

Solving Equations by Hybrid Evolutionary Computation Techniques

Jun He, Jiyou Xu, and Xin Yao

Abstract—Evolutionary computation techniques have mostly been used to solve various optimization and learning problems. This paper describes a novel application of evolutionary computation techniques to equation solving. Several combinations of evolutionary computation techniques and classical numerical methods are proposed to solve linear and partial differential equations. The hybrid algorithms have been compared with the well-known classical numerical methods. The experimental results show that the proposed hybrid algorithms outperform the classical numerical methods significantly in terms of effectiveness and efficiency.

Index Terms—Adaptation, hybrid algorithms, linear equations, partial differential equations.

I. INTRODUCTION

THERE has been a huge increase in the number of papers and successful applications of evolutionary computation techniques in a wide range of areas in recent years. Almost all of these applications can be classified as evolutionary optimization (either numerical or combinatorial) or evolutionary learning (supervised, reinforcement, or unsupervised). This paper presents a very different and novel application of evolutionary computation techniques in equation solving, i.e., solving linear and partial differential equations by simulated evolution.

One of the best-known numerical methods for solving linear equations is the successive overrelaxation (SOR) method [1]. However, it is often very difficult to estimate the optimal relaxation factor, which is a key parameter of the SOR method. This paper proposes a hybrid algorithm combining the SOR method with evolutionary computation techniques. The hybrid algorithm does not require a user to guess or estimate the optimal relaxation factor. The algorithm “evolves” it.

Unlike most other hybrid algorithms where an evolutionary algorithm is used as a wrapper around another algorithm (often a classical algorithm), the hybrid algorithm proposed in this paper integrates the SOR method with evolutionary computation techniques, such as recombination and mutation. It makes better use of a population by employing different equation-solving strategies for different individuals in the population. Then these individuals can exchange information through recombination. Experimental results show that the hybrid algorithm can solve equations within a small fraction of time needed by the classical SOR method for a number of problems we have tested.

Manuscript received September 28, 1999; revised February 3, 2000. This work was supported in part by the State Key Laboratory of Software Engineering, Wuhan University, Wuhan, China.

J. He and J. Xu are with the Department of Computer Science, Northern Jiaotong University, Beijing 100044, China (e-mail: jhe1998@263.net).

X. Yao is with the School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, U.K. (e-mail: x.yao@cs.bham.ac.uk).

Publisher Item Identifier S 1089-778X(00)04472-6.

Built on the work of solving linear equations, this paper also proposes two hybrid algorithms for solving partial differential equations, i.e., the Dirichlet problem [2]. Both hybrid algorithms use the difference method [2] to discretize the partial differential equations into a linear system first and then solve it.

Fogel and Atmar [3] used linear equation solving as test problems for comparing recombination and inversion operators and Gaussian mutation in an evolutionary algorithm. A linear system of the form

$$b_i = \sum_{j=1}^n a_{ij}(x_j), \quad i = 1, 2, \dots, n$$

was used in their study. The worth of an individual that encoded (x_1, \dots, x_n) was defined according to the error function

$$E = \sum_{i=1}^n E_i$$

where

$$E_i = \sum_{j=1}^n |a_{ij}(x_j) - b_i|, \quad i = 1, 2, \dots, n.$$

However, the emphasis of their study was not on equation solving, but rather on comparing the effectiveness of recombination relative to mutation. No comparison with classical equation-solving methods was given, and only small problems ($n = 10$) were considered [3]. The problems tested in this paper have up to 150 equations.

The rest of this paper is organized as follows. Section II describes the hybrid algorithm for solving linear equations, proves its convergence, presents its numerical results, and compares it with the SOR method. Section III gives two hybrid algorithms for solving partial differential equations, proves their convergence, presents their numerical results, and compares them with the SOR method. Section IV concludes the paper.

II. HYBRID ALGORITHM FOR SOLVING LINEAR EQUATIONS

Consider the following linear equations:

$$A\mathbf{x} = \mathbf{b} \quad (1)$$

where $A \in R^n \times R^n$ and $\mathbf{x}, \mathbf{b} \in R^n$.

Let

$$\begin{aligned} B &= I - \text{diag}(a_{ii}^{-1})A \\ \mathbf{d} &= \text{diag}(a_{ii}^{-1})\mathbf{b}. \end{aligned} \quad (2)$$

Then substituting (2) into (1), we have

$$\mathbf{x} = B\mathbf{x} + \mathbf{d}. \quad (3)$$

The SOR method [1] for solving (3) can be described as

$$x_i^{(k)} = (1 - \omega)x_i^{(k-1)} + \omega \left(\sum_{j=1}^{i-1} b_{ij}x_j^{(k)} + \sum_{j=i+1}^n b_{ij}x_j^{(k-1)} + d_i \right), \quad i = 1, \dots, n. \quad (4)$$

If we rewrite B as

$$B = L + U,$$

then (4) can be rewritten in the matrix form

$$\mathbf{x}^{(k)} = (1 - \omega)\mathbf{x}^{(k-1)} + \omega(L\mathbf{x}^{(k)} + U\mathbf{x}^{(k-1)} + \mathbf{d}).$$

That is

$$\mathbf{x}^{(k)} = \mathcal{L}_\omega \mathbf{x}^{(k-1)} + \mathbf{g}_\omega \quad (5)$$

where

$$\begin{aligned} \mathcal{L}_\omega &= (I - \omega L)^{-1}((1 - \omega)I + \omega U) \\ \mathbf{g}_\omega &= \omega(I - \omega L)^{-1}\mathbf{d} \end{aligned}$$

where $\omega \in (0, 2)$ is called the relaxation factor, which influences the convergence rate of the SOR method greatly. The optimal relaxation parameter has been discussed for some special matrix A [4]. But, in general, it is very difficult to estimate the prior optimal relaxation factor.

The key idea behind the hybrid algorithm that combines the SOR method and evolutionary computation techniques is to self-adapt the relaxation factor used in the SOR method. For different individuals in a population, different relaxation factors are used to solve equations. The relaxation factors will be adapted based on the fitness of individuals (i.e., based on how well an individual solves the equations).

A. The Algorithm and Its Convergence

Similar to many other evolutionary algorithms, the hybrid algorithm always maintains a population of approximate solutions to linear equations. Each solution is represented by an individual. The initial solution is usually generated by the SOR method using an arbitrary relaxation factor ω . Different individuals use different relaxation factors.

Recombination in the hybrid algorithm involves all individuals in a population. If the population size is N , then the recombination will have N parents and generates N offspring through linear combination. Mutation is achieved by performing one SOR iteration as given in (5). The mutation is stochastic because ω used in the iteration is generated at random between 0 and 2. The fitness of an individual is evaluated based on the error of an approximate solution. For example, given an approximate solution (i.e., an individual) \mathbf{x} , its error is defined by $\|e(\mathbf{x})\| = \|A\mathbf{x} - \mathbf{b}\|$. The relaxation factor is adapted after each generation, depending on how well an individual performs (in terms of error). The main steps of the hybrid algorithm are described as follows.

- 1) *Initialization*: Generate an initial population of approximate solutions to the linear equations using different arbitrary relaxation factors. Denote the initial population as $X^{(0)} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ where N is the population size. Let $k \leftarrow 0$ where k is the generation counter.
- 2) *Recombination*: Let $R = (r_{ij})_{N \times N}$ be an $N \times N$ matrix, which satisfies $r_{ij} \geq 0$ for $1 \leq i, j \leq N$ and $\sum_{j=1}^N r_{ij} = 1$ for $1 \leq i \leq N$. Then generate an intermediate population $X^{(k+C)} = \{\mathbf{x}_1^{(k+C)}, \dots, \mathbf{x}_N^{(k+C)}\}$ through the following recombination:

$$\begin{pmatrix} \mathbf{x}_1^{(k+C)} \\ \vdots \\ \mathbf{x}_N^{(k+C)} \end{pmatrix} = R \begin{pmatrix} \mathbf{x}_1^{(k)} \\ \vdots \\ \mathbf{x}_N^{(k)} \end{pmatrix}. \quad (6)$$

Many different methods for choosing matrix R can be used in practice. R can be chosen either deterministically or stochastically. This paper will show, in the next section, that even a very simple choice of R can work well for the hybrid algorithm.

- 3) *Mutation*: Generate the next intermediate population $X^{(k+M)}$ from $X^{(k+C)}$ as follows: for each individual $\mathbf{x}_i^{(k+C)}$ ($1 \leq i \leq N$) in population $X^{(k+C)}$, produce an offspring according to (5)

$$\mathbf{x}_i^{(k+M)} = \mathcal{L}_{\omega_i} \mathbf{x}_i^{(k+C)} + \mathbf{g}_{\omega_i}, \quad i = 1, \dots, N. \quad (7)$$

Only one iteration is carried out for each mutation. The mutation is stochastic because both \mathcal{L}_{ω_i} and \mathbf{g}_{ω_i} are stochastic. Both \mathcal{L}_{ω_i} and \mathbf{g}_{ω_i} depend on ω_i , which is adapted stochastically during the adaptation step after each iteration.

- 4) *Adaptation*: Let \mathbf{x} and \mathbf{y} be two individuals with relaxation factors ω_x and ω_y , respectively, and let $\|e(\mathbf{x})\| = \|A\mathbf{x} - \mathbf{b}\|$ and $\|e(\mathbf{y})\| = \|A\mathbf{y} - \mathbf{b}\|$ be their errors, respectively. Then the relaxation factors ω_x and ω_y are adapted as follows.

- a) If $\|e(\mathbf{x})\| > \|e(\mathbf{y})\|$, then move ω_x toward ω_y using

$$\omega'_x = (0.5 + p_x)(\omega_x + \omega_y) \quad (8)$$

where p_x is a random number in $(-0.01, 0.01)$, and move ω_y away from ω_x using

$$\omega'_y = \begin{cases} \omega_y + p_y(2 - \omega_y), & \text{if } \omega_y > \omega_x \\ \omega_y + p_y(0 - \omega_y), & \text{if } \omega_y < \omega_x \end{cases} \quad (9)$$

where p_y is a random number in $(0.008, 0.012)$.

- b) If $\|e(\mathbf{x})\| < \|e(\mathbf{y})\|$, adapt ω_x and ω_y in the same way as above, but reverse the role of ω_x and ω_y .
- c) If $\|e(\mathbf{x})\| = \|e(\mathbf{y})\|$, no adaptation.

The idea of adapting parameters can be applied to many different domains. For example, back-propagation algorithms for neural-network training can be accelerated using self-adaptive learning rates [5].

- 5) *Selection and Reproduction*: The best $N/2$ individuals in population $X^{(k+M)}$ will reproduce (i.e., each individual generates two offspring), and then form the next generation $X^{(k+1)}$ of N individuals.

6) *Halte*: If the error of the population: $\|e(X)\| = \min\{\|e(\mathbf{x})\|; \mathbf{x} \in X\}$ is less than a given threshold ε_0 , then the algorithm terminates; otherwise, go to step 2).

There are a few parameters in the above algorithm which may be tuned in practice to optimize the algorithm's performance. However, our experience with the algorithm so far has indicated that those parameters, e.g., matrix R , initial relaxation factors ω 's, etc., are not critical. Hybrid algorithms with different parameter settings can all outperform the classical SOR method. The experimental result and comparison are given in the next section.

The following theorem establishes the convergence of the hybrid algorithm.

Theorem 1: If there exists an ε ($0 < \varepsilon < 1$) such that, for the norm of \mathcal{L}_ω

$$\|\mathcal{L}_\omega\| < \varepsilon < 1$$

then

$$\lim_{k \rightarrow \infty} \mathbf{x}^{(k)} = \mathbf{x}^*$$

where \mathbf{x}^* is the solution to the linear equations, i.e.,

$$A\mathbf{x}^* = \mathbf{b}.$$

Proof: The individuals in the population at time k are $\mathbf{x}_i^{(k)}$, $i = 1, \dots, N$. Let the error between the approximate and exact solutions be $\mathbf{e}_i^{(k)} = \mathbf{x}_i^{(k)} - \mathbf{x}^*$. According to recombination

$$\mathbf{x}_i^{(k+C)} = \sum_{j=1}^N r_{ij} \mathbf{x}_j^{(k)}, \quad i = 1, \dots, N.$$

Since $\sum_{j=1}^N r_{ij} = 1$ and $r_{ij} \geq 0$, then for $i = 1, \dots, N$

$$\begin{aligned} \|\mathbf{e}_i^{(k+C)}\| &= \|\mathbf{x}_i^{(k+C)} - \mathbf{x}^*\| \\ &\leq \sum_{j=1}^N r_{ij} \|\mathbf{x}_j^{(k)} - \mathbf{x}^*\| \\ &\leq \max \left\{ \|\mathbf{e}_j^{(k)}\|; \quad j = 1, \dots, N \right\}. \end{aligned}$$

According to mutation, for $i = 1, \dots, N$

$$\mathbf{x}_i^{(k+M)} = \mathcal{L}_{\omega_i} \mathbf{x}_i^{(k+C)} + \mathbf{g}_{\omega_i}.$$

Since $\mathbf{x}^* = \mathcal{L}_{\omega_i} \mathbf{x}^* + \mathbf{g}_{\omega_i}$, then

$$\mathbf{x}_i^{(k+M)} - \mathbf{x}^* = \mathcal{L}_{\omega_i} (\mathbf{x}_i^{(k+C)} - \mathbf{x}^*).$$

Therefore

$$\begin{aligned} \|\mathbf{e}_i^{(k+M)}\| &\leq \|\mathcal{L}_{\omega_i}\| \|\mathbf{e}_i^{(k+C)}\| \\ &< \varepsilon \max \left\{ \|\mathbf{e}_j^{(k)}\|; \quad j = 1, \dots, N \right\}. \end{aligned}$$

According to selection, we have for $i = 1, \dots, N$

$$\begin{aligned} \|\mathbf{e}_i^{(k+1)}\| &\leq \|\mathbf{e}_i^{(k+M)}\| \\ &< \varepsilon \max \left\{ \|\mathbf{e}_j^{(k)}\|; \quad j = 1, \dots, N \right\}. \end{aligned}$$

This means that the sequence $\{\max\{\|\mathbf{e}_j^{(k+1)}\|; j = 1, \dots, N\}; k = 0, 1, \dots\}$ is strictly monotonic decreasing, and thus convergent. ■

The following theorem justifies the adaptation method for relaxation factors used in the hybrid algorithm.

Theorem 2: Let $\rho(\omega)$ be the spectral radius of matrix \mathcal{L}_ω , let ω^* be the optimal relaxation factor, and let ω_x and ω_y be the relaxation factors of individuals \mathbf{x} and \mathbf{y} , respectively. Assume $\rho(\omega)$ is monotonic decreasing when $\omega < \omega^*$, $\rho(\omega)$ is monotonic increasing when $\omega > \omega^*$, and $\rho(\omega_x) > \rho(\omega_y)$. Then

- 1) $\rho(\omega'_x) < \rho(\omega_x)$ when $\omega'_x = p(\omega_x + \omega_y)$ for $p \in (0, 1)$, and
- 2) $\rho(\omega'_y) < \rho(\omega_y)$ when $\omega'_y = \omega_y + \delta \text{sign}(\omega_y - \omega_x)$, where $\delta > 0$, and $\omega^* < \omega_y < \omega_x$ or $\omega_x < \omega_y < \omega^*$.

Proof: The first result can be derived directly from the monotonicity of $\rho(\omega)$. The second result can also be derived from the monotonicity of $\rho(\omega)$ by letting $\delta < |\omega^* - \omega_x|$. ■

B. Numerical Experiments

In order to evaluate the effectiveness and efficiency of the proposed hybrid algorithm, numerical experiments have been carried out on a number of problems. The first problem is to solve the following linear equations:

$$A\mathbf{x} = \mathbf{b}$$

where $a_{ii} = 2n$ and $b_i = i$ for $i = 1, \dots, n$, and $a_{ij} = j$ for $i \neq j$, $i, j = 1, \dots, n$. The dimension $n = 150$ in all of our experiments. The problem is to be solved with an error smaller than $\varepsilon_0 = 10^{-6}$.

Table I shows the numerical results achieved by the classical SOR method with different relaxation factors, i.e., $\omega = 1.0, 1.25, 1.5, 1.75$, which are common values used in the SOR method.

Table II gives the numerical results produced by the hybrid algorithm with different initial relaxation factors. The hybrid algorithm used in all of our experiments was very simple, and had population size two. That is, only two individuals were used. Two experiments were carried out using the hybrid algorithm, one with initial relaxation factors 1.0 and 1.25, and the other with initial relaxation factors 1.5 and 1.75. The number of iterations used to produce results in Table II and that used to produce results in Table II were the same. Table II lists the results of both individuals in a population for the two experiments.

Since only two individuals were used in a population in our experiment, the recombination matrix was chosen as follows: if the fitness of the first individual was better than the second, let

$$\begin{pmatrix} x_1^{(k+C)} \\ x_2^{(k+C)} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0.99 & 0.01 \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \end{pmatrix};$$

else, let

$$\begin{pmatrix} x_1^{(k+C)} \\ x_2^{(k+C)} \end{pmatrix} = \begin{pmatrix} 0.01 & 0.99 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \end{pmatrix}.$$

It is very clear from Tables I and II that the hybrid algorithm performed much better than the classical SOR method. Hybrid algorithms with different initial relaxation factors have all found

TABLE I
ERRORS PRODUCED BY THE CLASSICAL
SOR METHOD WITH DIFFERENT RELAXATION FACTORS

Iterations	$\omega = 1.0$	$\omega = 1.25$	$\omega = 1.50$	$\omega = 1.75$
100	2.22851e+02	3.77823e+02	6.04438e+02	9.79678e+02
200	1.12210e+02	2.33974e+02	3.92983e+02	6.40986e+02
300	4.67222e+01	1.32309e+02	2.61897e+02	3.94819e+02
400	1.33169e+01	5.94886e+01	1.51498e+02	2.30768e+02
500	8.54598e+00	5.03717e+01	1.65649e+02	3.94111e+02
600	2.40567e+00	1.73236e+01	6.80516e+01	1.90952e+02
700	1.26884e+00	1.63912e+01	8.84312e+01	2.90272e+02
800	5.44300e-01	8.37548e+00	5.39054e+01	2.17434e+02
900	1.48634e-01	4.53472e+00	4.11268e+01	1.98593e+02
1000	9.82843e-02	3.35902e+00	3.56386e+01	1.94783e+02

TABLE II
ERRORS PRODUCED BY THE HYBRID ALGORITHM WITH DIFFERENT INITIAL
RELAXATION FACTORS

Iteration	Hybrid Algorithm with initial $\omega = 1.0$ and 1.25		Hybrid Algorithm with initial $\omega = 1.5$ and 1.75	
100	2.30461e+01	2.32916e+01	4.04223e+01	4.05483e+01
200	5.70489e+00	5.81735e+00	2.26384e+01	2.29360e+01
300	5.11369e-01	5.26034e-01	6.15001e+00	6.16114e+00
400	4.92925e-02	4.99415e-02	2.73954e+00	2.77822e+00
500	2.38316e-03	2.42947e-03	4.09947e-01	4.19652e-01
600	4.63174e-05	4.72330e-05	2.56437e-02	2.57719e-02
700	4.51076e-07	4.61773e-07	1.98691e-03	2.01740e-03
800	3.57279e-09	3.64514e-09	5.64595e-05	5.74845e-05
900	1.64988e-11	1.51914e-11	6.09689e-07	6.23950e-07
1000	3.41061e-13	3.41061e-13	7.90861e-09	8.09719e-09

Note: The factors were adapted dynamically during execution of the algorithm. The two columns under the algorithm indicate two individuals. A total of ten independent runs were conducted. The average results are reported here.

approximate solutions with an error smaller than $\varepsilon_0 = 10^{-6}$ within 1000 iterations, while none of the classical SOR method could find an approximate solution with an error smaller than $\varepsilon_0 = 10^{-6}$ after 1000 iterations, no matter which relaxation factor had been used. After 1000 iterations, there was at least eight orders of magnitude difference between the error generated by the classical SOR method and that produced by the hybrid algorithm. Table III shows how ω changed dynamically as the hybrid algorithm progressed.

To evaluate the hybrid algorithm further, ten additional test problems, labeled from $P1$ to $P10$, with 150 variables were generated at random. That is, matrices A in problems $P1$ – $P10$ were all generated at random: a_{ii} 's were generated uniformly at random in $[16, 25]$, and a_{ij} 's ($i \neq j$) were generated uniformly at random in $[0, 150]$. $b_i = 1.0$ for all problems. All problems were required to be solved with an error smaller than $\varepsilon_0 = 10^{-6}$. The maximum number of iterations allowed was 1000.

Table IV compares the number of iterations needed by the classical SOR method and that needed by the hybrid algorithm to solve the linear equations to the given preciseness ($\varepsilon_0 =$

TABLE III
VALUE OF ω FOR DIFFERENT INDIVIDUALS AT DIFFERENT GENERATIONS
OF THE HYBRID ALGORITHM; RESULTS HAVE BEEN AVERAGED OVER
TEN INDEPENDENT RUNS

Iteration	Hybrid Algorithm with initial $\omega = 1.0$ and 1.25		Hybrid Algorithm with initial $\omega = 1.5$ and 1.75	
100	8.33748e-01	9.01997e-01	1.25062e+00	1.34361e+00
200	6.81233e-01	7.26442e-01	1.06422e+00	1.12993e+00
300	5.56617e-01	5.92962e-01	9.04964e-01	9.63276e-01
400	4.54796e-01	4.84459e-01	7.39422e-01	7.87605e-01
500	3.71602e-01	3.95837e-01	6.04161e-01	6.43561e-01
600	3.03626e-01	3.23427e-01	4.93644e-01	5.25838e-01
700	2.48084e-01	2.64264e-01	4.03343e-01	4.29648e-01
800	2.02703e-01	2.15923e-01	3.29561e-01	3.51054e-01
900	1.65623e-01	1.76425e-01	2.69275e-01	2.86836e-01
1000	2.67402e-01	3.32096e-01	2.20017e-01	2.34366e-01

10^{-6}). Two observations can be made immediately from the table. First, except for problems $P2$ and $P3$ where the SOR method with $\omega = 1.25$ performed the same as hybrid algorithms, the SOR method performed much worse than the hybrid algorithm for all other problems. Second, the SOR method was extremely sensitive to the relaxation factor ω , while the hybrid algorithm was very robust against different initial values of ω . This indicates that the simple adaptation scheme for relaxation factors had worked quite effectively in the hybrid algorithm.

III. HYBRID ALGORITHMS FOR PARTIAL DIFFERENTIAL EQUATIONS

Similar ideas as described in Section II can be applied to solve partial differential equations. This section proposes two hybrid algorithms combining evolutionary computation techniques with the difference method [2] for solving the Dirichlet problem [2].

The Dirichlet problem can be described as follows:

$$\begin{cases} Lu = f, & \text{in } \Omega \\ u|_{\Gamma} = \phi(x), & \text{on } \Gamma \end{cases} \quad (10)$$

where Ω is an open domain in R^2 , Γ is its boundary, and L is

$$Lu = - \sum_{i,j=1}^2 \frac{\partial}{\partial x_i} a_{ij} \frac{\partial u}{\partial x_j}.$$

We assume that matrix $\{a_{ij}(x)\}$ is symmetric and uniformly positive definite, and that $a(x) \geq 0$ is in Ω .

A. Two Hybrid Algorithms

Both algorithms introduced in this section are the simplest of their more general versions. That is, we only describe the version with two individuals, i.e., the population size is two in both cases. The first algorithm is similar to that given in Section II because it first discretizes the partial differential equation (10) into a linear system using the difference method, and then solves it using the algorithm given in Section II.

TABLE IV
NUMBER OF ITERATIONS NEEDED BY THE CLASSICAL SOR METHOD AND THAT NEEDED BY THE HYBRID ALGORITHM TO SOLVE THE TEN PROBLEMS TO THE GIVEN PRECISIONNESS ($\varepsilon_0 = 10^{-6}$)

Problem	SOR with	SOR with	Hybrid algorithm with initial $\omega = 1.25$ and 1.75	
	$\omega = 1.25$	$\omega = 1.75$		
P1	160	650	120	120
P2	20	100	20	20
P3	40	160	40	40
P4	60	480	50	50
P5	60	670	50	50
P6	60	> 1000	50	50
P7	100	> 1000	60	60
P8	120	> 1000	100	100
P9	70	> 1000	50	50
P10	260	160	120	120

The second algorithm differs from the first one in that it uses two different discretization methods. Each discretization leads to a different linear system. Each linear system is solved by an individual. The two individuals usually use different relaxation factors. There is no dynamic adaptation of relaxation factors as described in Section II because the two individuals solve two different linear systems.

Solving the Dirichlet problem as defined by (10) numerically is often done by using the difference method. A linear discrete system is then obtained

$$\begin{cases} A_h u_h = f_h, & \text{in } \Omega \\ u_h|_{\Gamma} = \phi_h, & \text{on } \Gamma. \end{cases} \quad (11)$$

This system can be solved by the classical SOR method or the hybrid algorithm introduced in Section II using iteration

$$u_h^{(k+1)} = \mathcal{L}_{\omega} u_h^{(k)} + g_h$$

where \mathcal{L}_{ω} denotes the iteration matrix.

The first hybrid algorithm for solving partial differential equations can be described as follows.

- 1) *Initialization*: Generate an initial population of two individuals $\{u_h^{(0)}, v_h^{(0)}\}$ with two different initial relaxation factors ω_u, ω_v . Let $k \leftarrow 0$ where k is the generation counter.
- 2) *Recombination*: Two parents $\{u_h^{(k)}, v_h^{(k)}\}$ are used to generate two offspring $u_h^{(k+C)}, v_h^{(k+C)}$ in recombination as follows:

$$\begin{cases} u_h^{(k+C)} = (1 - \eta_u) u_h^{(k)} + \eta_u v_h^{(k)} \\ v_h^{(k+C)} = (1 - \eta_v) v_h^{(k)} + \eta_v u_h^{(k)} \end{cases}$$

where $0 \leq \eta_u \leq 1$, $0 \leq \eta_v \leq 1$. Both η_u and η_v can be deterministic or stochastic. In the more general case of more than two individuals, the recombination given in Section II is used.

- 3) *Mutation*: Each individual (after recombination) is mutated as follows:

$$\begin{cases} u_h^{(k+M)} = \mathcal{L}_{\omega_u} u_h^{(k+C)} + g_u \\ v_h^{(k+M)} = \mathcal{L}_{\omega_v} v_h^{(k+C)} + g_v \end{cases}$$

where $\omega_u, \omega_v \in (0, 2)$.

- 4) *Adaptation*: Adapt the relaxation factor for each individual as described by the algorithm given in Section II.
- 5) *Selection and Reproduction*: The two mutated offspring replace the previous population, i.e.,

$$\begin{cases} u_h^{(k+1)} = u_h^{(k+M)} \\ v_h^{(k+1)} = v_h^{(k+M)}. \end{cases}$$

- 6) *Halt*: If the error of the best individual in the population is smaller than the given error ε_0 , then terminate the algorithm; otherwise, go to step 2).

The above algorithm can be regarded as a specific version of the hybrid algorithm given in Section II when the population size is two. The following theorem establishes the convergence of the hybrid algorithm, where its proof can refer to the proof of Theorem 1.

Theorem 3: If there exists some ε ($0 < \varepsilon < 1$) such that

$$\|\mathcal{L}_{\omega_u}\| < \varepsilon \quad \text{and} \quad \|\mathcal{L}_{\omega_v}\| < \varepsilon$$

for all \mathcal{L}_{ω_u} and \mathcal{L}_{ω_v} , then

$$\lim_{k \rightarrow \infty} u_h^{(k)} = u_h^*, \quad \lim_{k \rightarrow \infty} v_h^{(k)} = v_h^*$$

where u_h^* and v_h^* are solutions to (11).

Proof: The same as the proof for Theorem 1. ■

While the above hybrid algorithm is virtually the same as that given in Section II, the second hybrid algorithm described below is quite different. It uses different discretization methods to generate different linear systems, each of which is then solved by an individual. Let the two discretized systems of the Dirichlet problem, i.e., (10), be

$$\begin{cases} A_h u_h = f_h, & \text{in } \Omega \\ u_h|_{\Gamma} = \phi_h, & \text{on } \Gamma \end{cases} \quad (12)$$

and

$$\begin{cases} A'_h v_h = f'_h, & \text{in } \Omega \\ v_h|_{\Gamma} = \phi'_h, & \text{on } \Gamma. \end{cases} \quad (13)$$

Then the two individuals, which represent two approximate solutions to the above two systems, respectively, can be represented by $\{u_h, v_h\}$.

The second hybrid algorithm can be described as follows.

- 1) *Initialization*: Generate an initial population of two individuals $\{u_h^{(0)}, v_h^{(0)}\}$. Let $k \leftarrow 0$ where k is the generation counter.
- 2) *Recombination*: Similar to the first hybrid algorithm, recombination is based on weighted averaging

$$\begin{cases} u_h^{(k+C)} = (1 - \eta_u) u_h^{(k)} + \eta_u v_h^{(k)} \\ v_h^{(k+C)} = (1 - \eta_v) v_h^{(k)} + \eta_v u_h^{(k)} \end{cases} \quad (14)$$

where $0 \leq \eta_u \leq 1$ and $0 \leq \eta_v \leq 1$. Both can be chosen deterministically or stochastically.

- 3) *Mutation*: Mutation is applied to recombined individuals, and is equivalent to one iteration in the SOR method

$$\begin{cases} u_h^{(k+M)} = \mathcal{L}_{\omega_u} u_h^{(k+C)} + g_u \\ v_h^{(k+M)} = \mathcal{L}_{\omega_v} v_h^{(k+C)} + g_v. \end{cases} \quad (15)$$

- 4) *Selection and Reproduction*: The mutated offspring replace the parents. This is similar to generational evolutionary algorithms. That is

$$\begin{cases} u_h^{(k+1)} = u_h^{(k+M)} \\ v_h^{(k+1)} = v_h^{(k+M)}. \end{cases} \quad (16)$$

- 5) *Halt*: If the error of the best individual in the population is smaller than the given error ε_0 , then terminate the algorithm; otherwise, go to step 2).

It is worth noting that the second hybrid algorithm does not adapt relaxation factors dynamically. The relaxation factors are fixed after initialization. This is mainly because the optimal relaxation factor is problem dependent, and the two individuals in the second hybrid algorithm are approximate solutions to different systems. It is not reasonable to mix two relaxation factors together, as was done in Section II, although it is useful to adapt the relaxation factor within an individual. It is shown later in this section that the second hybrid algorithm can outperform the classical method, even without adaptation of relaxation factors.

The following theorem establishes the convergence of the second hybrid algorithm.

Theorem 4: If there exists an ε ($0 \leq \varepsilon \leq 1$) such that

$$\|\mathcal{L}_{\omega_u}\|_\infty < \varepsilon \quad \text{and} \quad \|\mathcal{L}_{\omega_v}\|_\infty < \varepsilon$$

then

$$\lim_{k \rightarrow \infty} (u_h^{(k)}, v_h^{(k)})^t = (u_h^{**}, v_h^{**})^t$$

where $(u_h^{**}, v_h^{**})^t$ are solutions to the following equations:

$$\begin{pmatrix} \mathcal{L}_{\omega_u}^{-1} - \eta_u I & (1 - \eta_u)I \\ (1 - \eta_v)I & \mathcal{L}_{\omega_v}^{-1} - \eta_v I \end{pmatrix} \begin{pmatrix} u_h^{**} \\ v_h^{**} \end{pmatrix} = \begin{pmatrix} \mathcal{L}_{\omega_u}^{-1} g_u \\ \mathcal{L}_{\omega_v}^{-1} g_v \end{pmatrix}. \quad (17)$$

Proof: Let the two individuals at time k be $u_h^{(k)}$ and $v_h^{(k)}$. According to recombination

$$\begin{cases} u_h^{(k+C)} = (1 - \eta_u)u_h^{(k)} + \eta_u v_h^{(k)} \\ v_h^{(k+C)} = (1 - \eta_v)v_h^{(k)} + \eta_v u_h^{(k)}. \end{cases}$$

Rewrite it in the matrix form

$$\begin{pmatrix} u_h^{(k+C)} \\ v_h^{(k+C)} \end{pmatrix} = \begin{pmatrix} (1 - \eta_u)I & \eta_u I \\ \eta_v I & (1 - \eta_v)I \end{pmatrix} \begin{pmatrix} u_h^{(k)} \\ v_h^{(k)} \end{pmatrix}$$

where I is a $n \times n$ unit matrix.

According to mutation

$$\begin{cases} u_h^{(k+M)} = \mathcal{L}_{\omega_u} u_h^{(k+C)} + g_u^{(k)} \\ v_h^{(k+M)} = \mathcal{L}_{\omega_v} v_h^{(k+C)} + g_v^{(k)}. \end{cases}$$

Rewrite it in the matrix form

$$\begin{pmatrix} u_h^{(k+M)} \\ v_h^{(k+M)} \end{pmatrix} = \begin{pmatrix} \mathcal{L}_{\omega_u} & 0 \\ 0 & \mathcal{L}_{\omega_v} \end{pmatrix} \begin{pmatrix} u_h^{(k+C)} \\ v_h^{(k+C)} \end{pmatrix} + \begin{pmatrix} g_u^{(k)} \\ g_v^{(k)} \end{pmatrix}.$$

According to selection

$$\begin{cases} u_h^{(k+1)} = u_h^{(k+M)} \\ v_h^{(k+1)} = v_h^{(k+M)}. \end{cases}$$

Consider recombination, mutation, and selection together, we have

$$\begin{pmatrix} u_h^{(k+1)} \\ v_h^{(k+1)} \end{pmatrix} = \begin{pmatrix} \mathcal{L}_{\omega_u} & 0 \\ 0 & \mathcal{L}_{\omega_v} \end{pmatrix} \begin{pmatrix} (1 - \eta_u)I & \eta_u I \\ \eta_v I & (1 - \eta_v)I \end{pmatrix} \cdot \begin{pmatrix} u_h^{(k)} \\ v_h^{(k)} \end{pmatrix} + \begin{pmatrix} g_u^{(k)} \\ g_v^{(k)} \end{pmatrix}.$$

Let

$$e^{(k+1)} = \begin{pmatrix} e_u^{(k+1)} \\ e_v^{(k+1)} \end{pmatrix} = \begin{pmatrix} u_h^{(k+1)} - u^{**} \\ v_h^{(k+1)} - v^{**} \end{pmatrix}.$$

We have

$$e^{(k+1)} = \begin{pmatrix} \mathcal{L}_{\omega_u} & 0 \\ 0 & \mathcal{L}_{\omega_v} \end{pmatrix} \begin{pmatrix} (1 - \eta_u)I & \eta_u I \\ \eta_v I & (1 - \eta_v)I \end{pmatrix} e^{(k)}.$$

Since $0 \leq \eta_u, \eta_v \leq 1$, we have

$$\left\| \begin{pmatrix} (1 - \eta_u)I & \eta_u I \\ \eta_v I & (1 - \eta_v)I \end{pmatrix} \right\|_\infty \leq 1.$$

Because $\|\mathcal{L}_{\omega_u}\|_\infty < \varepsilon$ and $\|\mathcal{L}_{\omega_v}\|_\infty < \varepsilon$, we get

$$\|e^{(k+1)}\|_\infty < \varepsilon \|e^{(k)}\|_\infty.$$

In other words, the sequence $\{\|e^{(k+1)}\|; k = 0, 1, \dots\}$ is strictly monotonic decreasing, and is convergent. ■

If $\eta_u = 1$ and $\eta_v = 1$, (17) becomes

$$\begin{pmatrix} \mathcal{L}_{\omega_u}^{-1} - I & 0 \\ 0 & \mathcal{L}_{\omega_v}^{-1} - I \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \mathcal{L}_{\omega_u}^{-1} g_u \\ \mathcal{L}_{\omega_v}^{-1} g_v \end{pmatrix}$$

where (u^{**}, v^{**}) is the solution to (12) and (13), respectively.

If $\eta_u \neq 1$ and $\eta_v \neq 1$, the relationship between $(u^*, v^*)^t$ and $(u^{**}, v^{**})^t$ is

$$\begin{pmatrix} u^{**} \\ v^{**} \end{pmatrix} = \begin{pmatrix} \mathcal{L}_{\omega_u}^{-1} - \eta_u I & (1 - \eta_u)I \\ (1 - \eta_v)I & \mathcal{L}_{\omega_v}^{-1} - \eta_v I \end{pmatrix}^{-1} \cdot \begin{pmatrix} \mathcal{L}_{\omega_u}^{-1} - I & 0 \\ 0 & \mathcal{L}_{\omega_v}^{-1} - I \end{pmatrix} \begin{pmatrix} u^* \\ v^* \end{pmatrix}$$

i.e., $(u^{**}, v^{**})^t$ is a linear combination of $(u^*, v^*)^t$, where u^* is the solution to $A_h u_h = f_h$ and v^* is the solution to $A'_h v_h = f'_h$.

B. Numerical Experiments

Numerical experiments have been carried out on a number of test problems in order to evaluate the strength and weakness

of proposed hybrid algorithms. The first problem is to solve the following partial differential equations:

$$\begin{cases} \Delta u = -100(x^2 + y^2) \sin(10xy) \\ \equiv f(x, y), & (x, y) \in (0, 1) \times (0, 1) \\ u(0, y) = 0, & y \in [0, 1] \\ u(1, y) = \sin(10y), & y \in [0, 1] \\ u(x, 0) = 0, & x \in [0, 1] \\ u(x, 1) = \sin(10x), & x \in [0, 1] \end{cases} \quad (18)$$

where $\Omega = (0, 1) \times (0, 1)$ and $f(x, y) = -100(x^2 + y^2) \sin(10xy)$. The exact solution to the problem is $u(x, y) = \sin(10xy)$.

1) *Results of the First Hybrid Algorithm:* Using the five-point difference method, we can obtain a discrete system for (18)

$$\begin{cases} (u_{i+1,j} - 2u_{i,j} + u_{i-1,j})/h^2 \\ + (u_{i,j+1} - 2u_{i,j} + u_{i,j-1})/h^2 \\ = f_{i,j}, & i, j = 1, \dots, n-1 \\ u_{0,j} = 0, & j = 0, \dots, n \\ u_{1,j} = \sin(10jh), & j = 0, \dots, n \\ u_{i,0} = 0, & i = 0, \dots, n \\ u_{i,1} = \sin(10ih), & i = 0, \dots, n \end{cases} \quad (19)$$

where the mesh $h = 0.01$, $n = 1/h$ and $f_{i,j} = -100((ih)^2 + (jh)^2) \sin(10ihjh)$.

The above discrete system can be solved by the SOR method as follows: for $i, j = 1, \dots, n-1$

$$u_{ij}^{(k+1)} = \omega \left(u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} - h^2 f_{i,j} \right) / 4 + (1 - \omega) u_{i,j}^{(k)} \quad (20)$$

where $\omega(0 < \omega < 2)$ is the relaxation factor.

The first hybrid algorithm was implemented in our experiment as follows.

- 1) Initialize the population with two individuals $u_{i,j}^{(0)} = 0.0$ and $v_{i,j}^{(0)} = 0.0$. Let $k \leftarrow 0$. Two different relaxation factors should be used for two individuals.
- 2) Recombine two individuals $u_{i,j}^{(k)}$ and $v_{i,j}^{(k)}$ as follows.
 - a) If the fitness of $u_{i,j}^{(k)}$ is higher than that of $v_{i,j}^{(k)}$, then

$$u_{ij}^{(k+C)} = u_{ij}^{(k)}, \quad v_{ij}^{(k+C)} = 0.5 \left(u_{ij}^{(k)} + v_{ij}^{(k)} \right).$$

- b) If the fitness of $v_{i,j}^{(k)}$ is higher than that of $u_{i,j}^{(k)}$, then

$$v_{ij}^{(k+C)} = v_{ij}^{(k)}, \quad u_{ij}^{(k+C)} = 0.5 \left(u_{ij}^{(k)} + v_{ij}^{(k)} \right).$$

- 3) Mutate $u_{ij}^{(k+C)}$ and $v_{ij}^{(k+C)}$ using one SOR iteration as described by (20): for $i, j = 1, \dots, n-1$

$$\begin{cases} u_{ij}^{(k+M)} = \omega_u \left(u_{i,j+1}^{(k+C)} + u_{i,j-1}^{(k+C)} + u_{i+1,j}^{(k+C)} + u_{i-1,j}^{(k+C)} - h^2 f_{i,j} \right) / 4 + (1 - \omega_u) u_{i,j}^{(k+C)} \\ v_{ij}^{(k+M)} = \omega_v \left(v_{i,j+1}^{(k+C)} + v_{i,j-1}^{(k+C)} + v_{i+1,j}^{(k+C)} + v_{i-1,j}^{(k+C)} - h^2 f_{i,j} \right) / 4 + (1 - \omega_v) v_{i,j}^{(k+C)}. \end{cases} \quad (21)$$

- 4) Adapt relaxation factors as follows.
 - a) If the fitness of $u_{ij}^{(k+M)}$ is higher than that of $v_{ij}^{(k+M)}$, then

$$\omega'_u = \begin{cases} \omega_u + p_u(2 - \omega_v), & \text{if } \omega_u > \omega_v \\ \omega_u + p_u(0 - \omega_v), & \text{if } \omega_u < \omega_v \end{cases}$$

and $\omega'_v = \omega_v + (0.5 + p_v) * (\omega_u - \omega_v)$, where $p_u \in (0.008, 0.012)$ and $p_v \in (-0.01, 0.01)$ are random numbers.

- b) If the fitness of $v_{ij}^{(k+M)}$ is higher than that of $u_{ij}^{(k+M)}$, then

$$\omega'_v = \begin{cases} \omega_v + p_v(2 - \omega_u), & \text{if } \omega_v > \omega_u \\ \omega_v + p_v(0 - \omega_u), & \text{if } \omega_v < \omega_u \end{cases}$$

and $\omega'_u = \omega_u + (0.5 + p_u) * (\omega_v - \omega_u)$, where $p_u \in (-0.01, 0.01)$ and $p_v \in (0.008, 0.012)$ are random numbers.

- 5) Replace the old generation with mutated offspring, i.e., $u_{ij}^{(k+1)} = u_{ij}^{(k+M)}$ and $v_{ij}^{(k+1)} = v_{ij}^{(k+M)}$. Let $k \leftarrow k+1$.
- 6) If the error $\|e\|$ of the best individual is smaller than the given error ε_0 , terminate the algorithm; otherwise, go to step 2). Here, we define

$$\begin{aligned} \|e\| &= \min\{\|e_u\|, \|e_v\|\} \\ \|e_u\| &= \max\{|u_{i,j} - u(ih, jh)|; i, j = 1, n-1\} \\ \|e_v\| &= \max\{|v_{i,j} - v(ih, jh)|; i, j = 1, n-1\}. \end{aligned}$$

Table V summarizes the numerical results produced by the SOR method and those by the first hybrid algorithm. The hybrid algorithm was initialized with the same relaxation factors as those used by the SOR method. It is clear from Table V that the hybrid algorithm performed significantly better than the SOR method. The hybrid algorithm was able to find a better approximate solution after 300 iterations than that could be found by the SOR method after 1000 iterations. Furthermore, different relaxation factors had a crucial impact on the performance of the SOR method. The SOR method with relaxation factor 1.25 performed much worse than that with relaxation factor 1.75. However, the hybrid algorithm is much more robust against changes in initial relaxation factors. This reduces the burden on the user to find/guess a near-optimal relaxation factor.

TABLE V
ERRORS PRODUCED BY THE CLASSICAL SOR METHOD AND BY THE
FIRST HYBRID ALGORITHM

Iteration	SOR with $\omega = 1.25$	SOR with $\omega = 1.75$	Hybrid algorithm with initial $\omega = 1.25$ and 1.75	
100	7.74876e-01	3.39587e-01	7.74876e-01	3.39587e-01
200	5.96559e-01	1.08033e-01	4.79187e-02	4.78746e-02
300	4.59065e-01	4.52751e-02	6.42171e-04	6.41561e-04
400	3.55212e-01	2.15914e-02	4.63378e-04	4.63317e-04
500	2.77599e-01	1.05872e-02	4.86865e-04	4.86853e-04
600	2.19625e-01	5.21141e-03	4.91363e-04	4.91360e-04
700	1.76055e-01	2.57598e-03	4.93147e-04	4.93145e-04
800	1.42990e-01	1.40235e-03	4.93906e-04	4.93906e-04
900	1.17434e-01	9.25236e-04	4.93586e-04	4.93586e-04
1000	9.73326e-02	7.10448e-04	4.94353e-04	4.94353e-04

To further evaluate the proposed hybrid algorithm, the following five more problems (problems P1–P5) were tested using both the SOR method and the hybrid algorithm:

- P1: $u(x, y) = 2xy$,
 $f(x, y) = 0$
- P2: $u(x, y) = 2x^3y + \cos(x)$,
 $f(x, y) = 12xy - \cos(x)$
- P3: $u(x, y) = xy^2 + xy^3 + x^2$,
 $f(x, y) = 2 + 2x + 6xy$
- P4: $u(x, y) = x^2 - y^2$,
 $f(x, y) = 0$
- P5: $u(x, y) = x \sin(y) + y \sin(x)$,
 $f(x, y) = -x \sin(y) - y \sin(x)$

where $u(x, y)$ gives the exact solution, and $f(x, y)$ gives the right-hand function of (18). All five problems are required to be solved within the given error $\varepsilon = 10^{-4}$. The maximum number of iterations allowed for each algorithm is 1000.

Table VI shows the number of iterations needed for each algorithm to find an approximate solution with an error less than $\varepsilon = 10^{-4}$. The error was checked every ten iterations. From the table, the hybrid algorithm performed consistently better than the SOR method. It could find an approximate solution within the given error in less than one-third of the time needed by the SOR method for four problems and in less than half the time for the other.

2) *Results of the Second Hybrid Algorithm:* The second hybrid algorithm made use of different discretizations of the partial differential equation (18). The first discretization was the same as that used previously for the first hybrid algorithm. The discrete system as given by (19) was obtained using the five-point difference method. The discrete system could be solved by the SOR method using iteration 20.

TABLE VI
NUMBER OF ITERATIONS NEEDED BY THE SOR METHOD AND THAT NEEDED
BY THE FIRST HYBRID ALGORITHM TO SOLVE THE FIVE PROBLEMS
TO THE GIVEN PRECISION ($\varepsilon_0 = 10^{-4}$)

Problem	SOR with $\omega = 1.25$	SOR with $\omega = 1.75$	Hybrid algorithm with initial $\omega = 1.25$ and 1.75	
P1	> 1000	990	270	270
P2	> 1000	> 1000	390	390
P3	> 1000	> 1000	380	380
P4	> 1000	380	160	160
P5	> 1000	980	260	260

The second discretization of partial differential equation (18) used the nine-point difference method. The second discrete system obtained was as follows:

$$\begin{cases} (-u_{i+2,j} + 16u_{i+1,j} - 30u_{i,j} \\ + 16u_{i-1,j} - u_{i-2,j})/h^2 \\ + (-u_{i,j-2} + 16u_{i,j-1} - 30u_{i,j} \\ + 16u_{i,j-1} - u_{i,j-2})/h^2 \\ = f_{i,j}, & i, j = 2, \dots, n-2 \\ u_{0,j} = 0, & j = 0, \dots, n \\ u_{1,j} = \sin(10jh), & j = 0, \dots, n \\ u_{i,0} = 0, & i = 0, \dots, n \\ u_{i,1} = \sin(10ih), & i = 0, \dots, n \end{cases} \quad (22)$$

where the mesh $h = 0.01$, $n = 1/h$ and $f_{i,j} = -100((ih)^2 + (jh)^2) \sin(10ihjh)$.

In the above system, the values of $\{v_{1,j}, v_{n-1,j}, v_{i,1}, v_{i,n-1}, i, j = 1, \dots, n-1\}$ were not given. We used the values from the five-point difference method instead.

The second discrete system could be solved by the SOR method using the following iteration: for $i, j = 2, \dots, n-2$

$$u_{i,j}^{(k+1)} = \omega \left(16u_{i,j+1}^{(k)} + 16u_{i,j-1}^{(k)} + 16u_{i+1,j}^{(k)} + 16u_{i-1,j}^{(k)} - u_{i,j+2}^{(k)} - u_{i,j-2}^{(k)} - u_{i+2,j}^{(k)} - u_{i-2,j}^{(k)} - h^2 f_{i,j} \right) / 60 + (1 - \omega) u_{i,j}^{(k)} \quad (23)$$

where $\omega (0 < \omega < 2)$ is the relaxation factor.

The implementation details of the second hybrid algorithm are as follows.

- 1) Initialize the population with two individuals: $u_{i,j}^{(0)} = 0.0$ and $v_{i,j}^{(0)} = 0.0$ for $i, j = 1, \dots, n-1$, where $u_{i,j}^{(0)}$ and $v_{i,j}^{(0)}$ are approximate solutions to the first and second discrete systems, respectively. Let $k \leftarrow 0$.
- 2) Recombine $u_{i,j}^{(k)}$ and $v_{i,j}^{(k)}$ to generate two individuals as follows:

$$\begin{aligned} u_{i,j}^{(k+C)} &= (1 - \eta_u) u_{i,j}^{(k)} + \eta_u v_{i,j}^{(k)}, & i, j = 2, \dots, n-2 \\ v_{i,j}^{(k+C)} &= (1 - \eta_v) v_{i,j}^{(k)} + \eta_v u_{i,j}^{(k)}, & i, j = 2, \dots, n-2 \end{aligned}$$

and

$$\begin{cases} v_{i,1}^{(k+C)} = u_{i,1}^{(k)}, & i = 1, \dots, n-1 \\ v_{i,n-1}^{(k+C)} = u_{i,n-1}^{(k)}, & i = 1, \dots, n-1 \\ v_{1,j}^{(k+C)} = u_{1,j}^{(k)}, & j = 1, \dots, n-1 \\ v_{n-1,j}^{(k+C)} = u_{n-1,j}^{(k)}, & j = 1, \dots, n-1. \end{cases}$$

- 3) Mutate $u_{i,j}^{(k+C)}$ and $v_{i,j}^{(k+C)}$ with one SOR iteration using (20) and (23), respectively

$$u_{ij}^{(k+M)} = (\omega_u + p_u) \left(u_{i,j+1}^{(k+C)} + u_{i,j-1}^{(k+C)} + u_{i+1,j}^{(k+C)} + u_{i-1,j}^{(k+C)} - h^2 f_{i,j} \right) / 4 + (1 - \omega_u) u_{i,j}^{(k+C)},$$

for $i, j = 1, \dots, n-1$ (24)

where $p_u \in (-0.01, 0.01)$ is a random number, and

$$v_{ij}^{(k+M)} = (\omega_v + p_v) \left(16v_{i,j+1}^{(k+C)} + 16v_{i,j-1}^{(k+C)} + 16v_{i+1,j}^{(k+C)} + 16v_{i-1,j}^{(k+C)} - v_{i,j+2}^{(k+C)} + v_{i,j-2}^{(k+C)} - v_{i+2,j}^{(k+C)} - v_{i-2,j}^{(k+C)} - h^2 f_{i,j} \right) / 60 + (1 - \omega_v) v_{i,j}^{(k+C)},$$

for $i, j = 2, \dots, n-2$ (25)

where $p_v \in (-0.01, 0.01)$ is a random number.

- 4) The mutated individuals replace the old individuals, i.e., $u^{(k+1)} = u^{(k+M)}$ and $v^{(k+1)} = v^{(k+M)}$. Let $k \leftarrow k+1$.
- 5) If the error $\|e\|$ (as defined in the first hybrid algorithm) is less than the given value ε_0 , then terminate the algorithm; otherwise, go to step 2).

Table VII gives the errors produced by the SOR method and the second hybrid algorithm. An interesting observation can be made from this table. The second hybrid algorithm was not able to converge as fast as the SOR method initially since no adaptation of relaxation factors was used. However, it kept improving its approximation constantly until 1600 iterations, while the SOR method was unable to improve its approximation significantly after 1100 iterations. The hybrid algorithm outperformed the SOR method quickly after 1200 iterations.

Five more problems (problems P1–P5) were also used as test problems for the second hybrid algorithm

- P1: $u(x, y) = 2xy,$
 $f(x, y) = 0$
- P2: $u(x, y) = x^2 - y^2,$
 $f(x, y) = 0$
- P3: $u(x, y) = \sin(10x) + \cos(10y),$
 $f(x, y) = -100 \sin(10x) - 100 \cos(10y)$
- P4: $u(x, y) = \sin(10x) \sin(10y),$
 $f(x, y) = -200 \sin(10x) \sin(10y)$
- P5: $u(x, y) = xy + \sin(10xy),$
 $f(x, y) = -100(x^2 + y^2) \sin(10xy)$

TABLE VII
 ERRORS PRODUCED BY THE SOR METHOD AND THE SECOND HYBRID ALGORITHM; RESULTS HAVE BEEN AVERAGED OVER TEN INDEPENDENT RUNS

Iteration	SOR with	SOR with	Hybrid algorithm with	
	$\omega = 1.25$	$\omega = 1.75$	$\omega = 1.75$ and 1.75	
100	7.74876e-01	3.39587e-01	4.24024e-01	4.30060e-01
200	5.96559e-01	1.08033e-01	1.64685e-01	1.66863e-01
300	4.59065e-01	4.52751e-02	7.54606e-02	7.62545e-02
400	3.55212e-01	2.15914e-02	3.98919e-02	4.02557e-02
500	2.77599e-01	1.05872e-02	2.22857e-02	2.24808e-02
600	2.19625e-01	5.21141e-03	1.26483e-02	1.27590e-02
700	1.76055e-01	2.57598e-03	7.21061e-03	7.27506e-03
800	1.42990e-01	1.40235e-03	4.11729e-03	4.15583e-03
900	1.17434e-01	9.25236e-04	2.35180e-03	2.37549e-03
1000	9.73326e-02	7.10448e-04	1.34283e-03	1.35805e-03
1100	8.12522e-02	6.08813e-04	7.65994e-04	7.76363e-04
1200	6.81884e-02	5.59487e-04	4.36153e-04	4.43753e-04
1300	5.74352e-02	5.35215e-04	2.47604e-04	2.53550e-04
1400	4.84904e-02	5.23154e-04	1.39782e-04	1.44779e-04
1500	4.09909e-02	5.17161e-04	9.45556e-05	8.25767e-05
1600	3.47030e-02	5.14227e-04	9.58485e-05	4.70044e-05

where $u(x, y)$ gives the exact solution, and $f(x, y)$ gives the right-hand side function in (18). Table VIII compares the results of the first and second hybrid algorithms.

It is interesting to note from Table VIII that the second hybrid algorithm, which used two different discretization but no adaptation of the relaxation factor, performed consistently better than the first hybrid algorithm for all five problems, although the difference was not huge. This appeared to indicate that having a finer discretization and mixing it with a different discretization helped to improve the performance of hybrid algorithms, even without self-adaptation of relaxation factors. The recombination operator used in the hybrid algorithms was able to exchange useful information between different individuals and find better solutions.

Table IX compares the error and the number of iterations used to find approximate solutions by the SOR and the second hybrid algorithm. The table shows that the second hybrid algorithm could find more accurate approximate solutions than the SOR method using a slightly higher number of iterations.

IV. FUTURE WORK AND CONCLUSIONS

This paper has proposed three hybrid algorithms for solving linear and partial differential equations. The significance of this work lies in the novel use of evolutionary computation techniques in an area where they had seldomly been used. The hybrid algorithms integrate the classical SOR method with evolutionary computation techniques. The recombination operator in the hybrid algorithms mixes two parents by a kind of averaging, which is similar to the intermediate recombination often used

TABLE VIII
 ERRORS PRODUCED BY THE FIRST AND SECOND HYBRID ALGORITHMS
 USING THE FIVE-POINT DIFFERENCE METHOD

Problem	First Hybrid Algorithm		Second Hybrid Algorithm	
	$\omega = 1.25$ and 1.75		$\omega = 1.75$ and 1.75	
P1	3.55271e-15	3.96378e-15	1.99840e-15	1.99840e-15
P2	1.55431e-15	1.66533e-15	8.88178e-16	9.99201e-16
P3	1.87232e-03	1.87232e-03	6.94864e-04	5.83984e-04
P4	9.63844e-04	9.63806e-04	3.81461e-04	2.76922e-04
P5	4.81240e-04	4.81140e-04	1.87149e-04	1.60460e-04

TABLE IX
 ERRORS AND NUMBER OF ITERATIONS NEEDED BY THE SOR METHOD
 AND THE SECOND HYBRID ALGORITHM; NUMBER OF ITERATIONS IS
 REPRESENTED BY THE NUMBER IN THE BRACKETS

Problem	SOR	Second Hybrid Algorithm	
	$\omega = 1.75$	$\omega = 1.75$ and 1.75	
P1	3.96378e-15 (5430)	1.99840e-15 (6460)	1.99840e-15 (6460)
P2	1.66533e-15 (5170)	8.88178e-16 (5850)	9.99201e-16 (5850)
P3	1.87232e-03 (1080)	6.94864e-04 (1130)	5.83984e-04 (1130)
P4	9.63806e-04 (970)	3.81461e-04 (1070)	2.76922e-04 (1070)
P5	4.81140e-04 (2810)	1.87149e-04 (3120)	1.60460e-04 (3120)

in evolution strategies [6], [7]. The mutation operator is equivalent to one iteration in the SOR method. However, the mutation is stochastic as a result of stochastic self-adaptation of the relaxation parameter ω .

The proposed hybrid algorithms differ from most evolutionary algorithms. They integrate evolutionary computation techniques with classical methods, rather than using an evolutionary algorithm as a wrapper around the classical method. The hybrid algorithms, as described in this paper, can have different variants, depending on how recombination and mutation are implemented. The algorithms can be made more stochastic by introducing random recombination. Our current implementation uses random mutation, but deterministic recombination.

Numerical experiments with various test problems have shown that the hybrid algorithms perform much better than the classical SOR method. They are more efficient and robust than the classical method. They are also very simple and easy to implement.

Because the SOR method is very sensitive to the relaxation factor ω , it is often necessary to guess an appropriate relaxation factor by trial and error. For example, we knew $\omega = 1.75$ gave the SOR method the best performance for the problems we considered here only after we had experimented with $\omega = 1, 1.25, 1.5, 1.75$. However, the hybrid algorithms presented in this paper were much more robust and much less dependent on the initial values of ω . There was no need for any preliminary experiments to estimate the relaxation factor. Hence, in general, the hybrid algorithms need much less time to use than the SOR method. In addition, the population nature of the hybrid algorithms makes parallelization of the algorithms very straightforward by running each individual on a processor.

Future work can be done to further improve the hybrid algorithms proposed in this paper. First, self-adaptation of relaxation factors can be introduced into the second hybrid algorithm for solving partial differential equations when different discretizations are used. Second, the impact of an increased population size on the performance of hybrid algorithms should be studied. Third, the impact of different recombination parameters (e.g., matrix R , η_u , and η_v) should be investigated. Fourth, the self-adaptation scheme for relaxation factors should be studied further. Fifth, parallel hybrid algorithms should be investigated. Lastly, the impact of multiple discretizations (more than two) on the performance of hybrid algorithms should be studied.

ACKNOWLEDGMENT

The authors are grateful to the three anonymous referees and D. Fogel for their constructive comments.

REFERENCES

- [1] J. Ortega, *Introduction to Parallel and Vector Solution of Linear System*. New York: Plenum, 1989.
- [2] G. F. P. J. F. Botha, *Fundamental Concepts in the Numerical Solution of Differential Equations*. Chichester, U.K.: Wiley, 1983.
- [3] D. B. Fogel and J. W. Atmar, "Comparing genetic operators with Gaussian mutations in simulated evolutionary process using linear systems," *Biol. Cybern.*, vol. 63, no. 2, pp. 111–114, 1990.
- [4] D. Young, *Iterative Solution of Large Linear System*. New York: Academic, 1971.
- [5] R. Salomon and J. L. van Hemmen, "Accelerating backpropagation through dynamic self-adaptation," *Neural Networks*, vol. 9, no. 4, pp. 589–601, 1996.
- [6] H.-P. Schwefel, *Evolution and Optimum Seeking*. New York: Wiley, , 1995.
- [7] T. Bäck, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: Comments on the history and current state," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 3–17, Jan. 1997.