Advances in Chemical Engineering and Science, 2013, 3, 30-36
http://dx.doi.org/10.4236/aces.2013.34A1005 Published Online October 2013 (http://www.scirp.org/journal/aces)

A Comparison between Evolutionary Metaheuristics and Mathematical Optimization to Solve the Wells Placement Problem

Ghazi AlQahtani, Ahmed Alzahabi, Ernee Kozyreff, Ismael R. de Farias Jr., Mohamed Soliman
Texas Tech University, Lubbock, USA
Email: ghazi.alqahtani@ttu.edu, ahmed.alzahabi@ttu.edu, ernee.kozyreff@ttu.edu, ismael.de-farias@ttu.edu, mohamed.soliman@ttu.edu

Received August 17, 2013; revised September 20, 2013; accepted October 4, 2013

ABSTRACT

The Wells Placement Problem (WPP) consists in choosing well locations within an oil reservoir grid to maximize the reservoir total oil production, subject to distance threshold between wells and number of wells cap constraints. A popular approach to WPP is Genetic Algorithms (GA). Alternatively, WPP has been approached in the literature through Mathematical Optimization. Here, we conduct a computational study of both methods and compare their solutions and performance. Our results indicate that, while GA can provide near-optimal solutions to instances of WPP, typically Mathematical Optimization provides better solutions within less computational time.

Keywords: Wells Placement; Genetic Algorithm; Integer Programming

1. Introduction

Consider an oil reservoir for which oil production totals over the entire reservoir area are given (or estimated). We are interested in the Wells Placement Problem (WPP), *i.e.*, the problem of maximizing the total oil production of the reservoir by selecting locations to build at most a certain number of vertical wells at a given minimum distance apart from one another.

WPP (and variations of it) has been the subject of a number of studies in the last decades [1-9]. The main approaches used to find solutions have been Genetic Algorithms (GA), which are a special type of Evolutionary Metaheuristics, and Integer Programming (IP), a subclass of Mathematical Optimization.

To the best of our knowledge, the earliest study of wells placement optimization using IP was the work conducted in 1974 by Rosenwald and Green [8], who developed a numerical optimization framework to select optimal positions of wells. Their objective was to minimize the difference between actual and desired flow curves by choosing optimum well locations from predefined location sets. In 1997, Vasantharajan and Cullick [9] introduced the term *reservoir quality* to represent reservoir connectivity adjusted by tortuosity, which was the surface area surrounding the flow by the volume contained. The well site selection process was performed through IP, with the goal of maximizing reservoir quality. Later, in 1999, Cullick *et al*. [4] improved the formulation of the problem by reducing its number of constraints.

On the other side, GA is the most widely used method to solve WPP optimization [1]. Guyaguler *et al*. [6] developed a hybrid method of GA linked with polytope search and coupled with a proxy generated by kriging. The objective function in this work was to maximize net present value by finding the optimum injection well locations. The output of this work indicates that using kriging as a proxy with the hybrid optimization framework reduced the computational expense considerably. This work was extended by Badru and Kabir [2] by including horizontal wells and linking the hybrid GA with experimental design to find the impact of uncertainty on recovery and the number of wells required. In 2006, Bangerth *et al*. [3] compared GA with four algorithms designed to solve WPP. GA was also compared with Covariance Matrix Adaptation-Evolution Strategy by Ding [5] to optimize well locations for different types of wells with calculated productivity index as the objective function. Onwunalu *et al*. [7] used GA combined with statistical proxy to establish a relationship between the variables and the objective function using unsupervised learning techniques.

Some of the articles cited above report computational results using either GA or IP, but we are not aware of a study that compares both. In this paper, we use GA and IP to solve instances of WPP, and report the computational results of those tests.

Specifically, our test reservoir sits on an $m \times n$ grid, where each of the mn sites in the grid represents a possible location for a well. For each $(i, j) \in I \times J$, where $I = \{1, \cdots, m\}$ and $J = \{1, \cdots, n\}$, a production total $p_{(i,j)}$ is associated with the site, and each pair of sites chosen needs to be at least D grid units apart from each other. Finally, the maximum number of well locations selected, N, is predetermined.

2. Evolutionary Metaheuristics

Evolutionary Metaheuristics use ideas of biology and Darwin's theory of evolution, such as reproduction, mutation, and "survival of the fittest", to provide solutions to problems that arise in other fields of study. Because one of the most used types of evolutionary metaheuristics in petroleum engineering is genetic algorithms, we developed a GA to tackle our problem of interest (for details on the structure and use of GA in general, see [10]).

2.1. Definition of Terms and Variables

Our GA starts with an *initial population* that is improved through *crossover*, *local search*, *mutation*, and the addition of new members to the population, which we call *intruders*. After a generation, the best fit individuals of the population are selected, the others are discarded, and a new generation begins. The algorithm runs for a predetermined number of generations, n_g, and returns the fittest individuals of the final population. The fitness of an individual, here, is the total oil production associated with it.

An individual y^z is represented by an N-array of ordered pairs (which we call *coordinates*), and denoted as

$$y^z = \left((i_1, j_1)^z, (i_2, j_2)^z, \cdots, (i_N, j_N)^z\right)$$

where $(i_N, j_N)^z$, $k \in \{1, \cdots, N\}$, represent the locations on the grid of the wells defined by y^z. Associated with y^z, the value p_z is the total oil production that results from choosing this individual, given by Equation (1).

$$p_z = \sum_{k=1}^{N} p_{(i_k, j_k)^z} \tag{1}$$

We denote the $m \times n$ production totals matrix by P, *i.e.*, an element of row i and column j of p is $p_{(i,j)}$, and we define $p_{(0,0)} = 0$ We let q be the sorted vector of the elements of P and its corresponding coordinates:

$$q = \left(\left(q_1, (r_1, s_1)\right), \left(q_2, (r_2, s_2)\right), \cdots, \left(q_{mn}, (r_{mn}, s_{mn})\right)\right)$$

where

$$q_1 \geq q_2 \geq \cdots \geq q_{mn},$$

$$q_k = p_{(r_k, s_k)},$$

and

$$\left\{(r_1, s_1), \cdots, (r_{mn}, s_{mn})\right\} = I \times J.$$

2.2. The Genetic Algorithm

The high level structure of our GA is as follows:

```
WPP-GA(P, m, n, D, N, n_g, n_p, n_c, n_i, Δ)
1  for z := 1 to n_p + 2n_c + n_i do
2  |  for k := 1 to N do
3  |  |  (i_k, j_k)^z := (0, 0);
4  |  end
5  end
6  INITIAL-POPULATION;
7  for g := 1 to n_g do
8  |  CROSSOVER;
9  |  LOCAL-SEARCH;
10 |  MUTATION;
11 |  SELECTION;
12 end
13 return y^1;
```

At each generation g, n_p is the number of individuals in the initial population, n_c is the number of crossover operations performed, and n_i is the number of intruders created. The value Δ is used in the sub-routine **LOCAL-SEARCH**, and will be explained later.

Lines 1 - 5 initialize the variables of the algorithm, which are the individuals of the population, at zero. Line 6 creates the initial population, and lines 7 - 12 perform evolutionary sub-routines for n_g generations. Finally, line 13 returns the best fit individual found.

Throughout the algorithm, and for all individuals considered, it is verified whether the distance of all locations in the grid associated with the individual satisfy the minimum distance constraint. When it does, we say that the individual is *feasible*. This verification is done using the following sub-routine.

```
CHECK-FEASIBILITY(y^t)
1  f := 1;
2  for k := 2 to N do
3  |  if (i_k, j_k)^t = (0, 0) then
4  |  |  continue;
5  |  end
6  |  (i_1, j_1) := (i_k, j_k)^t;
7  |  for ℓ := 1 to k − 1 do
8  |  |  if (i_ℓ, j_ℓ)^t = (0, 0) then
9  |  |  |  continue;
10 |  |  end
11 |  |  (i_2, j_2) := (i_ℓ, j_ℓ)^t;
12 |  |  if √((i_1 − i_2)^2 + (j_1 − j_2)^2) < D then
13 |  |  |  f := 0;
14 |  |  |  break;
15 |  |  end
16 |  end
17 |  if f = 0 then
18 |  |  break;
19 |  end
20 end
21 return f;
```


This routine takes as input an individual, y^t, and returns 1 if the individual is feasible, or 0 otherwise. The procedure simply computes the Euclidean distance between every pair of coordinates that define the individual, and checks whether the distance is at least D.

```
INITIAL-POPULATION(P, m, n, D, N, n_p)
 1  (i_1, j_1)^1 := (r_1, s_1);
 2  ℓ := 2;
 3  for k := 2 to mn do
 4  |  y^0 := ((i_1, j_1)^1, ..., (i_{ℓ-1}, j_{ℓ-1})^1, (r_k, s_k),
    |        (0,0), ..., (0,0));
 5  |  if CHECK-FEASIBILITY(y^0) = 1 then
 6  |  |  (i_ℓ, j_ℓ)^1 := (r_k, s_k);
 7  |  |  ℓ := ℓ + 1;
 8  |  end
 9  |  if ℓ > N then
10  |  |  break;
11  |  end
12  end
13  for z := 2 to n_p do
14  |  (i_1, j_1)^z := (rand({1, ..., m}), rand({1, ..., n}));
15  |  ℓ := 2;
16  |  for k := 1 to 10N do
17  |  |  i* := rand({1, ..., m});
18  |  |  j* := rand({1, ..., n});
19  |  |  y^0 := ((i_1, j_1)^z, ..., (i_{ℓ-1}, j_{ℓ-1})^z, (i*, j*),
    |  |        (0,0), ..., (0,0));
20  |  |  if CHECK-FEASIBILITY(y^0) = 1 then
21  |  |  |  (i_ℓ, j_ℓ)^z := (i*, j*);
22  |  |  |  ℓ := ℓ + 1;
23  |  |  end
24  |  |  if ℓ > N then
25  |  |  |  break;
26  |  |  end
27  |  end
28  end
29  return y^1, ..., y^{n_p};
```

The routine **INITIAL-POPULATION** can be divided into two main blocks: lines 1 - 12 and 13 - 29. The first block creates a "greedy" solution to the problem by choosing the coordinates of y^1, the first individual of the population, based on the vector q of sorted values of P. The second block creates the rest of the initial population by selecting random locations on the grid (using the function rand(A), which returns a random element of a finite set A), with the condition that each individual created is feasible.

We note that, for large values of N, it may be hard, and perhaps even impossible, to find N locations on the grid such that the minimum distance between each pair of locations is D. Our algorithm makes $10N$ attempts to find such locations, and, in the case that it is not successful, some of the coordinates of the individual being created remain as $(0,0)$.

The sub-routine **CROSSOVER** performs the reproduction of individuals. In this routine, two individuals of the population $y^1, \cdots, y^{n_p}$ are selected at random (lines 2 - 3), and the crossover point, represented by d, is also selected at random (line 4). In lines 5 - 12, the crossover occurs, and two new individuals are created. These new individuals are then checked for feasibility, and, in the case that they are not feasible, they are "erased" from the population by setting all of their coordinates to $(0,0)$ (lines 13 - 21). The procedure is repeated n_c times, and the routine creates $2n_c$ new individuals.

```
CROSSOVER(P, D, N, n_p, n_c, y^1, ..., y^{n_p})
 1  for k := 1 to n_c do
 2  |  a_1 := rand({1, ..., n_p});
 3  |  a_2 := rand({1, ..., n_p} \ {a_1});
 4  |  d := rand({1, ..., N});
 5  |  for ℓ := 1 to d do
 6  |  |  (i_ℓ, j_ℓ)^{2k-1} := (i_ℓ, j_ℓ)^{a_1};
 7  |  |  (i_ℓ, j_ℓ)^{2k} := (i_ℓ, j_ℓ)^{a_2};
 8  |  end
 9  |  for ℓ := d + 1 to N do
10  |  |  (i_ℓ, j_ℓ)^{2k-1} := (i_ℓ, j_ℓ)^{a_2};
11  |  |  (i_ℓ, j_ℓ)^{2k} := (i_ℓ, j_ℓ)^{a_1};
12  |  end
13  |  if CHECK-FEASIBILITY(y^{2k-1}) = 0 then
14  |  |  for ℓ := 1 to N do
15  |  |  |  (i_ℓ, j_ℓ)^{2k-1} := (0,0);
16  |  |  end
17  |  end
18  |  if CHECK-FEASIBILITY(y^{2k}) = 0 then
19  |  |  for ℓ := 1 to N do
20  |  |  |  (i_ℓ, j_ℓ)^{2k} := (0,0);
21  |  |  end
22  |  end
23  end
24  return y^{n_p+1}, ..., y^{n_p+2n_c};
```

Our GA then performs a local search in the grid, in the geographical sense, to try to improve every individual of the population.

```
LOCAL-SEARCH(P, D, N, n_p, n_c, Δ, y^1, ..., y^{n_p+2n_c})
 1  for z := 1 to n_p + 2n_c do
 2  |  ℓ := rand({1, ..., N});
 3  |  if (i_ℓ, j_ℓ)^z = (0,0) then
 4  |  |  continue;
 5  |  end
 6  |  (i*, j*) := (i_ℓ, j_ℓ)^z;
 7  |  b_1 := 0;
 8  |  b_2 := 0;
 9  |  for d_1 := -Δ to Δ do
10  |  |  if i* + d_1 < 1 or i* + d_1 > m then
11  |  |  |  continue;
12  |  |  end
13  |  |  for d_2 := -Δ to Δ do
14  |  |  |  if j* + d_2 < 1 or j* + d_2 > n then
15  |  |  |  |  continue;
16  |  |  |  end
17  |  |  |  if p_{(i*+d_1, j*+d_2)} ≤ p_{(i*+b_1, j*+b_2)} then
18  |  |  |  |  continue;
19  |  |  |  end
20  |  |  |  y^0 := ((i_1, j_1)^z, ..., (i_{ℓ-1}, j_{ℓ-1})^z, (i*+d_1, j*+d_2),
    |  |  |        (i_{ℓ+1}, j_{ℓ+1})^z, ..., (i_N, j_N)^z);
21  |  |  |  if CHECK-FEASIBILITY(y^0) = 1 then
22  |  |  |  |  b_1 := d_1;
23  |  |  |  |  b_2 := d_2;
24  |  |  |  end
25  |  |  end
26  |  end
27  |  (i_ℓ, j_ℓ)^z := (i* + b_1, j* + b_2);
28  end
29  return y^1, ..., y^{n_p+2n_c};
```

In line 2, a coordinate ℓ of individual y^z is chosen

at random, and, in case it is not $(0,0)$, a local search in the reservoir grid, within a square of area $(2\Delta+1)^2$ centered at $(i_\ell, j_\ell)^z$, is performed to try to find a location in the search area such that, if substituted in place of $(i_\ell, j_\ell)^z$, will maintain feasibility of the individual and increase its associated p_z value the most. The routine performs this local search for $y^1,\cdots,y^{n_p+2n_c}$, and returns their updated values.

The next genetic operator is mutation, which, again, is performed for every individual of the population $y^1,\cdots,y^{n_p+2n_c}$.

MUTATION$(P, D, N, n_p, n_c, y^1, \ldots, y^{n_p+2n_c})$

```
1  for z := 1 to n_p + 2n_c do
2  |  ℓ := rand({1, ..., N});
3  |  i* := rand({1, ..., m});
4  |  j* := rand({1, ..., n});
5  |  if p_(i*,j*) ≤ p_(i_ℓ,j_ℓ)^z then
6  |  |  continue;
7  |  end
8  |  y^0 := ((i_1, j_1)^z, ..., (i_ℓ−1, j_ℓ−1)^z, (i*, j*),
   |         (i_ℓ+1, j_ℓ+1)^z, ..., (i_N, j_N)^z);
9  |  if CHECK-FEASIBILITY(y^0) = 1 then
10 |  |  (i_ℓ, j_ℓ)^z := (i*, j*);
11 |  end
12 end
13 return y^1, ..., y^(n_p+2n_c);
```

This routine picks a coordinate ℓ from individual y^z at random and replaces it with another coordinate picked at random in the entire grid. If the replacement keeps y^z feasible and improves the value of p_z, then y^z is updated; otherwise, y^z remains unchanged.

Our last genetic operator, **INTRUDERS**, creates n_i individuals that enter the population "from outside". The intention of this procedure is to add variability to the population thus far created.

INTRUDERS(P, D, N, n_p, n_c, n_i)

```
1  for z := n_p + 2n_c + 1 to n_p + 2n_c + n_i do
2  |  u := rand({1, ..., min{10N, mn}});
3  |  (i_1, j_1)^z := (r_u, s_u);
4  |  ℓ := 2;
5  |  for k := 1 to 100N do
6  |  |  u := rand({1, ..., min{10N, mn}});
7  |  |  i* := r_u;
8  |  |  j* := s_u;
9  |  |  y^0 := ((i_1, j_1)^z, ..., (i_ℓ−1, j_ℓ−1)^z, (i*, j*),
   |  |         (0,0), ..., (0,0));
10 |  |  if CHECK-FEASIBILITY(y^0) = 1 then
11 |  |  |  (i_ℓ, j_ℓ)^z := (i*, j*);
12 |  |  |  ℓ := ℓ + 1;
13 |  |  end
14 |  |  if ℓ > N then
15 |  |  |  break;
16 |  |  end
17 |  end
18 end
19 return y^(n_p+2n_c+1), ..., y^(n_p+2n_c+n_i);
```

This routine is similar to **INITIAL-POPULATION**. The main differences are in lines 2 and 6, where the choice of the coordinates of the new individuals is more selective: they are chosen among the top $10N$ values of the P matrix. Thus, this routine brings to the population not only variability, but also members with high fitness.

Our final sub-routine selects and preserves the n_p best fit individuals from the entire population $y^1,\cdots,y^{n_p+2n_c+n_i}$, which will become the initial population of the next generation.

SELECTION$(P, y^1, \ldots, y^{n_p+2n_c+n_i})$

```
1  sort y^1, ..., y^(n_p+2n_c+n_i) in decreasing order of
   p_z, z ∈ {1, ..., n_p + 2n_c + n_i}
2  for z := n_p + 1 to n_p + 2n_c + n_i do
3  |  for k := 1 to N do
4  |  |  (i_k, j_k)^z := (0, 0);
5  |  end
6  end
7  return y^1, ..., y^(n_p+2n_c+n_i);
```

The sorting procedure in line 1 is, in fact, a renumbering of the individuals $y^1,\cdots,y^{n_p+2n_c+n_i}$ based on the values p_z, $z\in\{1,\cdots,n_p+2n_c+n_i\}$. As a clarifying example, suppose we give as input to this line $y^1=((1,2),(3,4))$, $y^2=((5,6),(7,8))$, and $y^3=((9,10),(11,12))$, with $p_1=10$, $p_2=30$, and $p_3=20$. Then the output will be $y^1=((5,6),(7,8))$, $y^2=((9,10),(11,12))$, and $y^3=((1,2),(3,4))$. Lines 2 - 7 simply erase variables $y^{n_p+1},\cdots,y^{n_p+2n_c+n_i}$ and keep $y^1,\cdots,y^{n_p}$ which are the surviving individuals that will start a new generation. (In case $g=n_g$, then there will not be another generation, and **WPP-GA** will return y_1, the individual with the best fitness found after n_g generations.)

A careful analysis of **WPP-GA** and its sub-routines shows that the algorithm terminates, but not necessarily with an optimal solution to the problem, *i.e.*, an individual whose fitness is the highest possible.

3. Mathematical Optimization

Mathematical Optimization approaches quantitative problems using tools such as linear algebra, calculus, and graph theory. In a sense, it is a more sophisticated method than Evolutionary Metaheuristics, and, in practice, usually requires bigger and more structured algorithms to solve a given problem.

The main advantage of using mathematical optimization, and, in our particular case, IP, is that a solution may be *proven* to be optimal, as opposed to GA, which does not guarantee optimality of the best solution found.

It is common practice, in IP, to formulate problems by defining an objective function to be maximized (or minimized), subject to constraints that define the problem in question. Here, the formulation of WPP is as follow:

$$\text{maximize} \sum_{(i,j)\in I\times J} p_{(i,j)} x_{(i,j)},$$

$$\text{subject to } x_{(i_1,j_1)} + x_{(i_1,j_2)} \le 1, \begin{cases} (i_1,j_1),(i_2,j_2) \in I\times J \\ \sqrt{(i_1-i_2)^2+(j_1-j_2)^2} < D \end{cases}$$

$$x_{(i,j)} \in \{0,1\} \qquad (i,j) \in I\times J,$$

where a variable $x_{(i,j)}$ represents the location (i,j) in the oil reservoir grid, and is equal to 1 if the location is chosen, or 0 otherwise. Hence, a feasible solution to this problem is a mn-vector of 0's and 1's that satisfies all constraints defined in the formulation shown. Note that this is a completely different way of representing a solution, in comparison with our GA. Nonetheless, the values of the objective function, here, and the fitness of the individuals, in the GA, are comparable, since they have the same meaning.

To solve our instances with the IP approach, we use a commercial solver based on *branch-and-cut* that can handle large-scale optimization problems. For details on branch-and-cut, the reader is referred to [11].

4. Computational Tests

We ran our computational tests in the Texas Tech High Performance Computing Center's 3.0 GHz CPUs with 16 GB RAM nodes. For the tests using IP, we used CPLEX 12 with default settings, except for the search strategy, which was set to Traditional Branch-and-Cut, and the relative and absolute gap tolerances, which were set to zero. Every test had a time limit of 3600 seconds.

Our test grid has $m = n = 100$, and we set the values of D to $8, 12, 16$, and 20. For each of these values, we tried to solve WPP with N set to $10, 20, \cdots$, up to the point at which either the time limit was reached, or the value of N was *not* reached, *i.e.*, neither method was able to find a solution that represented N well locations. For the GA, we set $n_g = 10000$, $n_p = 1000$, $n_c = 100$, $n_i = 10$, and $\Delta = 10$ for all tests.

The values of the P matrix were obtained with the commercial reservoir simulator Eclipse, using the *Quality Maps* approach (see [12] for details). Some main characteristics of our heterogeneous and anisotropic reservoir are: initial pressure of 4000 *psi*, porosity of 22%, and horizontal permeability average of 175 *mD* with standard deviation of 91.1 *mD*. The distance between two closest grid points is 300 *ft*, and the thickness of the reservoir is 75 *ft*.

Table 1 shows the results obtained using both GA and IP. The columns "Best Sol' n" displays the value of the best solutions found by both methods. (Note: in GA, this value is called the *fitness* of the best individual, whereas in IP it is called the *objective function value* of the best solution. We will use the latter expression in our analysis.) "Time" shows the computational time, in seconds, required for the test to finish. We note that, because of the settings used in CPLEX, all solutions obtained using IP are provably optimal precisely when the time is less than 3600 seconds. The column "Card" shows the cardinality of the solution found, *i.e.*, the number of well locations associated with that solution. Finally, "Gap" refers to the relative percentage difference between the GA and IP best solutions, and is computed using Equation (2).

$$\text{Gap} = \frac{\text{Best Sol'n with IP} - \text{Best Sol'n with GA}}{\text{Best Sol'n with IP}} \cdot 100 \tag{2}$$

From **Table 1**, it is clear that the problem gets harder as the values of D and N increase. On the IP section of the table, this is reflected in the time required to solve the instances. On the GA side, both the computational time and the cardinality of the best solution show the increase in the difficulty of the problem. For instance, for $D = 8$ and $N \ge 120$, the best solution that GA can find has cardinality 117, although solutions with greater cardinality (and better objective function value) do exist, as the IP results show.

Overall, IP was capable of finding better solutions than GA, and, most of the time, was faster. In every test, the best solution found by IP had an objective function value at least as good as the one found by the GA. In 8 cases, both methods found an optimal solution to the problem, and in 11 other cases, the gap between the two methods was less than 1%. This shows that GA can be effective for instances that are less challenging, but, for the harder instances, IP was notably more successful than GA.

The merit of being faster and finding better solutions is due not only to the differences in approaching the problem (IP vs. GA), but also a result of the algorithms used to solve the instances. While our GA has less than 500 lines of code, CPLEX is the product of professional software developers who worked on it for decades. The seemingly small values in the last columns reflect this contrast of algorithms: a gap as small as 0.1% can be very difficult to close, and thus finding a sub-optimal solution can be a much easier task than finding an optimal one. Moreover, the fact that CPLEX can guarantee the optimality of a solution is a feature that a GA does not have. It is only in comparison studies such as the present study, that one can visualize the effectiveness of a GA, in finding optimal solutions. Finally, we note that CPLEX is a multi-purpose solver that can handle a vast number of different problems, while our GA was developed specifically to tackle WPP.

5. Conclusions and Further Research

In this study, we compared the performance of GA and IP to solve instances of the problem of placing vertical wells in an $m \times n$ grid that sits on a reservoir, with the

Table 1. Summary of results using GA and IP.

D	N	GA			IP			Gap
		Best Sol'n	Time	Card	Best Sol'n	Time	Card	
8	10	55,187,158	112	10	55,187,158	33	10	0.00
	20	68,445,523	122	20	68,568,163	43	20	0.18
	30	77,462,025	124	30	77,462,025	48	30	0.00
	40	85,106,849	142	40	85,112,969	49	40	0.01
	50	92,306,494	150	50	92,306,494	54	50	0.00
	60	99,004,082	208	60	99,132,004	52	60	0.13
	70	104,894,614	191	70	105,383,497	57	70	0.46
	80	110,598,373	263	80	111,275,555	54	80	0.61
	90	115,751,327	320	90	116,836,492	74	90	0.93
	100	120,097,642	472	100	121,998,756	76	100	1.56
	110	121,859,490	568	110	126,733,273	97	110	3.85
	120	124,017,004	690	117	130,778,001	149	120	5.17
	130	124,017,004	762	117	134,144,399	284	130	7.55
	140	124,017,004	859	117	136,147,901	3,600	140	8.91
12	10	47,495,649	111	10	47,495,649	106	10	0.00
	20	56,566,624	129	20	56,566,624	130	20	0.00
	30	63,899,845	139	30	63,928,663	129	30	0.05
	40	70,334,844	182	40	70,392,256	141	40	0.08
	50	75,637,819	214	50	76,120,456	181	50	0.63
	60	77,731,339	356	56	80,639,104	263	60	3.61
	70	77,064,512	422	56	82,588,104	3,600	68	6.69
16	10	40,836,652	115	10	40,836,652	183	10	0.00
	20	48,708,586	126	20	48,708,586	243	20	0.00
	30	54,792,273	150	30	54,803,004	322	30	0.02
	40	56,570,909	240	34	58,673,867	1,534	40	3.58
	50	56,758,783	326	35	58,664,640	3,600	40	3.25
20	10	36,921,362	113	10	36,921,362	363	10	0.00
	20	43,793,101	135	20	43,829,509	333	20	0.08
	30	45,037,857	229	23	46,605,956	1,092	28	3.36

objective of extracting the maximum amount of oil from the reservoir. Our results indicate that GA can be effective for easier instances, but it lacks performance for harder ones. In comparison, CPLEX (which takes the IP approach) always found a solution at least as good as our GA, and most of the time faster. Moreover, IP has the advantage of finding provably optimal solutions, while GA is not able to guarantee that a solution is optimal.

The GA presented here can, of course, be modified and have its performance enhanced, either by actually changing the algorithm, or by simply adjusting its parameters. Similarly, the solution via IP can be made faster by considering different formulations of the problem and tuning some parameters on CPLEX. To maintain both approaches simplely, we chose not to vary those parameters, to provide a basic GA, and to use a straightforward formulation for our IP approach.

As further research, we suggest the study of *non-conventional* wells placement, a more challenging problem, since it involves wells that are not necessarily vertical, and thus may require a 3D-grid as the set of possible locations for the wells.

6. Acknowledgements

The authors acknowledge the High Performance Computing Center at Texas Tech University at Lubbock for providing resources that have contributed to the research results reported within this paper.
URL: http://www.hpcc.ttu.edu

We thank Dr. Jerome Onwunalu's for guidance and support, and for providing a GA code used in preliminary tests.

This research was partially supported by the Office of Naval Research (ONR) through grant N000141310041 to de Farias.

REFERENCES

[1] G. AlQahtani, R. Vadapalli, S. Siddiqui and S. Bhattacharya, "Well Optimization Strategies in Conventional Reservoirs," *Proceedings of SPE Saudi Arabia Section Technical Symposium and Exhibition*, Al-Khobar, 8-11 April 2012, 13 Pages. http://dx.doi.org/10.2118/160861-MS

[2] O. Badru and C. S. Kabir, "Well Placement Optimization in Field Development," *SPE Annual Technical Conference and Exhibition*, Denver, 5-8 October 2003, 9 Pages.

[3] W. Bangerth, H. Klie, M. F. Wheeler, P. L. Stoffa and M. K. Sen, "On Optimization Algorithms for the Reservoir Oil Well Placement Problem," *Computational Geosciences*, Vol. 10, No. 3, 2006, pp. 303-319. http://dx.doi.org/10.1007/s10596-006-9025-7

[4] A. S. Cullick, S. Vasantharajan and M. W. Dobin, "Determining Optimal Well Locations from a 3D Reservoir Model," US Patent No. US6549879 B1, 2003.

[5] D. Y. Ding, "Optimization of Well Placement Using Evolutionary Algorithms," *SPE EAGE Annual Technical Conference and Exhibition*, Rome, 9-12 June 2008, p. 912.

[6] B. Guyaguler, R. N. Horne, L. L. Rogers and J. J. Rosenzweig, "Optimization of Well Placement in a Gulf of Mexico Waterflooding Project," *SPE Annual Technical Conference and Exhibition*, Vol. 5, No. 3, 2000, pp. 110-

118.

[7] J. Onwunalu, M. Litvak, L. J. Durlofsky and K. Aziz, "Application of Statistical Proxies to Speed up Field Development Optimization Procedures," *International Conference and Exhibition*, Abu Dhabi, 3-6 November 2008, 14 Pages.

[8] G. W. Rosenwald and D. W. Green, "A Method for Determining the Optimum Location of Wells in a Reservoir Using Mixed-Integer Programming," *SPE Journal*, Vol. 14, No. 1, 1974, 12 Pages.

[9] S. Vasantharajan and A. S. Cullick, "Well Site Selection Using Integer Programming Optimization," *Proceedings of IAMG*'97, 22-26 September 1997, pp. 421-426.

[10] R. Haupt and S. Haupt, "Practical Genetic Algorithms," John Wiley and Sons, Hoboken, 2004.

[11] G. L. Nemhauser and L. A. Wolsey, "Integer and Combinatorial Optimization," John Wiley and Sons, Hoboken, 1988.

[12] P. S. Cruz, R. N. Horne and C. V. Deutsch, "The Quality Map: A Tool for Reservoir Uncertainty Quantification and Decision Making," *SPE Annual Technical Conference and Exhibition*, Vol. 7, No. 1, 2004, 9 Pages.

