MODERN DATA ANALYTICS

Applied AI and Machine Learning for Oil and Gas Industry

By

Tatyana Plaksina

This book is dedicated to my family:

Mohammed and Leila

Table of Contents

Introduction

Nowadays data and information that can be converted into data, are ubiquitous. The data from natural, engineered, and social systems are being collected via various channels and devices and come in different shapes and frequencies, which makes application of data analytics tools necessary and unavoidable. Energy industry in general and its petroleum branch in particular collect and manipulate enormous volumes of simulated, experimental, and field data on daily basis to optimize operations, extract higher value from various resources, and reduce their environmental footprint.

Data analytics is a booming and rapidly developing discipline in many engineering fields, and petroleum engineering is not an exception. In the last several years, we have experienced an unprecedented growth in application of artificial intelligence (AI) and machine learning (ML) techniques in almost every aspect of our industry starting from drilling automation for all types of reservoirs and ending with optimization of CO_2 huff-n-puff operations in liquid-rich shale formations. The number of research studies that apply various data analytics techniques grows so rapidly, that it is almost impossible for a practicing engineer or a lab researcher to dig through the entire volume of articles and conference papers to learn about suitable data analytics methods for their problems. This is the main reason why this book was created. It serves as a “one stop methodology shop” for practicing engineers, researchers, data analytics experts, and energy data enthusiasts to learn about the most interesting and successfully applied methods, and using analogy decide which technique can be applied to their new problem.

Data analytics is a vast discipline that encompasses various approaches and techniques that can take book volumes to list and describe. Realizing this, in this book I focus on the leading-edge techniques, known under umbrella terms of AI and ML (and even ML is sometimes considered a part of AI, but rigorous and thorough mathematical classification of these methods is not an objective of this data analytics handbook), and provide the key concepts behind various AI and ML algorithms. Once the reader develops the sense of how these methods manipulate data and what kind of output they can produce, I provide many analogies or examples of problems to which these techniques were applied. Examples from current petroleum literature are the key ingredient of this book because they give the reader the opportunity to match his/her own engineering problem with the most suitable data analytics technique.

This book has three main parts: data analytics with AI, data analytics with fuzzy logic, and data analytics with ML. Subsequently, these three main parts are subdivided into subfields where it is needed. For example, the AI data analytics part feature two types of algorithms: evolutionary computation and swarm intelligence.

Use this book as a desk manual for data intensive problems if you are a practicing engineer or data analyst, and as a textbook if you are a student or a researcher in the academia. It will give you a broad and yet sufficiently detailed introduction to the growing field of data analytics for engineering applications.

PART I: Data Analytics with AI

AI is a large discipline within computational science that is being applied today to problems that range from speech recognition to analysis of social system behavior. In petroleum engineering, however, we do not need to create chat-bots or to predict when employees will resign from a company. Rather, we use AI as optimization methods to maximize or minimize certain objective functions. Thus, we will consider application of AI to data analytics in the context of optimization.

Optimization is a relatively old field of study in petroleum engineering that has been used for estimation, prediction, and determination of various parameters since the 1940s. Optimization was and is still used for a wide variety of engineering problems starting from well drilling, well placement and pipeline conditions in field development, to production forecasting and interpretation of experimental results from labs (Wang, 2003). Although there are many ways to classify optimization algorithms, we will adhere to these three main types: linear, integer, and non-linear programming techniques (**Figure 1**).

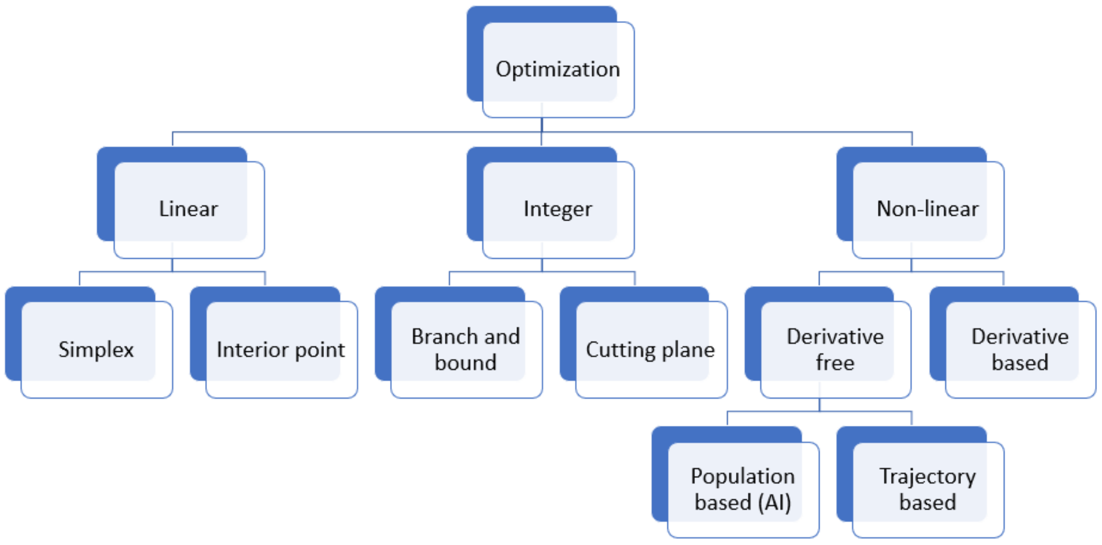

Figure 1. AI algorithms in relation to other types of optimization algorithms.

Linear optimization includes algorithms for which the objective function and the constraints are linear (Carroll Jr and Horne, 1992). The best examples of linear optimization are simplex and the interior point algorithms. Though linear optimization has its merits and applications in engineering (particularly, industrial systems engineering), it has a disadvantage of being computationally intensive (and even prohibitive) due to a large number of iterations required for the algorithms to converge (Klee and Minty, 1970).

Another class of algorithms - *integer optimization* – offers flexibility of working with integer or mixed continuous and integer values in control vectors and output vectors, but they also have a drawback of being computationally intense. A couple of the best examples of integer optimization algorithms are the cutting plane technique and the branch and bound method (Gononrl, 1958; Land and Doig, 2010). In petroleum engineering this type of optimization was used to optimize well placement and well control.

The third type of optimization that includes AI algorithms, is *non-linear optimization*. These techniques are applied to problems that have non-linear objectives and/or constraints and are usually subdivided in gradient-based and gradient-free algorithms. Gradient-based methods require calculation of the objective function's gradient to steer the search toward the global optimum. This can only be done if such gradient exists and can be calculated. The most popular and widely applied examples of gradient-based optimization methods are the steepest descent/ascent, Newton's method, quasi-Newton method, etc. (Watson et al., 1980; Fujii and Horne, 1995; Chen, 2013). These methods are applicable if the objective function is differentiable at all points of its domain, there is no need to access numerical simulator code to extract gradient, and the objective function is not a result of some experiment, for which the precise mathematical expression cannot be established.

In many cases, whether we are dealing with experimental data (e.g. lab data on fluid properties), simulated data (e.g. production history from simulated wells), or data collected in the field (e.g. downhole and surface gauge readings from the entire field), the expression of the objective function and its properties (like differentiability) are not known and cannot be known. Thus, none of the above-mentioned approaches can be used meaningfully to extract valuable insight from these data. For the problems with such objective functions, the data analyst or engineer must resort to heuristic or derivative-free methods. These algorithms can be customized and improved in running speed if the function's search domain is known. The only drawback of these methods is that finding the value of the global optimum is not guaranteed because that would require infinite number of iterations (Kamrani, 2010; Mohagheghian, 2016). Thus, solutions obtained with these methods are almost always near optimal.

For the purposes of this book, heuristic or gradient-free methods can be subdivided into two categories: trajectory-based and population-based. As a rule, a trajectory-based algorithm maintains only one solution at each iteration, while a population-based method usually keeps a population of solutions (Kamrani, 2010).

In the next chapter, you will be introduced to the first class of AI algorithms – *evolutionary computation*. These methods are usually population-based and use some evolution of the population of solutions to converge to a nearly optimal solution. All algorithms are provided with their detailed workflows and some interesting examples of oil and gas applications.

Chapter One: Evolutionary Computation

Evolutionary computation is a type of AI and a class of optimization methods that emulate evolution of some species or a population over time similarly to the biological evolution. These algorithms are usually population-based stochastic solvers that use trial and error as the main search driving mechanism.

Evolutionary algorithms start with an initial set or population of possible solutions and update them from one iteration to another. At each iteration the population is altered by removing less “fit” solutions and replacing them with new stochastically generated solutions. The equivalent biological terms are *mutation* for alteration and *selection* for removal. Upon reaching a termination criterion or criteria, the final population of candidates is expected to have an increased “fitness” (improved objective function values) from that of the initial population.

In this chapter, you will be introduced to several most widely used AI population-based methods including Genetic Algorithm (GA) with its multi-objective modification Non-dominated Sorting Genetic Algorithm (NSGA), Differential Evolution (DE), and Covariance Matrix Adaptation – Evolutionary Strategy (CMA-ES). You will also see how you can go from your specific problem (say, optimization of the number of hydraulic fracture stages or the number of infill wells) to the level of mathematical abstraction that allows application of these algorithms. At the end of the chapter, you will familiarize yourself with applications of evolutionary AI algorithms in petroleum industry with all key references.

Genetic Algorithms (GA)

The idea of *evolution* and the survival of the fittest is not new in natural sciences, but for mathematical optimization purposes, it was first suggested by Holland (1992). GA is a versatile stochastic evolutionary algorithm that can be easily customized to work with binary or continuous variables as well as one or multiple objective functions.

GA has three main components that need to be defined before it can be applied to an optimization problem (Velez-Langs, 2005):

- A *chromosome*, which serves as a representation of an optimization control vector with *n* unknowns. Each *gene* in a chromosome represents either one variable or a part of one variable (for example, you can divide the entire length of your horizontal well in equal 80 intervals and have a sequence of 80 0's and 1's signifying perforations if the value is 1 and no perforations if the value is 0);

- A number that represents the size of a GA *population* (depending on the problem, it can range from several to several hundred chromosomes);
- Operations (or sometimes referred to as operators) of *selection*, *mutation*, and *crossover* that can be used to evolve chromosomes from one generation (*parents*) to the next one (*offspring*).

A typical and generic GA workflow is shown in **Figure 2**.

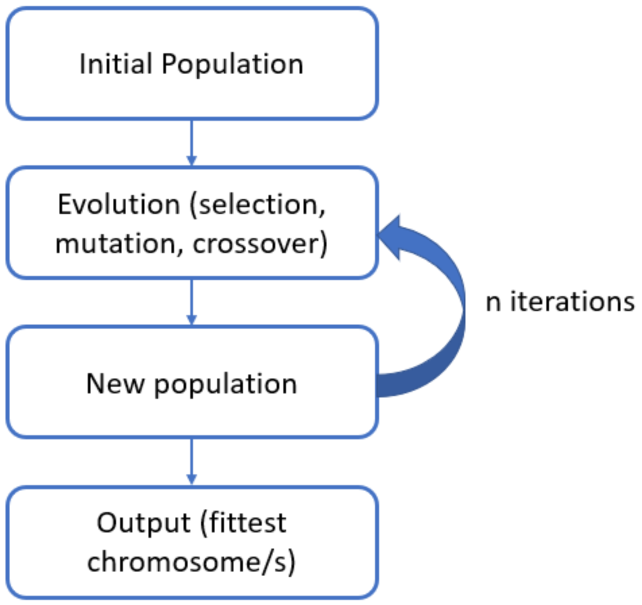
Initial Population
Evolution (selection, mutation, crossover)
n iterations
New population
Output (fittest chromosome/s)

Figure 2. GA workflow (Holland, 1992).

To start optimization using GA, you need to specify the initial population and its size. The quality of this initial population of control vectors (also named candidates or chromosomes or individuals) will define how fast GA will converge toward nearly optimal solution/s. There are many techniques to generate the initial population, and the most common of them is random sampling of the search domain.

Say, your GA chromosome for a horizontal well placement optimization problem has two variables: the horizontal wellbore length (*L*) and the number of wellbores (*N*). Based on engineering feasibility and the size of a field for which you are optimizing these two variables, you can define the domain *L~[1000 m; 2000 m]* and *N~[0, 100]*. Once the ranges for each variable are defined, you can randomly sample different values for each chromosome *(L,N)* and create the initial population using these randomly sampled chromosomes.

To perform the evolution step, GA workflow evaluates *fitness* (also referred to as *objective*) function for each chromosome (Algosayir, 2012). After that, there are many variations that GA can take based on how you as a user want to steer the evolution. For example, GA can select two chromosomes with the highest fitness values (selection) and start crossing over these chromosomes to produce the entire new population of offspring (Fathinasab and Ayatollahi, 2016). Based on the number of points at which parental chromosomes are cut and crossed over, the crossover operation can be one-point crossover, two-point crossover, or multiple points crossover (**Figure 3**).

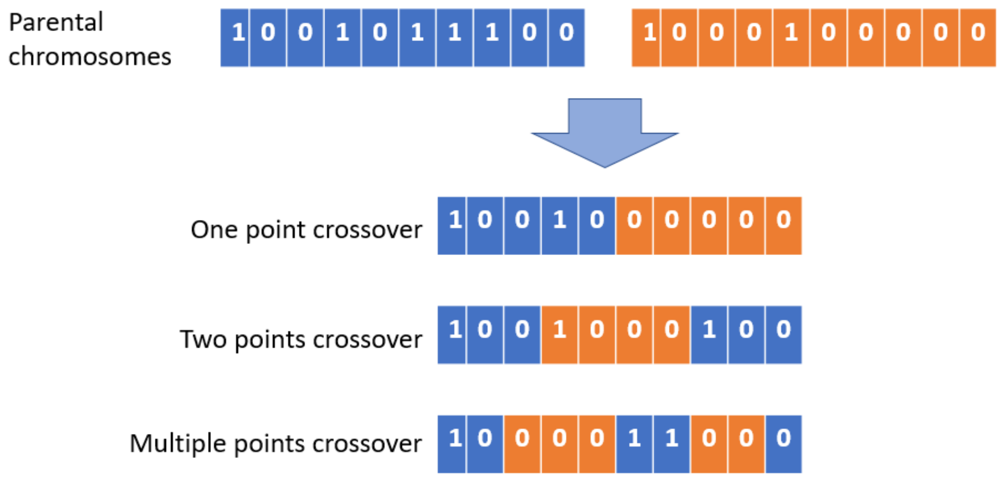

Figure 3. Types of crossover GA operation.

After crossover operation, some offspring chromosomes are mutated (changed) in random locations (genes) to prevent fast convergence and keep exploring the search domain (**Figure 4**). The next step of the evolution involves accepting new chromosomes and parental chromosomes into the new population and elimination of old chromosomes with low fitness values (Filgueiras et al., 2016).

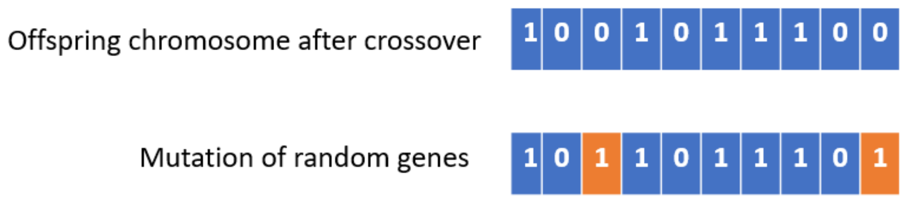

Figure 4. Mutation GA operation.

After this step, evolution of the current generation is over, and GA proceeds to the next iteration unless some termination criteria have been achieved. Termination criteria might include the maximum number of iterations (e.g. $n=1000$), absence of significant improvement of the fitness value (e.g. *tolerance* < 0.0001) or any other user-defined criterion.

Consider graphical representation of an arbitrary fitness function of two variables *f(x,y)* with one global maximum shown by a star (**Figure 5**). Note that the function is a three-dimensional surface that is represented by several projections on the *x-y* plane (similar to isopachs in geologic maps).

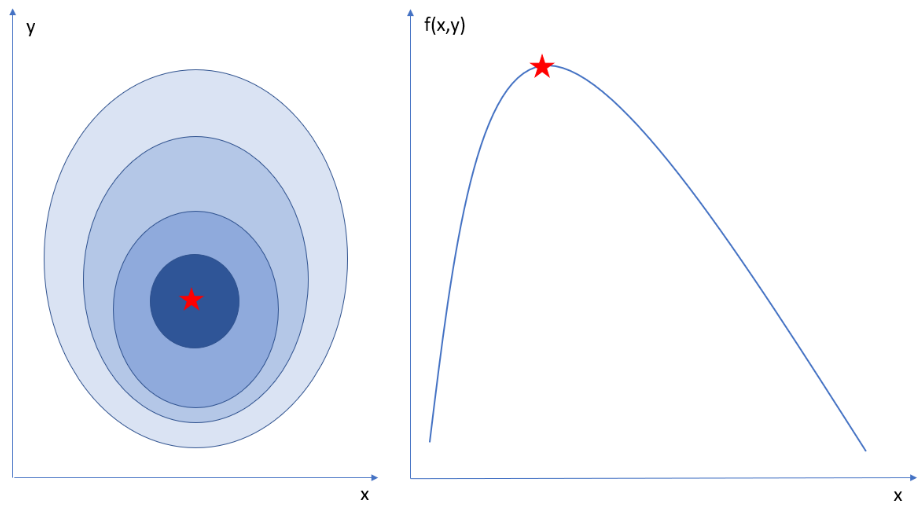

Figure 5. An arbitrary fitness (objective) function with one global maximum.

To observe how GA searches for the global optimum, refer to **Figure 6**. In this example, the evolution starts from iteration 1, for which the initial population of chromosomes randomly samples the domain *(x,y)*. During some iteration k ($k<n$), the population is converging toward the global optimum. Upon termination of GA at the final

iteration n, the final population of chromosomes is clustered closely around the global maximum, but none of them has achieved the actual optimal value. Thus, the solution obtained from any finite number of GA iterations for large domains (domains that cannot be searched using exhaustive search) is almost always nearly optimal.

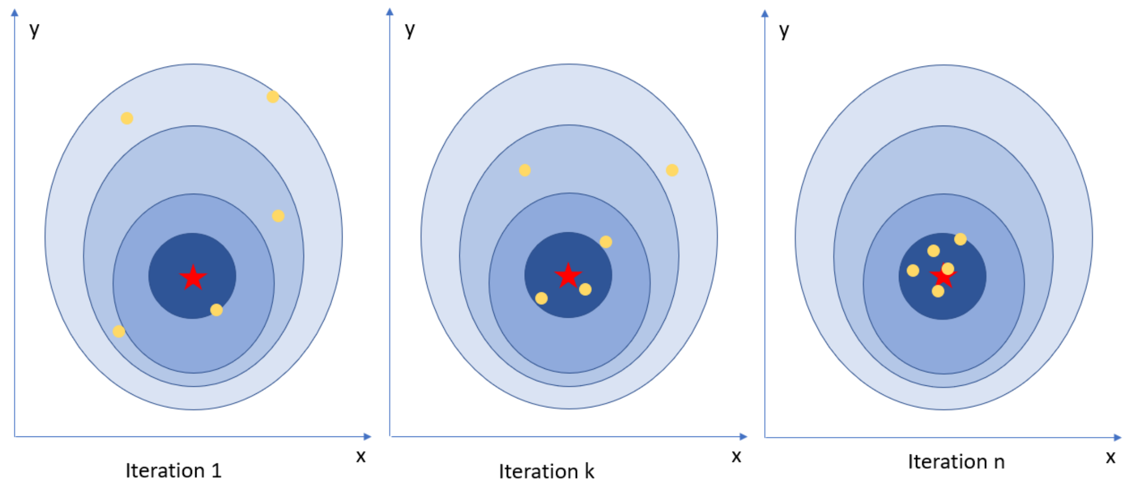

Figure 6. An example of GA search for the global maximum from iteration 1 to *n*.

GA is a very popular optimization technique due to its following advantages:

- GA does not have many parameters that require tuning before it can be applied to the problem. Among these parameters you need to specify only the size of the GA population, the maximum number of iterations, types of crossover, and number of genes in mutation;
- GA works with the population of possible solutions instead of just one candidate solution (Ab Wahab et al., 2015; Mohagheghian, 2016);
- GA can be easily modified from deterministic rules for parameter setting to stochastic ones (Algosayir, 2012);
- During GA evolution, the algorithm manipulates the entire chromosome (control vector) instead of each individual variable (Algosayir, 2012);
- GA does not use gradient to steer the search and uses direct fitness (objective) function evaluations (Bittencourt and Horne, 1997);

- GA can be easily enhanced by other algorithms to achieve better performance. For example, the converged population of GA solutions can be used in some other method as starting points for more targeted search for the optimal solution (Güyagüler et al., 2002);

- GA can be parallelized inside the same iteration as chromosomes within the same population can be used independently of each other for function evaluation purposes (Mariajayaprakash et al., 2015).

Although GA is an efficient and versatile optimization technique, it does have some drawbacks. For example,

- Random sampling of the search domain can be detrimental to GA's performance because some randomly selected candidates might fall outside of the appropriate region. Thus, additional procedures that check whether the initial chromosomes are within the domain or not should be implemented;

- GA evolutionary process and its speed of convergence are dependent on the quality of the initial population (Bittencourt and Horne, 1997);

- If the problem is sufficiently complex (chromosomes are long and the search domain has a complex shape), GA can be very slow and computationally intensive (Ballester and Carter, 2007).

GA is a global optimization algorithm that searches a nearly optimal solution for one objective function. Though this is good enough for most problems faced by our industry, we start to encounter more frequently problems that have not one but multiple objectives. For example, in optimization of CO_2 injection during CO_2 huff-n-puff process in liquid rich shale wells, operators might want to have more than one objective: maximum condensate production and maximum CO_2 volumes injected for CO_2 sequestration purposes. These two objectives could be conflicting or non-conflicting depending on the system's behavior. Thus, there is a need for a modification of GA that can handle multiple objective functions, which is addressed by the algorithm described below.

Non-dominated Sorting Genetic Algorithm (NSGA)

Multi-objective optimization (MOO) has become a rapidly developing aspect of data analytics in petroleum engineering and energy industry in general. MOO allows to assess various scenarios in presence of multiple operational and/or economic goals (or objectives) that can be conflicting or not. NSGA is one of MOO algorithms that uses GA strategy to explore complex domains formed by several objective functions.

To understand how NSGA searches for a set of nearly optimal solutions, it is instrumental to know two main approaches to solving MOO problems (Konak et al., 2005). The first approach is to formulate an *aggregate objective function* F_{agr} that includes two or more objectives $F_1, \ldots F_n$. To solve the problem using an aggregate objective function, the user must specify the weight for each individual objective as shown below for two objectives F_1 and F_2 (Marler and Arora, 2004):

$$F_{agr} = a_1 F_1 + a_2 F_2, where\ a_1 + a_2 = 1.$$

Therefore, these weights represent relative importance of the objectives. For example, if you have one objective as cumulative oil production with the weight of 0.8 and one objective as cumulative water production with the weight of 0.2 (all weights should add up to 1), then keeping oil production high is more important than keeping water production low. Although the aggregate function approach can be very effective, it is not always possible to define weights from the get-go and any small changes in these weights lead to vastly different solutions (Das and Dennis, 1997).

To avoid putting the task of defining weights for each objective on the user, researchers proposed another approach (Pareto-based approach), which produces a set of nearly optimal solutions called the *Pareto optimal set*. In the Pareto set (**Figure 7**) each solution is not better or worse than another one in some general sense, but rather each solution is better in one objective by sacrificing value in another (Sreekanth et al., 2012). **Figure 7** shows an arbitrary MOO problem with two objectives. The plot shows the space of solutions that are *dominated* (in the dashed circular region) and *non-dominated* (they are located on the Pareto front). It is also common for Pareto-based MOO to rank solutions with the Pareto optimal solutions being in Rank 1 and the rest of dominated solutions are in Rank 2, etc.

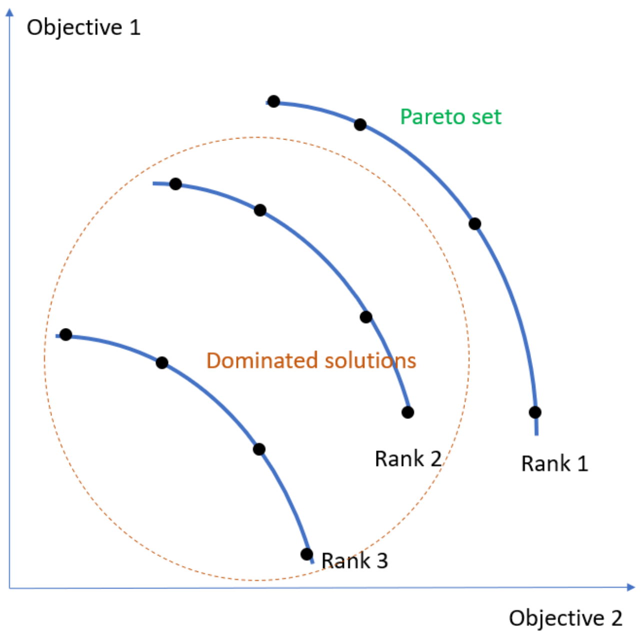

Figure 7. A generic MOO problem with two objectives and the Pareto set of non-dominated solution.

NSGA can be computationally expensive, because it generates a large set of solutions that need to be ranked. Thus, some improvements were proposed, and the algorithm's complexity was reduced from $O(MN^3)$ to $O(MN^2)$, where M is the number of objectives (in our case two) and N is the size of the GA population (Deb et al., 2002). The researchers improved NSGA's performance using elitism (preserving the fittest chromosome from the parental generation to the offspring generation) and avoiding the sharing parameter (a parameter that is similar to weights in the aggregate function approach).

The workflow of the NSGA-II method is shows below as pseudo-code.

```
for every p∈P, S_p=∅, np=0
      for every q∈P,
          if p dominates q, Sp=S_p∪{q},        % add q to set dominated by p
               else if q dominates p, np=np+1;% increment counter of p
          if np=0, prank=1, F1=F1∪{p};         % p belongs to the first front
```

```
i=1;                                              % start the front counter
while Fi≠0,Q=∅                                    % Q stores the next front
        for every p∈Fi
            for every q∈S_p,nq=nq-1,
                if nq=0, qrank=i+1,Q=Q∪{q} ;  % q is on the next front
i=i+1,
Fi=Q.
```

For each solution *p* we compute the domination counter n_p, which denotes the number of solutions dominated by this solution, and the set S_p containing the solutions dominated by the solution *p*. Thus, this improved workflow requires only $O(MN^2)$ comparisons instead of $O(MN^3)$ required by the old NSGA workflow.

To understand how NSGA-II operates inside one iteration, let us consider a schematic workflow in **Figure 8**. The method takes a population of chromosomes (the population size is *N*) that are randomly generated during the first iteration or obtained from the previous iteration, and using crossover and mutation, augments it with additional *N* chromosomes (offspring). Then, the entire augmented set is sorted based on the value of the objectives and subsequently ranked. The highest rank (Rank 1) includes all non-dominated solutions. NSGA-II maintains the population size *N*, therefore, all low-ranking solutions starting from the sorted *N+1* to *2N* chromosomes are discarded. Finally, the fittest *N* chromosomes are accepted and passed onto the next iteration or output as the result (if the maximum number of iterations has been reached).

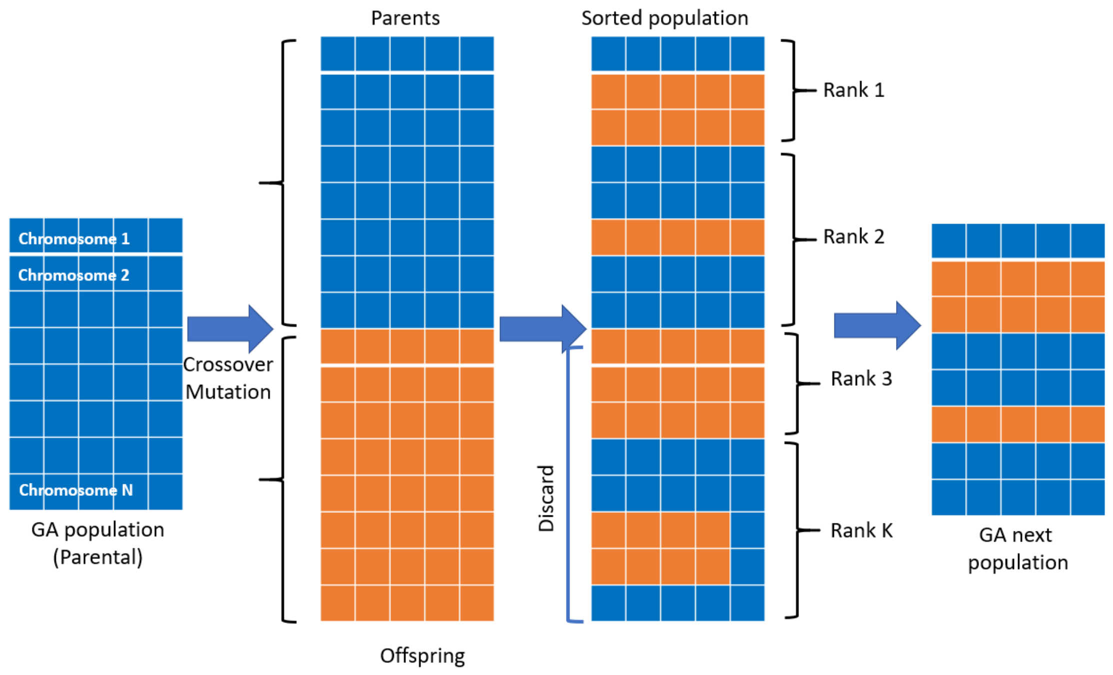

Figure 8. NSGA-II workflow inside one iteration.

In the last decade, NSGA-II has been widely applied to various petroleum engineering problems that optimize more than one objective. For many problems with two objectives, like history matching (Park et al., 2013) or well placement in shale reservoirs (Plaksina and Gildin, 2015), NSGA-II performed really well. However, there is evidence that NSGA-II becomes inefficient and computationally costly when the number of objectives increases to three or more. To address this drawback and add the capability to handle three and more than three objectives efficiently, Deb and Jain (2014) developed a modified version of their NSGA algorithm with reference-point-based non-dominated sorting approach and named it NSGA-III. Some examples of NSGA-II applications will be considered in the Applications section of this chapter.

Differential Evolution (DE)

Another evolutionary population-based algorithm that combines GA and Adaptive Random Search (ARS) is Differential Evolution (DE) method (Boender and Romeijn, 1995; Maria, 1998). ARS is a gradient-free direct-search optimization method that looks for the optimal solution using a hypersphere around the current solution and iteratively adapts the radius of the hypersphere (known as step size adaptation). At its basis, DE is similar to GA because it uses the same idea of evolution using various genetic operators. The key difference between these two AI algorithms is that DE mostly relies on mutation rather than crossover to evolve chromosomes from one generation to another (Ab Wahab et al., 2015).

DE has four main steps in its iterative workflow: generation of the initial population, mutation, crossover, and selection. First, a fixed number of chromosomes of candidate solutions (*NP*) is generated using the ranges defined for each variable inside chromosomes and a uniform random value in the range from 0 to 1 (Storn and Price, 1995). Second, this initial population of chromosomes is evolved using mutation. Mutation is performed by adding the difference of two random chromosomes from the current population to another randomly selected chromosome scaled by a factor *F*. Third, the crossover operator is applied to the entire population to increase its diversity. Unlike in GA, in DE crossover has the crossover probability rate (*CR*) and can be of two types: exponential or binomial. Final, during the selection step, every chromosome’s fitness value in the current population is compared to that of the corresponding chromosome in the initial population and the chromosome with the lower or higher (for a minimization or maximization problem, respectively) objective function value is accepted into the next generation. This evolutionary workflow is repeated until some termination criterion (e.g. the maximum number of iterations reached) is achieved (**Figure 9**).

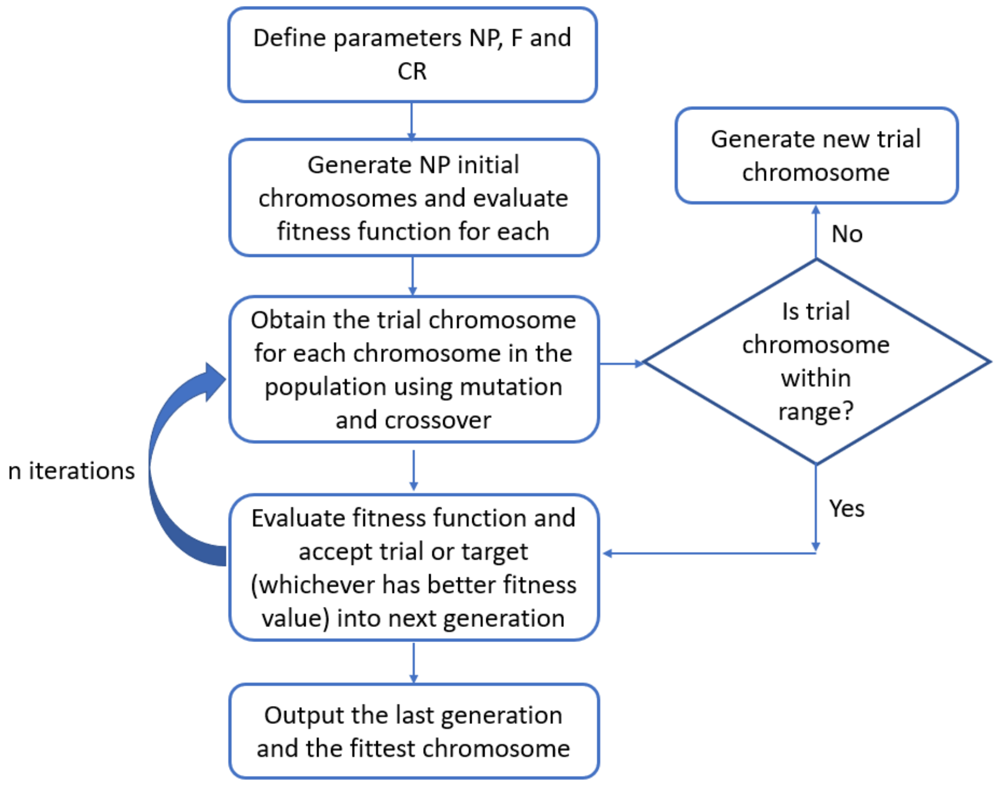
Define parameters NP, F and CR
Generate NP initial chromosomes and evaluate fitness function for each
Obtain the trial chromosome for each chromosome in the population using mutation and crossover
Evaluate fitness function and accept trial or target (whichever has better fitness value) into next generation
Output the last generation and the fittest chromosome
Generate new trial chromosome
No
Is trial chromosome within range?
Yes
n iterations

Figure 9. DE algorithm workflow.

DE requires specification of three main parameters: the size of population (*NP*), the scaling factor (*F*), and the crossover constant (*CR*). There are many approaches to define these parameters, but the most popular is to choose *NP* equal to 5–10 times of the number of the variables in a given problem, and *CR* and *F* are within the ranges between 0.1 to 1 and 0.4 to 1, respectively (Price et al., 2006).

Like all numerical algorithms, DE has some advantages and disadvantages over other methods. Among DE's advantages the researchers highlight (Khademi et al., 2010; Ab Wahab et al., 2015; Mirzabozorg, 2015):

- Ability to optimize non-differentiable, non-linear and multimodal functions;
- Relative simplicity, for example, DE requires only a couple of control parameters to steer the optimization;
- Good convergence toward the global optimum in consecutive independent runs.

It was also observed that DE's performance improves if the size of the population is increased and the chromosome with the best fitness function value is preserved from one generation to another (elitism). However, increasing of the population size should be done cautiously to avoid excessive computational costs.

Covariance Matrix Adaptation – Evolutionary Strategy (CMA-ES)

The last evolutionary algorithm covered in this chapter is Covariance Matrix Adaptation – Evolution Strategy (CMA-ES) that was developed by Hansen and Ostermeier (1996). CMA-ES is an algorithm designed for difficult non-linear non-convex optimization problems in continuous domains that relies on some distribution model of the candidate population to explore the search space (Greenwood, 2001).

This algorithm uses selection and adaptation strategy of the sample population while preserving and modifying the main strategy parameters, which is the convergence property of previous generations (*covariance matrix*) and utilizing this knowledge to produce the next generation population. In each generation the parent (parental control vector of variables) for the next generation is calculated with a weighted average of λ selected candidates from μ children (offspring) generated at the current generation using a (λ, μ)-selection. The next generation population is generated by sampling a multivariate normal distribution of the covariance matrix with the variance at the generation g over the mean of the generation $mN(m^{(g)},s^{(g)2}\ C^{(g)})$ (Hansen and Ostermeier, 1996; Greenwood, 2001). The step size (g) determines the overall variance of the mutation at generation g. The variable property of step size s at each generation plays an important role in controlling the premature convergence and close convergence to global optima (Gagganapalli, 2015). **Figure 10** shows CMA-ES general workflow that is similar to all evolutionary algorithms discussed above.

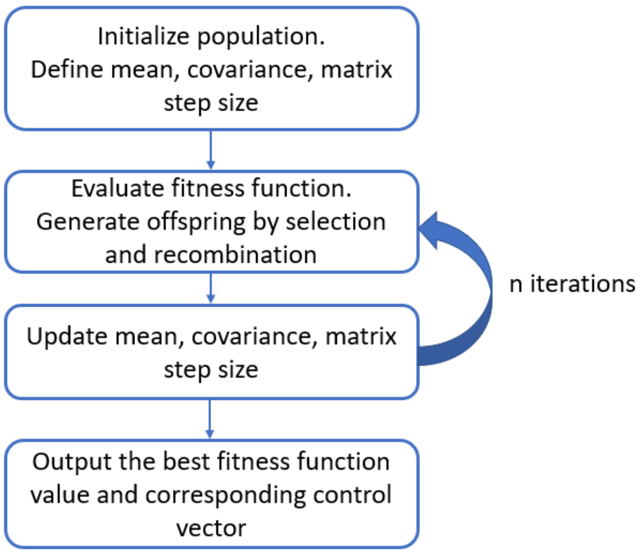
Initialize population.
Define mean, covariance, matrix step size
Evaluate fitness function.
Generate offspring by selection and recombination
Update mean, covariance, matrix step size
n iterations
Output the best fitness function value and corresponding control vector

Figure 10. General workflow of CMA-ES (Gagganapalli, 2015).

CMA-ES starts its search for a global optimum in D dimensions by first generating μ D-dimensional continuous (real valued) candidates that are sampled from a multivariate normal distribution around the mean *m* at any generation. The expression for such sampling can be written as follows:

$$x^{(g+1)}{}_k \sim N(m^{(g)}, (s^{(g)})^2\, C^{(g)}),\ for\ k=1,\ldots,K$$

Thus, to go to the next generation $(g+2)$, the parameters $m^{(g+1)}$, $C^{(g+1)}$, $s^{(g+1)}$ must be calculated. The CMA-ES crossover is achieved by calculating the *mean vector* for every generation and then mutating this vector to generate the children (offspring). The mean vector $m^{(g)}$ for generation *g* is the weighted average of the μ selected best individuals in the order of the fitness ranking of the objective function from the sample space $xk^{(g+1)}$ for $k = 0,1,2,\ldots,\mu$ (Stangeland, 2015).

The *Covariance Matrix Adaption* determines the varying mutation for the child (offspring) population in the evolutionary process. The offspring at some generation are sampled according to the multivariate normal distribution in *Rn*, while *recombination* is a selection of a new mean at generation $(g+1)$. CMA-ES mutation amounts to the sampling of the normal distribution of covariance matrix multiplied by the step size around the mean. The dependencies between the candidates in the distribution are represented by the covariance matrix.

CMA-ES offers many advantages over other optimization algorithms including:

- It performs well on non-separable and/or badly conditioned problems because its iterative process approximates a positive definite matrix (Hansen, 2006);
- Unlike quasi-Newton methods, CMA-ES does not use or approximate gradients and does not assume or require their existence. This makes the method feasible on non-convex and non-continuous problems, as well as on multimodal and/or noisy problems (Hansen, 2006);
- CMA-ES is a particularly reliable algorithm for local optimization, and it performs well on global optimization problems (Gagganapalli, 2015);
- CMA-ES does not require tedious parameter tuning and in most cases the choice of the strategy's internal parameters is completely automated (Gagganapalli, 2015).

While the above-mentioned points are very attractive characteristics, CMA-ES also has several disadvantages that are important to keep in mind when selecting it as an optimization tool:

- For relatively simple functions that can be optimized within several iterations, CMA-ES can be slower than other algorithms (e.g. multilevel coordinate search) (Hansen and Ostermeier, 2001);
- On simple or convex-quadratic functions, other second order derivative-based methods tend to be faster than CMA-ES (Hansen and Ostermeier, 2001);
- For separable functions, CMA-ES might have difficulty finding all comparable solutions (Stangeland, 2015).

Applications

In this section, you will see how researchers applied all methods described above to solve various optimization problems in petroleum engineering. All examples provided have corresponding references in case you see a strong parallel with your data analytics problem and would like to review the details of their solution.

GA has been applied to a multitude of oil and gas E&P problems for more than a decade. Below, you can review some interesting examples of GA application.

To predict formation permeability from well logs in South Pars gas field in the Persian Gulf, Saemi et al. (2007) proposed to enhance an algorithm for the auto-design of neural networks with GA. Farshi (2008) used a combination of continuous GA (GA that uses real numbers as genes inside chromosomes) and binary GA (GA that uses only 0's and 1's as genes inside chromosomes) to optimize vertical and directional well placement. The author demonstrated that such hybridization approach yielded higher objective function values in shorter computational time.

GA can also be used for recovery processes optimization. For example, Chen et al. (2010) combined GA with the Tabu search method to optimize EOR process by controlling parameters such as water alternative gas (WAG) ratio, injection cycle time, water and gas injection rates, and bottomhole pressures of the oil producers. The researchers reported that this combination of GA and the Tabu method significantly improved the convergence speed, increased the recovery factor and the Net Present Value (NPV) of the project. Another EOR optimization is shown in Edmunds et al. (2010) work that optimized a steam and solvent cycling process using GA method. This application reduced the physical cumulative steam to oil ratio (CSOR) to the value close to one.

Another interesting application of GA to steam and solvent injection in oil sands and fractured carbonate reservoirs is demonstrated in the work of Algosayir (2012). The author brought together GA, simulated annealing, orthogonal arrays and response surface proxy techniques to achieve better performance. The results showed that using a proxy saved up to 95% of the computational time, while application the orthogonal arrays (with the minimax criterion) improved the convergence. Another EOR process, steam-assisted gravity drainage (SAGD), was optimized by combined binary and continuous GA by Chen (2013). In this solution the author used a numerical simulator to optimize the steam injection rates in a saturated oil reservoir.

GA has been applied to the problems of infill well placement and well drilling. For example, Salmachi et al. (2013) optimized infill well placement in coal bed methane reservoirs by integrating GA and reservoir simulator (for fast NPV objective function evaluation). To optimize drilling operations in an abnormally pressured formation in a Louisiana offshore field, Guria et al. (2014) used binary GA on a control vector consisting of equivalent circulation density, rotary speed of the drill bit, weight on drill bit, and the Reynolds number in drill bit nozzles to optimize drilling depth, drilling time, and the cost of drilling functions.

Because history matching is a type of optimization problem (minimization of the difference between observed and simulated data), it is natural to expect multiple applications of GA in history matching. For example, a modification of GA with altered crossover and mutation rates was applied by Xu et al. (2015) to history matching of the simulation data with the experimental results of the vapor extraction (VAPEX) heavy oil recovery process. Using this modified GA, the researchers achieved 71% reduction of the computational cost and were able to match the data with the less than 1% error in comparison to the error obtained with the conventional GA.

GA has been also applied to optimization of unconventional assets (shale and tight reservoirs) development. For example, in addition to simultaneous perturbation stochastic approximation (SPSA) and covariance matrix adaptation evolution strategy (CMA-ES), GA was used to place horizontal wells inside a shale gas reservoir and optimally space hydraulic fracture (HF) stages along them (Ma et al., 2015). The results of this study show that all three approaches (SPSA, GA, and CMA-ES) can handle various HF stage spacing in geologic systems with homogeneous and heterogeneous petrophysical properties. It was also observed that for increasing number of control variables inside the control vector, evolutionary methods GA and CMA-ES converged faster to higher objective function values than SPSA.

Plaksina and Gildin (2017) presented a step-by-step workflow in which the problem of selecting optimal HF stage spacing and the number of horizontal wells was solved using binary GA. Specifically, the authors defined the control vector as a binary sequence encoding the number of horizontal wells (constrained by the geometry of the field and HF half-length), followed by about 80 genes encoding segments of the horizontal well, in which HF stage could be placed (1 is for HF stage and 0 is for no HF stage inside this segment), and ending with the number corresponding to the HF half-length (constrained by physical ability to break rock and maintain fracture open). The workflow included the procedure that checked whether the number of wells and the HF half-length are staying inside the allowed range during all iterations and in each chromosome to avoid falling out of the search domain (**Figure 11**). If any of these numbers ended up outside the allowed ranges, the algorithm would continue regenerating them until they fall inside the search domain. The chromosomes evolved over the specified number of iterations using elitism, crossover, and mutation, and for each chromosome the NPV function was evaluated. Each NPV function evaluation required one run of a numerical simulator (Eclipse was chosen for this work), which provided 10-year gas production history resulting from the production arrangement encoded by the evaluated chromosome.

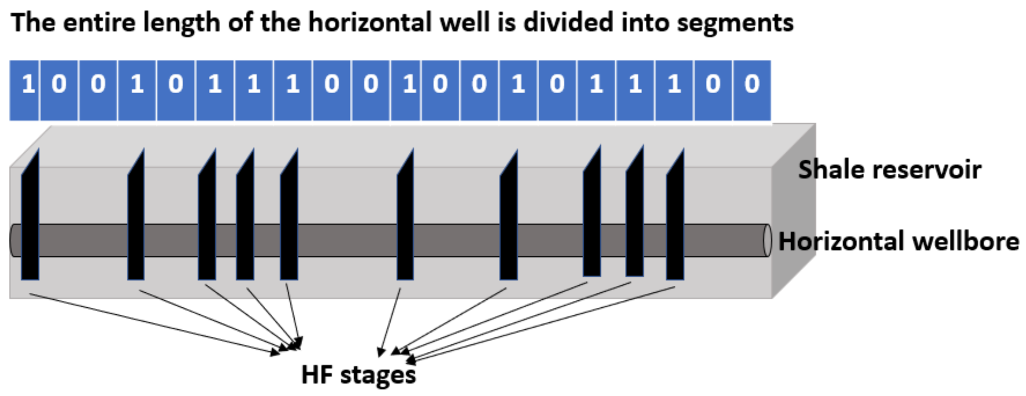

Figure 11. Conceptual representation of HF stage and well placement problem in shale gas reservoirs.

Similarly, Rahmanifard and Plaksina (2018) used GA among other algorithms to solve the same problem of well and HF stage placement in shale gar reservoirs, but instead of costly reservoir simulator calls (which would take about 4 minutes to evaluate one chromosome), in this study the authors used a fast analytical method (Wattenbarger slab model) to obtain 10-year gas production history in seconds.

Now, let us consider some applications of NSGA-II to petroleum engineering problems. MOO is a relatively new concept in petroleum data analytics, so there is a limited number of examples available to date. For example, Park et al. (2013) used NSGA-II to history match actual production history and the output of a numerical model in presence of two objectives: water cut and changes in water saturation.

Another interesting application of NSGA-II is shown in the work of Plaksina and Gildin (2015), in which the authors solved the problem of optimal placement of horizontal wells and HF stages along them in the Barnett shale in presence of two objectives. They considered two cases: conflicting objectives (long-term discounted NPV vs. cumulative water production) and non-conflicting objectives (long-term discounted NPV vs. short-term discounted NPV). The authors demonstrated that for conflicting objectives the Pareto front had multiple solutions depending on relative importance of objectives (similar to weights in the aggregate function approach), while for non-conflicting objectives the Pareto front had only one solution. Thus, if it is not clear from the statement of the problem whether multiple objectives are conflicting or not (say, you have more than two of them), you can draw conclusions about conflict or lack of such from the shape of the Pareto front.

Another interesting application of MOO can be found in the work by Kanfar and Clarkson (2017), in which the authors history match both flowback (early time production data when the well is still cleaning after HF treatment) and late time production data to the output of their numerical model to obtain a better estimate of the liquid-rich shale formation permeability. The authors used NSGA-II to optimize three objective functions: oil misfit (the difference between observed and modeled oil rates), gas misfit (the difference between observed and modeled gas rates), and water misfit ((the difference between observed and modeled water rates)). Prior to this study, history matching for such reservoirs (to correct formation permeability by minimizing misfit between actual and modeled data) was performed with either flowback or production data, but the authors showed how to use both types of data to obtain even better formation permeability estimates.

DE has been also successfully applied to history matching problems and production design optimization. For example, Wang and Buckley (2006) used DE algorithm to obtain the match between modeled capillary pressures and relative permeabilities and observed core flooding data. Decker and Mauldon (2006) optimized fracture shapes and sizes in hydraulically fractured wells using a cost function as the objective function. In their work, the authors demonstrate that DE algorithm is an applicable tool to estimate of key fracture characteristics obtained by geological analysis. Jahangiri (2007) used DE in combination with inflow control devices and DE to optimize production constraints and maximize oil production from smart wells.

DE algorithm can also be used for geostatistical applications. For example, Zhang et al. (2009) applied DE to obtain optimal solutions of variogram properties in geostatistical systems. In their study, the authors found that DE is more cost effective and stable in comparison to GA in matching the key variogram variables with experimental data. Hajizadeh et al. (2009) studied performance of DE and the neighborhood algorithm (NA) in history matching of a black oil model and concluded that DE performed better than NA in matching the field data. Later, Hajizadeh et al. (2010) also investigated convergence and robustness of different population-based optimization algorithms such as DE, NA, and ant colony optimization (ACO) solving a history matching problem and found that DE had the fastest convergence rate and lowest misfit value, while NA did not perform for the problem with a large number of unknowns.

Zhang et al. (2014) applied DE to a micro-seismic study and developed a robust velocity model for more accurate data interpretation. Awotunde and Mutasiem (2014) used DE for drilling operations optimization and minimized the cost of drilling operations and drilling time. Finally, Mirzabozorg (2015) compared performance of DE and PSO algorithm as history matching tools for a homogeneous 2D and a heterogeneous 3D SAGD reservoir models. The author's results show that DE is more efficient than PSO and provided quantitative justification for this observation.

To finish the review of petroleum related application of evolutionary algorithms, let us consider some interesting optimization problems that used CMA-ES as the primary tool. CMA-ES is a relatively new algorithm, but it is gaining wide popularity and the research studies featuring this method are becoming more frequent in the last five years.

Fonseca et al. (2013) used CMA-ES to improve computational efficiency of EnOpt (CMA-EnOpt) that was applied to optimization of waterflooding of a multilayered geomodel of a reservoir with multiple sealing and leaky faults. The authors used inflow control values settings for injector and producer wells and undiscounted NPV as the fitness function for optimization. To draw a conclusion about the improved performance, the authors compared EnOpt and CMA-EnOpt and observed a slight increase in computational speed and the objective function value for the latter method. However, the study highlights that CMA-EnOpt is preferable because of its near-independence of the initial choice of parameters and robustness.

Similarly to GA, CMA-ES has been applied to various workflows for optimal well (both vertical and horizontal) placement and completion/production optimization using numerical reservoir simulation. For example, Ma et al. (2015) used CMA-ES in addition to GA and Simultaneous Perturbation Stochastic Approximation (SPSA) algorithms to place horizontal wells and HF stages along these wells in a dry gas shale reservoir. The authors observed that while SPSA was the most efficient in terms of running time, GA and CMA-ES explored the search space better, and thus, achieved higher objective function (discounted NPV) values with CMA-ES showing the highest results. Siddiqui et al. (2015) also applied several optimization algorithms (PSO, DE and CMA-ES) to the problem of vertical well placement and production rate optimization during gas cycling in gas condensate reservoirs. The authors found that the best performance was achieved when both well placement and production optimization were done simultaneously, while in terms of the algorithms they observed the highest NPV value in DE application cases. In another well placement study, Forouzanfar et al. (2016) investigated the effect of performing simultaneous versus sequential optimization of choosing well trajectories and controlling their life-cycle production. The authors applied both approaches to the PUNQ reservoir model and concluded that simultaneous well placement/control approach using CMA-ES yielded better results in terms of NPV.

CMA-ES was also applied to EOR process optimization. For example. Jamal et al. (2016) used CMA-ES to optimize parameters in polymer alternating gas (PAG) EOR process for mature conventional reservoirs. The authors used a numerical simulator to design the EOR process and its steps and applied CMA-ES and PSO to optimize the main parameters of the developed process.

One of the latest applications of CMA-ES is featured in the numerical CO_2 sequestration study by Lu et al. (2019). The authors model CO_2 sequestration and surfactant alternating gas (SAG) processes using coupled reservoir flow and geomechanics simulation approach and optimized well control parameters using CMA-Es and GA to achieve optimal performance (maximize CO_2 storage volume). The results of this study show that optimized SAG process yielded higher sequestration volumes than regular CO2 sequestration operations.

In the next chapter, you will be introduced to another class of AI algorithms, swarm intelligence, and learn how they are different from the methods presented in this chapter. You will also learn some interesting applications of these algorithms to petroleum engineering problems.

References

Ab Wahab, M., Nefti-Meziani, S., and Atyabi, A. (2015). A comprehensive review of swarm optimization algorithms. PloS one **10**(5): e0122827.

Algosayir, M. M. (2012). Optimization of steam/solvent injection methods: application of hybrid techniques with improved algorithm configuration, University of Alberta.

Awotunde, A. A. and M. A. Mutasiem (2014). Efficient drilling time optimization with differential evolution. SPE Nigeria Annual International Conference and Exhibition, Society of Petroleum Engineers.

Ballester, P. and Carter J. (2007). A parallel real-coded genetic algorithm for history matching and its application to a real petroleum reservoir. Journal of Petroleum Science and Engineering **59** (3): 157-168.

Bittencourt, A. and Horne, R. (1997). Reservoir development and design optimization. SPE Annual Technical Conference and Exhibition, Society of Petroleum Engineers.

Boender, C. and Romeijn, H. (1995). Stochastic methods. Handbook of global optimization, Springer**:** 829-869.

Carroll Jr, J. and Horne, R. (1992). Multivariate optimization of production systems. Journal of Petroleum Technology **44** (07): 782-831.

Chen, S., Li, H., Yang, D. et al. (2010). Optimal parametric design for water-alternating-gas (WAG) process in a CO_2-miscible flooding reservoir. Journal of Canadian Petroleum Technology **49** (10): 75-82.

Chen, Z. (2013). A genetic algorithm optimizer with applications to the SAGD process, University of Calgary.

Das, I. and Dennis, J. (1997). A closer look at drawbacks on minimizing weighted sums of objectives for Pareto set generations in multi-criteria optimization problems. Structural and multidisciplinary Optimization **14** (1): 63-69.

Deb, Kand Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. IEEE Transactions on Evolutionary Computation, **18** (4), 577-601.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitism multi-objective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6** (2): 182-197.

Decker, J. and Mauldon, M. (2006). Determining size and shape of fractures from trace data using a differential evolution algorithm. Golden Rocks 2006, The 41st US Symposium on Rock Mechanics (USRMS), American Rock Mechanics Association.

Edmunds, N., Peterson, J., and Moini, B. (2010). Method for viscous hydrocarbon production incorporating steam and solvent cycling, Google Patents.

Farshi, M. (2008). Improving genetic algorithms for optimum well placement, Stanford University.

Fathinasab, M. and Ayatollahi, S. (2016). On the determination of CO_2–crude oil minimum miscibility pressure using genetic programming combined with constrained multivariable search methods. Fuel **173**: 180-188.

Filgueiras, P., Portela, N., Silva, S., et al. (2016). Determination of saturates, aromatics, and polars in crude oil by 13C NMR and support vector regression with variable selection by genetic algorithm. Energy & Fuels **30** (3): 1972-1978.

Fonseca, R., Leeuwenburgh, O., Van den Hof., P., et al. (2013). Improving the ensemble optimization method through covariance matrix adaptation (CMA-EnOpt). SPE Reservoir Simulation Symposium, Woodlands, TX.

Forouzanfar, F., Poquioma, W., and Reynolds, A. (2016). Simultaneous and sequential estimation of optimal placement and controls of wells with a covariance matrix adapttion algorithm. SPE Journal April: 501- 521.

Fujii, H. and Horne, R. (1995). Multivariate optimization of networked production systems. SPE Production & Facilities **10** (03): 165-171.

Gagganapalli, S. (2015). Implementation and evaluation of CMA-ES algorithm. Circulation **701**: p. 8888.

GononrI, R. E. (1958). Outline of an algorithm for integer solutions to linear programs.

Greenwood, G. (2001). Finding solutions to NP problems: philosophical differences between quantum and evolutionary search algorithms. Proceedings of the 2001 Congress on Evolutionary Computation.

Guria, C., Goli, K., and Pathak, A. (2014). Multi-objective optimization of oil well drilling using elitist non-dominated sorting genetic algorithm. Petroleum Science **11**(1): 97-110.

Güyagüler, B., Horne, R., Rogers, L. et al. (2002). Optimization of well placement in a Gulf of Mexico waterflooding project. SPE Reservoir Evaluation & Engineering **5** (03): 229-236.

Hajizadeh, Y., Christie, M., and Demyanov, V. (2009). Application of differential evolution as a new method for automatic history matching. Kuwait International Petroleum Conference and Exhibition, Society of Petroleum Engineers.

Hajizadeh, Y., Christie, M., and Demyanov, V. (2010). Comparative study of novel population-based optimization algorithms for history matching and uncertainty quantification: PUNQ-S3 revisited. Abu Dhabi International Petroleum Exhibition and Conference, Society of Petroleum Engineers.

Hansen, N. (2006). The CMA evolution strategy: a comparing review, in “Towards a new evolutionary computation.” Springer. p. 75-102.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. Proceedings of IEEE International Conference on Evolutionary Computation.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. Evolutionary computation **9** (2): 159-195.

Holland, J. (1992). Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence, MIT Press, Cambridge, MA.

Jahangiri, H. (2007). Production optimization using smart well technology with differential evolution algorithm. Graduate Student Symposium, University of Southern California.

Jamal, M., Al-Nuaim., S., et al. (2016). Optimal parameter selection in a polymer alternating gas PAG process. SPE KSA Annual Technical Symposium and Exhibition, Dammam, Saudi Arabia.

Kanfar, M. and Clarkson, C. (2017). Reconciling flowback and production data: a novel history matching approach for liquid rich shale wells. Journal of Natural Gas Science and Engineering **33**: 1134-1148.

Khademi, M., Rahimpour, M., and Jahanmiri, A. (2010). Differential evolution (DE) strategy for optimization of hydrogen production, cyclohexane dehydrogenation and methanol synthesis in a hydrogen-permselective membrane thermally coupled reactor. International journal of hydrogen energy **35** (5): 1936-1950.

Klee, V. and Minty, G. (1970). How good is the simplex algorithm, DTIC Document.

Konak, A., Coit, D., and Smith, A. (2006). Multi-objective optimization using genetic algorithms: a tutorial. Reliability Engineering and System Safety **91**: 992-1007.

Land, A. and Doig, A. (2010). An automatic method for solving discrete programming problems. 50 Years of Integer Programming 1958-2008, Springer**:** 105-132.

Lu, X., Ganis, B., and Wheeler, M. (2019). Optimal design of CO_2 sequestration with three-way coupling of flow-geomechanics simulations and evolution strategy. SPE Reservoir Simulation Conference, Galveston, TX.

Ma, X., Gildin, E., and Plaksina, T. (2015). Efficient optimization framework for integrated placement of horizontal wells and hydraulic fracture stages in unconventional gas reservoirs."Journal of Unconventional Oil and Gas Resources **9**: 1-17.

Maria, G. (1998). IDENTIFICATION/DIAGNOSIS-Adaptive random search and short-cut techniques for process model identification and monitoring. AIChE Symposium Series, New York, NY: American Institute of Chemical Engineers, 1971-c2002.

Mariajayaprakash, A., Senthilvelan, T., and Gnanadass, R. (2015). Optimization of process parameters through fuzzy logic and genetic algorithm–a case study in a process industry. Applied Soft Computing **30**: 94-103.

Marler, R. and Arora, J. (2004). Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization **26** (6): 369-395.

Mirzabozorg, A. (2015). Incorporation of engineering knowledge in history matching, optimization, and uncertainty assessment frameworks with application to the SAGD process, University of Calgary.

Mohagheghian, E. (2016). An application of evolutionary algorithms for WAG optimisation in the Norne Field, Memorial University of Newfoundland.

Park, H., Datta-Gupta, A., and King, M. (2013). Handling conflicting multiple objective using pareto-based evolutionary algorithms during history matching of reservoir performance. SPE Reservoir Simulation Symposium, The Woodlands, Texas, USA.

Plaksina, T. and Gildin E. (2015). Practical handling of multiple objectives using evolutionary strategy for optimal placement of hydraulic fracture stages in unconventional gas reservoirs. Journal of Natural Gas Science and Engineering **27**: 443-451.

Plaksina, T. and Gildin, E. (2017). Rigorous integrated evolutionary workflow for optimal exploitation of unconventional gas assets. International Journal of Energy Optimization and Engineering **6** (1): 101-122.

Price, K., Storn, R., and Lampinen, J. (2006). Differential evolution: a practical approach to global optimization, Springer Science & Business Media.

Rahmanifard, H. and Plaksina, T. (2018). Application of fast analytical approach and AI optimization techniques to hydraulic fracture stage placement in shale gas reservoirs. Journal of Natural Gas Science and Engineering **52**: 367-378.

Saemi, M., Ahmadi, H., and Varjani, A. (2007). Design of neural networks using genetic algorithm for the permeability estimation of the reservoir. Journal of Petroleum Science and Engineering **59** (1): 97-105.

Salmachi, A., Sayyafzadeh, M., and Haghighi, M. (2013). Infill well placement optimization in coal bed methane reservoirs using genetic algorithm. Fuel **111**: 248-258.

Siddiqui, M., Al-Nuaim, S., and Khan, R. (2015). Well placement and rate optimization for gas cycling in gas condensate reservoirs. SPE Middle East Oil and Gas Show, Manama, Bahrain.

Skeekanth, J., Datta, B., and Mohapatra, P. (2012). Optimal short-term reservoir operation with integrated long-term goals. Water Resources Management **26**: 2833-2850.

Stangeland, K. (2015). Positioning in electromagnetic fields. University of Stavanger, Norway.

Storn, R. and Price, K. (1995). Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces, Berkeley. CA: International Computer Science Institute.

Velez-Langs, O. (2005). Genetic algorithms in oil industry: An overview. Journal of Petroleum Science and Engineering **47** (1): 15-22.

Wang, J. and Buckley, J. (2006). Automatic history matching using differential evolution algorithm. International Symposium of the Society of Core Analysis, Trondheim.

Wang, P. (2003). Development and applications of production optimization techniques for petroleum fields, Stanford University.

Watson, A., Seinfeld, J., Gavalas, G. et al. (1980). History matching in two-phase petroleum reservoirs. Society of Petroleum Engineers Journal **20** (06): 521-532.

Xu, S., Zhang, M., Zeng, F. et al. (2015). Application of genetic algorithm (GA) in history matching of the vapour extraction (VAPEX) heavy oil recovery process. Natural Resources Research **24** (2): 221-237.

Zhang, D., Gong, X., and Peng, L. (2009). Estimating geostatistics variogram parameters based on hybrid orthogonal differential evolution algorithm. International Symposium on Intelligence Computation and Applications, Springer.

Zhang, J., Zhang, H., Yu, J. et al. (2014). Fast one-dimensional velocity model determination using station-pair differential times based on the differential evolution method in microseismic monitoring. SEG Technical Program Expanded Abstracts 2014, Society of Exploration Geophysicists**:** 4832-4836.

Chapter Two: Swarm Intelligence

Swarm intelligence (SI) is another broad group of AI algorithms that mimic collective behavior of decentralized agents (organisms, elements, or particles) that pursue a common goal or objective. Similarly to evolutionary computation, SI maintains a population of elements (sometimes called *boids*) that interact between each other, and based on the information received from their environment, steer their search toward the global optimum. Many of SI algorithms were directly inspired by behavior of natural biological systems and species, in which each decentralized individual contributes to the "greater good" of the entire community or colony.

SI imposes a set of rules on each boid that mimic a somewhat random individual behavior, and yet interaction between these individual boids leads to "intelligent" collective behavior. Examples of SI in the nature include systems like bacterial colonies, ant colonies, schools of fish, packs of wolves, flocks of birds, etc. (Abraham et al., 2006; Engelbrecht, 2006).

Main characteristics of SI that distinguish this class of algorithms from other AI algorithms are decentralization of all boids, their self-organization, communication and cooperation of boids in apparent absence of some centralized control system. As Bonabeau and Meyer (2001) point out that despite simplicity of individual behavior of boids, the resulting behavior of the system of boids (SI systems) can be very complex. In the last couple of decades, petroleum engineering experienced a surge of interest in application of SI algorithms to various optimization problems (Edelen, 2003; Kamrani, 2010; Ganesan et al., 2013; Alam et al., 2014; Senthilkumar, 2014). In this chapter, you will be introduced to three popular SI algorithms that were successfully applied to discrete and continuous optimization problems in the oil and gas industry: Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Imperialist Competitive Algorithm (ICA).

Particle Swarm Optimization (PSO)

PSO was proposed by Eberhart and Kennedy (1995) as an optimization technique that mimics behavior of natural systems such as schools of fish or swarms of insects. Similarly to evolutionary computation algorithms discussed in the previous chapter, PSO has a population of boids (particles or candidates) that are randomly generated at the initial step of the workflow, and uses an objective function to evaluate each candidate and steer the search. Furthermore, PSO updates the population of the candidates based on their fitness and uses some degree of randomization in the search for the global optimum. The main difference of PSO from any other evolutionary algorithm is that PSO does not modify its candidates using mutation or crossover. Instead, PSO considers two main parameters of each candidate: a vector that stores the candidate's position in the search space and the candidate's velocity (Kaewkamnerdpong and Bentley, 2005).

At the step of initialization, PSO randomly generates the particles' (candidates') positions and velocities. After that, at each iteration, PSO updates position and velocity of each particle until satisfactory solutions or some termination criteria are achieved. Thus, each particle in the population travels through the search space using its regularly updated velocity function and traces the trajectory of its best positions and the best positions of the entire swarm (Mohamed et al., 2011). In addition to this, the particle's trajectory records the best encountered positions and the best population (swarm) size. To update the particle's position for the next iteration, PSO combines this particle's best position (*pbest*) with the other particles' best positions (*gbest*) (Mirzabozorg, 2015). **Figure 12** shows a generic trajectory of a PSO particle.

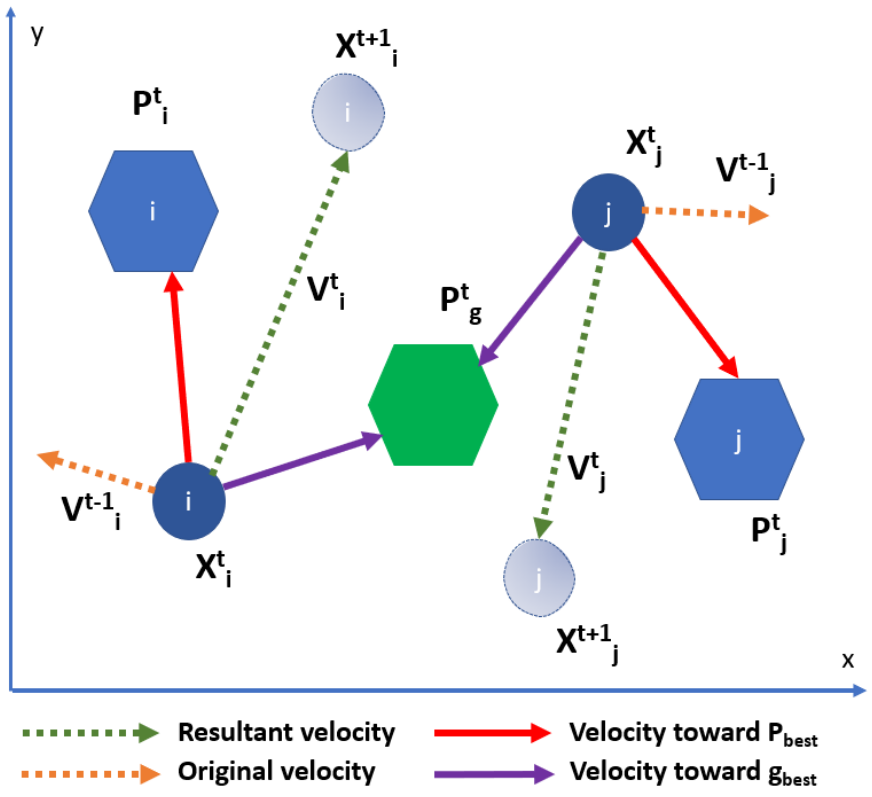

Figure 12. Trajectory of an inertia weight particle in PSO. (adapted from Mohagheghian, 2016).

A generic workflow of PSO algorithm with inertia weight implementation is shown in **Figure 13**.

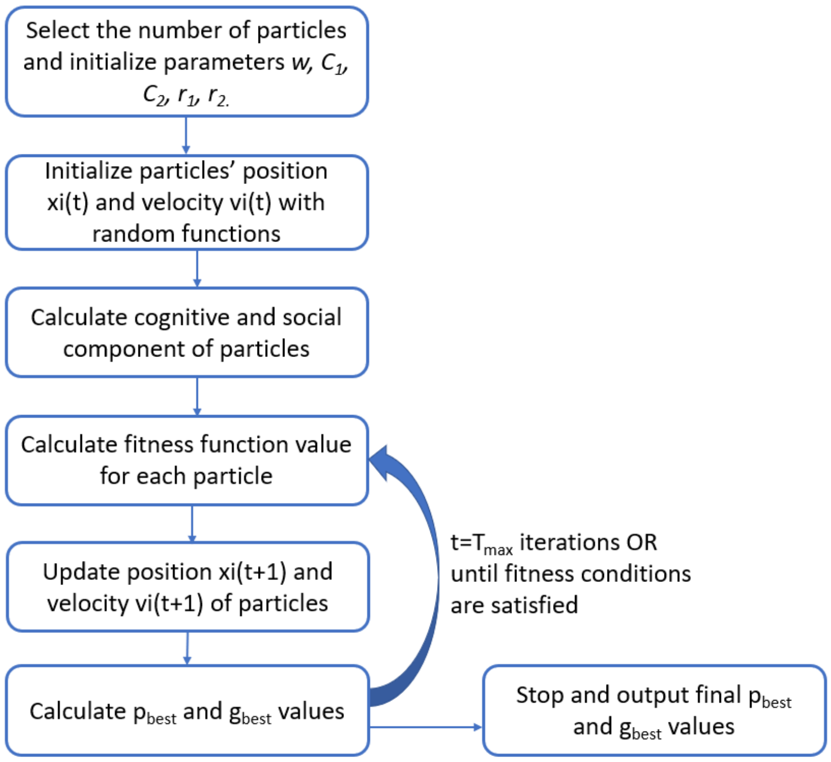
Select the number of particles and initialize parameters *w, C₁, C₂, r₁, r₂.*
Initialize particles' position xi(t) and velocity vi(t) with random functions
Calculate cognitive and social component of particles
Calculate fitness function value for each particle
Update position xi(t+1) and velocity vi(t+1) of particles
Calculate pbest and gbest values
t=Tmax iterations OR until fitness conditions are satisfied
Stop and output final pbest and gbest values

Figure 13. General PSO workflow (Kathrada, 2009; Assareh et al., 2010).

Note that w is the inertia weight that is usually varied linearly decreasing from w_{max} to w_{min} during the optimization process as follows:

$$w(t+1) = w_{min} + (w_{max} - w_{min})\frac{(T_{max}-t)}{T_{max}},$$

where T_{max} is specified by the user as the maximum number of iterations and $w_{min} = 0.4$ and $w_{max} = 0.9$ are experimentally established and suggested by Shi et al. (1998). It was demonstrated that inertia weight variation leads to better convergence and results than the application of a fixed inertia value. Thus, inertia weight PSO uses the following equations to calculate velocities v_{ij} and positions x_{ij} of the particles in each iteration t:

$$v_{ij}(t+1) = wv_{ij}(t) + C_1 r_1 \left(P_{ij} - x_{ij}(t)\right) + C_2 r_2 \left(P_{gj} - x_{ij}(t)\right)$$

$$if\ v_{ij} > V_{max.j}\ \ then\ v_{ij} = V_{max,j}$$

$$else\ if\ v_{ij} < -V_{max,j}\ then\ v_{is} = -V_{max,j}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1),$$

where r_1 and r_2 are the numbers sampled randomly from the interval [0,1] and C_1 and C_2 are the acceleration constants for individual and global bests. As a rule, $C_1=C_2=2$ is selected, but these values can be modified based on your problem and convergence rate (Kathrada, 2009).

Among strengths of PSO, the reader can note relative ease in use (it requires a small number of parameters as the input data), straightforward implementation, acceptable efficiency in the global optimum search, and flexibility. Among disadvantages of PSO, the reader should be aware of its tendency to converge slowly (weak local search ability). This disadvantage can be mitigated by increasing the population size, introduction of the dynamic velocity adjustment, and a sub-swarm approach to increase convergence toward a global optimum. (Atyabi and Powers, 2013; Chang and Yu, 2013; Ab Wahab et al., 2015).

Ant Colony Optimization (ACO)

Ant colony optimization (ACO) is also a population-based AI algorithm that is designed to solve complex optimization problems. Similarly to PSO, ACO uses a set of boids called *artificial ants* (or simply *ants*) to search the solution space of a given optimization problem. In ACO, any problem is reformulated into the problem of finding the best path on a weighted graph. The ants investigate the graph and construct the solutions. These solutions are constructed in a stochastic fashion and are governed by a *pheromone model* (a *pheromone matrix*), which is a set of parameters associated with the graph components (such as nodes or edges). The values of the pheromone matrix are constantly updated as the ants explore the solution space (Dorigo et al., 1996).

Let us consider ACO and its application using the traveling salesman problem (TSP). In the TSP, you need to optimize the route given a set of locations (e.g. cities as graph nodes) and the distances between them (as values on graph edges). In the context of the TSP, the optimal solution is the shortest route (the minimal total travelled distance) that visits each city once and only once. Because in the TSP there is no limitation on movement from one city to another, the graph representing all possible routes is fully connected and the number of vertices represents the number of cities. The lengths of the edges in the graph are proportional to the distances between the cities (vertices). To apply ACO, let us assign pheromone values and heuristic values to the edges of the graph. Pheromone values are modified during the search and represent the cumulated experience of all ants involved in the search, while heuristic values depend on the nature of the problem and for the TSP, they represent the inverse of the lengths of the edges.

In the TSP, ACO constructs the solutions as follows. The entire population of ants starts exploration from randomly selected cities (vertices of the graph). After that, during each iteration each ant moves along the edges of the graph keeping memory of its path. The ant never visits the same vertex (city) two times and chooses only those edges that lead to some new vertex (city). Ants use the probabilistic rule to choose which edge to follow among those that lead to yet another unvisited vertex. This rule is governed by pheromone values and some heuristic information. According to this information, the higher the pheromone and the heuristic values assigned to an edge, the higher the probability that an ant will prefer this edge. After the entire population of ants has completed traversing the graph, the pheromone values assigned to the edges are updated. Note that all pheromone values are initially low, but then receive additional boosts proportional to the quality of the solutions to which it belongs (there is one solution per ant). This workflow is repeated for the specified number of iterations or until some termination criterion has been reached.

To define application of ACO in more rigorous mathematical terms, let us consider a combinatorial optimization problem (COP) that can be defined as a triplet (S, Ω, f), where S is a search space defined over a finite set of discrete decision variables, Ω is a set of constraints for the variables, and $f : S \rightarrow R+0$ is an objective (fitness) function to be optimized. Using this notation, the search space S is a set of discrete variables X_i , i=1,…,n, with values $v_{ji} \in D_i = \{v_{1i},\dots,v_{|Di|i}\}$. Elements of S are values for each variable X_i that has a value v_{ji} assigned from its domain D_i. The set of feasible solutions $S\Omega$ consists of the elements of S that satisfy all the constraints in the set Ω. Thus, a solution $s^* \in S\Omega$ is called a global optimum if and only if the inequality $f(s*) \le f(s)\ \forall s \in S\Omega$ is satisfied. The set of all global optimal solutions is denoted by $S^*\Omega \subseteq S\Omega$ and, therefore, solving a COP requires finding at least one s* such that $s^* \in S^*\Omega$.

In ACO, the population of ants constructs a solution to a COP by traversing (travelling along) a fully connected graph that can be defined as follows. Each decision variable $X_i = v_{ji}$ is called a *solution component* and denoted by c_{ij}. The set of all possible solution components is denoted by C. Then, the COP graph $GC(V,E)$ is defined by assigning the components C to either the set of vertices V or with the set of edges E. Furthermore, a pheromone trail value τ_{ij} that changes at each iteration is associated with each component c_{ij}. The main purpose of the pheromone values is to allow modelling of the probability distribution of different solution components.

Figure 14 shows a general ACO workflow with all steps. Note that there are modifications of ACO that change pheromone matrix and manipulate the heuristic information to achieve faster convergence or to force broader search of the domain.

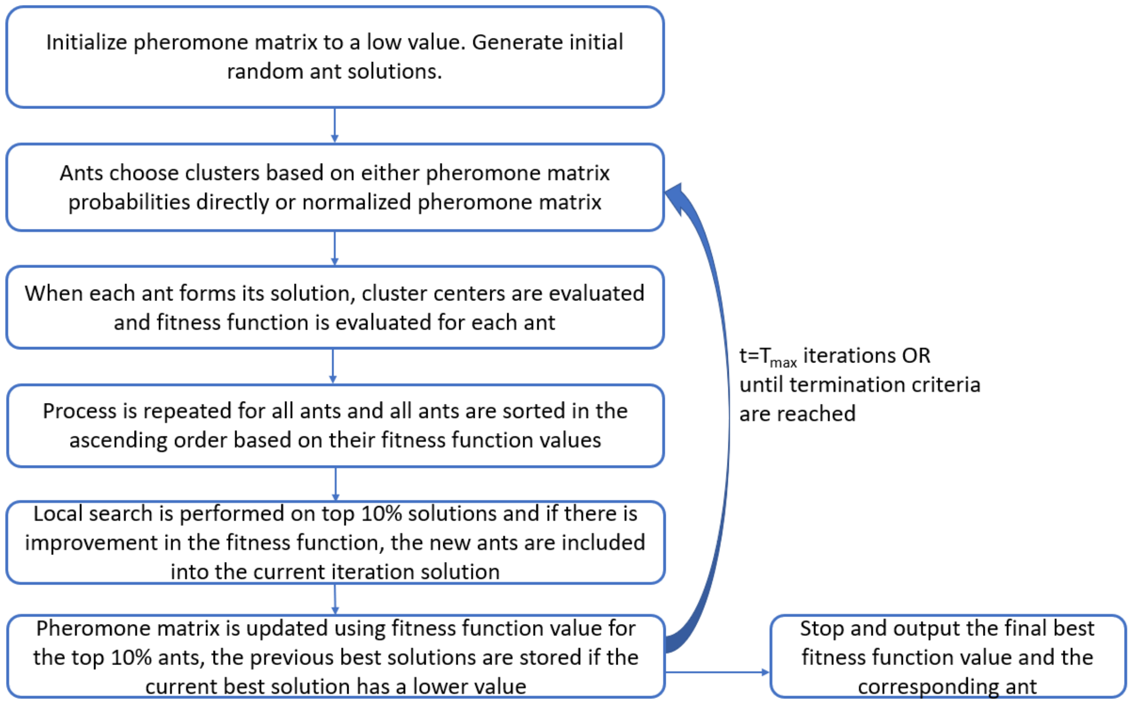
Initialize pheromone matrix to a low value. Generate initial random ant solutions.
Ants choose clusters based on either pheromone matrix probabilities directly or normalized pheromone matrix
When each ant forms its solution, cluster centers are evaluated and fitness function is evaluated for each ant
Process is repeated for all ants and all ants are sorted in the ascending order based on their fitness function values
Local search is performed on top 10% solutions and if there is improvement in the fitness function, the new ants are included into the current iteration solution
Pheromone matrix is updated using fitness function value for the top 10% ants, the previous best solutions are stored if the current best solution has a lower value
t=T_max iterations OR until termination criteria are reached
Stop and output the final best fitness function value and the corresponding ant

Figure 14. General workflow of ACO (adapted from Popa et al., 2012).

Imperialist Competitive Algorithm (ICA)

Imperialist Competitive Algorithm (ICA) is an SI algorithm that takes competitive behavior of countries as its basis to construct solutions. Specifically, ICA models interaction between empires and colonies and the struggle for dominance between them. This method was first introduced by Atashpaz-Gargari and Lucas (2007) and is also an example of a population-based AI algorithm. **Figure 15** shows a general workflow of ICA which starts with generation of the initial population of possible solutions that are represented by countries in this method. Then, these countries are divided into two groups: imperialists (about 10-13% of the entire population) and colonies. Colonies are distributed among the imperialists based on values assigned to each imperialist. These values denote relative strengths of the empires (an empire is an imperialist + colonies) and are pre-set at the initialization step. Mimicking competition among counties in the physical world, in ICA imperialists start competing and capturing more colonies. During ICA iterative process, stronger (higher valued) imperialists try to capture colonies from weaker empires until weaker ones wane out of existence. The ICA workflow stops when the maximum number of iterations has been reached or some other user-specified termination criterion has been encountered.

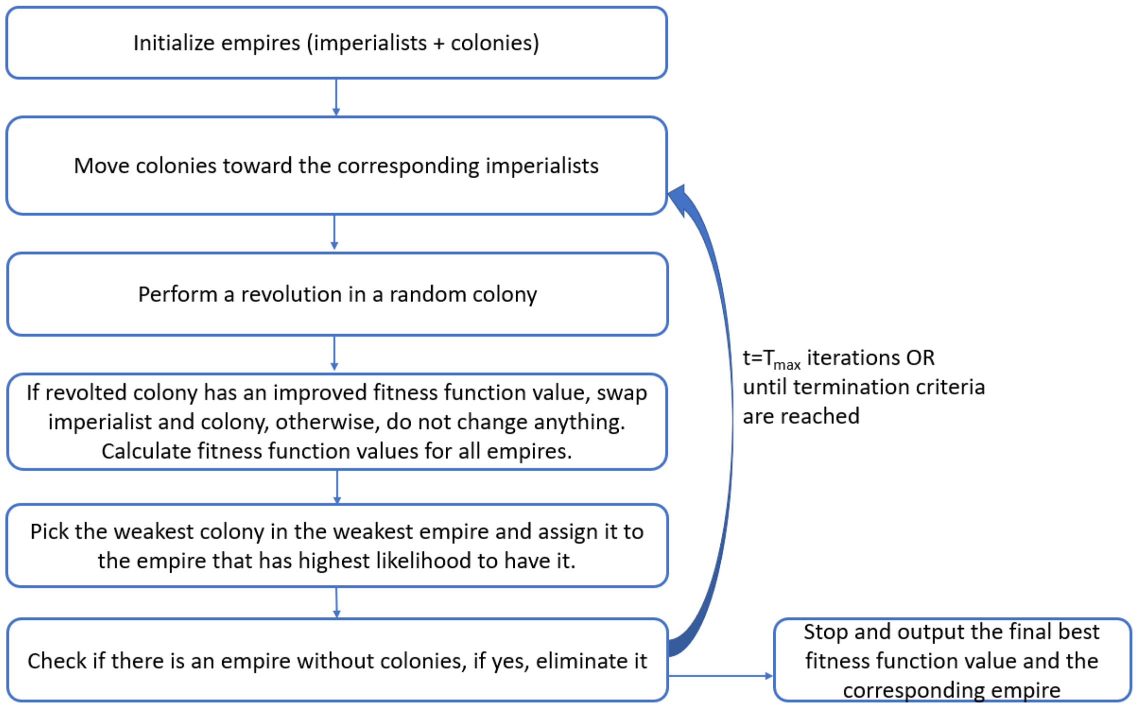
Initialize empires (imperialists + colonies)
Move colonies toward the corresponding imperialists
Perform a revolution in a random colony
If revolted colony has an improved fitness function value, swap imperialist and colony, otherwise, do not change anything. Calculate fitness function values for all empires.
Pick the weakest colony in the weakest empire and assign it to the empire that has highest likelihood to have it.
Check if there is an empire without colonies, if yes, eliminate it
t=T_max iterations OR until termination criteria are reached
Stop and output the final best fitness function value and the corresponding empire

Figure 15. ICA workflow.

The process of finding a nearly optimal solution in ICA involves movement of colonies and their assimilation by empires. Let us consider assimilation process in more detail. After the initialization step, when all solutions (countries) are generated and divided into imperialists and colonies, the colonies start to move toward respective imperialists based on a model that uses two tuning parameters β and θ. Refer to **Figure 16**, to see two type of colony movement allowed by ICA. **Figure 16a** shows movement with the distance x in the direction of the imperialist. Note that x is sampled randomly from the interval $[0, \beta d]$, where d is the distance between the imperialist and the moving colony. **Figure 16b** shows movement with the distance x, but with a deviation angle θ from the straight path between the imperialist and the moving colony. Note that the angle θ is randomly sampled from the interval [-*θ*, *θ*] and measured in radians. This type of movement is useful when an optimization problem has two or more variables being optimized because it extends the searching capabilities of ICA. Specific values for the parameters *β* and *θ* can be tuned based on the nature of the optimization problem. Some suggested values are $\beta = 2$ and $\theta = \pi/4$ rad.

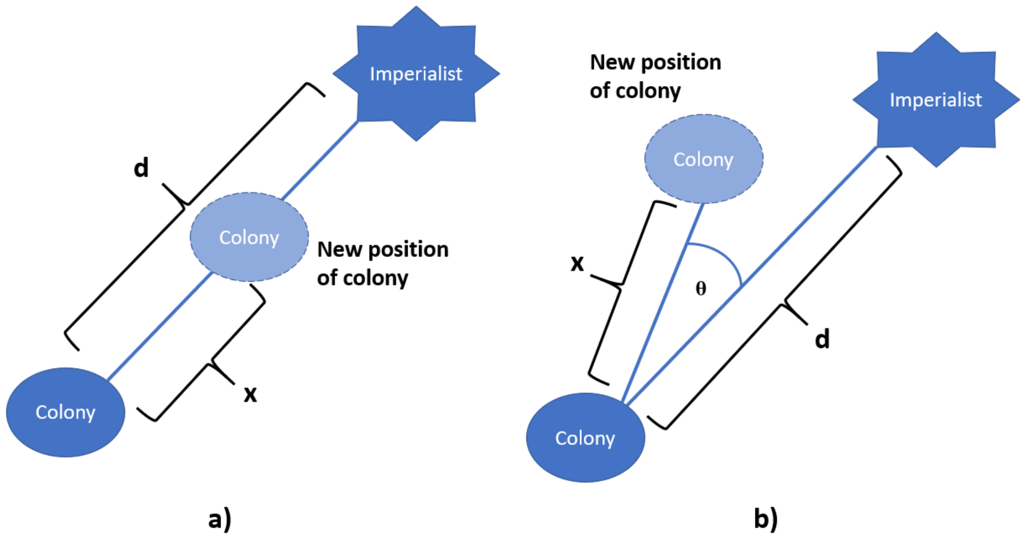

Figure 16. The process of assimilation of colonies in ICA. a) movement along the shortest distance between the imperialist and the colony, b) movement at an angle from the shortest distance between the imperialist and the colony.

The process of revolution is analogous to mutation in GA and signifies a sudden change in a colony's position to prevent fast convergence to some local optimum. Revolution mechanism also encourages the algorithm to keep exploring the search domain. In ICA, the country that is supposed to revolt and move to a new position is chosen randomly. After the revolution is complete, ICA evaluates the fitness function using this new position, and if the value is an improvement in comparison to the function value of the corresponding imperialist, the revolted colony and the imperialist swap their positions.

ICA workflow typically runs until all empires collapse except for one with the best fitness function value. At this step, the algorithm reports the solution and its corresponding fitness function value. However, for the cases when one solution cannot be achieved in finite computational time, it is a good practice to include some additional termination criteria, for example, the maximum number of iterations or absence of the fitness function improvement for certain number of iterations.

The authors of ICA tested it on several most common optimization problems such as TSP and reported good performance of the algorithm. Furthermore, among advantages of ICA the authors highlighted its scalability (dimensionality of the problem can be easily increased) and flexibility (ICA can be hybridized with other algorithms to improve local searches) properties, its applicability to both continuous and discrete optimization problems, weaker dependency of the initialization process, and a relatively low chance of trapping in some local optimum due to the revolution process. Among ICA's disadvantages, one can list the need to tune parameters to a specific problem, tendency to converge prematurely especially in multimodal problems, and the need to modify the algorithm for discrete problems as the original formulation of ICA is only for continuous domain.

Applications

PSO is one the most frequently applied SI algorithms in petroleum engineering because it was particularly well suited for continuous function optimization (Eberhart and Kennedy, 1995), neural network training (Shen et al., 2006), static function optimization (Shi and Eberhart, 1999), dynamic function optimization (Blackwell and Branke, 2004), multimodal optimization (Brits et al., 2002), and data clustering (Cohen and de Castro, 2006). However, wide application of PSO to petroleum related problems started less than a decade ago.

Similarly to GA, PSO has been used for history matching real production data and the output of the corresponding numerical model. For example, Kathrada (2009) adapted PSO for history matching of finite difference simulation models. However, his results show that even a highly efficient modification of PSO such as Flexi-PSO does not give a guarantee that the obtained predictions (oil/gas production forecast) will cover the true reservoir behavior range. This study suggests that perhaps metaheuristic algorithms by themselves (without some modification and hybridization with other methods) should not be used a reliable first line tools for history matching problems.

Some researchers took advantage of PSO to solve petroleum economics problems and used both empirical data and numerically simulated high-resolution data. One such study was conducted by Onwunalu and Durlofsky (2010) who used socioeconomic indicators including population, GDP (gross domestic product), import and export data (empirical data component) and the NPV (that has a simulated data component) as the objective function and applied GA and PSO to determine optimal well type (including vertical, directional, and dual-lateral) and its location. Another petroleum economics problem was solved by Assareh et al. (2010) who developed PSO and GA demand estimation models (PSO-DEM and GA-DEM) in exponential and linear forms to predict future oil demand until 2030. After comparing both algorithms, the authors conclude that the PSO model produces lower average relative errors.

PSO has been also applied to optimize EOR processes using numerical simulators for fast objective function evaluation. For example, Wang and Qiu (2013) used three different modifications of PSO to optimize oil recovery from a heavy oil reservoir and compared their relative performance. The authors concluded that conventional PSO achieved the highest values of the objective function.

Humphries et al. (2014) combined PSO with the generalized pattern search to solve well placement (well location) and control problem. Their study shows that combing these two approaches sequentially (one after another) has better performance than applying both strategies simultaneously.

Another set of EOR studies that employ PSO includes the work by Zhou et al. (2016) who integrated uniform design (UD) into the initialization process of PSO and used this hybrid approach to maximize the NPV of a cyclic steam stimulation project. The authors pointed out that combining PSO with UD at the initialization step improved the quality of the initial population and, therefore, the convergence rate. Moreover, their approach led to more economic and technically feasible scenarios for heavy oil reservoirs development.

Mohagheghian (2016) also used GA and PSO algorithms to optimize hydrocarbon WAG process in the E-segment of the Norne field. He compared the optimized results to the reference cases and concluded that the best overall values of NPV found by GA and PSO were 13.8% and 14.2% higher, respectively. In addition to this, the author observed improvements in incremental recovery factor used as the objective function. He obtained an increase of 14.2% in the case of GA and 16.2% in the case of PSO.

ACO has not gained as much popularity for petroleum engineering applications as PSO, but there are some interesting examples that we can consider. One of the first applications of ACO includes reservoir history matching (which one of the most obvious choices as it is inherently a minimization problem) using an example field from the Gulf of Mexico by Hajizadeh et al. (2009). In this study, the authors matched actual production history and the output of a simulation model using ACO and neighborhood algorithm (NA) and concluded that ACO converged faster with better misfit values than NA. However, NA maintained wider population diversity and, thus, came up with a wider range of uncertainty for the history matched parameters and production forecast.

Popa et al. (2012) applied ACO to a waterflooding optimization problem in a mature field and developed a systematic workflow that can be applied to any field. The authors used data analytics tool and associative data modelling to find field blocks that can be developed using similar strategies (waterflooding patterns) and then applied waterflooding optimization within those blocks. Note that the authors solved the problem from technical perspective only (injection/production) and did not include economic considerations into their objective function which leaves room for additional contribution in the future.

ACO was also used for geophysical applications such as automated fault detection for reservoir characterization (Zhe and Haming, 2012). The authors pointed out that fault tracking had been mostly done manually and was a formidable task for large fields. To identify faults, a geophysicist would need to visually analyze seismic data and input all faults into a geomodel for further flow simulation and production forecasting. The authors proposed and tested orientation constraint ACO to automatically pick faults from the seismic data cube and observed that the algorithm could identify most of the faults that were also picked by manual tracking.

Another interesting application of ACO is in drilling engineering. Jiang and Samuel (2016) optimized the rate of penetration that can be used real-time during drilling operations. The authors combined ACO with artificial neural network (ANN) to predict ROP and estimate bit life. ANN was used to create ROP that would normally be acquired in the field in real time during drilling operations, while ACO was used to optimize ROP and predict bit life span.

Unlike PSO and ACO, ICA has not been widely used in petroleum engineering because of its relative novelty and the need to tune parameters to match specific optimization problems. Among one of the most prominent applications of ICA is optimization of well placement for field development by Al Dossary and Nasrabadi (2015). The authors considered three scenarios (a vertical well placed in a reservoir with heterogeneously distributed permeability, a vertical well placed in a channeled reservoir, and a horizontal well placed in a channeled reservoir) and compared performance of ICA with that of GA. In addition to this, they calibrated the results using the exhaustive search (the size of the problem allowed trying every possible location for vertical wells and establishing the true global optimum) and arrived at a conclusion that ICA had better performance (it terms of computational time efficiency and ability to achieve a nearly optimal solution) than GA for the well placement problem.

In the next chapter, you will be introduced to the basics of fuzzy logic and its hybridization with other algorithms and learn more about its data analytics applications in petroleum engineering.

References

Abraham, A., Guo, H., and Liu, H. (2006). Swarm intelligence: foundations, perspectives and applications. Swarm Intelligent Systems, Springer: 3-25.

Ab Wahab, M., Nefti-Meziani, S., and Atyabi, A. (2015). A comprehensive review of swarm optimization algorithms. PloS one **10** (5): e0122827.

Alam, S., Dobbie, G., Koh, Y., et al. (2014). Research on particle swarm optimization based clustering: a systematic review of literature and techniques. Swarm and Evolutionary Computation **17**: 1-13.

Assareh, E., Behrang, M., Assari, M. et al. (2010). Application of PSO (particle swarm optimization) and GA (genetic algorithm) techniques on demand estimation of oil in Iran. Energy **35** (12): 5223-5229.

Al Dossary, M. and Nasrabadi, H. (2015). Well placement optimization using imperialist competition algorithm. SPE Reservoir Characterisation and Simulation Conference and Exhibition, Abu Dhabi.

Atashpaz-Gargari, E. and Lucas, C. (2007). Imperialist competitive algorithm: an algorithm for optimization inspired by imperialist competition. IEEE Congress on Evolutionary Computation, 4661-4667.

Atyabi, A. and Powers, D. (2013). Cooperative area extension of PSO-transfer learning vs. uncertainty in a simulated swarm robotics. ICINCO (1).

Blackwell, T. and Branke, J. (2004). Multi-swarm optimization in dynamic environments. Workshops on Applications of Evolutionary Computation, Springer.

Bonabeau, E. and Meyer, C. (2001). Swarm intelligence: a whole new way to think about business. Harvard Business Review **79** (5): 106-115.

Brits, R., Engelbrecht, A., and Van den Bergh, F. (2002). A niching particle swarm optimizer. Proceedings of the 4th Asia-Pacific conference on simulated evolution and learning, Singapore: Orchid Country Club.

Chang, Y. and Yu, G. (2013). Multi-sub-swarm PSO classifier design and rule extraction.

Cohen, S. and De Castro, L. (2006). Data clustering with particle swarms. 2006 IEEE International Conference on Evolutionary Computation, IEEE.

Dorigo, M., Maniezzo., V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B, **26** (1): 29-41.

Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. Proceedings of the sixth international symposium on micro machine and human science, New York, NY.

Edelen, M. (2003). Swarm intelligence and stigmergy: robotic implementation of foraging behavior.

Engelbrecht, A. (2006). Fundamentals of computational swarm intelligence, John Wiley & Sons.

Ganesan, T., Elamvazuthi, I.. Shaari, K. et al. (2013). Swarm intelligence and gravitational search algorithm for multi-objective optimization of synthesis gas production. Applied Energy **103**: 368-374.

Hajizadeh, Y., Christie, M., and Demyanov, V. (2009). Ant colony for history matching. SPE EUROPEC/EAGE Annual Conference and Exhibition, Amsterdam.

Humphries, T., Haynes, R., and James, L. (2014). Simultaneous and sequential approaches to joint optimization of well placement and control. Computational Geosciences **18** (3-4): 433-448.

Jiang, W. and Samuel, R. (2016). Optimization of rate of penetration in a convoluted drilling framework sing ant colony optimization. IADC/SPE Drilling Conference and Exhibition, Fort Worth.

Kaewkamnerdpong, B. and Bentley, P. (2005). Perceptive particle swarm optimisation: an investigation. Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005., IEEE.

Kathrada, M. (2009). Uncertainty evaluation of reservoir simulation models using particle swarms and hierarchical clustering, Heriot-Watt University.

Kamrani, E. (2010). Modeling and forecasting long-term natural gas (NG) consumption in Iran, using particle swarm optimization (PSO).

Mirzabozorg, A. (2015). Incorporation of engineering knowledge in history matching, optimization, and uncertainty assessment frameworks with application to the SAGD process, University of Calgary.

Mohagheghian, E. (2016). An application of evolutionary algorithms for WAG optimisation in the Norne Field, Memorial University of Newfoundland.

Mohamed, L., Christie, M., and Demyanov, V. (2011). History matching and uncertainty quantification: multiobjective particle swarm optimisation approach. SPE EUROPEC/EAGE Annual Conference and Exhibition.

Onwunalu, J. and Durlofsky, L. (2010). Application of a particle swarm optimization algorithm for determining optimum well location and type. Computational Geosciences **14** (1): 183-198.

Popa, A., Sivakumar, K., and Cassidy, S. (2012). Associative data modeling and ant colony optimization approach for waterflood analysis. SPE Western Regional Meeting, Bakersfield, CA.

Senthilkumar, S. (2014). Practical applications of swarm intelligence and evolutionary computation, hybrid soft computing. International Journal of Swarm Intelligence and Evolutionary Computation.

Shen, Q., Shi, W., Yang, X.et al. (2006). Particle swarm algorithm trained neural network for QSAR studies of inhibitors of platelet-derived growth factor receptor phosphorylation. European Journal of Pharmaceutical Sciences **28** (5): 369-376.

Shi, Y. and Eberhart, R. (1998). Parameter selection in particle swarm optimization. Annual Conference on Evolutionary Programming, San Diego.

Wang, X. and Qiu, X. (2013). Application of particle swarm optimization for enhanced cyclic steam stimulation in a offshore heavy oil reservoir. arXiv preprint arXiv:1306.4092.

Zhe, Y. and Hanming, G. (2012). An automatic fault tracking approach based on ant colony algorithm. SEG Annual Meeting, Las Vegas.

Zhou, Q., Wu, W., Liu, D. et al. (2016). Estimation of corrosion failure likelihood of oil and gas pipeline based on fuzzy logic approach. Engineering Failure Analysis **70**: 48-55.