Comput Geosci (2013) 17:249–268
DOI 10.1007/s10596-012-9328-9

ORIGINAL PAPER

Hybrid differential evolution and particle swarm optimization for optimal well placement

E. Nwankwor · A. K. Nagar · D. C. Reid

Received: 15 February 2012 / Accepted: 9 October 2012 / Published online: 17 November 2012

Abstract There is no gainsaying that determining the optimal number, type, and location of hydrocarbon reservoir wells is a very important aspect of field development planning. The reason behind this fact is not farfetched—the objective of any field development exercise is to maximize the total hydrocarbon recovery, which for all intents and purposes, can be measured by an economic criterion such as the net present value of the reservoir during its estimated operational life-cycle. Since the cost of drilling and completion of wells can be significantly high (millions of dollars), there is need for some form of operational and economic justification of potential well configuration, so that the ultimate purpose of maximizing production and asset value is not defeated in the long run. The problem, however, is that well optimization problems are by no means trivial. Inherent drawbacks include the associated computational cost of evaluating the objective function, the high dimensionality of the search space, and the effects of a continuous range of geological uncertainty. In this paper, the differential evolution (DE) and the particle swarm optimization (PSO) algorithms are applied to well placement problems. The results emanating from both algorithms are compared with results obtained by applying a third algorithm called hybrid particle swarm differential evolution (HPSDE)—a product of the hybridization of DE and PSO algorithms. Three cases involving the placement of vertical wells in 2-D and 3-D reservoir models are considered. In two of the three cases, a *max-mean objective* robust optimization was performed to address geological uncertainty arising from the mismatch between real physical reservoir and the reservoir model. We demonstrate that the performance of DE and PSO algorithms is dependent on the total number of function evaluations performed; importantly, we show that in all cases, HPSDE algorithm outperforms both DE and PSO algorithms. Based on the evidence of these findings, we hold the view that hybridized metaheuristic optimization algorithms (such as HPSDE) are applicable in this problem domain and could be potentially useful in other reservoir engineering problems.

Keywords Differential evolution · DE · Particle swarm optimization · PSO · Hybridization · HPSDE · Reservoir simulation · Well placement optimization

E. Nwankwor (✉) · A. K. Nagar · D. C. Reid
Centre for Applicable Mathematics and Systems Science,
Department of Mathematics and Computer Science,
Liverpool Hope University, Liverpool, UK
e-mail: e.nwankwor@liverpool.ac.uk

A. K. Nagar
e-mail: nagara@hope.ac.uk

D. C. Reid
e-mail: reidd@hope.ac.uk

E. Nwankwor
Department of Geosciences, University of Liverpool,
Liverpool, UK

1 Introduction

During the field development planning phase of hydrocarbon reservoirs, optimal well configuration is arguably the most important decision input. It is generally a nontrivial task which has a significant bearing on the asset value of the project, as it can potentially determine the recoverability of the hydrocarbon in place.

The fact, however, is that the very nature of subsurface reservoirs make well configuration optimization a very challenging problem. Hydrocarbon reservoir models are generally a system of highly nonlinear equations with large number of time-varying states, time-invariant parameters, and geological uncertainties. It is therefore essential that optimization algorithms for this purpose are efficient, reliable, and robust. In this work, we apply three metaheuristic algorithms—differential evolution (DE), particle swarm optimization (PSO), and hybrid particle swarm differential evolution (HPSDE)—to well placement problem. We compare the results from all three algorithms; and we show that on the average, the HPSDE algorithm outperforms both the DE and the PSO algorithms. We also show that the performance of DE and PSO are dependent on the total number of simulations performed.

The DE is a population-based metaheuristic algorithm that is modeled on Darwin's evolutionary principle of "survival of the fittest." It was introduced as a computational intelligence paradigm in [60], and it is a member of the evolutionary algorithm family. Like the well-known genetic algorithm (GA), it is based on the theory of natural selection, and they both use evolutionary operators such as crossover, mutation, and selection to maintain a population of potential solutions. In both DE and GA algorithms, the selection operator is the sole mechanism for choosing the best individuals from the population in every generation (or iteration); thus, many researchers have reported DE as an improved version of GA [16]. It is noted, however, that there are salient differences between both algorithms. While GA relies either on binary or real-valued (continuous) strings, DE operates directly on floating point vectors; whereas GA maintains a genetic link from one generation to another, DE is an abstraction of evolution at individual behavioral level; and most importantly, GA relies mainly on the crossover operator to explore the search space, while a special form of mutation operator effects the working of DE. In other words, crossover and mutation mechanisms are the dominant operator in GA and DE algorithms, respectively. Over the past years, DE has been shown to be a simple yet versatile metaheuristic algorithm for real parameter optimization and global optimization in general [16, 56, 61]. It is arguably one of the hottest topics in today's computational intelligence research domain, and the spurt in interest in this subject is evident from the breathtaking wide array of application areas—science, engineering, statistics, economics, and finance. However, this does not appear to be the case in reservoir engineering or in the area of oil field development planning. To the best of our knowledge, besides [68] and [27] where DE is employed in history matching, there has not been much in the literature on the application of DE in reservoir engineering problem domains.

The PSO algorithm was originally introduced in terms of social and cognitive behavior by Kennedy and Eberhart [21, 35]. As a computational paradigm, it attempts to mimic the social interaction exhibited by social animal groups such as flocks of birds and schools of fish. Like most biologically inspired optimization algorithms, PSO is population-based, and the individuals that make up the population are referred to as particles. The popularity of PSO stems from the simplicity of its implementation, and the computational efficiency of information-sharing within the algorithm—as it derives its internal communications from the social behavior of the particles that make up the population. Thus, there are various applications of PSO in diverse engineering and computational science disciplines across the literature. However, like DE, the application of PSO in reservoir engineering is exiguous. We note that [20] and [47] are the only known applications of PSO in well placement optimization problem domain.

Well configuration optimization often entails finding the number, type, location, and possibly drilling sequence of reservoir wells in order to maximize recovery factor, production rates, and asset value. It is a common practice to associate wells to reservoir grid block cell centers where they are represented by source or sink terms (depending on whether they produce fluid from or inject fluid into the reservoir), and therefore the optimization variables are typically real-valued integers. Since determining the number of wells is clearly an integer problem, the combination of this optimization problem with the optimization of the production settings of the wells leads to a mixed-integer nonlinear problem (MINLP) [36]. However, due to issues bothering on nonconvexity, such MINLPs are extremely difficult to solve. Another drawback for MINLPs is that they require far too many evaluations of the objective function (which in this case entails full reservoir simulation), and this renders it less effective in reservoir engineering applications. To this end, most of the solution methods in well configuration optimization problems are either gradient-based local optimization methods or gradient-free stochastic optimization techniques.

Local optimization methods attempt to find the optimum by iteratively improving upon an initial guess (well placement) until the optimal (albeit local) is determined. The main drawback of these methods is to effectively find the improving directions in which to

alter the initial guess. Bangerth et al. [3] compared the performance of two local techniques, the finite difference gradient (FDG) and the simultaneous perturbation stochastic approximation (SPSA), against the performance of a global technique—the very fast simulated annealing (VFSA)—in optimizing the placement of vertical wells in a 2-D reservoir model. The FDG method attempts to find improving directions by perturbing each of the well location by one grid block in every direction. The obvious drawback of this method is the requirement of $2n + 1$ number of objective function evaluations to compute an improving direction for any n to-be-placed wells. The SPSA method which was earlier employed in [59] is an approximate gradient-based method where the gradients are approximated by random perturbations. To compute the derivative, a random direction in which to alter the wells is generated, and if this change in position of the wells does not yield an improvement in the objective function, then the opposite direction is automatically assumed. This algorithm was shown to perform better than GA in the optimization of vertical wells, and it is noted that the computational requirement for this method is less expensive, as improving direction is definitely found in at most two reservoir simulations. However, the disadvantage of the SPSA algorithm is that the assumed optimal configuration may generally not be the "steepest" one. Another drawback appears in the calculation of new solutions: the step size must be chosen carefully, otherwise there is a risk of finding "solutions" which are not feasible. Thus, the assumed efficiency of this method is questionable. The VFSA technique which is based on standard simulated annealing (SA) shares some semblance with most stochastic approximation algorithms. In all cases considered, both the VFSA and SPSA outperformed the FDG method. Wang et al. [67] applied a gradient-based steepest descent algorithm in optimizing the number and placement of injection wells in a 2-D reservoir model, while Sarma and Chen [57] applied a gradient-based algorithm where the derivative of the objective function is computed with respect to continuous well locations, thereby allowing for arbitrary step size and search directions. A gradient-based algorithm was utilized in [73] for the optimal placement of vertical wells in a 2-D reservoir model. The methods applied in [57] and [73] are based on the same principle. The difference, however, is in the derivatives used and the method of its computation. While the derivative of the objective function in [57] is with respect to continuous well locations, the derivative in [73] is with respect to flow rates. At each discrete time step, an adjoint formulation was used to compute the "rate gradients" for each of the low rate "pseudowells" that are placed at the eight neighboring grid blocks surrounding a current well position. The derivatives (with respect to flow rate) at the pseudowells are then summed, and the well is moved in the direction of the pseudowell with the largest summed gradient. In terms of computational efficiency, these gradient-based approaches are highly reliable; however, they have their drawbacks. The nonconvex nature of the underlying optimization problem inevitably means that they generally contain multiple optima; hence, they are prone to be trapped in local solutions. In addition, discontinuous derivatives arising from the nonsmooth nature of the optimization surface may pose significant problems. It is also important to recognize the fact that gradient information is often not readily available. Adjoint-based formulations, which are a popular and efficient way of computing derivatives, are invasive with respect to the flow simulator. They are therefore only feasible with full access to, and detailed knowledge of, the simulator source code [13]. Besides, the objective function value may be computed with some noise, this therefore means that any computation of derivative estimates is susceptible to lots of inaccuracies.

The gradient-free stochastic optimization techniques by their nature are generally noninvasive with respect to the flow simulator. They treat the simulator as a black box—only objective function values are required and no explicit gradient computations are involved. These methods are therefore much easier to implement than, for example, gradient-based adjoint techniques. However, in a typical "no-free-lunch" fashion (see [70]), this advantage is counterbalanced by a significant deterioration in computational efficiency when compared to gradient-based adjoint formulation approaches. In other words, the gradient-free global methods will tolerate lower performance measures in the hope of finding the global optimum, as opposed to the computationally efficient gradient-based local optimal solutions. This appears to be the reason why there are many applications of stochastic optimization algorithms in the well placement optimization problem domain. In this regard, the binary GA (hereafter referred to simply as GA) appears to be the most frequently used technique. Becker and Song [5] applied SA in optimizing the placement and schedule of horizontal wells that were originally cast as a traveling salesman problem. The use of neural networks (NN) was applied in [11], while GA was employed in [1, 6, 23, 26, 40, 46, 48]. A combination of NN and GA was applied in [72]; PSO was applied in [20, 47], and covariance matrix adaptation evolution strategy

was applied in [8]. Apparently, there appears to be no instance or citation of DE in this problem domain. We note that both DE and PSO are relatively new nondeterministic approaches for global optimization (both algorithms were introduced in 1995); we allude that their relative "young age" may be the reason why their application in well optimization problem domain is almost nonexistent, or at best, few and far between.

Since inception, DE has been employed in various science and engineering problem domains—neural network training [15, 28, 50, 51, 63], optimization of mechanical design [38], aerodynamic shape optimization problem [43, 56], power transmission expansion planning [62], and economic dispatch problems for nonsmooth objective functions [49, 69]. In the same vein, PSO has been used in diverse application areas—dynamic economic dispatch problems [12], quadratic assignment problems [24], pole shape optimization [10], and neural networks training [9, 21, 35]. Despite the popularity and application of both algorithms in the aforementioned areas of science and engineering, they are almost nonexistent in the well placement optimization problem domain, but for [20] and [47]. In [47] the standard PSO was applied to various well optimization problems, and it was shown that on the average, PSO outperforms GA in maximizing the net present value (NPV) of the reservoir models. This finding (result from comparing the performance of GA and PSO) provides the main motivation for this work. We compare the performance of DE and PSO algorithms, and our choice is borne out of the fact that DE is a deviant of GA—the crossover operator in GA is replaced with a special type of differential operator for reproducing offspring in subsequent generations.

The results from both algorithms (DE and PSO) are further compared with the results emanating from a hybrid of both algorithms—HPSDE. The robustness of the results is enhanced by incorporating geological uncertainty in two of the three examples considered. To this end, we espouse [65] where multiple geological realizations were used to account for reservoir uncertainties. In all cases, the performance measure relating to all three algorithms are based on averaged results over multiple runs of the algorithms. Detailed comparison of the results from DE, PSO and HPSDE algorithms highlights some interesting findings. It shows that on the average, HPSDE outperformed both DE and PSO algorithms. The paper is structured as follows: in Section 2, DE, PSO, and HPSDE algorithms are described, and the implementation of all three algorithms as used in this work is presented in Section 3. In Section 4, we present the well optimization examples considered in this work, and discuss the results. Finally, we draw conclusion and propose future work direction in Section 5.

2 DE algorithm

In 1995, Rainer Storn and Kenneth Price proposed a new floating point-encoded, population-based evolutionary algorithm for global optimization, and the new algorithm was named differential evolution. It derived its name from a special kind of differential operator which they invoked when creating new offspring of its population [16]. Being an evolutionary algorithm, DE is based on the Darwin's principle of survival of the fittest, a strategy in which the individuals in a population evolve by improving their fitness value through the probabilistic operations of mutation, recombination, and selection. The individuals are evaluated with respect to their fitness against a defined objective function, and those with superior fitness are selected to compose the population of the next generation. Since inception, several DE strategies have evolved, and a comprehensive naming notation to classify these strategies is presented in [61]. As a standard, the nomenclature of DE strategies is consistent with the DE/*a*/*b*/*c* format, where *a* represents a string denoting the target of the mutation operation, *b* defines the number of difference vectors used in the mutation, and *c* stands for the type of crossover employed. Based on the standard and notations defined above, the most widely used DE strategy is the DE/rand/1/bin. This strategy indicates that the mutation target is randomly selected from the population, and the mutation is performed using a single difference vector, as well as a uniform binomial crossover. The basic DE algorithm consists of four distinct events which are represented as shown in Fig. 1.

The first step is the initialization of a population of candidate solutions N_p at iteration $k = 1$, and this is given by

$$N_p(k) = \left[X_1(k), X_2(k), \ldots, X_{Np}(k)\right] \tag{1}$$

where each candidate solution $X_i(k)$ is a D-dimensional vector containing as many real-valued parameters as the problem dimension D. Each of the candidate solution is given by

$$X_i(k) = \left[x_{i,j}(k), x_{i,j}(k), \ldots, x_{i,D}(k)\right] \tag{2}$$

where $i = 1, 2, \ldots, N_p$ and $j = 1, 2, \ldots, D$.

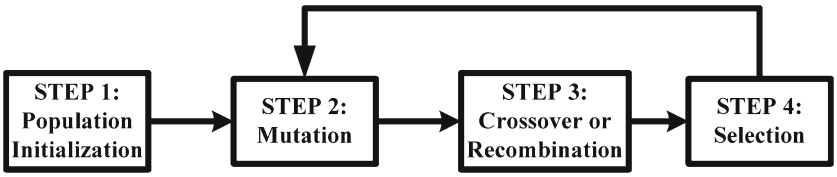


Fig. 1 Basic DE algorithm procedure

Typically, each decision parameter in every candidate solution of the initial population is assigned a randomly chosen value from a predefined feasible numerical bound. In other words, the j-th component of the i-th population member at the initialization step is given by

$$x_{i,j}(1) = x_j^L + \text{rand}(0,1) \cdot \left(x_j^U - x_j^L\right) \tag{3}$$

where x_j^L and x_j^U are, respectively, the lower and upper bound of the j-th component, and rand (0,1) is a uniformly distributed random number between 0 and 1.

Once the population has been initialized, the corresponding fitness value is evaluated and stored in memory for future reference. In each generation, a mutant vector is created for each i-th population member by randomly choosing three parameter vectors from the current population. A scalar number F is used to scale the difference of any two of the three randomly chosen vectors, and the scaled difference is added to the third one. We can express the mutation process of the j-th component of each vector as follows:

$$v_i(k) = x_{r1,j}(k) + F \cdot \left(x_{r2,j}(k) - x_{r3,j}(k)\right) \tag{4}$$

where $F \in [0, 1+]$ is a user-defined constant known as the scaling (or mutation) factor, and vector indices $r1$, $r2$, and $r3$ are randomly chosen with $r1$, $r2$, and $r3$ $\in\{1,2,\ldots,N_p\}$.

Note that $r1 \neq r2 \neq r3 \neq i$ and that x_{r1}, x_{r2}, and x_{r3} are selected anew for each parent vector in every generation. The magnitude and direction of the mutation step is defined by the difference between two of the three randomly chosen population vectors; and this makes the mutation operation to exhibit a self-adaptive behavior, such that the average mutation length decreases as the population converges [61]. The third step is the crossover or recombination scheme. The main purpose of this process is to increase the potential diversity of the population by mixing the parameters of the mutant vector with the target vector according to a selected probability distribution. There are mainly two kinds of crossover schemes—binomial and exponential. The result of the crossover step at iteration k is the birth of a trial vector which is defined as follows:

$$U_i(k) = \left[u_{i,j}(k), u_{i,j}(k), \ldots, u_{i,D}(k)\right]. \tag{5}$$

For a maximization problem, the binomial crossover scheme is performed on each of the D-dimensional variables according to the following equation:

$$u_{i,j}(k) = \begin{cases} v_{i,j}(k), & \text{if rand}(0,1) \leq \text{CR} \\ x_{i,j}(k), & \text{else} \end{cases} \tag{6}$$

where CR is a user-defined crossover rate which is usually in the range of [0,1].

The crossover rate controls the diversity of the population and aids the algorithm to avoid getting stuck in local optima. At the end of the iteration, the selection operator is applied to determine which one of the target and the trial vectors would survive in the next iteration, i.e., at iteration $k = k + 1$. This operator compares the fitness of the trial vectors against the corresponding target vectors and selects the better solution according to the following equation:

$$X_i(k+1) = \begin{cases} U_i(k), & \text{if } f(U_i(k)) \geq f(X_i(k)) \\ X_i(k), & \text{else} \end{cases} \tag{7}$$

where $f(x)$ is a fitness value. Thus, if the new trial vector yields a better fitness value, it automatically replaces its target in the next iteration; otherwise, the target vector is retained in the population. In other words, the population must either get better (w.r.t. the fitness value) or remain constant; it never deteriorates. Figure 2 is a simple flowchart that illustrates the DE algorithm. The control parameters for this algorithm are the mutation factor F, the crossover rate CR, and the population size N_p.

2.1 Treatment of infeasible solutions

Evolutionary algorithms such as DE were originally proposed to solve unconstrained optimization problems. The application of these algorithms on boundary-constrained problems such as well placement optimization problem may result in solutions that violate the physical boundary of the search space. All such bound offending values are termed infeasible solutions, and a comprehensive review of methods for preserving feasibility of solutions is available in [44]. In this work, however, we employ the "out-of-bound value" technique [17, 39, 44]. This involves the use of specialized operators to create and retain candidate solutions that are feasible. It is implemented in accordance to the

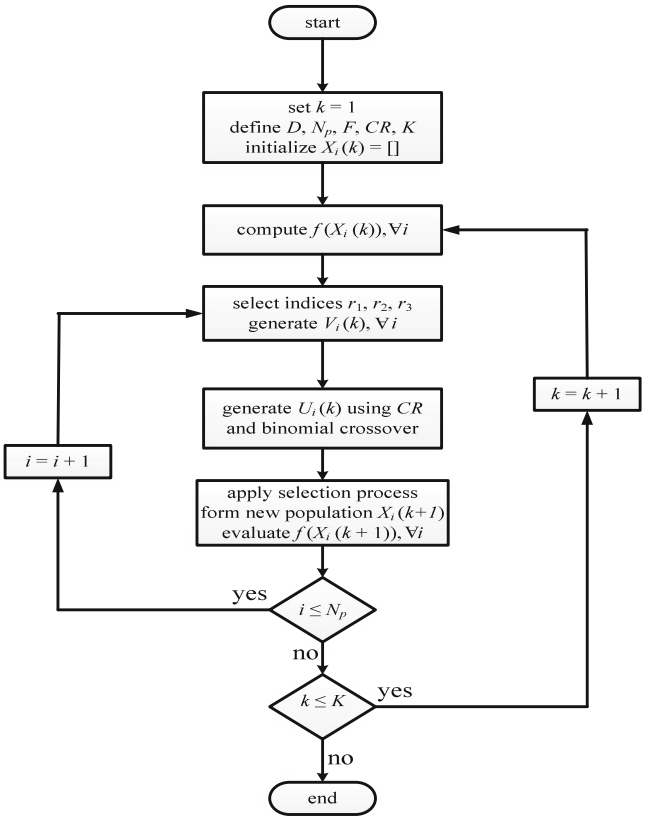


Fig. 2 Flowchart showing the DE algorithm

following equation:

$$x_{i,j}(k) = \begin{cases} x_{i,j}^{\min} & \text{if } x_{i,j}(k) \leq x_{i,j}^{\min} \\ x_{i,j}^{\max} & \text{if } x_{i,j}(k) \geq x_{i,j}^{\max} \\ x_{i,j}(k) & \text{otherwise} \end{cases} \tag{8}$$

where $x_{i,j}^{\min}$ and $x_{i,j}^{\max}$ are, respectively, the minimum and maximum bound of the j-th component of the i-th population member. The application of Eq. 8 ensures that all bound offending values are reset to boundary values.

2.2 PSO algorithm

It can be said that the basic idea behind the PSO algorithm is the simulation of the social behavior metaphor of bird flocks and fish schools. It is a population-based stochastic algorithm which has been widely and efficiently deployed in nonlinear optimizations of varying complexities. Introduced in 1995 [21, 35], its popularity has gained momentum because of its low memory requirement, high computational efficiency, and easy-to-implement properties. Being a population-based algorithm, the individuals of the population are referred to as particles, and a collection of particle at any given iteration is called the swarm. The particles are flown through the search space, with each particle representing a possible or potential solution of the optimization problem. In any given iteration, a particle's fitness is based on a performance function related to the optimization problem, or in other words, the position of each particle is continually adjusted according to its relative fitness and position to other particles that make up the swarm. The movement of the particles across the search space is influenced by two factors: information from iteration-to-iteration and information from particle-to-particle interactions. Based on iteration-to-iteration information, the particle stores in its memory the best solution attained so far, and it experiences an attraction towards this solution (called *pbest*) as it traverses across the problem search space. On the other hand, the outcome of the particle-to-particle information is that each particle stores in its memory the best solution (called *gbest*) attained by any particle in the swarm and experiences an attraction towards this solution. The first and second factors are, respectively, referred to as the cognitive and social components of the algorithm. At the end of each iteration, the *pbest* and *gbest* are updated for each particle, and this update process continues iteratively until the desired result is converged upon, or it is determined that an acceptable solution cannot be found within available computational limit.

At iteration k, if the position of the i-th particle of the swarm across the search space is represented by a D-dimensional vector $\mathbf{x}_i(k)$, and the velocity of this particle is given by vector $\mathbf{v}_i(k)$, if the best position found in the search space by particle i up to iteration k is represented by another vector $\mathbf{y}_i(k)$, and if the best position found by any of the particles in the neighborhood of particle i up to iteration k is represented by yet another vector $\mathbf{y}^*(k)$, then we can mathematically represent all four vectors as:

$$\mathbf{x}_i(k) = \left[x_{i,1}(k), x_{i,2}(k), \ldots, x_{i,D}(k)\right] \tag{9}$$

$$\mathbf{v}_i(k) = \left[v_{i,1}(k), v_{i,2}(k), \ldots, v_{i,D}(k)\right] \tag{10}$$

$$\mathbf{y}_i(k) = \left[y_{i,1}(k), y_{i,2}(k), \ldots, y_{i,D}(k)\right] \tag{11}$$

$$\mathbf{y}^*(k) = \left[y_{i,1}^*(k), y_{i,2}^*(k), \ldots, y_{i,D}^*(k)\right]. \tag{12}$$

At the next iteration $(k+1)$, the position and velocity vectors are updated accordingly, and the new position

of particle i can be computed with respect to its previous position $\mathbf{x}_i(k)$ by adding an updated velocity vector to the previous position vector:

$$\mathbf{x}_i(k+1) = \mathbf{x}_i(k) + \mathbf{v}_i(k+1). \tag{13}$$

The elements of the updated velocity vector $\mathbf{v}_i(k+1)$ are given by

$$v_{i,j}(k+1) = v_{i,j}(k) + c_1 r_{1,j}\left(y_{i,j}(k) - x_{i,j}(k)\right) + c_2 r_{2,j}\left(y_j^*(k) - x_{i,j}(k)\right) \tag{14}$$

where $j = 1, 2, \ldots, D$ represents the components or dimension of the search space; c_1 and c_2 are constants called cognitive and social scaling parameters, respectively; and $r_{1,j}$ and $r_{2,j}$ are random numbers drawn from a uniform distribution [0, 1].

Equations 13 and 14 represent the classical version of the PSO algorithm [18]. The concept of an inertia weight (ω) was developed to better control the exploration and exploitation abilities of the algorithm. It was incorporated into the algorithm and was first reported in the literature in 1998 by Shi and Eberhart [58]. The resulting velocity update equation is given by

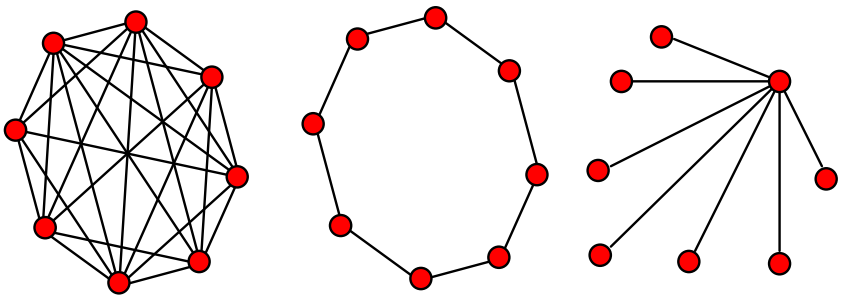
$$v_{i,j}(k+1) = \underbrace{\omega\, v_{i,j}(k)}_{\text{inertia term}} + \underbrace{c_1 r_{1,j}\left(y_{i,j}(k) - x_{i,j}(k)\right)}_{\text{cognitive term}} + \underbrace{c_2 r_{2,j}\left(y_j^*(k) - x_{i,j}(k)\right)}_{\text{social scaling term}} \tag{15}$$

Equations 13 and 15 define the standard version of the PSO algorithm. As indicated above, Eq. 15 is the sum of three components, namely the inertia, the cognitive, and the social scaling components. The inertia component (in apparent reference to its relationship with the inertia weight ω), defines the particle's momentum, and it causes the particle to continue in the direction in which it is moving at iteration k in accordance to the second law of motion. The cognitive component (in apparent reference to its relationship with the cognitive parameter c_1) captures the particle's memory with respect to its previously attained best position; it provides a velocity component in this direction and is responsible for local search. The third term, which is called the social component (in apparent reference to its relationship with the social scaling parameter c_2), represents information stored in memory about the best position of any particle in the neighborhood of particle i and causes movement towards this particle. This component is responsible for global search. Thus, the position of each particle at every instance is determined by its momentum, its memory, and the collective experience of other particles in the swarm.

Of the three components highlighted above, it appears that the social component have the greatest influence on the overall performance of the PSO algorithm. This is because an individual particle (on its own) has little or no power to solve any problem whatsoever; problem solving can only take place when the particles in the swarm interact. In other words, the effectiveness or otherwise of problem solving by PSO is a population-wide phenomenon, emerging from the individual behaviors of the particles through their interactions with one another (the swarm), and in accordance to some sort of communication structure called neighborhood topologies.

The topology typically consists of bidirectional edges connecting pairs of particles, so that if a particle j is in the neighborhood of another particle i, then particle i is also in j's neighborhood. Each particle communicates with some other particles and is affected by the best point found by any member of its topological neighborhood. Several types of PSO neighborhood topologies have been reported; however, it is noted that PSO algorithms with small neighborhoods perform better on complex problems while PSO algorithms with large neighborhoods perform better for simple problems [34]. The k-best topology which was proposed in [34] connects every particle to its k-nearest particles in the topological space. Generally speaking, there are as many neighborhoods as there are particles in a given swarm. This is so because each of the particles can form its own neighborhood in its own right. However, when $k = 1$, the neighborhoods of individual particles are the same for all particles. In this special case, we have one neighborhood which has a star topology where each particle has a direct link to every other particle in the neighborhood as shown in Fig. 3a. Although they have fast convergence properties, PSO algorithms using this topology are susceptible to premature convergence and are generally referred to as *global best* or "gbest" algorithms [21, 22, 34].

With $k = 2$, this becomes the circle (or ring) topology where each particle is directly linked to two adjacent particles in its topological space as shown in Fig. 3b. There are diverse neighborhood topologies that have been reported in the PSO literature. This includes the wheel topology, which effectively isolates the particles from one another, as information is communicated to other particles through a focal (or central) particle as shown in Fig. 3c. We note that besides the star topology, PSO algorithms using other topologies are referred to as *local best* or "lbest" algorithms [22]. The flowchart illustrating the standard PSO algorithm is shown in Fig. 4.



(a) Star topology (b) Ring or Circular topology (c) Wheel topology

Fig. 3 Examples of PSO neighborhood topologies. **a** Star topology. **b** Ring or circular topology. **c** Wheel topology

Like in DE, one of the challenges in implementing PSO in boundary-constrained optimization problems is the issues that arise from infeasible solutions. Direct use of Eqs. 13 and 15 on boundary-constrained problems may result to solutions that violate the physical boundary. To handle this issue, we employ the "absorb" technique [14, 47] by applying the following equations:

$$x_{i,j}(k+1) = \begin{cases} l_j \text{ if } x_{i,j}(k+1) < l_j \\ u_j \text{ if } x_{i,j}(k+1) > u_j \end{cases} \quad (16)$$

$$v_{i,j}(k+1) = \begin{cases} 0 \text{ if } x_{i,j}(k+1) < l_j \\ 0 \text{ if } x_{i,j}(k+1) > u_j \end{cases} \quad (17)$$

where l_j and u_j are the lower and upper bounds of the j-th component of the search space.

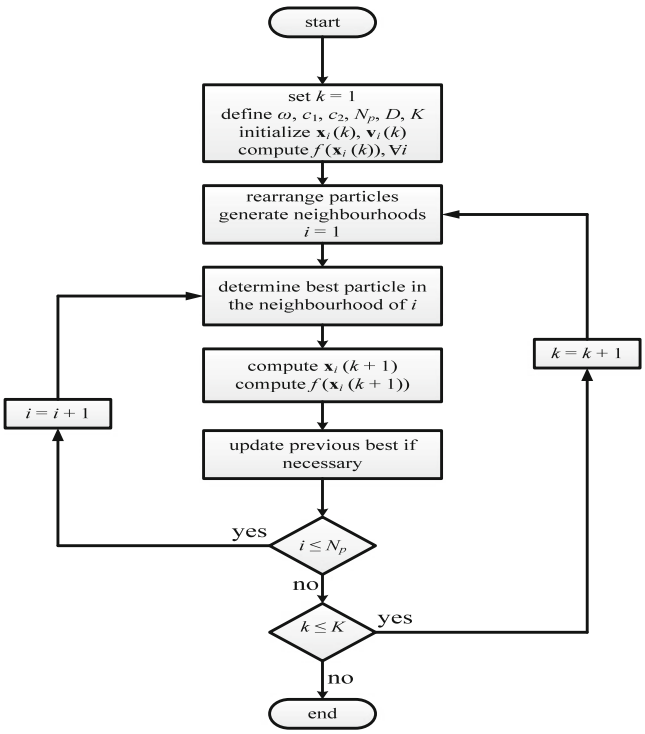


Fig. 4 Flowchart showing the PSO algorithm

While Eq. 16 has the effect of moving infeasible solutions to the nearest boundary by setting all variables outside the feasible region to the nearest bound, Eq.17 has the effect of halting the affected particles by setting their velocities to zero. In any case, the absorb technique works in a very similar fashion with the out-of-bound value technique that we employ in DE.

2.3 Hybridization and HPSDE algorithm

Though classified as global optimization techniques, DE and PSO have their own drawbacks. They are susceptible to premature convergence which can lead to potential solutions being trapped in local optima. This shortcoming is even more pronounced in domains where the search space is nonlinear, noncontinuous, and nonsmooth, as often the case in many reservoir engineering problems. To overcome this disadvantage, researchers have employed various hybridization techniques to create hybrid metaheuristic algorithms that are more robust and effective in problem solving. In a general sense, hybridization is simply an attempt at combining the good traits of participating algorithms or concepts, with the ultimate view of improving the efficiency and capabilities of the newly created "hybrid" algorithm. We justify the use of hybridization as a direct consequence of the so called no-free-lunch theorem. Wolpert and Macready [70] established that any elevated performance over one class of problem by any algorithm is offset by performance over another class of problem. This underlines the fact that no single optimization technique can solve all optimization problems optimally.

Generally speaking, most of the hybrid metaheuristics that have been published in the literature can be loosely grouped into three categories: those created by combining one metaheuristic algorithm with another metaheuristic algorithm; those developed by combining a standard metaheuristic algorithm with mathematical

operators; and those developed by incorporating evolutionary operators (selection, mutation, and crossover) into nonevolutionary metaheuristic algorithms.

Among the three evolutionary operators, mutation appears to be the most commonly applied operator in the hybridization of nonevolutionary metaheuristic algorithms such as PSO. The purpose of applying mutation to PSO is to increase the diversity of the population and enable the PSO to escape local optima [7, 31, 37, 41, 45, 54]. In [33], mutation alongside crossover and elitism is incorporated into PSO, and the resulting algorithm outperformed both PSO and GA in recurrent network design problem. The selection operator (which entails copying the particles with the best performance into the next generation) was applied to a PSO algorithm in [2], and this led to a continuous retention of the best-performing particles. Løvbjerg et al. [42] showed that incorporating the crossover operation in PSO algorithms effects information-swap between individual particles. A quadratic approximation operator (QA) is used in [19] and [18] to hybridize a binary GA and PSO, respectively. In both cases, the QA operator was used to update a part of the population, while the remaining of the population is updated by GA or PSO as the case may be. The result showed a substantial improvement in the performance of the hybrid algorithm when used to solve a set of 15 benchmark problems.

Collision-avoiding mechanisms are designed in [7] and [37] to prevent particles from colliding with each other, and a "dissipative particle swarm" is designed in [71] by adding negative entropy into PSO to discourage premature convergence. Better results were obtained in [55] by applying a hybrid of PSO and GA in the profiled corrugated horn antenna optimization problem, while a PSO hybrid with GA crossover operator was introduced in [32] to optimize difficult real number optimization problems. Genetical swarm optimization is presented by systematically combining PSO and GA in [25]. The population in each iteration is divided into two parts, and these parts are evolved with the two techniques, respectively. They are then recombined in the updated population and further divided into two random parts for another run of GA and PSO in the next iteration.

Poli et al. [52, 53] proposed a hybrid PSO based on genetic programming (GP), and the GP was used to evolve new laws for the control of particles' movement for specific classes of problems. Ant colony optimization is combined with PSO in [29], while DE and PSO are combined in [30]. The particles in the swarm drift according to position update equation, but occasionally, DE is applied to replace poorly performing particles with better ones while retaining their velocity. The DEPSO algorithm in [74] basically involves the use of DE and canonical PSO operators in alternate generations. The hybrid achieved better results than PSO in problems with higher dimensionality.

From the foregoing, it is evident that a wide array of hybridized metaheuristic algorithms have been designed and implemented for the purpose of improving the performance and problem-solving capabilities of the participating algorithms. A comprehensive (but not exhaustive) review is available in [4]. In this paper, we employ a hybrid of DE and PSO referred to as hybrid particle swarm differential evolution (HPSDE). This is a modified version of the algorithm proposed in [64, 74]. It starts off as a standard DE algorithm up to the point where the trial vector is generated. If the fitness of the trial vector is better than the corresponding target vector, then it is included in the population; otherwise, the algorithm activates the PSO phase and generates a new candidate solution using the position and velocity update equations. The method is repeated iteratively with the hope of finding better solutions or till optimum values are reached. The inclusion of PSO phase creates a perturbation in the population, which in turn helps in maintaining diversity of the population and producing an optimal solution [64]. The HPSDE flowchart is depicted in Fig. 5.

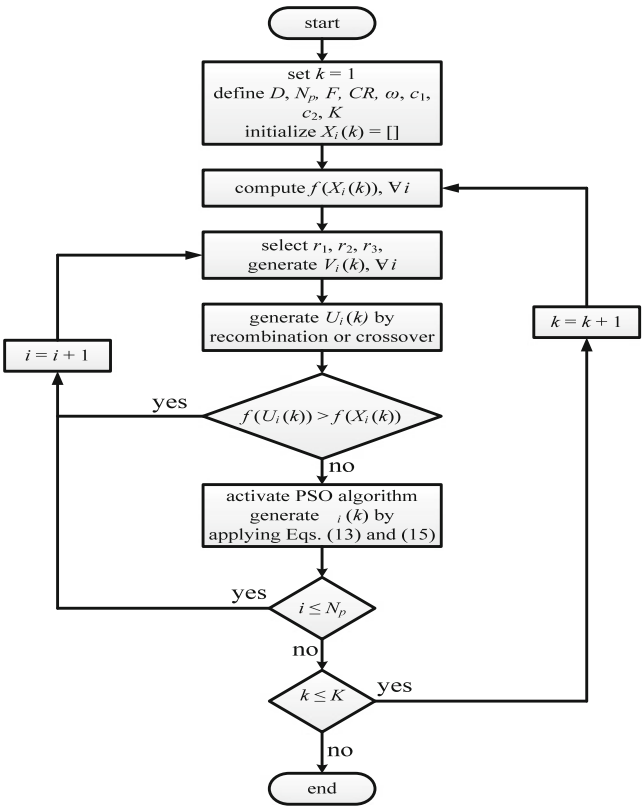


Fig. 5 Flowchart illustrating the HPSDE algorithm

3 Implementation of well placement optimization problem

In this section, we describe the implementation of DE, PSO, and HPSDE algorithms in the well placement optimization problem. The algorithms are given, and a step-by-step implementation is presented.

3.1 Implementation of DE in well placement problem

We describe the implementation of the DE algorithm for a well placement optimization problem. Algorithm 1 presents the steps in the DE/1/rand/bin strategy for a maximization problem. It is adapted and modified from [61] and implemented in MATLAB®. The first step signifies the beginning of the algorithm, and step 2 assigns values to DE parameters N_p, F, CR, D, and K. In step 3 of the algorithm, the population is initialized such that each component $x_{i,j}(k)$, $\forall i \in \{1,2,\ldots,N_p\}, \forall j \in \{1,2,\ldots,D\}$ are made of random elements drawn from predefined lower (L) and upper (U) bounds in accordance to Eqs. 2 and 3. Step 4 computes the objective function of the initialized population, and the evaluated objective function is saved in step 6 for future reference. Steps 9–11 compute a mutant vector $v_i(k)$ in accordance with Eq. 4.

Step 12 is used in order to generate a trial vector U_i in accordance with Eq. 6. Following the birth of the trial vector, if any element of the "newly born" vector are outside the feasible region of the search space, step 13 is activated to modify and adjust the trial vector within the feasible region. Step 14 evaluates the objective function of the trial vectors. Steps 18–25 describe the selection process; the objective function of the trial vector is compared against the target vector in order to determine the population of the next iteration. The algorithm terminates when the maximum number of iterations K is reached.

Algorithm 1 DE Algorithm

1: Set iteration index $k = 1$
2: Define $D, N_p, F = 0.5, CR = 0.1, K$
3: Initialize $X_i(k) : x_{i,j}(k) \sim \text{rand}\,(L_j, U_j) \forall j, \forall i$
4: Compute objective function, $f(X_i(k)), \forall i$
5: **while** $k \le K$ **do**
6: Save, $f(X_i(k)), \forall i$
7: $i = 1$
8: **while** $i \le N_p$ **do**
9: Select $r_1, r_2, r_3 \in \{1, 2, ..., N_p\}, r_1 \ne r_2 \ne r_3 \ne i$
10: Randomly select $j \in \{1, 2, ..., D\}$
11: Compute mutant vector $v_i(k), \forall i$
12: Apply Eq. 6 to generate trial vector $U_i(k), \forall i$
13: Apply Eq. 8 if necessary
14: Compute objective function, $f(U_i(k)), \forall i$
15: $i = i + 1$
16: **end while**
17: $i = 1$
18: **while** $i \le N_p$ **do**
19: **if** $f(U_i(k)) > f(X_i(k))$ **then**
20: $X_i(k+1) = U_i(k)$
21: **else**
22: $X_i(k+1) = X_i(k)$
23: **end if**
24: $i = i + 1$
25: **end while**
26: $k = k + 1$
27: **end while**

3.2 Implementation of PSO in well placement problem

The implementation of the PSO algorithm in well placement optimization problem is given below; it is adopted from [47]. Algorithm 2 presents the steps as implemented in MATLAB®. Like Algorithm 1, the PSO algorithm presented here is for a maximization problem. Step 1 signifies the beginning of the algorithm. Step 2 initializes the values of the PSO parameters ω, c_1, c_2, N_p, and K. Step 3 initializes each component of the particle position, $x_{i,j}(k)$, with random elements drawn from a uniform distribution U, such that $\text{U}, \forall j \in \{1,2,\ldots,D\}, \forall i \in \{1,2,\ldots,N_p\}$. In step 4, the components of the velocity vector, $v_{i,j}(k)$, is initialized in a similar fashion as in step 3. Step 5 computes the objective function of all particles. Step 6 updates the previous best position for each particle. The particle indices (positions of particles in the array of particles) are permuted in step 8, and the neighborhoods for each particle are generated in step 9.

Step 12 determines the best particle in the neighborhood of particle i. The elements of the updated velocity vector of new particle, $v_{i,j}(k+1)$, are computed in accordance with Eq. 15 in step 13. Steps 15–19 update all components of the position of particle i. In step 17, infeasible solutions are modified using Eqs. 16 and 17. Step 20 evaluates the objective function $f(\mathbf{x}_i(k+1))$ based on the new particle position. Steps 24–31 update the previous best positions for each particle, $y_i(k)$, if the new objective function value, $f(\mathbf{x}_i(k+1))$, is better than that at the previous best position, $f(\mathbf{y}_i(k))$. The algorithm terminates when the maximum number of iterations K is reached.

Algorithm 2 PSO Algorithm

1: Set iteration index $k = 1$
2: Define $N_p, \omega = 0.721, c_1 = c_2 = 1.193, K$
3: Initialize $\mathbf{x}_i(k) : x_{i,j}(k) \sim \mathrm{U}(l_j, u_j) \forall j, \forall i$
4: Initialize $\mathbf{v}_i(k) : v_{i,j}(k) \sim \mathrm{U}(0,1) \forall j, \forall i$
5: Compute objective function, $f(\mathbf{x}_i(k)), \forall i$
6: $\mathbf{y}_i(k) = \mathbf{x}_i(k), \forall i$
7: **while** $k \leq K$ **do**
8: Permute the particle indices
9: Generate neighborhood for each particle
10: $i = 1$
11: **while** $i \leq N_p$ **do**
12: Determine best particle in neighborhood of particle i
13: Compute $\mathbf{v}_i(k+1)$ using Eq. 15
14: $j = 1$
15: **while** $j \leq D$ **do**
16: $x_{i,j}(k+1) = x_{i,j}(k) + v_{i,j}(k+1)$
17: Apply Eqs. 16 and 17 if necessary
18: $j = j + 1$
19: **end while**
20: Compute objective function, $f(\mathbf{x}_i(k+1)), \forall i$
21: $i = i + 1$
22: **end while**
23: $i = 1$
24: **while** $i \leq N_p$ **do**
25: **if** $f(\mathbf{x}_i(k+1)) > f(\mathbf{y}_i(k))$ **then**
26: $\mathbf{y}_i(k+1) = \mathbf{x}_i(k+1)$
27: **else**
28: $\mathbf{y}_i(k+1) = \mathbf{y}_i(k)$
29: **end if**
30: $i = i + 1$
31: **end while**
32: $k = k + 1$
33: **end while**

3.3 Implementation of HPSDE in well placement problem

The HPSDE algorithm is a hybrid of DE and PSO. It begins with DE algorithm up to the point where the trial vectors are generated. The fitness of the trial vector is compared with that of the corresponding target vector to determine if it is included in the population of the next iteration or if it is updated using a global best PSO algorithm. By so doing, we combine the global information obtained via PSO algorithm into the DE algorithm thereby maintaining a fair balance between the exploration and exploitation factors of the algorithms. Algorithm 3 presents the steps in HPSDE for a maximization problem. It is adapted and modified from [64] and implemented in MATLAB®.

Algorithm 3 HPSDE Algorithm

1: Set iteration index $k = 1$
2: Define $D, N_p, F, K, CR, \omega, c_1, c_2$
3: Initialize $X_i(k) : x_{i,j}(k) \sim \mathrm{rand}\,(L_j, U_j) \forall j, \forall i$
4: Compute objective function, $f(X_i(k)), \forall i$
5: **while** $k \leq K$ **do**
6: Save, $f(X_i(k)), \forall i$
7: $i = 1$
8: **while** $i \leq N_p$ **do**
9: Select $r_1, r_2, r_3 \in \{1, 2, ..., N_p\}, r_1 \neq r_2 \neq r_3 \neq i$
10: Randomly select $j \in \{1, 2, ..., D\}$
11: **while** $j \leq D$ **do**
12: Compute mutant vector $v_i(k), \forall i$
13: **end while**
14: Apply Eq. 6 to generate trial vector $U_i(k), \forall i$
15: Apply Eq. 8 if necessary
16: Compute objective function, $f(U_i(k)), \forall i$
17: $i = i + 1$
18: **end while**
19: $i = 1$
20: **while** $i \leq N_p$ **do**
21: **if** $f(U_i(k)) > f(X_i(k))$ **then**
22: $X_i(k+1) = U_i(k)$
23: **else** activate PSO algorithm
24: $j = 1$
25: **while** $j \leq D$
26: $\mathbf{v}_i(k+1) = \omega v_i(k) + c_1 r_1 (pbest_i(k) - X_i(k)) + c_2 r_2 (gbest_i(k) - X_i(k))$
27: $\bar{X}_i(k) = X_i(k) + \mathbf{v}_i(k+1)$
28: **end while**
29: **end if**
30: **if** $f(\bar{X}_i(k)) > f(X_i(k))$ **then**
31: $X_i(k+1) = \bar{X}_i(k)$
32: **else**
33: $X_i(k+1) = X_i(k)$
34: **end if**
35: $i = i + 1$
36: **end while**
37: $k = k + 1$
38: **end while**

The algorithm begins in step 1 with the initialization of the first iteration $k = 1$. Step 2 assigns values to DE and PSO parameters. Step 3 initializes a population of vectors $X_i(k)$ such that each component $x_{i,j}(k), \forall i \in \{1,2,\ldots,N_p\}, \forall j \in \{1,2,\ldots,D\}$ are made of random elements drawn from predefined L and U bounds in accordance to Eqs. 2 and 3. Step 4 computes the objective function of the initialized population, and the computed objective function is saved in step 6 for

future reference. In steps 9–12, we compute a mutant vector $v_i(k)$ in accordance with Eq. 4, while a trial vector U_i is generated in step 14 in accordance with Eq. 6. If any element of the trial vector is outside the feasible region of the search space, step 15 is activated in order to modify and adjust the trial vector within the feasible region. In step 16, the objective function of the trial vectors is evaluated. Steps 21 and 22 compare the objective function of the trial vector with that of the corresponding target vector in order to determine the population of the next iteration. The fitness value of the trial vector must be greater than the fitness value of the target vector in order to make it to the next iteration; otherwise, the algorithm uses the PSO velocity and position update equation to generate new candidate solution as illustrated in steps 23–27. In steps 30–33, the objective function of the newly generated vectors $\overline{X}_i\,(k)$ is evaluated and compared with the fitness of corresponding target vectors to determine the population of the next iteration. The algorithm continues iteratively until it terminates when the predefined maximum number of iterations K is attained.

4 Objective function formulation and optimization

For any given hydrocarbon reservoir, determining the optimal well configuration that maximizes the recovery factor over a time interval [0, T] can be posed as an optimization problem. In every practical sense, the maximization of the recovery factor (objective function) for a water-flooded reservoir is equivalent to any of the following:

1. Maximizing the cumulative volumes of hydrocarbon produced at terminal time T
2. Maximizing the water saturation of the reservoir at terminal time T
3. Minimizing the volume of hydrocarbon in place at terminal time T

However, the objective for most E&P companies is to maximize the economic value of the asset. The commonly used economic criterion for this purpose is the NPV [1, 3, 5, 8, 13, 20, 46, 47, 57, 65, 72, 73]. Therefore, we designate NPV as the objective function in all problems considered in this work. For all potential solutions (well configuration), the NPV is computed from the fluid production profiles generated as a result of simulation run associated with corresponding well placements. Thus, the NPV is a measure of the cash flow (CF) generated from sale of produced volumes of oil. Following a slight modification of the economic model described in [47], we define the NPV as the total oil revenues minus the capital expenditure (CAPEX) and the operation cost (OPEX), in combination with a discount factor d, which represents the time value of money (i.e., interest rate or inflation). This can be represented mathematically as follows:

$$\mathrm{NPV} = \sum_{t=1}^{T} \frac{\mathrm{CF}_{(t)}}{(1+d)^t} - \mathrm{CAPEX} \tag{18}$$

where T is the terminal time or total production years, d is the discount factor, and $\mathrm{CF}_{(t)}$ represents the cash flow at time t. The cash flow at any time is given by

$$\mathrm{CF}_{(t)} = \mathrm{REV}_{(t)} - \mathrm{OPEX}_{(t)} \tag{19}$$

where $\mathrm{REV}_{(t)}$ is the revenue accrued from sale of products at time t, and $\mathrm{OPEX}_{(t)}$ represents the operating expenditure (or cost of production) at time t. Both quantities are usually measured in US dollars.

For a two-phase (oil and water) flow reservoir model, the values of $\mathrm{REV}_{(t)}$ and $\mathrm{OPEX}_{(t)}$ at any time (t) are, respectively, given by

$$\mathrm{REV}_{(t)} = p_{(t)}^{\mathrm{oil}}\, \upsilon_{(t)}^{\mathrm{oil}} \tag{20}$$

$$\mathrm{OPEX}_{(t)} = p_{(t)}^{\mathrm{w,p}}\, \upsilon_{(t)}^{\mathrm{w,p}} + p_{(t)}^{\mathrm{w,i}}\, \upsilon_{(t)}^{\mathrm{w,i}} \tag{21}$$

where $p_{(t)}^{\mathrm{oil}}$ is the price of oil at time t, $p_{(t)}^{\mathrm{w,p}}$ is the cost of producing water in the production wells, and $p_{(t)}^{\mathrm{w,i}}$ is the cost of injecting water in the injection wells at time t; all three quantities are measured in dollars per barrel. On the other hand, $\upsilon_{(t)}^{\mathrm{oil}}$ is the total volume of oil produced at time t, while $\upsilon_{(t)}^{\mathrm{w,p}}$ and $\upsilon_{(t)}^{\mathrm{w,i}}$ (all measured in barrels) represent the total volumes of water produced (from production wells) and injected (in injection wells), respectively.

The CAPEX represents the total cost to drill and complete all wells; it is noted that as far as production is concerned, CAPEX is incurred at time $t = 0$. It is computed as follows:

$$\mathrm{CAPEX} = \sum_{w=1}^{n^{\mathrm{w}}} \left(C_{\mathrm{w}}^{\mathrm{top}} + L_{\mathrm{w}}^{\mathrm{main}} + C^{\mathrm{drill}} \right) \tag{22}$$

where n^{w} is the total number of wells, $C_{\mathrm{w}}^{\mathrm{top}}$ is the cost of drilling the main bore to the top of the reservoir (in US dollars), $L_{\mathrm{w}}^{\mathrm{main}}$ is the length of the main bore (in meters), and C^{drill} is the cost of drilling within the reservoir (in dollars per meter).

The values of the parameters for NPV computation are given in Table 1 below. We assume that time-dependent parameters such as d, $p_{(t)}^{\mathrm{oil}}$, $p_{(t)}^{\mathrm{w,p}}$, and $p_{(t)}^{\mathrm{w,i}}$ are constant over the time period [0, T].

Table 1 NPV computation parameters

Parameters	Symbol	Value
Price of oil	p^{oil}	\$50/bbl
Water production cost	$p^{w,p}$	\$10/bbl
Water injection cost	$p^{w,i}$	\$5/bbl
Discount factor	d	0.1
Drilling cost per meter	C^{drill}	$\$5.3 \times 10^4$/m
Drilling cost to reservoir top	C_w^{top}	$\$50 \times 10^6$

We now apply the algorithms DE, PSO, and HPSDE to three optimization problems involving placement of vertical wells in 2-D and 3-D reservoir models. For each of the problems, five optimization runs of the algorithms are performed, and the results are averaged over the number of runs to determine the relative performance of each algorithm. The choice of five runs is based on suggestion in [13, 66], and the importance of comparing the average performance of the algorithms (over multiple optimization runs) stems from the nondeterministic nature of these algorithms. It is noted that this reduces both the effects caused by different distributions of the initial solutions and the randomness resulting from the probabilistic operators in the algorithms. To further reinforce fairness in the results; the control parameters used in each of the three algorithms are the same in all the problems considered. These factors afford us the ability to draw a more general conclusion with respect to the performance of the algorithms.

All applications in this work are model-based, and the simulations are performed using MATLAB® Reservoir Simulation Toolbox 2011a. Since our applications are model-based, we note the inevitable presence of geological uncertainty—a direct consequence of the fact that reservoir models are a "crude" approximation of real physical oil reservoirs. To address this mismatch between the physical reservoir and the reservoir model, we employ a robust optimization strategy in the first two examples. By robust optimization, the optimization procedure is carried out over a set of realizations which explicitly accounts for the geological uncertainty in the models [65]. The robust optimization objective adopted in this work is the *max-mean objective*, which basically seeks to maximize the average performance measure associated with each of the realizations. It is given by

$$\langle \text{NPV} \rangle = \left(\frac{1}{R} \sum_{i=1}^{R} \text{NPV}_i \right) \tag{23}$$

where <NPV> is the expected NPV, and R is the number of geological realization.

4.1 Case 1: placement of a single producer

For the first example, we consider optimizing the placement of a single producer well in a 2-D reservoir model with $45 \times 45 \times 1$ grid blocks of dimensions of 10 m $\times$ 10 m $\times$ 10 m. In order to address the mismatch between model and physical reservoir, we incorporate geological uncertainty by considering ten realizations of the model. Usually, it is common to decide upon a few sources of uncertainty that presumably have the largest impact on the model, and, to this end, we choose the permeability distribution. Thus, the realizations are generated based on varying permeability distribution. The system contains oil and connate water. The initial pressure and saturation (connate water saturation) are 350×10^5 Pa and 0.2, respectively, and both are uniform throughout the reservoir model. There are no aquifers and no water injection; thus, only oil is produced. The remaining system properties are given in Table 2. The reservoir is simulated for 10 years with the single producer placed in grid block position corresponding to the results obtained from the five runs of each of the algorithms. It is constrained to operate at a bottom hole pressure (BHP) of 65×10^5 Pa, and the cumulative volume of oil produced is used to compute the NPV (by applying Eq. 18) corresponding to each optimization run of the algorithms.

The computed NPVs are averaged over the number of runs to reflect the relative performance of each algorithm. In the same way, the average NPV achieved by each of the algorithm is computed for the remaining realizations, and, using Eq. 23, <NPV> of the reservoir model is computed for each of the algorithm.

We perform a sensitivity analysis in order to examine the effect of different population size and iteration number combinations on the performance of the three algorithms. To this end, we consider six population size and iteration combinations: (5, 10), (10, 10), (20, 10), (10, 20), (10, 40), and (10, 80), respectively. Note that in three of the six population size and iteration combinations, the number of iteration is held constant (while the population size is varied), and the population size is held constant (while the maximum number of iteration is varied) in the remaining three combinations.

Table 2 Systems properties

Properties	Symbol	Value
Porosity	ϕ	0.3
Oil viscosity	μ_o	10^{-3} Pas
Oil compressibility	c_o	10^{-10} Pa^{-1}
Rock compressibility	c_r	1.8×10^{-10} Pa^{-1}

For each of the population size and iteration combination, we perform five optimization runs of the algorithms and computed the average NPV relating to each algorithm. Figure 6 shows the corresponding <NPV> for each algorithm against the total number of simulations per realization, given by $N_p \times K$. It is obvious in all six population size and iteration number combinations that HPSDE (red line) algorithm greatly outperforms both DE (blue line) and PSO (green line) algorithms.

The advantage of the HPSDE algorithm over DE and PSO algorithms is very much pronounced in cases where the population size and iteration combination are low. For example, at the respective population size and iteration combination of five and ten, an <NPV> of $\$175.8 \times 10^6$ was attained for HPSDE. This represents a 24% increase in the <NPV> of $\$141.7 \times 10^6$ and a 52% increase in the <NPV> of $\$115 \times 10^6$ achieved by the PSO and the DE algorithms, respectively. The comparative advantage of the performance of HPSDE over PSO and DE algorithms becomes less remarkable as the total number of simulations per realization (given by $N_p \times K$) increases from 50 to 800. For a population size of ten and maximum iteration of 80, the HPSDE algorithm achieves an <NPV> of $\$303.7 \times 10^6$, while the PSO and the DE algorithms attained a near-convergence <NPV> of $\$261 \times 10^6$ and $\$259.9 \times 10^6$, respectively. This performance of the PSO and DE algorithms is 14% less than the performance of HPSDE.

Another interesting observation from the view points of the DE and PSO algorithms is that the performance of one algorithm over the other in well placement optimization problem appears to be dependent on the total number of simulations. In all six population size and iteration number combinations, the DE algorithm attained higher <NPV> than the PSO algorithm at very low simulation per realization. For example, in the first population size and iteration combination (i.e., $N_p = 5$, $K = 10$), DE outperforms PSO algorithm from the start till 15 simulations per realization. After this "threshold" number of simulations, the PSO achieved better <NPV> than DE. This pattern is observed in all six population size and iteration number combinations. However, the threshold number of simulation after which PSO outperforms DE varies from one population size and iteration number combination to the other. In the last population size and iteration combination, though PSO outperforms DE from simulation per realization of 22, both algorithms (PSO and DE) achieve a near-convergent <NPV> of $\$261 \times 10^6$ and $\$259.9 \times 10^6$ from a total simulation per realization of 744 to 800. In any case, the HPSDE algorithm outperforms both the DE and PSO algorithms.

4.2 Case 2: placement of two wells—a producer and an injector

In this example, we consider optimizing the placement of a producer and an injector in a 2-D reservoir model where the oil is to be replaced by water in a simple waterflooding process. The reservoir model has $50 \times 50 \times 1$ grid blocks, each of the grid blocks has a dimension of 10 m × 10 m × 10 m; initial pressure of the reservoir is 350×10^5 Pa and the residual oil and connate water saturation is 0.2. Again, we consider ten

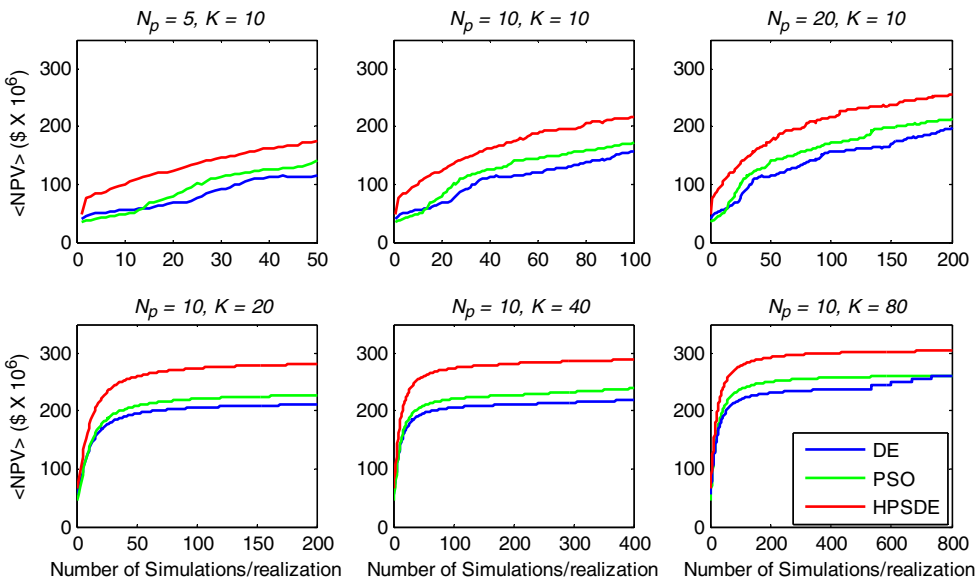


Fig. 6 <NPV> of DE, PSO, and HPSDE versus number of simulations per realization for different population size and maximum iteration number combinations

realizations of the model, so that optimization results are robust against geological uncertainty. Unlike in example 1 (which did not require relative permeability), we employ the Corey model for relative permeability, with Corey exponents $n_o = n_w = 2.45$ and relative permeability endpoints for oil and water 0.9 and 0.65, respectively. The relative permeability curve is depicted in Fig. 7, and the remaining system properties are given in Table 3.

In mathematical probability, we often encounter occupancy problems where the usual task is to find the total number of possible placement of m different balls into n bins (read, placement of two nonidentical wells in 2,500 possible grid blocks).

However, in this case, we are not interested in the total number of possible outcomes. We are only interested in the outcome that yields the highest NPV in each realization of the reservoir model.

For each of the wells, there are three optimization variables $\{x, y, I\}$, which results in a total of six variables. The Cartesian coordinates of each well location is represented by the variables x and y, while the variable $I \in (0,1)$ is a binary indicator that represents the well type (i.e., $I = 0$ designates a production well, and $I = 1$ designates an injection well). Such binary indicator was employed in [72], and we have adopted it in this work because of simplicity and ease of implementation.

Using a population size of 20 and maximum iterations of 100, five optimization runs of DE, PSO, and HPSDE algorithms are performed to determine the placement of the injector and producer. Both wells are placed at locations corresponding to the solutions from each run of the algorithms. The system is simulated for 10 years, with the producer constrained to operate at a BHP of 65×10^5 Pa and the injector operating at a BHP of 140×10^5 Pa. The volumes produced are used to compute the NPV in accordance with Eq. 18, and the computed NPVs are averaged over the number of runs to determine the average NPV associated with each algorithm. Accordingly, this is repeated for each realization of the model and the <NPV> for the reservoir model corresponding to each algorithm is computed using Eq. 23. The results are plotted and shown in Fig. 8.

Table 3 Systems properties

Properties	Symbol	Value
Oil viscosity	μ_o	10^{-3} Pas
Water viscosity	μ_w	10^{-3} Pas
Oil compressibility	c_o	10^{-10} Pa^{-1}
Rock compressibility	μ_r	1.8×10^{-10} Pa^{-1}

The DE and the PSO algorithms yielded <NPV> of $\$543.1 \times 10^6$ and $\$536.9 \times 10^6$, respectively. Both performance values are below the <NPV> of $\$580.8 \times 10^6$ that is achieved using the HPSDE. Again, it is observed that DE achieved better results than PSO when the number of simulation is below a threshold value. This is consistent with the pattern observed in the first example (see Section 4.1). Beyond this threshold number of simulations (54 in this case), PSO outperforms DE until after 1,588 simulations, when both algorithms begin to converge to the same value of <NPV>. We also note that DE attained nominally better <NPV> than PSO after 1,720 simulations. In any case, however, HPSDE achieved better <NPV> values than the duo of DE and PSO algorithms. The water saturation maps of the reservoir from the best run of the algorithms are shown in Fig. 9.

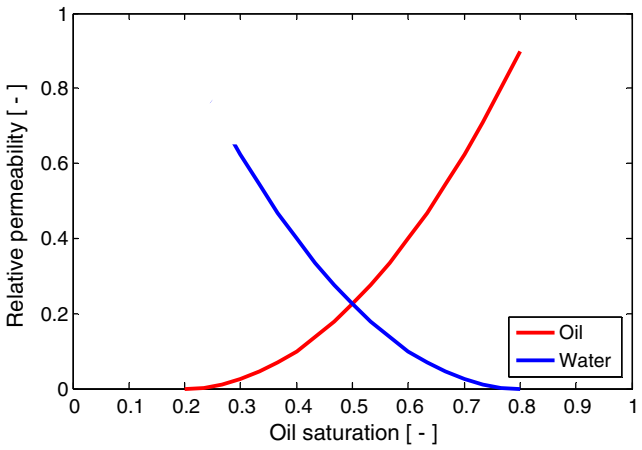


Fig. 7 Relative permeability curve for oil and water

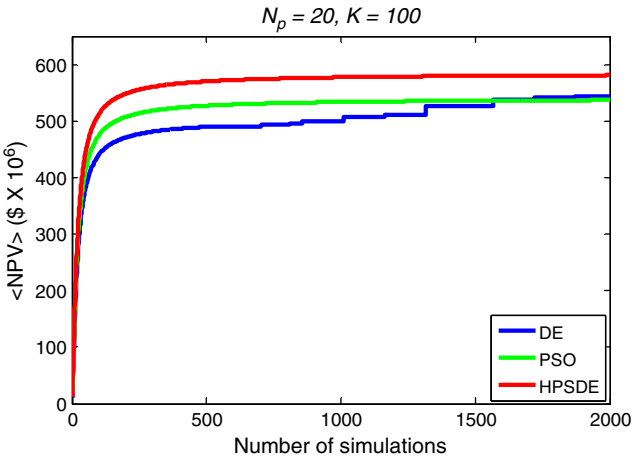


Fig. 8 <NPV> of DE, PSO, AND HPSDE algorithms versus number of simulations for case 2

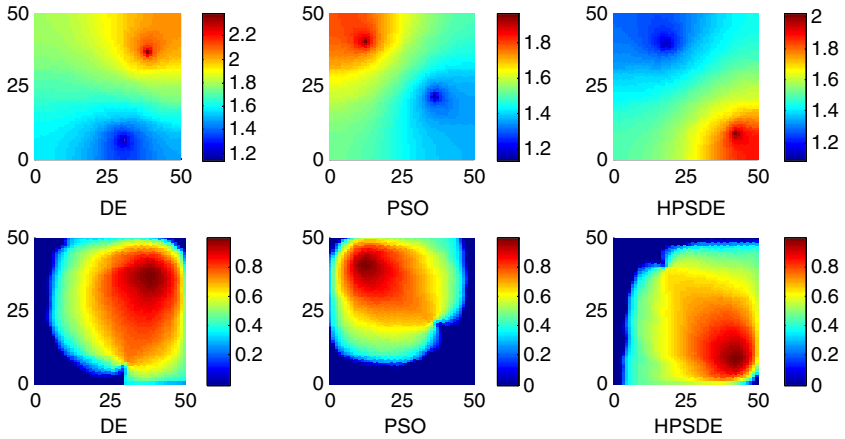


Fig. 9 Initial pressure in 106 Pa (*top row*) and water saturation map (*bottom row*) from realization with best performance of DE, PSO, and HPSDE runs

4.3 Case 3: placement of nine wells in a 3-D reservoir

A 3-D reservoir model is considered in this example. It contains 50 × 50 × 8 grid blocks with dimension of 10 m × 10 m × 10 m. The fluid, geological, and system properties are the same as in case 2, and Fig. 10 shows the permeability distribution of the model. The task is to determine the optimal type and placement of nine wells (injectors and producers) for a waterflooding operation. In this example, we ignore the effects of geological uncertainty; therefore, the performance measure is the simple NPV resulting from the fluid profile generated. Like in case 2, each of the nine wells has three optimization variables $\{x, y, I\}$, which results in a total of 27 variables. The variables x and y represent the Cartesian coordinates of each well location, and the variable $I \in (0,1)$ is a binary indicator that represents the well type (producer or injector). Using a population size of 50 and maximum iterations of 100, five runs of DE, PSO, and HPSDE algorithms are used to determine the placement of the wells. The system is simulated for 10 years, with the producers constrained to operate at a BHP of 65×10^5 Pa and the injectors at 100×10^5 Pa.

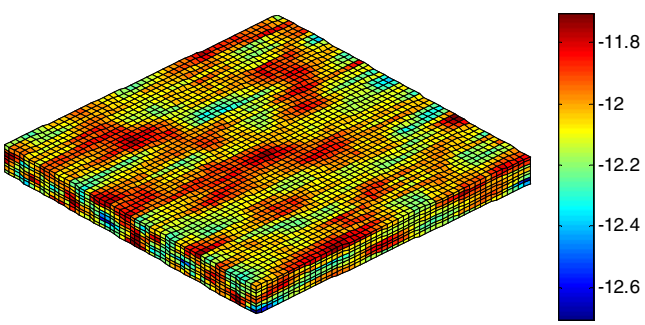


Fig. 10 Permeability field (in millidarcys) for case 3

It is important to note that there were more producers than injectors in the best optimization runs of the algorithms. In this regard, the well split of the producers to injectors are 5:4, 5:4, and 6:3 for DE, PSO, and HPSDE algorithms, respectively. The reservoir water saturation maps from well location obtained from best runs of the algorithms are shown in Fig. 11.

The fluid profiles are used to compute the NPV corresponding to each optimization run of the algorithms. This is achieved by applying Eq. 18, and the computed NPVs are averaged over the number of optimization runs to determine the average performance measure (NPV) associated with each of the algorithm. The results are plotted and shown in Fig. 12.

At the end of the total number of simulations, the performance of DE algorithm was higher than that of PSO algorithm. It achieved a maximum NPV of $\$46.3 \times 10^9$, which is 1.7% higher than the NPV of $\$45.3 \times 10^9$ attained by PSO algorithm. With an NPV of $\$49.1 \times 10^9$, the performance of HPSDE represents a 5.7% rise in the performance of DE and a 7.7% increase in the performance of PSO. Though there were periods when DE and PSO converge to near and same NPV measure, the DE outperformed the PSO algorithm at very low and very high numbers of simulations. The better performance of DE over PSO at low number of simulation is consistent with the pattern observed in cases 1 and 2; DE better performance (than PSO) at very high number of simulation was only observed in case 2. Although the exact reason for this is unknown at this time, we note that the overall performance of PSO is better than the overall performance of DE over reasonable number of simulations. In all cases, however, both algorithms did not achieve better NPV than the HPSDE algorithm. This result is consistent with the

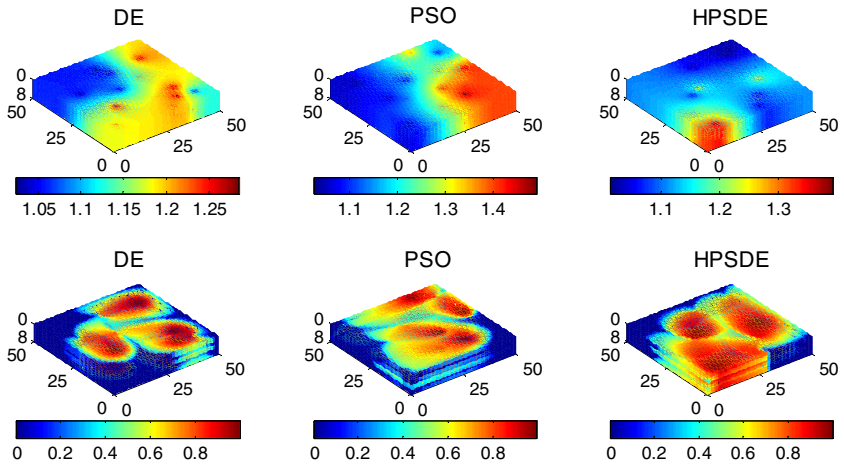


Fig. 11 Initial pressure in 106 Pa (*top row*) and water saturation maps (*bottom row*) from best optimization runs of DE, PSO, and HPSDE algorithms

results from the first and second cases where max-mean objective robust optimization was performed.

Furthermore, we compared the performance of the three stochastic algorithms against the computed NPV for a specific well pattern with same number of wells—an inverted nine-spot arrangement (eight producers and one injector placed at the center). The inverted nine-spot arrangement yielded an NPV of $\$41.9 \times 10^9$. This is considerably lower than the NPV attained by any of the three metaheuristic algorithms.

In fact, the NPV attained by the DE algorithm is 10% higher than that of the inverted nine-spot well pattern; PSO and HPSDE algorithms yielded an NPV that is, respectively, 8 and 17% higher than the NPV achieved by the inverted nine-spot well arrangement. This reinforces the belief that metaheuristic algorithms are able to provide better results than specific well pattern arrangement of same economic constraints (i.e., same number of wells).

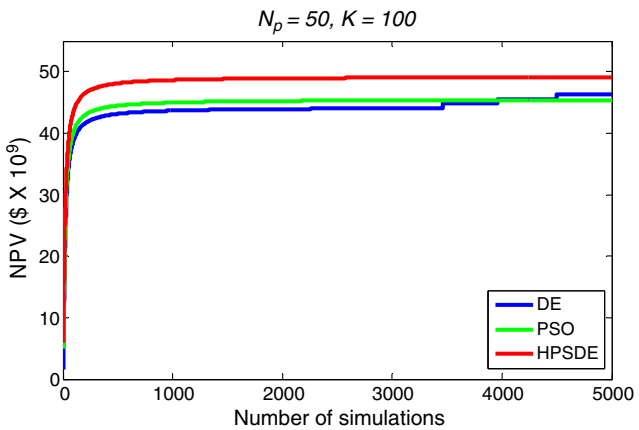


Fig. 12 NPV of DE, PSO, and HPSDE algorithms for case 3

5 Conclusions

Well configuration is an important decision input that can ultimately determine a reservoir's production profile and, therefore, the recoverability of the reservoir. For all intents and purposes, the recoverability is a direct measure of the economic value of the portfolio or the NPV of the asset. In this paper, we applied three metaheuristic algorithms in well placement optimization problems. One of the algorithms (HPSDE) is a hybrid of the other two algorithms—DE and PSO. With NPV as performance measure, we considered three examples involving the placement of one, two, and nine vertical wells. Based on suggestion from [13, 66], five optimization runs of each of the algorithms are performed in each examples considered, and the results are averaged over the number of optimization runs. The HPSDE algorithm consistently outperformed DE and PSO algorithms. In two of the examples, we factored in geological uncertainty by addressing the discrepancies between physical reservoir and reservoir model. To this end, we performed a max-mean objective robust optimization of the performance measure, and HPSDE yielded better results than DE and PSO. In the third example, we also compared the performance (NPV) of the metaheuristic algorithms with the NPV attained via a specific well pattern with same number of wells (inverted nine-spot arrangement). The stochastic algorithms yielded higher NPV than the specific well pattern arrangement. We also showed that the performance of DE and PSO is, to an extent, dependent on the total number of simulations. DE attained higher NPV than PSO at very low and very high number of total simulations. However, in all examples considered, the overall performance of PSO is better than that of DE. More importantly, we note that HPSDE outperformed both algorithms in all cases.

While we note that these findings are interesting and potentially useful, we acknowledge that there are issues or limitations that still need to be addressed. Chief among these limitations is the issue of control parameter tuning. For instance, in the second and third examples, there are instances where DE outperformed PSO and vice versa. It is important to understand how these behaviors are influenced by relevant control parameters of the algorithm. We note that DE parameters ($F = 0.5$, CR $= 0.1$) used in this work are adopted from [61], and PSO parameters ($c_1 = c_2 = 1.193$, $\omega = 0.721$) are adopted from [47]. For reasons bothering on fair comparison of results, all three algorithms were used without parameter tuning of any kind. Although the population size and the maximum number of iteration are largely dependent on the complexity of the underlying optimization problem and the number of optimization variables, we believe that effective parameter tuning (which will be computational expensive, as it will require extra function evaluations) would further enhance the performance of the algorithms.

A closely related limitation is the issue of usability in practical field development optimization scenarios. Indeed, a hybridized metaheuristic optimization algorithm such as HPSDE is potentially a viable and promising alternative in reservoir engineering optimization problems; however, issues of usability have to be addressed before it can be deployed for practical use in the industry. In some sense, the usability limitation is intertwined with parameter tuning, as usability would be undoubtedly enhanced if parameter tuning is sorted out in the design-end of the algorithm, as opposed to the user-end. This is so because it is generally unrealistic for industrial end-users to waste expensive function evaluations in correcting the inherent weakness of the design phase of an algorithm. We also note that the performance of HPSDE algorithm and, indeed, other hybridized stochastic algorithms, could be further improved by incorporating into the algorithms, knowledge (such as the problem structure) and relevant information about the underlying optimization problem. We plan to consider these issues in our future work and we also plan to apply these algorithms in nonconventional well optimization problems.

The above limitations notwithstanding, this work demonstrates the potential benefit of hybridized metaheuristic algorithms over standard stochastic techniques (such as DE and PSO) in reservoir engineering applications. Besides the fact that these findings are promising, the applicability of HPSDE algorithm in well placement optimization problem shows that hybridization could be key to unlocking some of the challenging optimization problems in field development optimization and reservoir engineering in general.

Acknowledgements We are grateful to SINTEF ICT for providing the computational resources used in this work. Special thanks to Knut-Andreas Lie and Bård Skaflestad for all the many insightful discussions and suggestions. This work was fully funded by the Centre for Applicable Mathematics and Systems Science through the Intelligent Energy Systems research grant.

References

1. Aitokhuehi, I., Durlofsky, L.J., Artus, V., Yeten, B., Aziz, K.: Optimization of advanced well type and performance. In: 9th European Conference on the Mathematics of Oil Recovery, Cannes (2004)
2. Angeline, P.J.: Using selection to improve particle swarm optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 84–89. Anchorage, AL, USA (1998)
3. Bangerth, B.L., Klie, W.H., Wheeler, M.F., Stoffa, P.L., Sen, M.K.: On optimization algorithms for the reservoir oil well placement problem. Comput. Geosci. **10**, 303–319 (2006)
4. Banks, A., Vincent, J., Anyakoha, C.: A review of particle swarm optimization. Part II: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. Nat. Comput.: Int. J. **7**(1), 109–124 (2008)
5. Beckner, B.L., Song, X.: Field development planning using simulated annealing—optimal economic well scheduling and placement. In: SPE Annual Technical Conference and Exhibition (SPE 30650), Dallas (1995)
6. Bittencourt, A.C., Horne, R.N.: Reservoir development and design optimization. In: SPE Annual Tech. Conf. and Exhibition (SPE 38895), San Antonio (1997)
7. Blackwell, T., Bentley, P.J.: Don't push me! collision-avoiding swarms. In: IEEE Congress on Evolutionary Comput. pp. 1691–1696. Honolulu, Hawaii, USA (2002)
8. Bouzarkouna, Z., Ding, D.Y., Auger, A.: Well placement optimization with the covariance matrix adaptation evolution strategy and meta-models. Comput. Geosci. **16**, 75–92 (2011)
9. Braendler, D., Hendtlass, T.: The suitability of particle swarm optimisation for training neural hardware. In: International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE, pp. 190–199. Springer, NY (2002)
10. Brandstatter, B., Baumgartner, U.: Particle swarm optimization–mass-spring system analogon. IEEE Trans. Magn. **38**, 97–1000 (2002)
11. Centilmen, A., Ertekin, T., Grader, A.S.: Applications of neural networks in multiwell field development. In: SPE Annual Technical Conference and Exhibition (SPE 56433), Houston (1999)
12. Chakrabarti, R., Chattopadhyay, P.K., Basu, M., Panigrahi, C.K.: Particle swarm optimization technique for dynamic economic dispatch. J. Inst. Eng. India **87**, 48–54 (2006)
13. Ciaurri, D.E., Mukerji, T., Durlofsky, L.J.: Derivative-free Optimization for Oil Field Operations Studies in Comput. Intell., vol. 359, pp. 19–55. Computational Optimization and Applications in Engr. and Ind. (2011)
14. Clerc, M.: Particle Swarm Optimization. iSTE, London (2006)
15. Das, S., Konar, A.: A swarm intelligence approach to the synthesis of two-dimensional IIR filters. Eng. Appl. Artif. Intell.

20(8), 1086–1096 (2007). doi:10.1016/j.engappai.2007.02.004. Accessed 18 Mar 2011
16. Das, S., Abraham, A., Konar, A.: Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives. www.softcomputing.net/aciis.pdf. Accessed: 18 Mar 2011
17. Davendra, D., Zenlinka, I., Onwubolu, G.: Hybrid Differential Evolution—Scatter Search Algorithm for Permutative Optimization Evolutionary Computation, InTech, Vienna, Austria (2009)
18. Deep, K., Bansal, J.C.: Hybridization of particle swarm optimization with quadratic approximation. J. Oper. Res. **46**, 3–24 (2009)
19. Deep, K., Das, K.N.: Quadratic approximation based hybrid genetic algorithm for function optimization. Appl. Math. Comput. **203**(1), 86–98 (2008)
20. Dong, X., Wu, Z., Dong, C., Chen, K., Wang, H.: Optimization of vertical well placement by using a hybrid particle swarm optimization. Wuhan Univ. J. Nat. Sci. **16**(3), 237–240 (2011)
21. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, pp. 39–43. Nagoya, Japan (1995)
22. Engelbrecht, A.P.: Fundamentals of Computational Swarm Intel. Wiley, West Sussex (2005)
23. Farshi, M.M.: Improving genetic algorithms for optimum well placement. MSc. Thesis, Stanford University (2008)
24. Gong, T., Tuson, A.L.: Particle swarm optimization for quadratic assignment problems—a forma analysis approach. Int. J. Comput. Intell. Res. **4**, 177–185 (2008)
25. Grimaccia, F., Mussetta, M., Zich, R.E.: Genetical swarm optimization: self-adaptive hybrid evolutionary algorithm for electromagnetics. IEEE Trans. Antennas Propag. **55**(3), 781–785 (2007)
26. Guyaguler, B., Horne, R.N.: Uncertainty assessment of well placement optimization. In: SPE Annual Tech. Conf. and Exhibition (SPE 71625), New Orleans, LA (2001)
27. Hajizadeh, Y., Christie, M., Demyanov, V.: History Matching with Differential Evolution Approach; A Look at New Search Strategies SPE EUROPEC/EAGE Annual Conference and Exhibition, Barcelona, Spain ISBN 978–90–73781–86–3 (2010). doi:10.2118/130253-MS
28. Haykin, S.: Neural Networks. Macmillan, New York (1999)
29. Hendtlass, T., Randall, M.: A survey of ant colony and particle swarm metaheuristics and their application to discrete optimization problems. In: Proc. of the Inaugural Workshop on Artificial Life (AL'01), pp. 15–25 (2001)
30. Hendtlass, T.: A combined swarm differential evolution algorithm for optimization problems. In: Proceedings of the 14th Int. Conf. on Ind. and Eng. App. of Artificial Intell. and Expert Systems. Lecture Notes in Computer Science, vol. 2070, pp. 11–18 Springer, Berlin (2001)
31. Higashi, N., Iba, H.: Particle swarm optimization with Gaussian mutation. In: Proceedings of the IEEE Swarm Intell. Symposium, pp. 72–79. Indianapolis, IN (2003)
32. Jian, M., Chen, Y.: Introducing recombination with dynamic linkage discovery to particle swarm optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 85–86 (2006)
33. Juang, C.F.: A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. IEEE Trans. Syst. Man Cybern., Part B, Cybern. **34**(2), 997–1006 (2004)
34. Kennedy, J.: Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: Proceedings of Cong. of Evolutionary Computation, vol. 3, pp. 1931–1938. IEEE, New York (1999)
35. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: IEEE, Neural Networks Council Staff, IEEE Neural Networks Council (eds.) Proc. IEEE International Conference on Neural Networks, pp. 1942–1948. IEEE, Los Alamitos (1995)
36. Kosmidis, V.D., Perkins, J.D., Pistikopoulos, E.N.: A mixed integer optimization formulation for the well scheduling problem on petroleum fields. Comput. Chem. Eng. **29**(7), 1523–1541 (2005)
37. Krink, T., Vesterstrøm, J.S., Riget, J.: Particle swarm optimization with spatial particle extension. In: Proceedings of the 4th Congress on Evolutionary Computation, pp. 1474–1479 (2002)
38. Lampinen, J., Zelinka, I.: Mechanical engineering design by differential evolution. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimisation, pp. 127–146. McGraw-Hill, London (1999)
39. Lampinen, J.: A constraint handling approach for the differential evolution algorithm. In: Proc. the Congress on Evolutionary Computation, vol. 2, pp. 1468–1473 (2002)
40. Litvak, M., Gane, B., Williams, G., Mansfield, M., Angert, P., Macdonald, C., McMurray, L., Skinner, R., Walker, G.J.: Field development optimization technology. Paper SPE 106426 presented at the SPE Reservoir Simulation Symposium, Houston (2007)
41. Løvbjerg, M., Krink, T.: Extending particle swarms with self-organized criticality. In: Proc. of the 4th Congress on Evolutionary Computation, pp. 1588–1593 (2002)
42. Løvbjerg, M., Rasmussen, T., Krink, T.: Hybrid particle swarm optimizer with breeding and subpopulations. In: Proc. of the 3rd Genetic and Evolutionary Computation Conference (GECCO-2001), vol. 1, pp. 469–476 (2001)
43. Madavan, N.: Aerodynamic shape optimisation using hybrid differential evolution. In: AIAA-2003–3792, 21st AIAA Applied Aerodynamic Conference, Orlando, Florida, USA (2003)
44. Michalewicz, Z., Schoenauer, M.: Evolutionary algorithms for constrained parameter optimization problems. Evol. Comput. **4**(1), 1–32 (1996)
45. Miranda, V., Fonseca, N.: New evolutionary particle swarm algorithm (EPSO) applied to voltage/VAR control. In: The 14th Power Systems Computation Conference (PSCC'02), Seville, Spain (2002)
46. Montes, G., Bartolome, P., Udias, A.L.: The use of genetic algorithms in well placement optimization. In: SPE Latin American and Caribbean Petroleum Engineering Conference, SPE 69439 (2001)
47. Onwunalu, J., Durlofsky, L.J.: Application of a particle swarm optimization algorithm for determining optimum well location and type. Comput. Geosci. **14**(1), 183–198 (2010)
48. Onwunalu, J.: Optimization of nonconventional well placement using genetic algorithms and statistical proxy. Master's Thesis, Stanford University (2006)
49. Perez-Guerrero, R.E., Cedeno-Maldonado, J.R.: Economic power dispatch with non-smooth cost functions using differential evolution. In: Proc. the 37th Annual North American Power Symposium 2005, pp. 183–190 (2005)
50. Plagianakos, V.P., Vrahatis, M.N.: Parallel evolutionary training algorithms for 'hardware-friendly' neural networks. Nat. Comput. **1**, 307–322 (2002)
51. Plagianakos, V.P., Vrahatis, M.N.: Training neural networks with threshold activation functions and constrained integer weights. In: IEEE Int. Joint Conf. on Neural Networks (IJCNN 2000), Como, Italy (2000)

52. Poli, R., Di Chio, C., Langdon, W.B.: Exploring extended particle swarms: a genetic programming approach. In: Beyer, H.-G., et al. (eds.) GECCO 2005: Proceedings of the 2005 Conf. on Genetic and Evolutionary Computation, pp. 169–176. Washington, DC (2005)
53. Poli, R., Langdon, W.B., Holland, O.: Extending particle swarm optimization via genetic programming. In: Keijzer, M., et al. (eds.) Lecture Notes in Computer Science. Proceedings of the 8th European Conference on Genetic Programming, vol. 3447, pp. 291–300. Springer, Berlin, Lausanne, Switzerland (2005)
54. Ratnaweera, A., Halgamuge, S.K., Watson, H.C.: Self-organizing hierarchical particle swarm optimizer with time varying accelerating coefficients. IEEE Trans. Evol. Comput. **8**(3), 240–255 (2004)
55. Robinson, J., Sinton, S., Rahmat-Samii, Y.: Particle swarm, genetic algorithm, and their hybrids: optimization of a profiled corrugated horn antenna. In: IEEE International Symposium on Antennas & Propagation, pp. 314–317. San Antonio, Texas (2002)
56. Rogalsky, T., Derksen, R.W., Kocabiyik, S.: Differential evolution in aerodynamic optimization. Can. Aeronaut. Space Inst. J. **46**, 183–190 (2000)
57. Sarma, P., Chen, W.H.: Efficient well placement optimization with gradient-based algorithms and adjoint models. In: Paper SPE 112257 Presented at the 2008 SPE Intelligent Energy Conf. and Exhib Amsterdam (2008)
58. Shi, Y., Eberhart, R.C.: A modified particle swarm optimizer. In: Proc. IEEE International Conf. on Evol. Comput., pp. 69–73. IEEE, Piscataway, NJ (1998)
59. Spall, J.C.: Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Trans. Automat. Contr. **37**(3), 332–341 (1992)
60. Storn, R., Price, K.: Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. Rep. TR-95–012, Inter. Computer Science Institute (ICSI) (1995)
61. Storn, R., Price, K.: Differential evolution—simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**, 341–359 (1997)
62. Sum-Im, T., Taylor, G.A., Irving, M.R., Song, Y.H.: A differential evolution algorithm for multistage transmission expansion planning. In: Proc. the 42nd International Universities Power Engineering Conference (UPEC 2007), pp. 357–364. Brighton, UK (2007)
63. Tasoulis, D.K., Plagianakos, V.P., Vrahatis, M.N.: Differential evolution algorithms for finding predictive gene subsets in microarray data. In: Artificial Intelligence Applications and Innovations. IFIP Int'l Federation for Information Processing, vol. 204, pp. 484–491 (2006)
64. Thangaraj, R., Pant, M., Abraham, A., Bouvry, P.: Particle swarm optimization: hybridization perspectives and experimental illustrations. Appl. Math. Comput. **217**, 5208–5226 (2011)
65. Van Essen, G.M., Zandvliet, M.J., Van den Hof, P.M.J., Bosgra, O.H., Jansen, J.D.: Robust waterflooding optimization of multiple geological scenarios. SPE Journal **14**(1), 202–210 (2009)
66. Vasiljevic, D., Golobic, J.: Comparison of the classical dumped least squares and genetic algorithm in the optimization of doublets. In: Proceedings of the First Workshop on Soft Computing, pp. 200–204. Nagoya, Japan (1996)
67. Wang, C., Li, G., Reynolds, A.C.: Optimal well placement for production optimization. In: Paper SPE 111154 Presented at SPE Eastern Regional Meeting, Lexington (2007)
68. Wang, J., Buckley, J.S.: Automatic history matching using differential evolution algorithm. In: Int'l Symposium of the Soc. of Core Analysts Trondheim, Norway (2006)
69. Wang, S.K., Chiou, J.P., Liu, C.W.: Non-smooth/non-convex economic dispatch by a novel hybrid differential evolution algorithm. IEE Proc. Gener. Transm. Distrib. **1**(5), 793–803 (2007)
70. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)
71. Xie, X., Zhang, W., Yang, Z.: A dissipative particle swarm optimization. In: IEEE Congress on Evolutionary Computation, pp. 1456–1461. Honolulu, Hawaii, USA (2002)
72. Yeten, B.: Optimum deployment of nonconventional wells. Ph.D. thesis, Stanford University (2003)
73. Zandvliet, M.J., Handels, M., van Essen, G.M., Brouwer, D.R., Jansen, J.D.: Adjoint-based well-placement optimization under production constraints. SPE J. **13**(4), 392–399 (2008)
74. Zhang, W.J., Xie, X.F.: DEPSO: hybrid particle swarm with differential evolution operator. In: IEEE International Conference on Systems, Man and Cybernetics (SMCC), pp. 3816–3821. Washington DC, USA (2003)