pM = 0.02, pC = 0.7	1.57991	0.00617904	11	287291	116454	3.17895	1.29888
	pM = 0.02, pC = 0.8	1.58098	0.0150071	11	317961	117065	3.51105	1.29438
	pM = 0.02, pC = 0.9	1.58097	0.0155307	11	243287	94913.7	2.69368	1.05391
	pM = 0.04, pC = 0.7	1.57748	0.000944315	3	184932	122786	2.04852	1.36379
	pM = 0.04, pC = 0.8	1.57723	0.000628544	4	171758	112654	1.90154	1.24874
	pM = 0.04, pC = 0.9	1.57729	0.000612651	3	180570	106035	2.00037	1.18175
	pM = 0.06, pC = 0.7	1.57713	0.0004759	3	112387	78218.2	1.24333	0.872829
	pM = 0.06, pC = 0.8	1.57709	0.000317366	1	128754	105258	1.43276	1.17183
	pM = 0.06, pC = 0.9	1.57699	0.000325912	2	98605.7	64505.3	1.08893	0.718013

function, it replaces the corresponding target vector in the next generation; otherwise the target is retained in the population.

If a target vector $\vec{X}_{j,G}$ is replaced with the corresponding trial vector $\vec{U}_{j,G}$, the best vector in the population $\vec{X}_{\text{best},G}$ may also be updated by $\vec{U}_{j,G}$, provided the latter yields a better value of the objective function.

3.5. Improved differential evolution

A proper trade-off between exploration and exploitation is necessary for efficient operation of a population-based stochastic search technique such as the DE. The DE/target-to-best/1, in its present form, favors exploitation only, since all the vectors are attracted by the same best position found so far by the entire population, thereby converging faster towards the same point.

For the PEMFC optimization problem, two kinds of neighborhood models are used for the DE. The first one is a local neighborhood model, where each vector is mutated using the best position found so far in a small neighborhood of it and not in the entire population. The second one, a global mutation model, takes into account the globally best vector of the entire population at current generation for mutating a population member.

A vector's neighborhood is the set of other parameter vectors that it is connected to; it considers their experience when updating its position. The graph of inter-connections describes the neighborhood structure. Generally, neighborhood connections are independent of the positions pointed to by the vectors. In the local model, whenever a parameter vector points to a good region of the search space, it only directly influences its immediate neighbors. Its second degree neighbors will only be influenced after those directly connected to them become highly successful themselves. Thus, there is a delay in the information spread through the population regarding the best position of each neighborhood. Therefore, the attraction to specific points is weaker, which prevents the population from getting trapped in local minima. Vectors belonging to a local neighborhood are not necessarily local in the sense of their geographical proximity or similarity of fitness values. Overlapping neighborhoods are created in DE according to the order of the indices of the population members.

Finally, the local and the global models are combined using a weight factor that appears as a new parameter in the algorithm. The weight factor may be tuned in many different ways. The neighborhoods of different vectors are chosen randomly and not according to their fitness values or geographical locations on the fitness landscape. This preserves the diversity of the vectors belonging to the same neighborhood. The mutation algorithm is explained in the next sub-section.

3.5.1. Improved mutation

For every vector $\vec{X}_{i,G}$ a neighborhood of radius k is defined (where k is a non-zero integer from 0 to $(NP-1)/2$, as the neighborhood size must be smaller than the population size, i.e. $2k+1 \leq NP$), consisting of vectors $\vec{X}_{i-k,G}, \ldots, \vec{X}_{i,G}, \ldots, \vec{X}_{i+k,G}$.

Table 7
Experimental results: two DEGL/SAW variants (high noise, target fitness 1.8864).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (1.8864)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
DEGL/SAW/bin	$k = 6, \alpha = \beta = 0.7$, Cr = 0.7	1.57677	9.03362e-16	0	568.967	158.626	0.000666667	0.00253708
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.8	1.57677	9.03362e-16	0	449.567	149.904	0.000333333	0.00182574
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.9	1.57677	9.03362e-16	0	411.8	126.458	0.000333333	0.00182574
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.7	1.57677	9.03362e-16	0	597.4	166.306	0.00166667	0.00379049
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.8	1.57677	9.03362e-16	0	537.367	198.152	0	0
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.9	1.57677	9.03362e-16	0	475.967	103.988	0	0
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.7	1.57677	9.03362e-16	0	767.1	270.66	0.00166667	0.00379049
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.8	1.57677	9.03362e-16	0	634.967	254.846	0.000666667	0.00253708
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.9	1.57677	9.03362e-16	0	574.1	175.563	0.000333333	0.00182574
DEGL/SAW/exp	$k = 6, \alpha = \beta = 0.7$, Cr = 0.7	1.57677	9.03362e-16	0	466.533	139.749	0.000666667	0.00253708
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.8	1.57677	9.03362e-16	0	463.1	119.16	0	0
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.9	1.57677	9.03362e-16	0	517.867	151.082	0	0
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.7	1.57677	9.03362e-16	0	535.433	154.51	0	0
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.8	1.57677	9.03362e-16	0	541.167	191.305	0	0
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.9	1.57677	9.03362e-16	0	577.533	159.69	0.001	0.00305129
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.7	1.57677	9.03362e-16	0	645.267	226.88	0	0
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.8	1.57677	9.03362e-16	0	582.367	248.955	0.00133333	0.00345746
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.9	1.57677	9.03362e-16	0	678.3	281.233	0.00133333	0.00345746

Table 8
Experimental results: traditional DE variants (high noise, target fitness 1.8864).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (1.8864)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
DE/rand/1/bin	$F = 0.7$, Cr = 0.7	1.57677	9.03362e-16	0	1457.27	538.702	0.0103333	0.0076489
	$F = 0.7$, Cr = 0.8	1.57677	9.03362e-16	0	1322.2	418.684	0.009	0.00711967
	$F = 0.7$, Cr = 0.9	1.57677	9.03362e-16	0	1158.43	372.657	0.00833333	0.00592093
	$F = 0.8$, Cr = 0.7	1.57677	9.03362e-16	0	1987.03	822.659	0.0163333	0.010662
	$F = 0.8$, Cr = 0.8	1.57677	9.03362e-16	0	1630.5	583.966	0.0123333	0.008172
	$F = 0.8$, Cr = 0.9	1.57677	9.03362e-16	0	1339.27	512.284	0.0103333	0.00808717
	$F = 0.9$, Cr = 0.7	1.57677	9.03362e-16	0	2638.27	1199.42	0.0236667	0.0147352
	$F = 0.9$, Cr = 0.8	1.57677	9.03362e-16	0	2047.67	710.878	0.0173333	0.00944433
	$F = 0.9$, Cr = 0.9	1.57677	9.03362e-16	0	1837.73	646.086	0.015	0.00820008
DE/rand/1/exp	$F = 0.7$, Cr = 0.7	1.57677	9.03362e-16	0	1354.63	532.872	0.0113333	0.00730297
	$F = 0.7$, Cr = 0.8	1.57677	9.03362e-16	0	1488.27	551.432	0.0106667	0.00827682
	$F = 0.7$, Cr = 0.9	1.57677	9.03362e-16	0	1204.2	421.779	0.00766667	0.00678911
	$F = 0.8$, Cr = 0.7	1.57677	9.03362e-16	0	1722.57	608.139	0.0136667	0.00808717
	$F = 0.8$, Cr = 0.8	1.57677	9.03362e-16	0	1618.9	616.207	0.0116667	0.00874281
	$F = 0.8$, Cr = 0.9	1.57677	9.03362e-16	0	1694.17	583.031	0.014	0.00770132
	$F = 0.9$, Cr = 0.7	1.57677	9.03362e-16	0	2091.57	637.116	0.0166667	0.00758098
	$F = 0.9$, Cr = 0.8	1.57677	9.03362e-16	0	1967.97	753.961	0.0163333	0.00999425
	$F = 0.9$, Cr = 0.9	1.57677	9.03362e-16	0	2383.6	844.639	0.0196667	0.00964305
DE/best/1/bin	$F = 0.7$, Cr = 0.7	1.62435	0.0877117	0	783.8	332.964	0.00266667	0.00449776
	$F = 0.7$, Cr = 0.8	1.59037	0.0517305	0	643	259.416	0.00133333	0.00345746
	$F = 0.7$, Cr = 0.9	1.60396	0.0704976	0	489.467	171.895	0	0
	$F = 0.8$, Cr = 0.7	1.59716	0.0622141	0	968.767	413.771	0.00533333	0.0062881
	$F = 0.8$, Cr = 0.8	1.62435	0.0877143	0	877.033	268.139	0.002	0.00406838
	$F = 0.8$, Cr = 0.9	1.61755	0.0829519	0	620.2	168.214	0.00133333	0.00345746
	$F = 0.9$, Cr = 0.7	1.61755	0.0829543	0	1267.73	566.633	0.008	0.00886683
	$F = 0.9$, Cr = 0.8	1.66019	0.104132	0	1010.43	411.192	0.00633333	0.00668675
	$F = 0.9$, Cr = 0.9	1.65743	0.133988	1	976.379	337.202	0.00551724	0.00572351
DE/best/1/exp	$F = 0.7$, Cr = 0.7	1.57677	1.27802e-05	0	564.133	196.552	0	0
	$F = 0.7$, Cr = 0.8	1.63594	0.12441	1	582.138	226.786	0.00137931	0.00441114
	$F = 0.7$, Cr = 0.9	1.60396	0.0704976	0	582.433	209.574	0.000666667	0.00253708
	$F = 0.8$, Cr = 0.7	1.63794	0.0950392	0	785.967	268.2	0.00166667	0.00379049
	$F = 0.8$, Cr = 0.8	1.60396	0.0704967	0	703.533	238.824	0.00166667	0.00379049
	$F = 0.8$, Cr = 0.9	1.60396	0.0704948	0	753.667	277.155	0.00266667	0.00449776
	$F = 0.9$, Cr = 0.7	1.68461	0.135596	1	1117.34	407.303	0.00586207	0.00682288
	$F = 0.9$, Cr = 0.8	1.67435	0.149956	2	1284.36	532.158	0.00785714	0.00786796
	$F = 0.9$, Cr = 0.9	1.67872	0.103693	0	1139.37	582.26	0.00566667	0.00727932

The vectors are assumed to be organized on a ring topology with respect to their indices, such that vectors $\overrightarrow{X}_{NP,G}$ and $\overrightarrow{X}_{2,G}$ are the two immediate neighbors of vector $\overrightarrow{X}_{1,G}$. The neighborhood topology is static and is defined on the set of indices of the vectors. Although various neighborhood topologies (e.g., star, wheel, pyramid, 4-clusters, and circular) have been proposed in the literature for the particle swarm optimization (PSO) algorithms, after some initial experimentation over numerical benchmarks, it was observed that in the case of DE (where the population size is usually larger than in the case of PSO) the circular or ring topology provides better performance compared to other neighborhood structures.

For each member of the population, a local donor vector is created by employing the best (fittest) vector in the neighborhood of that member and any two other vectors chosen from the same neighborhood. The model may be expressed as:

$$\overrightarrow{L}_{i,G} = \overrightarrow{X}_{i,G} + \alpha(\overrightarrow{X}_{n\text{best}_i,G} - \overrightarrow{X}_{i,G}) + \beta(\overrightarrow{X}_{p,G} - \overrightarrow{X}_{q,G}), \tag{15}$$

where the subscript $n\text{best}_i$ indicates the best vector in the neighborhood of $\overrightarrow{X}_{i,G}$ and $p,q \in [i-k, i+k]$ with $p \neq q \neq i$. Similarly the global donor vector is created as:

$$\overrightarrow{g}_{i,G} = \overrightarrow{X}_{i,G} + \alpha\left(\overrightarrow{X}_{g\text{best},G} - \overrightarrow{X}_{i,G}\right) + \beta(\overrightarrow{X}_{r_1,G} - \overrightarrow{X}_{r_2,G}), \tag{16}$$

where the subscript $g$best indicates the best vector in the entire population at generation G, and $r_1,r_2 \in [1,NP]$ with $r_1 \neq r_2 \neq i$. α and β are the scaling factors.

Table 9
Experimental results: genetic algorithm (high noise, target fitness 1.8864).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (1.8864)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
GA	pM = 0.02, pC = 0.7	1.57991	0.00617904	0	9574.9	18615.3	0.100333	0.205971
	pM = 0.02, pC = 0.8	1.58098	0.0150071	0	10048.5	17971	0.106	0.197495
	pM = 0.02, pC = 0.9	1.58097	0.0155307	0	5358.33	9910.03	0.0543333	0.10944
	pM = 0.04, pC = 0.7	1.57748	0.000944315	0	4254	7641.13	0.041	0.0845414
	pM = 0.04, pC = 0.8	1.57723	0.000628544	0	3777.67	7370.22	0.0366667	0.08172
	pM = 0.04, pC = 0.9	1.57729	0.000612651	0	2494.3	4211.39	0.0216667	0.0477842
	pM = 0.06, pC = 0.7	1.57713	0.0004759	0	1991.8	3329.35	0.0166667	0.0361351
	pM = 0.06, pC = 0.8	1.57709	0.000317366	0	2077.27	1874.85	0.019	0.0204011
	pM = 0.06, pC = 0.9	1.57699	0.000325912	0	1421.67	1648.09	0.00966667	0.0190251

Table 10
Experimental results: two DEGL/SAW variants (high noise, second set of bounds, target fitness 1.8864).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (1.8864)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
DEGL/SAW/bin	$k = 6$, $\alpha = \beta = 0.7$, Cr $= 0.7$	1.88629	9.03362e-16	0	2368.53	371.778	0.032	0.00714384
	$k = 6$, $\alpha = \beta = 0.7$, Cr $= 0.8$	1.88629	9.03362e-16	0	1972.7	263.688	0.0276667	0.00626062
	$k = 6$, $\alpha = \beta = 0.7$, Cr $= 0.9$	1.88629	9.03362e-16	0	1585.8	216.546	0.0206667	0.00365148
	$k = 6$, $\alpha = \beta = 0.8$, Cr $= 0.7$	1.88629	9.03362e-16	0	2421.37	522.236	0.0346667	0.00899553
	$k = 6$, $\alpha = \beta = 0.8$, Cr $= 0.8$	1.88629	9.03362e-16	0	1965.2	406.228	0.0266667	0.00711159
	$k = 6$, $\alpha = \beta = 0.8$, Cr $= 0.9$	1.88629	9.03362e-16	0	1520.57	252.089	0.0206667	0.00449776
	$k = 6$, $\alpha = \beta = 0.9$, Cr $= 0.7$	1.88629	9.03362e-16	0	2876.37	601.92	0.0416667	0.0105318
	$k = 6$, $\alpha = \beta = 0.9$, Cr $= 0.8$	1.88629	9.03362e-16	0	2323.33	394.228	0.0326667	0.00827682
	$k = 6$, $\alpha = \beta = 0.9$, Cr $= 0.9$	1.88629	9.03362e-16	0	1861.27	323.84	0.0253333	0.00681445
DEGL/SAW/exp	$k = 6$, $\alpha = \beta = 0.7$, Cr $= 0.7$	1.88629	9.03362e-16	0	1884.17	297.61	0.0256667	0.00678911
	$k = 6$, $\alpha = \beta = 0.7$, Cr $= 0.8$	1.88629	9.03362e-16	0	1853.6	347.356	0.0253333	0.00819307
	$k = 6$, $\alpha = \beta = 0.7$, Cr $= 0.9$	1.88629	9.03362e-16	0	1826.03	283.479	0.024	0.00674665
	$k = 6$, $\alpha = \beta = 0.8$, Cr $= 0.7$	1.88629	9.03362e-16	0	1874.37	291.614	0.0243333	0.00568321
	$k = 6$, $\alpha = \beta = 0.8$, Cr $= 0.8$	1.88629	9.03362e-16	0	1950.93	292.384	0.0283333	0.00698932
	$k = 6$, $\alpha = \beta = 0.8$, Cr $= 0.9$	1.88629	9.03362e-16	0	1955.97	287.799	0.0286667	0.00571346
	$k = 6$, $\alpha = \beta = 0.9$, Cr $= 0.7$	1.88629	9.03362e-16	0	2209.47	386.275	0.0316667	0.00791478
	$k = 6$, $\alpha = \beta = 0.9$, Cr $= 0.8$	1.88629	9.03362e-16	0	2270.07	401.462	0.0326667	0.00868345
	$k = 6$, $\alpha = \beta = 0.9$, Cr $= 0.9$	1.88629	9.03362e-16	0	2074.9	502.075	0.0293333	0.0101483

In the above two equations, the first perturbation term on the right hand side (the one multiplied by α) is an arithmetical recombination operation, while the second term (the one multiplied by β) is the differential mutation. Thus in both the global and local mutation models, mutated recombinants, not pure mutants, are generated.

The local and global donor vectors are combined using a scalar weight $w \in (0,1)$ to form the actual donor vector of the proposed algorithm:

$$\vec{V}_{i,G} = w\vec{g}_{i,G} + (1-w)\vec{L}_{i,G} \tag{17}$$

Table 11
Experimental results: traditional DE variants (high noise, second set of bounds, target fitness 1.8864).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (1.8864)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
DE/rand/1/bin	$F = 0.7$, Cr $= 0.7$	1.88629	9.03362e-16	0	5153.53	1051.51	0.0746667	0.0171672
	$F = 0.7$, Cr $= 0.8$	1.88629	9.03362e-16	0	4667.53	874.562	0.0663333	0.0129943
	$F = 0.7$, Cr $= 0.9$	1.88629	9.03362e-16	0	4579.13	558.609	0.0656667	0.0085836
	$F = 0.8$, Cr $= 0.7$	1.89283	0.0358466	1	5983.21	934.269	0.0858621	0.0152403
	$F = 0.8$, Cr $= 0.8$	1.90592	0.0599089	3	5659.04	880.677	0.08	0.0141421
	$F = 0.8$, Cr $= 0.9$	1.88629	9.03362e-16	0	4734.63	952.261	0.0686667	0.014077
	$F = 0.9$, Cr $= 0.7$	1.89283	0.0358466	1	6830.1	1216.28	0.10069	0.0185031
	$F = 0.9$, Cr $= 0.8$	1.90592	0.0599089	3	6147.48	967.84	0.0896296	0.0160484
	$F = 0.9$, Cr $= 0.9$	1.89938	0.0498131	2	5537.93	975.581	0.0796429	0.0150264
DE/rand/1/exp	$F = 0.7$, Cr $= 0.7$	1.88629	9.03362e-16	0	4392.93	676.041	0.062	0.0121485
	$F = 0.7$, Cr $= 0.8$	1.88629	9.03362e-16	0	4625.77	942.82	0.066	0.0161031
	$F = 0.7$, Cr $= 0.9$	1.88629	9.03362e-16	0	4526.53	765.141	0.065	0.0113715
	$F = 0.8$, Cr $= 0.7$	1.88629	9.03362e-16	0	4799.13	808.474	0.069	0.0129588
	$F = 0.8$, Cr $= 0.8$	1.89283	0.0358466	1	5377.55	666.506	0.0768966	0.0119832
	$F = 0.8$, Cr $= 0.9$	1.88629	9.03362e-16	0	4841.13	877.159	0.07	0.0128654
	$F = 0.9$, Cr $= 0.7$	1.89938	0.0498131	2	5902.07	1307.07	0.0842857	0.0200792
	$F = 0.9$, Cr $= 0.8$	1.88629	9.03362e-16	0	5930.97	958.235	0.0846667	0.0138298
	$F = 0.9$, Cr $= 0.9$	1.90592	0.0599089	3	5635.07	1173.18	0.0822222	0.0180455
DE/best/1/bin	$F = 0.7$, Cr $= 0.7$	1.9321	0.0844616	7	1841.04	321.297	0.0204348	0.00474654
	$F = 0.7$, Cr $= 0.8$	1.91247	0.0678842	4	1447.69	208.368	0.0176923	0.00429669
	$F = 0.7$, Cr $= 0.9$	1.96483	0.0978305	12	1281.5	181.121	0.015	0.00514496
	$F = 0.8$, Cr $= 0.7$	1.95174	0.0941378	10	2100.8	390.891	0.0275	0.00638666
	$F = 0.8$, Cr $= 0.8$	1.97137	0.0989562	13	1755.24	327.807	0.0188235	0.00485071
	$F = 0.8$, Cr $= 0.9$	1.98312	0.108959	14	1417.5	240.042	0.018125	0.00403113
	$F = 0.9$, Cr $= 0.7$	1.95828	0.0962326	11	2472.68	499.481	0.0315789	0.00764719
	$F = 0.9$, Cr $= 0.8$	1.98635	0.102285	15	2284.07	456.603	0.0293333	0.00798809
	$F = 0.9$, Cr $= 0.9$	2.02095	0.0978766	20	1822.7	514.595	0.021	0.00737865
DE/best/1/exp	$F = 0.7$, Cr $= 0.7$	1.92556	0.0798786	6	1448.08	394.147	0.0170833	0.00750604
	$F = 0.7$, Cr $= 0.8$	1.91901	0.0744232	5	1386.88	277.57	0.0168	0.00690411
	$F = 0.7$, Cr $= 0.9$	1.9321	0.0844621	7	1408.87	261.467	0.0169565	0.00470472
	$F = 0.8$, Cr $= 0.7$	1.95828	0.0962326	11	1676.89	328.233	0.02	0.00471405
	$F = 0.8$, Cr $= 0.8$	1.93865	0.0883078	8	1720.55	348.682	0.0227273	0.007025
	$F = 0.8$, Cr $= 0.9$	1.97792	0.0996255	14	1640.31	371.167	0.019375	0.0057373
	$F = 0.9$, Cr $= 0.7$	2.00409	0.0978309	18	2301.67	388.049	0.0275	0.00753778
	$F = 0.9$, Cr $= 0.8$	1.97792	0.0996252	14	2076.25	454.627	0.026875	0.00946485
	$F = 0.9$, Cr $= 0.9$	2.0041	0.0978299	18	2284.75	655.402	0.0291667	0.011645

Table 12
Experimental results: genetic algorithm (high noise, second set of bounds, target fitness 1.8864).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (1.8864)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
GA	pM = 0.02, pC = 0.7	1.88644	0.000268273	8	221216	140005	2.45136	1.54585
	pM = 0.02, pC = 0.8	1.88646	0.000312775	10	175238	114280	1.9525	1.27019
	pM = 0.02, pC = 0.9	1.88641	0.000146409	9	184262	118690	2.04857	1.32266
	pM = 0.04, pC = 0.7	1.88634	6.60373e-05	4	114746	102648	1.27423	1.1456
	pM = 0.04, pC = 0.8	1.88636	8.38883e-05	7	156458	145940	1.73261	1.61893
	pM = 0.04, pC = 0.9	1.88637	8.33742e-05	6	133116	144561	1.47083	1.60243
	pM = 0.06, pC = 0.7	1.88647	0.000513473	8	107302	119203	1.18591	1.32507
	pM = 0.06, pC = 0.8	1.88637	0.00014122	6	102471	103552	1.13375	1.14154
	pM = 0.06, pC = 0.9	1.88637	0.000119726	7	106923	102146	1.18348	1.13775

This version is called DEGL (DE with global and local neighborhoods) [8]. The rest of the algorithm is similar to DE/rand/1/bin. DEGL uses a binomial crossover scheme. When the weight factor w is made self-adaptive [8], a new variant, called DEGL/SAW (DEGL with self-adaptive weight), results.

In each generation, the vectors belonging to a DE population are perturbed sequentially. If a target vector $\vec{X}_{i,G}$ is replaced with the corresponding trial vector $\vec{U}_{i,G}$, the neighborhood-best $\vec{X}_{n\text{best},G}$ and the globally best vector $\vec{X}_{g\text{best},G}$ may also be updated by $\vec{U}_{i,G}$, provided the latter yields a better value of the objective function.

4. Simulation procedure and results

Six different variants of differential evolution were coded in C++ and executed: DEGL/SAW/bin, DEGL/SAW/exp, DE/rand/1/bin, DE/rand/1/exp, DE/best/1/bin, and DE/best/1/exp. Each of these six DEs was run, separately, with several parameter values, giving a total of nine parameter settings for each DE. (The problem parameters – x_1, x_2, etc. – are not to be confused with the algorithm parameters.) In addition, a three-operator genetic algorithm (GA) was also coded in C++ and run with nine different parameter sets. The programs were executed on the following platform: CPU: Intel Core i7-750 3.06 GHz (8 MB Cache); RAM: 6 GB DDR3-1600 (PC-12800) (CAS latency 8); Compiler: gcc 4.6.1; Operating System: Linux Mint 12; and Linux Kernel Version: 3.0.0.

Table 1 lists the stack parameters and operating conditions used for running the simulations in this paper. Table 2 shows the parameter values used to generate (via equation (12)) the V_{stack} values which, in turn, are used via the following equation to obtain the $V_{\text{stack-actual}}$ values:

$$V_{\text{stack-actual}} = V_{\text{stack}} + \mathcal{N}(0,\sigma) \tag{18}$$

where $\mathcal{N}(0,\sigma)$ represents a Gaussian noise with mean zero and standard deviation σ. Two levels of noise – high ($\sigma = 1/3$) and low ($\sigma = 1/6$) – are used in this paper. (The parameter values in Tables 1 and 2 are taken from Ref. [18]). (Recall that $V_{\text{stack-model}}$ is calculated as V_{stack} using equation (12).) For computation of V_{stack} from equation (12), the value of E_{Nernst} was fixed at 1.197374 V, a value that was obtained by putting $i = 0$ (open circuit) in equations (2) and (3) and plugging in the resulting values of the two partial pressures in equation (1).

Recall from Section 3.1 that two separate sets of bounds (Table 3) are used for chromosome initialization. The population size was fixed at 100 for the genetic algorithm and 70 for differential evolution. There were a total of 7 × 9 = 63 variants, and each variant of each algorithm was executed for 30 independent runs (by varying the seed of the random number generator). No systematic (algorithm) parameter tuning was attempted; after some preliminary runs, (algorithm) parameter settings that gave reasonably good solutions were used. For obtaining the errors in the calculation of fitness (equation (13)), fifteen pairs of current and voltage

Table 13
Experimental results: two DEGL/SAW variants (low noise, target fitness 0.4).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (0.4)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
DEGL/SAW/bin	$k = 6, \alpha = \beta = 0.7$, Cr = 0.7	0.392983	2.25841e-16	0	3019.83	651.26	0.0303333	0.00850287
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.8	0.392983	2.25841e-16	0	2303.27	513.273	0.021	0.00607425
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.9	0.392983	2.25841e-16	0	1564.53	361.487	0.012	0.00610257
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.7	0.392983	2.25841e-16	0	3481.4	725.123	0.0353333	0.0100801
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.8	0.392983	2.25841e-16	0	2515.7	608.144	0.0236667	0.00668675
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.9	0.392983	2.25841e-16	0	1778	445.079	0.015	0.00682288
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.7	0.392983	2.25841e-16	0	4700.67	1511.05	0.0513333	0.0187052
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.8	0.392983	2.25841e-16	0	2866.07	945.667	0.0283333	0.0108543
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.9	0.392983	2.25841e-16	0	2049.13	425.385	0.019	0.0048066
DEGL/SAW/exp	$k = 6, \alpha = \beta = 0.7$, Cr = 0.7	0.392983	2.25841e-16	0	2573.7	707.786	0.025	0.00937715
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.8	0.392983	2.25841e-16	0	2531	943.902	0.0233333	0.011547
	$k = 6, \alpha = \beta = 0.7$, Cr = 0.9	0.392983	2.25841e-16	0	2372.83	616.086	0.0216667	0.00791478
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.7	0.392983	2.25841e-16	0	3040	880.668	0.028	0.0103057
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.8	0.392983	2.25841e-16	0	2604.03	533.184	0.025	0.00731083
	$k = 6, \alpha = \beta = 0.8$, Cr = 0.9	0.392983	2.25841e-16	0	2691.83	565.69	0.0246667	0.00730297
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.7	0.392983	2.25841e-16	0	3559.53	1046.89	0.0356667	0.0116511
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.8	0.392983	2.25841e-16	0	3763.37	971.048	0.0386667	0.0107425
	$k = 6, \alpha = \beta = 0.9$, Cr = 0.9	0.392983	2.25841e-16	0	3399.87	819.533	0.0336667	0.00850287

Table 14
Experimental results: DE variants (low noise, target fitness 0.4).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (0.4)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
DE/rand/1/bin	$F = 0.7$, Cr = 0.7	0.392983	2.25841e-16	0	11974.9	2034.15	0.128667	0.0231537
	$F = 0.7$, Cr = 0.8	0.392983	2.25841e-16	0	7609.87	1216.29	0.078	0.0142393
	$F = 0.7$, Cr = 0.9	0.392983	2.25841e-16	0	5267.93	800.132	0.0526667	0.00907187
	$F = 0.8$, Cr = 0.7	0.392983	2.25841e-16	0	14832.7	3614.1	0.158333	0.0406909
	$F = 0.8$, Cr = 0.8	0.392983	2.25841e-16	0	9459.4	2089.74	0.099	0.0223375
	$F = 0.8$, Cr = 0.9	0.392983	2.25841e-16	0	7207.13	1131.54	0.0746667	0.0127937
	$F = 0.9$, Cr = 0.7	0.392983	2.25841e-16	0	20147.1	5917.5	0.217333	0.0655446
	$F = 0.9$, Cr = 0.8	0.392983	2.25841e-16	0	15493.3	12077.2	0.168	0.135173
	$F = 0.9$, Cr = 0.9	0.392983	2.25841e-16	0	9430.4	2342.48	0.101	0.0277116
DE/rand/1/exp	$F = 0.7$, Cr = 0.7	0.392983	2.25841e-16	0	8397.93	2421.69	0.0863333	0.0276035
	$F = 0.7$, Cr = 0.8	0.392983	2.25841e-16	0	10004	2205.74	0.104667	0.0241737
	$F = 0.7$, Cr = 0.9	0.392983	2.25841e-16	0	8831.37	1752.13	0.0933333	0.0186313
	$F = 0.8$, Cr = 0.7	0.392983	2.25841e-16	0	12232.6	3084.77	0.129667	0.0347884
	$F = 0.8$, Cr = 0.8	0.392983	2.25841e-16	0	13177.6	2518.07	0.141	0.0279593
	$F = 0.8$, Cr = 0.9	0.392983	2.25841e-16	0	11329	3142.6	0.120667	0.0346344
	$F = 0.9$, Cr = 0.7	0.396857	0.0212179	1	17108.7	3357.34	0.185172	0.0373804
	$F = 0.9$, Cr = 0.8	0.392983	2.25841e-16	0	18852.9	16035.7	0.202667	0.172845
	$F = 0.9$, Cr = 0.9	0.396857	0.0212179	1	16238.1	3155.47	0.173793	0.0347865
DE/best/1/bin	$F = 0.7$, Cr = 0.7	0.420103	0.0499948	7	3540.13	1038.51	0.0321739	0.0120441
	$F = 0.7$, Cr = 0.8	0.420698	0.0511904	7	2857.74	1189.4	0.0282609	0.0140299
	$F = 0.7$, Cr = 0.9	0.412352	0.0440512	5	1596.48	365.048	0.0124	0.00597216
	$F = 0.8$, Cr = 0.7	0.416227	0.0472801	6	5835.33	2290.01	0.0575	0.0247158
	$F = 0.8$, Cr = 0.8	0.427847	0.0541668	9	3711.38	1230.82	0.0347619	0.0136452
	$F = 0.8$, Cr = 0.9	0.429819	0.0545159	11	2736	809.2	0.0242105	0.00961237
	$F = 0.9$, Cr = 0.7	0.43947	0.0579058	12	7262.39	2384.21	0.075	0.0252633
	$F = 0.9$, Cr = 0.8	0.455563	0.0596263	16	5151.43	1195.34	0.0514286	0.0156191
	$F = 0.9$, Cr = 0.9	0.470099	0.164994	11	3909.21	996.205	0.0373684	0.0119453
DE/best/1/exp	$F = 0.7$, Cr = 0.7	0.396859	0.0212174	1	2600.48	983.322	0.0237931	0.0120753
	$F = 0.7$, Cr = 0.8	0.412352	0.0440512	5	2925.6	1105.99	0.0268	0.0131403
	$F = 0.7$, Cr = 0.9	0.427847	0.0541668	9	2536.1	930.03	0.022381	0.0109109
	$F = 0.8$, Cr = 0.7	0.412353	0.0440507	5	3995.4	1608.6	0.0388	0.0183303
	$F = 0.8$, Cr = 0.8	0.416227	0.0472801	6	4538.17	1398.08	0.045	0.0164184
	$F = 0.8$, Cr = 0.9	0.412352	0.0440512	5	4247.16	1707.77	0.0416	0.0188591
	$F = 0.9$, Cr = 0.7	0.467324	0.097189	16	5165.07	2044.91	0.0521429	0.0235922
	$F = 0.9$, Cr = 0.8	0.420101	0.0499961	7	6480	2871.95	0.066087	0.0327163
	$F = 0.9$, Cr = 0.9	0.443344	0.0585747	13	7109.94	1948.36	0.0741176	0.0212305

values ($J = 15$ in equation (13)) were used, following Ref. [20]. From each run, statistics were recorded for:

- best-of-run fitness after exactly 500,000 evaluations,
- the number of fitness evaluations required to reach a target fitness for the first time in the run. If a run fails to produce the target fitness by 500,000 evaluations, the run is marked as a "failure."

Both the DE and the GA being stochastic processes, the comparative performance of the 63 versions was analyzed using the following metrics:

- Solution quality, as given by the mean and standard deviation (over 30 runs) of best-of-run fitnesses at exactly 500,000 evaluations;
- Frequency of finding a solution, given by the number of "failed" runs out of the 30.
- Time to find an acceptable solution, as measured by the mean and standard deviation of the number of fitness evaluations required to reach the target fitness and also by the mean and standard deviation of the CPU time consumed to reach the target; the mean and the standard deviation were computed from only the successful runs, that is, from 30 - #failures runs.

Table 15
Experimental results: GA (low noise, target fitness 0.4).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (0.4)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
GA	pM = 0.02, pC = 0.7	0.420185	0.0322589	18	263555	136355	2.91083	1.50865
	pM = 0.02, pC = 0.8	0.417361	0.0280841	16	149087	86384.5	1.64429	0.947935
	pM = 0.02, pC = 0.9	0.417725	0.0306961	18	181993	130944	2.02417	1.45698
	pM = 0.04, pC = 0.7	0.40438	0.0245208	7	173979	121275	1.92565	1.35569
	pM = 0.04, pC = 0.8	0.399213	0.0142372	6	178834	129290	1.98458	1.44342
	pM = 0.04, pC = 0.9	0.40076	0.02411	4	165971	135161	1.83692	1.50446
	pM = 0.06, pC = 0.7	0.39327	0.000373735	0	133960	121023	1.489	1.35513
	pM = 0.06, pC = 0.8	0.398857	0.0175172	3	123922	103112	1.37741	1.15188
	pM = 0.06, pC = 0.9	0.394132	0.00528365	1	126162	99322.7	1.39414	1.10613

Table 16
Experimental results: two DEGL/SAW variants (low noise, second set of bounds, target fitness 0.4716).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (0.4716)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
DEGL/SAW/bin	$k = 6, \alpha = \beta = 0.7$, Cr $= 0.7$	0.471574	2.82301e-16	0	2911.6	421.419	0.0286667	0.00681445
	$k = 6, \alpha = \beta = 0.7$, Cr $= 0.8$	0.471574	2.82301e-16	0	2277.53	292.469	0.0213333	0.00434172
	$k = 6, \alpha = \beta = 0.7$, Cr $= 0.9$	0.471574	2.82301e-16	0	1970.3	404.027	0.0183333	0.00592093
	$k = 6, \alpha = \beta = 0.8$, Cr $= 0.7$	0.471574	2.82301e-16	0	3134.43	489.04	0.0303333	0.0076489
	$k = 6, \alpha = \beta = 0.8$, Cr $= 0.8$	0.471574	2.82301e-16	0	2483.57	395.992	0.023	0.00466092
	$k = 6, \alpha = \beta = 0.8$, Cr $= 0.9$	0.471574	2.82301e-16	0	1853.07	281.515	0.0166667	0.00479463
	$k = 6, \alpha = \beta = 0.9$, Cr $= 0.7$	0.471574	2.82301e-16	0	3415.47	676.068	0.0353333	0.00776079
	$k = 6, \alpha = \beta = 0.9$, Cr $= 0.8$	0.471574	2.82301e-16	0	2824.97	661.762	0.0263333	0.00999425
	$k = 6, \alpha = \beta = 0.9$, Cr $= 0.9$	0.471574	2.82301e-16	0	2237.9	284.949	0.0196667	0.00490133
DEGL/SAW/exp	$k = 6, \alpha = \beta = 0.7$, Cr $= 0.7$	0.471574	2.82301e-16	0	2282.13	413.097	0.02	0.00454859
	$k = 6, \alpha = \beta = 0.7$, Cr $= 0.8$	0.471574	2.82301e-16	0	2067.13	322.887	0.019	0.0048066
	$k = 6, \alpha = \beta = 0.7$, Cr $= 0.9$	0.471574	2.82301e-16	0	2312.4	383.126	0.0213333	0.00434172
	$k = 6, \alpha = \beta = 0.8$, Cr $= 0.7$	0.471574	2.82301e-16	0	2323.93	321.148	0.021	0.00402578
	$k = 6, \alpha = \beta = 0.8$, Cr $= 0.8$	0.471574	2.82301e-16	0	2377.67	432.92	0.022	0.00550861
	$k = 6, \alpha = \beta = 0.8$, Cr $= 0.9$	0.471574	2.82301e-16	0	2469.43	435.172	0.0223333	0.00568321
	$k = 6, \alpha = \beta = 0.9$, Cr $= 0.7$	0.471574	2.82301e-16	0	2786.77	394.91	0.0263333	0.00614948
	$k = 6, \alpha = \beta = 0.9$, Cr $= 0.8$	0.471574	2.82301e-16	0	2887.43	490.321	0.028	0.00761124
	$k = 6, \alpha = \beta = 0.9$, Cr $= 0.9$	0.471574	2.82301e-16	0	2819	372.015	0.0256667	0.00568321

Tables 4–6 show the summary of results of DEGL, DE/best, DE/rand and GA runs for $\sigma = 1/3$, the first set of bounds, and target fitness = 1.578. The relative performance of these algorithms for an easier target of 1.8864 is given in Tables 7–9. Tables 10–12 show how the performance changes when the second set of bounds is used in place of the first.

The effect of the change of the noise level to $\sigma = 1/6$ is shown in Tables 13–15 (harder target) and Tables 16–18 (easier target).

A few cases in the tables in this section show the mean CPU time as zero; this is due to the inherent limitation in the time-reporting precision of the implementation platform.

Table 17
Experimental results: DE variants (low noise, second set of bounds, target fitness 0.4716).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (0.4716)			CPU seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
DE/rand/1/bin	$F = 0.7$, Cr $= 0.7$	0.471574	2.82301e-16	0	7027.73	1221.31	0.0733333	0.0142232
	$F = 0.7$, Cr $= 0.8$	0.471574	2.82301e-16	0	6054.5	947.315	0.0606667	0.0108066
	$F = 0.7$, Cr $= 0.9$	0.471574	2.82301e-16	0	5549.4	652.335	0.0566667	0.00660895
	$F = 0.8$, Cr $= 0.7$	0.476667	0.0278955	1	8075.86	1460.45	0.0841379	0.0159278
	$F = 0.8$, Cr $= 0.8$	0.471574	2.82301e-16	0	7138.33	1288.36	0.0743333	0.0140647
	$F = 0.8$, Cr $= 0.9$	0.476667	0.0278955	1	6396.76	894.683	0.0655172	0.00948164
	$F = 0.9$, Cr $= 0.7$	0.476667	0.0278955	1	9299.62	1501.33	0.0962069	0.0169903
	$F = 0.9$, Cr $= 0.8$	0.476667	0.0278955	1	8167.93	1158.86	0.0848276	0.0129892
	$F = 0.9$, Cr $= 0.9$	0.476667	0.0278955	1	7186.07	921.684	0.0737931	0.0117758
DE/rand/1/exp	$F = 0.7$, Cr $= 0.7$	0.471574	2.82301e-16	0	5726.5	983.823	0.0583333	0.0105318
	$F = 0.7$, Cr $= 0.8$	0.471574	2.82301e-16	0	5865.4	802.976	0.058	0.00961321
	$F = 0.7$, Cr $= 0.9$	0.471574	2.82301e-16	0	5950.37	822.481	0.0596667	0.00927857
	$F = 0.8$, Cr $= 0.7$	0.471574	2.82301e-16	0	7197.7	955.102	0.0723333	0.0110433
	$F = 0.8$, Cr $= 0.8$	0.476667	0.0278955	1	7047.48	1078.27	0.072069	0.0123576
	$F = 0.8$, Cr $= 0.9$	0.471574	2.82301e-16	0	6817.6	1239.54	0.0693333	0.0131131
	$F = 0.9$, Cr $= 0.7$	0.471574	2.82301e-16	0	7950.97	1042.79	0.082	0.0118613
	$F = 0.9$, Cr $= 0.8$	0.471574	2.82301e-16	0	7734.93	1603.18	0.0813333	0.0188887
	$F = 0.9$, Cr $= 0.9$	0.471574	2.82301e-16	0	7736.8	1633.91	0.08	0.0189373
DE/best/1/bin	$F = 0.7$, Cr $= 0.7$	0.497039	0.0579147	5	2401	432.855	0.02	0.005
	$F = 0.7$, Cr $= 0.8$	0.528935	0.0770039	11	1999.68	420.131	0.0178947	0.00418854
	$F = 0.7$, Cr $= 0.9$	0.522505	0.0732574	10	1426.75	266.587	0.01	0.00561951
	$F = 0.8$, Cr $= 0.7$	0.517412	0.0712136	9	2710.19	817.771	0.0242857	0.00978337
	$F = 0.8$, Cr $= 0.8$	0.537783	0.0770069	13	2205.24	437.99	0.02	0.005
	$F = 0.8$, Cr $= 0.9$	0.518749	0.0736285	9	1815.57	271.852	0.0161905	0.00497613
	$F = 0.9$, Cr $= 0.7$	0.542877	0.0775276	14	3344.31	645.688	0.030625	0.00853913
	$F = 0.9$, Cr $= 0.8$	0.562313	0.075524	18	2575.5	512.61	0.02	0.00603023
	$F = 0.9$, Cr $= 0.9$	0.571941	0.0726368	20	2282	487.815	0.018	0.00632456
DE/best/1/exp	$F = 0.7$, Cr $= 0.7$	0.502132	0.0621608	6	1770.21	397.542	0.0158333	0.00653863
	$F = 0.7$, Cr $= 0.8$	0.497039	0.0579153	5	1837.08	418.389	0.014	0.00645497
	$F = 0.7$, Cr $= 0.9$	0.507225	0.0657277	7	1812.7	366.808	0.0152174	0.00665348
	$F = 0.8$, Cr $= 0.7$	0.507225	0.0657277	7	2290.3	604.15	0.0213043	0.00757049
	$F = 0.8$, Cr $= 0.8$	0.527598	0.0748872	11	2317.68	487.795	0.02	0.00666667
	$F = 0.8$, Cr $= 0.9$	0.549307	0.0793882	15	2199.8	424.799	0.0186667	0.0063994
	$F = 0.9$, Cr $= 0.7$	0.563248	0.0761314	18	2504.83	641.497	0.02	0.00603023
	$F = 0.9$, Cr $= 0.8$	0.553062	0.0775281	16	2854.07	688.841	0.0264286	0.00841897
	$F = 0.9$, Cr $= 0.9$	0.532691	0.076131	12	3043.17	589.392	0.0294444	0.00937595

Table 18
Experimental results: GA variants (low noise, second set of bounds, target fitness 0.4716).

Algorithm	Algorithm parameters	Best fitness after 500,000 evaluations		No. of evaluations until target fitness (0.4716)			CPU Seconds to target	
		Mean	Std. Dev.	Failures	Mean	Std. Dev.	Mean	Std. Dev.
GA	pM = 0.02, pC = 0.7	0.471709	0.000213533	15	222626	125555	2.51667	1.42005
	pM = 0.02, pC = 0.8	0.471786	0.000761441	13	189845	139433	2.13706	1.57447
	pM = 0.02, pC = 0.9	0.472149	0.00171254	14	267734	111218	3.02062	1.26385
	pM = 0.04, pC = 0.7	0.471635	0.000106712	13	208707	137022	2.36471	1.5437
	pM = 0.04, pC = 0.8	0.471637	0.000169794	11	185910	142019	2.09158	1.60667
	pM = 0.04, pC = 0.9	0.471625	8.6781e-05	10	182141	104188	2.029	1.17169
	pM = 0.06, pC = 0.7	0.471611	6.69683e-05	8	128398	98920.9	1.44318	1.11053
	pM = 0.06, pC = 0.8	0.471607	7.19073e-05	7	145422	97336.1	1.64087	1.10243
	pM = 0.06, pC = 0.9	0.471629	0.000135102	8	206962	166435	2.35682	1.90449

The results show that the improved differential evolution algorithm DEGL/SAW outperforms (i) the genetic algorithm, and (ii) the standard variants of differential evolution. The DEGL/SAW/exp is the winner, outperforming all the other algorithms on at least one metric. The GA is the worst performer on all three metrics.

To investigate whether the differences in performance can be attributed to chance, unpaired t -tests are performed between the best-performing GA (from among the nine parameter settings in Table 6 or 9 or 15 or 12 or 18, depending on the case) and the worst-performing DEGL (from among the nine parameter settings in Table 4 or 7 or 13 or 10 or 16) in each case. Results of unpaired t -tests are presented in Table 19. The null hypothesis $H_0{:}\mu_1{-}\mu_2 = 0$ is tested against the one-sided alternative $H_1{:}\mu_1{-}\mu_2 > 0$ where μ_1 is the (population) mean fitness of the genetic algorithm method and μ_2 that of the competitor. A level of significance of 0.05 is chosen. Since the sample variances are unequal, the standard two-sample t -test cannot be used. Therefore, the Smith–Satterthwaite test [11] corresponding to unequal variances is employed. The test statistic is given by

$$t - \text{statistic} = \frac{\overline{x}_1 - \overline{x}_2 - 0}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \quad (19)$$

and its sampling distribution can be approximated by the t -distribution with

$$\frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(s_1^2/n_1\right)^2}{n_1 - 1} + \frac{\left(s_2^2/n_2\right)^2}{n_2 - 1}} \quad (20)$$

degrees of freedom (rounded down to the nearest integer), where $\overline{x}_1, \overline{x}_2$ are the sample means, s_1, s_2 are the sample standard deviations, and n_1, n_2 the sample sizes.

Table 19 presents, for each test case, the following:

- t-statistic;
- degrees of freedom (df);
- 95% confidence interval for the difference of the two means, $\overline{x}_1 - \overline{x}_2 \pm t_{0.025}$ at the specified df $\times \sqrt{(s_1^2/n_1) + (s_2^2/n_2)}$.

The results show that for each of the test cases, the t -statistic exceeds $t_{.05}$ for the relevant degrees of freedom. Therefore the null hypothesis is rejected in all of these cases. Again, none of the confidence intervals contains zero. This leads to the conclusion that the improvements produced by the DEGL method are statistically significant.

Table 20 shows a representative solution (the seven evolved parameters as well as the resulting fitness) from each of the algorithms used in this paper.

Table 19
Results of Smith–Satterthwaite test.

n_1	$\overline{x}_1$	s_1	n_2	$\overline{x}_2$	s_2	t-statistic	d.f.	95% Confid. Intvl.
Solution quality (best GA vs. worst DEGL)								
30	1.576990	0.000326	30	1.576770	9.03362e-16	3.697285	29	0.000098–0.000342
30	1.886340	0.000066	30	1.886290	9.03362e-16	4.147070	29	0.000025–0.000075
30	0.393270	0.000374	30	0.392983	2.25841e-16	4.206092	28	0.000147–0.000427
30	0.471607	0.000072	30	0.471574	2.82301e-16	2.513631	28	0.000006–0.000060
Cost (#evaluations) of finding the target fitness (best GA vs. worst DEGL/saw/bin)								
28	98605.700000	64505.300000	30	5954.800000	1411.010000	7.598648	27	67630.704797–117671.095203
30	1421.670000	1648.090000	30	767.100000	270.660000	2.146628	30	31.904242–1277.235758
24	102471.000000	103552.000000	30	2876.370000	601.920000	4.711695	23	55860.626986–143328.633014
27	123922.000000	103112.000000	30	4700.670000	1511.050000	6.007374	26	78418.300945–160024.359055
22	128398.000000	98920.900000	30	3415.470000	676.068000	5.926048	21	81114.564677–168850.495323
Cost (#evaluations) of finding the target fitness (best GA vs. worst DEGL/saw/exp)								
28	98605.700000	64505.300000	30	4955.830000	1097.370000	7.681254	27	68631.881565–118667.858435
30	1421.670000	1648.090000	30	645.267000	226.880000	2.556173	30	156.173141–1396.632859
24	102471.000000	103552.000000	30	2270.070000	401.462000	4.740414	23	56467.255113–143934.604887
27	123922.000000	103112.000000	30	3763.370000	971.048000	6.054946	26	79357.915295–160959.344705
22	128398.000000	98920.900000	30	2887.430000	490.321000	5.951133	21	81642.960796–169378.179204
Cost (CPU seconds) of finding the target fitness (best GA vs. worst DEGL/saw/bin)								
28	1.088930	0.718013	30	0.063333	0.015610	7.556619	27	0.747096–1.304097
24	1.133750	1.141540	30	0.041667	0.010532	4.686574	23	0.609957–1.574210
30	0.009667	0.019025	30	0.001667	0.003790	2.258774	31	0.000768–0.015232
27	1.377410	1.151880	30	0.051333	0.018705	5.981247	26	0.870250–1.781904
22	1.443180	1.110530	30	0.035333	0.007761	5.946051	21	0.915365–1.900328

Table 20
Solutions generated by the algorithms ($\sigma = 1/3$, bounds of Ref. [20]).

Algorithm	x_1	x_2	x_3	x_4	λ	R_C (Ω)	B (V)	Fitness
DEGL/SAW/bin; $\alpha = \beta = 0.8, Cr = 0.9$	−0.8534	0.00223603	3.6e-05	−0.000156943	24	0.0001	0.0355533	1.57677
GA; $p_m = 0.02$, $p_c = 0.9$	−1.19969	0.00340152	5.10013e-05	−0.000145572	15.4727	0.0001	0.0251981	1.66285
DEGL/SAW/exp; $\alpha = \beta = 0.8, Cr = 0.8$	−0.929071	0.00245031	3.6e-05	−0.000156943	24	0.0001	0.0355533	1.57677
DE/rand/1/bin; $F = 0.9, Cr = 0.8$	−1.05125	0.00279628	3.6e-05	−0.000156943	24	0.0001	0.0355533	1.57677
DE/best/1/bin; $F = 0.7, Cr = 0.9$	−1.08897	0.00290308	3.6e-05	−0.000156943	24	0.0001	0.0355533	1.57677
DE/rand/1/exp; $F = 0.9, Cr = 0.9$	−0.949475	0.00250809	3.6e-05	−0.000156943	24	0.0001	0.0355533	1.57677
DE/best/1/exp; $F = 0.8, Cr = 0.8$	−0.987863	0.00261679	3.6e-05	−0.000156943	24	0.0001	0.0355533	1.57677

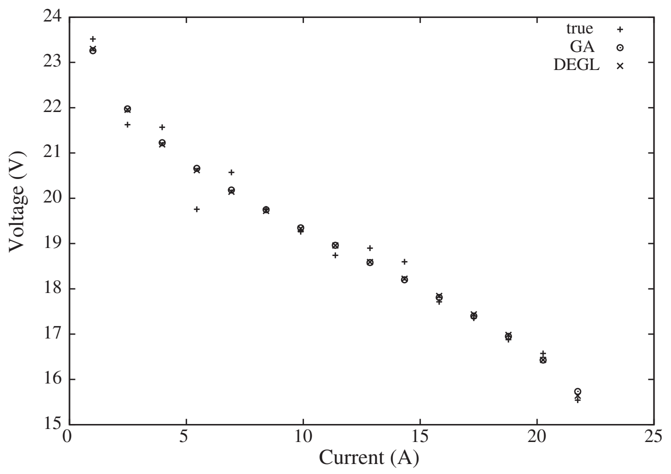

Fig. 1. Comparative polarization characteristics (DEGL and GA) corresponding to the bounds in Ref. [20].

Fig. 1 shows the polarization characteristics produced by DEGL/SAW/bin and GA (single-run solutions described in the first two rows in Table 20) against the "true" values. The data in this plot correspond to the bounds of Ref. [20] and to $\sigma = 1/3$. The polarization behavior displayed in this figure is representative of the solutions obtained from the other runs.

Fig. 2 provides a condensed, graphical description of the comparative performance of the seven algorithms; in all cases, the averaging is over 30 runs, $\sigma = 1/6$, target fitness = 0.4, and the bounds are those in Ref. [20].

While the numerical results change with a switch from the first set of bounds to the second, the qualitative relative performance of the seven competing algorithms does not change with the bounds (or with the algorithm parameter settings, or with the noise level).

Fig. 3 shows the convergence characteristics – the progression with time toward the optimal or near-optimal solution – of DEGL/SAW/bin with $\sigma = 1/6$ and the bounds of Ref. [20]. The plot shows how the best-of-generation (that is, best-so-far) fitness (averaged over 30 independent runs, all with $k = 6$, $\alpha = \beta = 0.8$, $Cr = 0.9$) improves with generation. The best fitness is obtained fairly early in a typical run, causing the last part of the plot to be flat well before generation 100 is reached.

It can be argued that in equation (1) one should not plug in any hydrogen or oxygen partial pressure values that are calculated from equations (2) and (3) using a non-zero current. In the traditional fuel cell modeling literature, however, this is common practice (see, for example, Refs. [22,14,24,25]). In the present paper, therefore, the whole suite of simulation runs was repeated with E_{Nernst} allowed to change with current following equation (1) (instead of holding it constant at 1.197374 V). The results produced by this step, not presented here because of space limitations, did not in any way affect the relative performance of the seven algorithms. In fact, the numerical differences between the newly-calculated voltages and their constant-E_{Nernst} counterparts turned out to be negligibly small (given the stack parameter values and operating conditions in this paper, this is not surprising). In this context, it is to be noted that (i) equation (5) requires that $i \geq 1$ A, (ii) equation (7) requires that $i_{\mathrm{den}} < i_{\mathrm{limit,den}}$, (iii) the present model, not unlike Refs. [18,20], ignored fuel crossover and internal currents which may sometimes be important for low-temperature fuel cells such as PEMFCs.

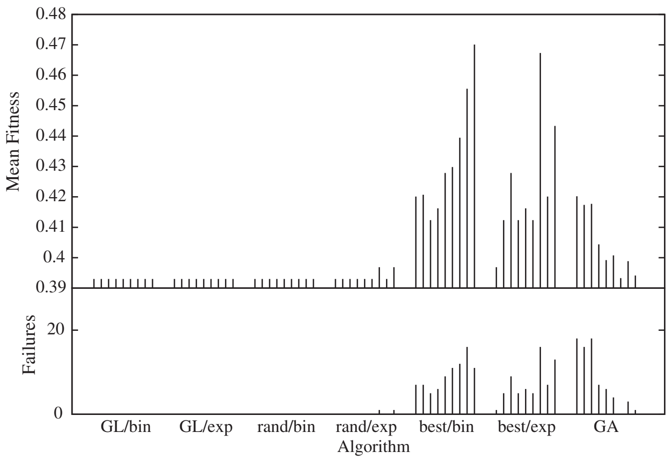

Fig. 2. Comparative results (two DEGL variants, four DE variants, and GA) corresponding to the bounds in Ref. [20].

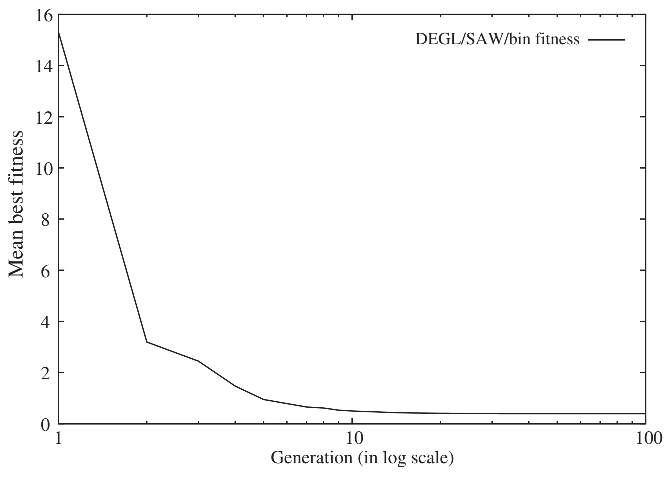

Fig. 3. Convergence characteristics (averaged over 30 runs of DEGL/SAW/bin).

5. Conclusion

Proton exchange membrane fuel cells are fast emerging as a viable, alternative energy source for a wide variety of application areas. Optimal (or near-optimal) choice of parameters for PEMFC modeling is an open research problem. This paper developed a differential evolution approach for solving the problem. The improved differential mutation strategy of using the local and global neighborhood led to a more efficient search for an optimal set of parametric coefficients. The DEGL results were seen to be statistically significantly superior regardless of the algorithmic parameter settings used. Numerical results produced by the algorithm were seen to be of superior quality than those of the competitors. The present approach outperformed its best-known state-of-the-art competitor on three metrics:

- The solution quality obtained at a given computational cost;
- The computational cost needed to find a solution of a given quality.
- The consistency (frequency) with which the algorithm finds an acceptable solution.

Acknowledgments

Two anonymous referees provided valuable feedback on an earlier version. The work of U. K. Chakraborty is supported in part by the following grants: US National Science Foundation grants CNS-1049427, CNS-1138359, and a 2011 UM Research Board Award.

References

[1] Amphlett JC, Baumert RM, Mann RF, Peppley BA, Roberge PR, Harris TJ. Performance modeling of the Ballard Mark IV solid polymer electrolyte fuel cell I: mechanistic model development. J Electrchem Soc 1995;142(1): 1–8.
[2] Chakraborty UK. Static and dynamic modeling of solid oxide fuel cell using genetic programming. Energy 2009;34:740–51.
[3] Chakraborty UK. An error in solid oxide fuel cell stack modeling. Energy 2011; 36:801–2.
[4] Chakraborty UK. Genetic programming model of solid oxide fuel cell stack: first results. Int J Inf Commun Technol 2008;1(3/4):450–8.
[5] Chakraborty UK, editor. Advances in differential evolution. Heidelberg: Springer; 2008.
[6] Chraim F, Karaki S. Fuel cell applications in wireless sensor networks. Austin: IEEE IMTC; 2010. pp. 1320–1325.
[7] Correa JM, Farret FA, Canha LN, Simoes MG. An electrochemical-based fuel-cell model suitable for electrical engineering automation approach. IEEE Trans Industr Electr 2004;51(5):1103–12.
[8] Das S, Abraham A, Chakraborty UK, Konar A. Differential evolution with a neighborhood-based mutation operator. IEEE Trans Evol Comput 2009; 13(3):526–53.
[9] Friede F, Raël S, Davat B. Mathematical model and characterization of the transient behavior of a PEM fuel cell. IEEE Trans Power Electronics 2004;19: 1234–41.
[10] Fuel cell Handbook. 5th ed. EG&G Services Parsons Inc; 2000 [US Department of Energy].
[11] Johnson RA. Miller & Freund's probability and statistics for Engineers. 6th ed. Pearson; 2000.
[12] Konar A. Computational intelligence: principles, techniques and applications. Springer; 1995.
[13] Larminie J, Dicks A. Fuel cell systems explained. 2nd ed. Wiley; 2003.
[14] Li YH, Choi SS, Rajakaruna S. An analysis of the control and operation of a solid oxide fuel-cell power plant in an isolated system. IEEE Trans Energy Conversion 2005;20:381–7.
[15] Lund H. Renewable energy strategies for sustainable development. Energy 2007;32:912–9.
[16] Mann RF, Amphlett JC, Hooper MAI, Jensen HM, Peppley BA, Roberge PR. Development and application of a generalised steady-state electrochemical model for a PEM fuel cell. J Power Sources 2000;86:173–80.
[17] Meyers JP, Maynard HL. Design considerations for miniature PEM fuel cells. J Power Sources 2002;109:76–88.
[18] Mo Z-J, Zhu X-J, Wei L-Y, Cao G-Y. Parameter optimization for a PEMFC model with a hybrid genetic algorithm. Int J Energy Res 2006;30:585–97.
[19] Nguyen TV, White RE. A water and heat management model for proton-exchange-membrane fuel cells. J Electrochem Soc 1993;140(8): 2178–86.
[20] Ohenoja M, Leiviska K. Validation of genetic algorithm results in a fuel cell model. Int J Hydrogen Energy 2010;35:12618–25.
[21] Outeiro MT, Chibante R, Carvalho AS, de Almeida AT. A parameter optimized model of a proton exchange membrane fuel cell including temperature effects. J Power Sources 2008;185:952–60.
[22] Padulles J, Ault GW, McDonald JR. An integrated SOFC plant dynamic model for power systems simulation. J Power Sources 2000;86:495–500.
[23] Storn R, Price KV. Differential evolution. Springer; 2005.
[24] Wang C, Nehrir MH, Shaw SR. Dynamic models and model validation for PEM fuel cells using electrical circuits. IEEE Trans Energy Conver 2005;20(2): 442–51.
[25] Wang C, Nehrir MH. A physically based dynamic model for solid oxide fuel cells. IEEE Trans Energy Conver 2007;22(4):887–97.