Journal of Materials Science and Chemical Engineering, 2014, 2, 59-73
Published Online October 2014 in SciRes. http://www.scirp.org/journal/msce
http://dx.doi.org/10.4236/msce.2014.210009

Scientific Research

A Computational Comparison between Optimization Techniques for Wells Placement Problem: Mathematical Formulations, Genetic Algorithms and Very Fast Simulated Annealing

Ghazi D. AlQahtani[1], Ahmed Alzahabi[2], Timothy Spinner[2], Mohamed Y. Soliman[2]

[1]Saudi Aramco, Dhahran, KSA
[2]Texas Tech University, Lubbock, TX, USA
Email: ghazi.qahtani.1@aramco.com, ahmed.alzahabi@ttu.edu, timothy.spinner@ttu.edu, mohamed.soliman@ttu.edu

Received August 2014

Abstract

This study considers several computational techniques for solving one formulation of the wells placement problem (WPP). Usually the wells placement problem is tackled through the combined efforts of many teams using conventional approaches, which include gathering seismic data, conducting real-time surveys, and performing production interpretations in order to define the sweet spots. This work considers one formulation of the wells placement problem in heterogeneous reservoirs with constraints on inter-well spacing. The performance of three different types of algorithms for optimizing the well placement problem is compared. These three techniques are: genetic algorithm, simulated annealing, and mixed integer programming (IP). Example case studies show that integer programming is the best approach in terms of reaching the global optimum. However, in many cases, the other approaches can often reach a close to optimal solution with much more computational efficiency.

Keywords

Wells Placement Optimization Integer Programming Simulated Annealing Genetic Algorithm

1. Introduction

After characterization of a reservoir through seismic, radiological, and other means of survey, the drainage of the reservoir must be carefully planned. During this process, factors that are considered include the connectivity of reservoir volumes, the overall drainage volume that could be achieved from each potential well location, and constraints due to the presence of other fluid volumes [1]. Often, in order to simplify this process, each possible well location is screened for suitability and production capability then ranked accordingly, in a process referred to as sweet spot identification. Quality factor maps were introduced by Gutteridge and Gawith in order to rank well locations based on connectivity rating and productivity index [2]. Alternatively, given 3-dimensional models of lithofacies, porosity, and permeability, Deutsch proposed the geo-objects method in which cells in the computational grid model of the reservoir were combined into separate bodies based on their measure of connectivity [3]. In a more recent work, Henery *et al*. integrated geological model and reservoir properties based on log data to build 3D reservoir property volumes considering facies, and then added monthly production to identify sweet spots which works for well optimum locations [4].

Once, a value is assigned to each potential well location, the optimal combination of well locations that are to be drilled and produced must be determined. This is the wells placement problem (WPP). Most often, it is assumed that the interaction between separate wells can be ignored, as this assumption greatly reduces computational complexity [1]. Vasantharajan and Cullick proposed the combination of quality maps with an integer programming (IP) solution in order to solve the wells placement problem [5]. For computationally complex problems where conventional optimization techniques require large solution times, metaheuristics techniques offer a way to potentially reach an acceptable solution more efficiently [6]. Of this category, genetic algorithm (GA) and simulated annealing (SA) type techniques are two common methods for solving a wide variety of optimization problems [7]. Genetic algorithm has been shown to outperform human engineers by achieving a higher net present value and oil recovery index [8] and reported as the most common method used for well placement optimization [9]. GA has been shown to perform similarly to a covariance matrix adaptation solution, which is another type of evolutionary algorithm [10]. Earlier, Bangerth *et al*. had reported better results were achieved from the use of two types of simulated annealing algorithms [11].

An earlier work compared the use of genetic algorithm and integer programming to solve this problem [12]. For each case studied, it was found that IP was capable of finding solutions at least as good as those obtained from genetic algorithm. The current work adds in a simulated annealing algorithm to the comparison, while also using a redesigned genetic algorithm. The next section contains the formulation of the optimization problem as well as details on the simulated annealing, genetic algorithm and integer programming techniques used to solve the wells placement problem. After that, results of a case study are presented to provide comparison of the efficacies of the three algorithms. Finally, a short summary and conclusions are presented.

2. Algorithm Design

The genetic algorithm and simulated annealing type algorithms both make random changes to the set of chosen well locations. Both algorithms performance depends heavily on the amount of effort spent tuning their parameters to a particular problem. The problem formulation features constraints which become difficult to satisfy when either the minimum well spacing or size of the chosen set becomes large, especially when changes to the chosen set are made randomly. The difficulty of checking the distance constraints presents a heavy computational burden.

2.1. Representing Well Locations

There were two approaches considered for representing the set of potential wells locations. One way is by enumerating the potential x and y coordinates of potential well locations, $i_x \in 1,\dots,N_x$ and $i_y \in 1,\dots,N_y$ where i_x is the index of the potential well location's x-coordinate, i_y is the index of the potential well location's y-location, while N_x and N_y are the total number of potential well x-coordinates and y-coordinates, respectively. Then the set of selected locations $\{l_k\}_{k=1}^{Nwell}$ is a series of pairs $\left\{\left(i_x, i_y\right)_k\right\}_{k=1}^{Nwell}$, where integer k indexes the selected wells. In this case, performing mutations in genetic type algorithms or perturbing the system parameter in simulated annealing type algorithms corresponds to changes in integers i_x or i_y for one or more selected well locations k.

An alternative way for representing potential well locations is to index every possible well location by integer m, and for the case where potential well locations are aligned on a square grid, m ranges from 1 to $(N_x \times N_y)$. Now the selected set of wells is a set of integers $\{m_k\}_{k=1}^{Nwell}$. This second method was chosen for use within the genetic and simulated annealing algorithms applied for this work. Every potential well location still has a pair of associated x- and y-coordinate, $\left(i_x, i_y\right)_m = \left(i_{x,m}, i_{y,m}\right)$. However, indexing the potential well locations with single index m allowed for a new way for programming the check of minimum spacing constraints. In this work, the satisfaction of the spacing constraints between every pair of potential wells m and n $\left(m \in 1,\ldots,\left(N_x \times N_y\right),\right.$ $\left.n \in 1,\ldots,(N_x \times N_y)\right)$ is precomputed and the result stored in coexistence matrix C, such that element

$$C_{m,n} = C_{n,m} = \begin{cases} 1, & d\left(m,n\right) \geq d_{\min} \\ 0, & d\left(m,n\right) < d_{\min} \end{cases}$$

where $d\left(m,n\right)$ is the distance between potential well locations m and n. Then, during the GA or SA algorithms, checking if the minimum distance constraint is met between potential locations m and n is reduced to seeing whether $C_{m,n}$ is 1. This eliminates the need to repeatedly calculate the distances between the same wells.

Along with formation of the coexistence matrix, both the GA and SA algorithms are initiated with the creation of an initial set of selected wells, called an individual in GA and referred to as the parameter in SA. This initial set is formed by a greedy method as outlined in **Algorithm 1**. While the SA algorithm requires only the use of three parameters at any one time, GA requires the creation of an entire population of individuals. In the current application, this population consists of different possible sets of well locations.

2.2. Genetic Algorithm Design

Previous work by the authors that applied a genetic algorithm included modules for mutation, cross-over, local search, population intrusion, and selection [12]. Of these types of operations, the genetic algorithm applied in the current work used only the mutation and selection modules, in order to focus on achieving high performance from the mutation operation. The mutation operation on a given set S consists of adding a new well location m, and removing any wells of set S that are within the minimum spacing of m. This is referred to as well replacement operation whose algorithm is detailed in **Figure 1**.

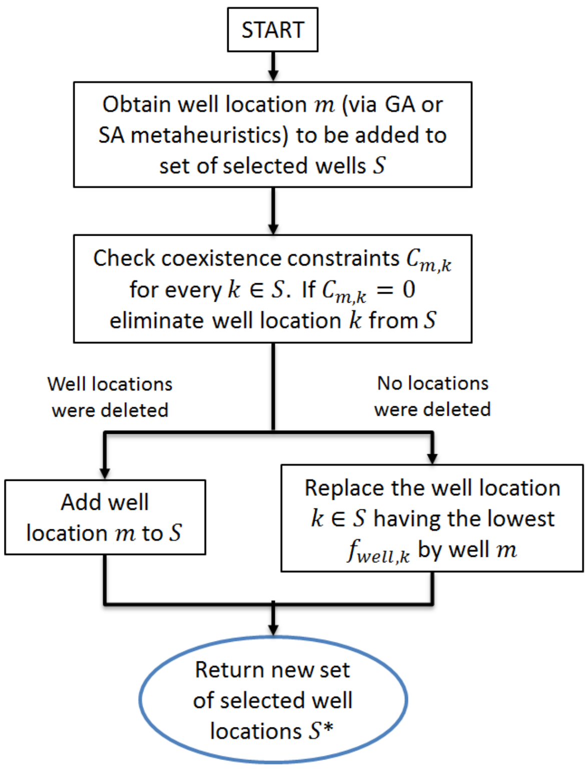

START
Obtain well location m (via GA or SA metaheuristics) to be added to set of selected wells S
Check coexistence constraints $C_{m,k}$ for every $k \in S$. If $C_{m,k} = 0$ eliminate well location k from S
Well locations were deleted
No locations were deleted
Add well location m to S
Replace the well location $k \in S$ having the lowest $f_{well,k}$ by well m
Return new set of selected well locations S^*

Figure 1. Algorithm flowchart for the well replacement operation.

```
$S$ = GREEDY_SET ($d_{min}$, $N_{well}$)
1  enumerate well locations $1 \ldots N_x \times N_y$ based on
   descending values of $f_{well}$
   % create coexistence matrix $C$:
2    for $m = 1 \ldots N_x \times N_y$
3      for $n = m \ldots N_x \times N_y$
4        $C_{m,n} = C_{n,m} = \begin{cases} 1, & d(m,n) \geq d_{min} \\ 0, & d(m,n) < d_{min} \end{cases}$
5  place location 1 in the set of selected well
   locations $S$
6  for $m = 2 \ldots N_x \times N_y$
7    if $C_{m,n} = 1$ for all $n \in S$
8      add $m$ to set $S$
9      if size($S$)=$N_{well}$
10       break; % creation of set S is complete
```

Algorithm 1. Pseudocode for greedy creation of selected well location set.

To create an initial population of individuals for the genetic algorithm, the initial individual is mutated in parallel, Npop/6 times. For each mutation, a well location that is not in the set of selected locations is added to this set. Once the initial population is created, the genetic algorithm loops over the number of generations. During each generation, a given set of well locations is mutated through the well replacement operation, five times serially. This increases the population size from Npop/6 to Npop. Due to the spacing constraints a given set S end up with less than Nwell well locations after the well replacement operation. Therefore, after well replacement, it is attempted to fill each set using a greedy method, wherein the highest objective valued well locations that meet the constraints with all of the existing locations within the set are added to the set. At the end of every generation, a selection of the sets that move onto the next generation is made. The highest Npop/6 individuals are chosen for the start of the next iteration, and the remaining individuals are discarded. After the algorithm runs for either the prescribed number of generations or exceeds the maximum time, the best set of well locations is simply chosen based upon the highest objective value of all existing individuals. The pseudocode for the genetic algorithm is presented as **Algorithm 2**.

2.3. Very Fast Simulated Reannealing Algorithm Design

In contrast to GA where mutations are carried out in parallel, within simulated annealing, a single parameter vector is changed serially, akin to having a single individual in genetic algorithm. However, every change to the parameter vector must pass the acceptance criteria or else the change is rejected. In the SA algorithm used, first the system parameter (set of selected well locations) and coexistence constraint matrix are created in the same way as they were for genetic algorithm. During the simulated annealing algorithm, presented in pseudocode as **Algorithm 3**, the size of parameter change is made according to the current temperature, and both the temperature and parameter size change decrease according to a cooling schedule as the number of iterations advances. As the cooling proceeds, the parameter will move to a local minimum in the objective function. To give the parameter the chance to move out of this minimum, the cooling schedule is periodically reset, which is called reannealing.

In the current work, the system parameter is the set selected of well locations. The size of the parameter change was chosen to be the number of selected well locations that must be deleted in order to accommodate adding a new well while meeting the spacing constraints. Therefore, the parameter size change is dependent on which replacement well not in the set is chosen to be added. As cooling proceeds, only wells which do not conflict with many of the existing wells in set S will be considered due to the decreasing temperature. After a replacement well is chosen, it is used in the well replacement operation to a new potential parameter, set S_p. S_p will only replace the current set of wells S if it meets the acceptance criteria. The probability of accepting a set S_p with a lower objective value than the current *S* also decreases as cooling proceeds, which forces the parameter towards a local minimum towards a local minimum towards the end of an annealing cycle. However, replacing S with an S_p having lower objective value is possible at the beginning of each cooling cycle,

```
$S^* = \mathrm{GA}(d_{min}, N_{well})$
$S_1$ = GREEDY_SET($d_{min}, N_{well}$) with obj. value $F_1$
% create initial population of individuals:
  for $j = 2 \dots Npop/6$
    randomly choose well location $m \notin S_1$
    $S_j$ =WELL_REPLACEMENT($S_1, m$)
    fill open locations in set $S_j$ with greedy
    search
for $i = 1 \dots$ Number_of_generations
  for $j = 1 \dots$ Npop/6
    % MUTATION:
    for $k = 1 \dots 5$
      $\ell = (k-1) \times {}^{Npop}/_6 + j; \ell' = k \times {}^{Npop}/_6 + j$
      randomly choose well location $m \notin S_\ell$
      $S_{\ell'}$ = WELL_REPLACEMENT($S_\ell, m$)
      fill open locations in set $S_{\ell'}$
      calculate objective function
      $F_{\ell'} = \sum_{n \in S_{\ell'}} f_{well,n}$
  % WELL SELECTION:
  RANK individuals $\{S_1 \dots S_{Npop}\}$ based on $F_\ell$
  Keep highest ranked $(Npop/6)$ individuals
  and renumber them as $\{S_1 \dots S_{Npop/6}\}$
Choose optimal achieved set of well locations
$S^* = S_m$ where set $S_m$ satisfies $F_m = \max_j F_j$
```

Algorithm 2. Pseudocode for genetic algorithm.

```
$S^* = \mathrm{VFSR}(d_{min}, N_{well})$
$S_1$ = GREEDY_SET($d_{min}, N_{well}$) with objective value $F^*$
for $i = 1 \dots$ number_of_reanneals
  $S = S^*$ with objective value $F = F^*$
  for $j = 1 \dots$ cooling_steps
    set temperature parameter $T_j$
    % select location of replacement well:
      for $m = 1 \dots N_x \times N_y$, $m \notin S$
        $\delta(m) = \sum_{n \in S} (C_{m,n} == 0)$
      find $\Delta$ according to cooling schedule $T_j$
      choose well $m$ from the set $m \in \{n \mid \delta(n) = \Delta\}$

    $S_p$ = WELL_REPLACEMENT($S, m$)
    fill open locations in set $S_p$
    $F_p = \sum_{n \in S_p} f_{well,n}$
    % acceptance criteria:
      if $F_p \geq F$
        $S = S_p$ & $F = F_p$
        if $F_p \geq F^*$
          $S^* = S_p$ & $F^* = F_p$;
      else
        $\Delta F = F - F_p$
        draw random number $u \in U[0,1]$
        if $u < \exp(-{}^{\Delta F}/_{\Delta T})$
          $S = S_p$ & $F = F_p$
```

Algorithm 3. Pseudocode for very fast simulated reannealing.

helping the parameter to jump out of the local minima, with hope that it can reach a lower minimum.

Within the simulated annealing algorithm, a set of wells with the highest achieved objective value is retained and only replaced when a new set exceeds its objective value. At the start of every reannealing cycle, the parameter to be changed is set to this best set. After either the total number of desired reanneals occurs or the maximum computational time limit is reached, the best set of wells achieved having the highest objective value is returned to the user.

3. Optimization via Mathematical Formulations

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 the wells placement problem was presented in [12].

4. Optimization Computations

The computations were carried on a single machine that has 2.4 GHz Quad Core CPU with 32 GB RAM. For tests using IP, the formulations generated were written in MATLAB and the optimization models were resolved using Gurobi 5.6 solver with default settings which were set to Branch-and-Cut solution strategy and the absolute gap tolerances were set to zero. The other controls used are the same as described in [12].

Our test grid has $m = n = 100$, and we set the values of D to $6, 8, 10, 12, 14, 16$, 18 and 20. For each of these values, we tried to solve WPP with N set to $10, 20, \ldots$, up to the point at which either the time limit of 3600 seconds was reached, or the value of N was not reached; *i.e.*, the point at which either method was unable to find a solution that represented N well locations. For the GA, we set $n_g = 10000$, $n_p = 1000$, $n_m = 5$. For the VFSA, we set the cooling steps per anneal $= 1000$, and the reanneals $= 500$, and search space of 2000 for GA and VFSA tests.

The values of the P matrix were obtained with the commercial reservoir simulator Eclipse, using the Quality Maps approach (see Da Cruz *et al*. 2004 for details). The main characteristics of our heterogeneous and anisotropic reservoir are as follows: 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 the two closest grid points is 300 ft, and the thickness of the reservoir is 75 ft.

The results obtained using GA, VFSA and IP are shown in two sets. The first set includes performance comparison in terms of objective function values achieved and elapsed computation time. This set of comparison can be found in **Figures 2-9** when $D =$ 6, 8, 10, 12, 14, 16, 18, and 20, respectively. Further, the term "Best Sol'n" is used in the first set of figures to represent the value of the best objective function solutions found by any of the methods. (Note: in GA, this value is called the fitness of the best individual, whereas in VFSA and IP it is called the objective function value of the best solution. We will use the latter expression in our analysis.) The expression "Time" shows the computational time, in seconds, required for the test to finish. We note that, because of the settings used in Gurobi, all solutions obtained using IP are provably optimal precisely when the time is less than 3600 seconds.

The second set of results includes variation between the three methods investigated with regard to the number of wells each method placed. This set shows, in the same figures, the actual number of wells accommodated by each method. The second set of the results can be found in the form of flow charts in **Figures 10-17** when $D =$ 6, 8, 10, 12, 14, 16, 18, and 20, respectively.

In the second set, the term "Card Gap" is used to refer to the cardinality constraint, which shows the actual number of wells each method were able to accommodate, relative to the assigned value of N, based on equation (1).

$$\text{Card Gap} = \frac{N - (GA \text{ or VFSA or IP) Card}}{N} \times 100 \tag{1}$$

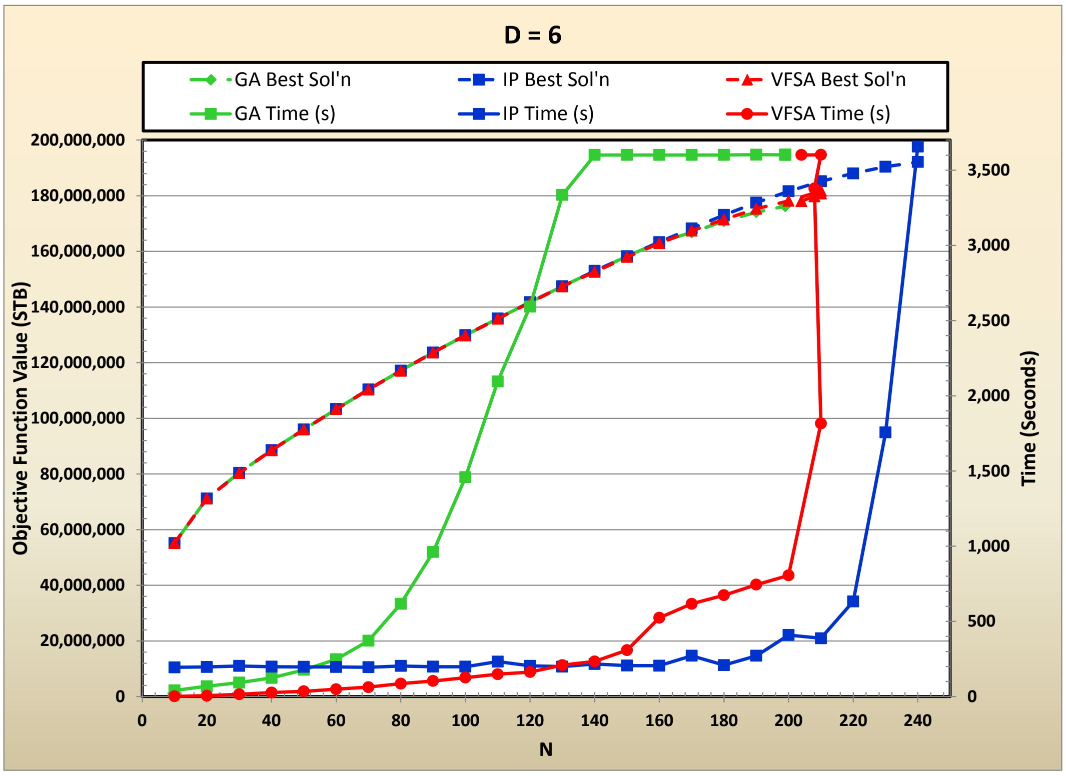


Figure 2. GA, VFSA and IP performance comparison in the subject of time and objective function values for $D = 6$.

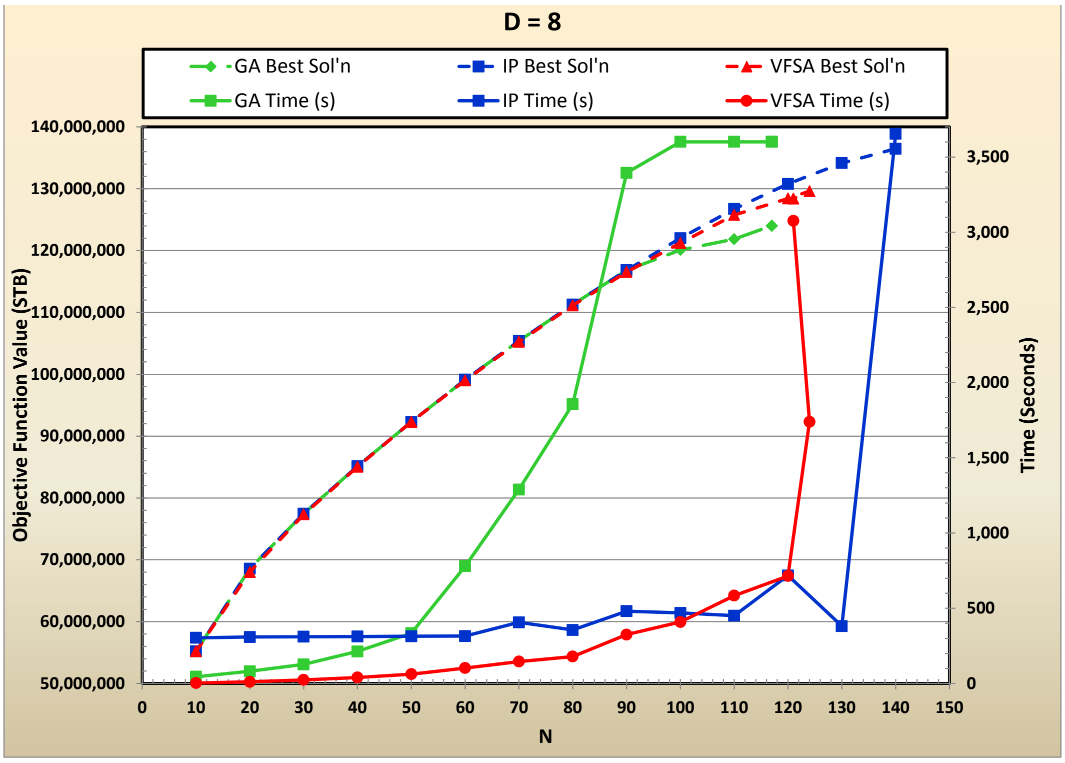


Figure 3. GA, VFSA and IP performance comparison in the subject of time and objective function values for $D = 8$.

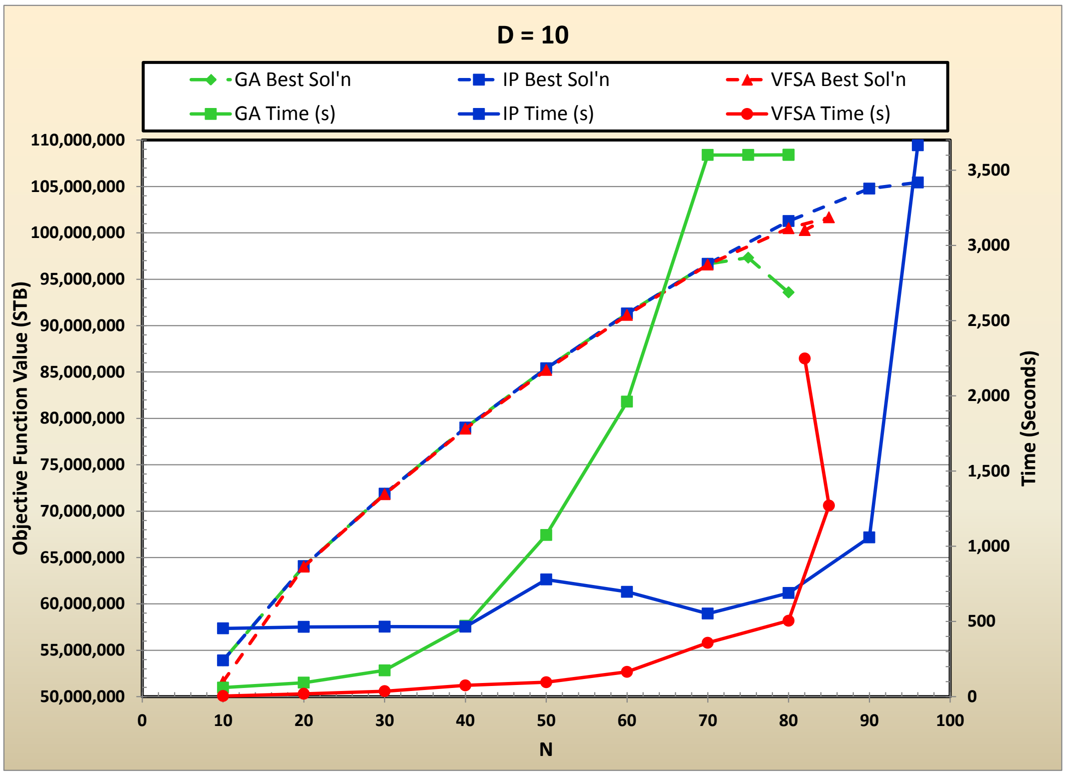


Figure 4. GA, VFSA and IP performance comparison in the subject of time and objective function values for $D = 10$.

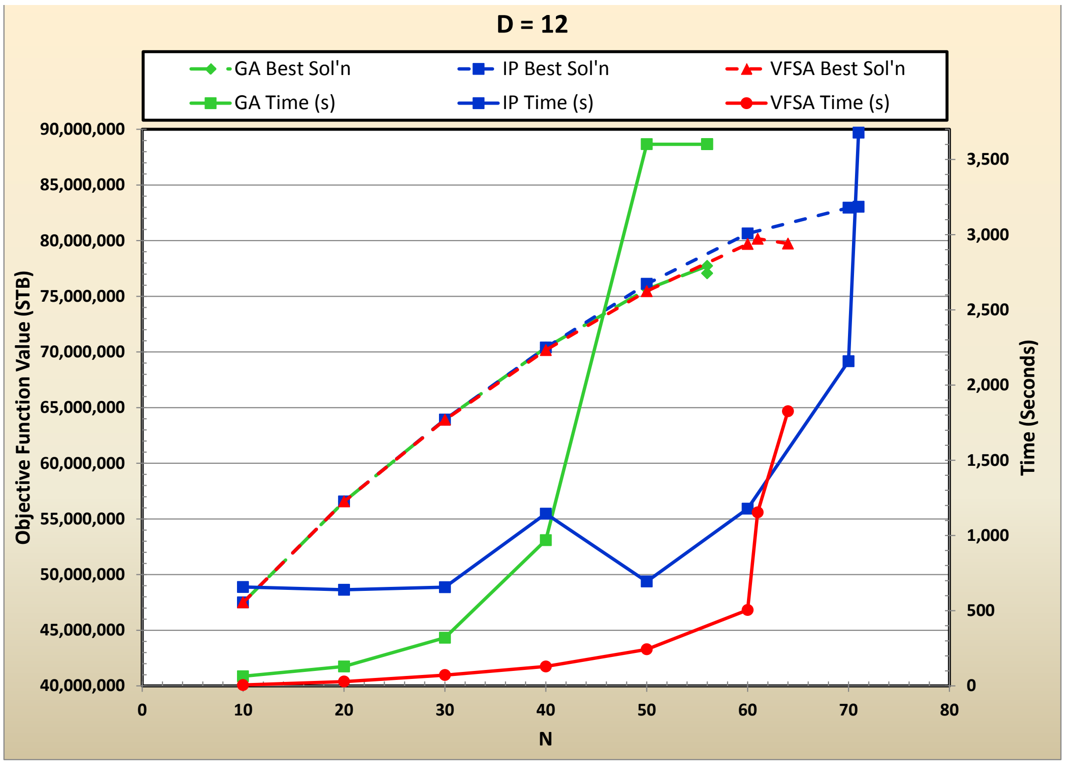


Figure 5. GA, VFSA and IP performance comparison in the subject of time and objective function values for $D = 12$.

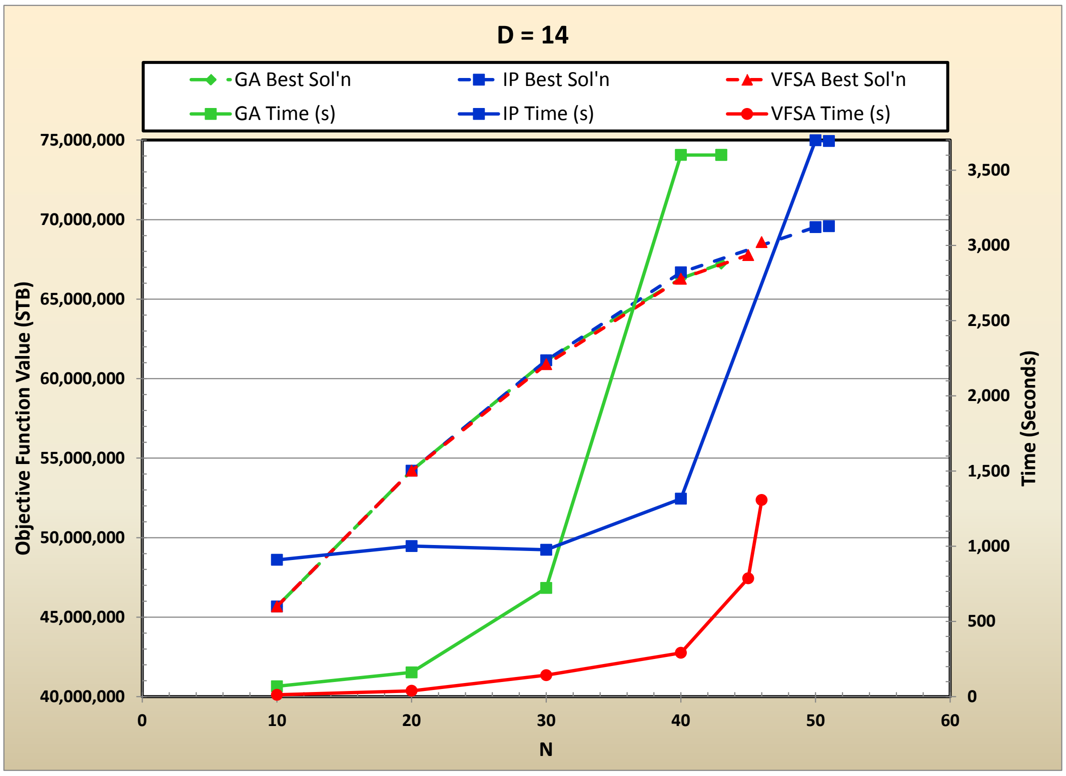


Figure 6. GA, VFSA and IP performance comparison in the subject of time and objective function values for *D* = 14.

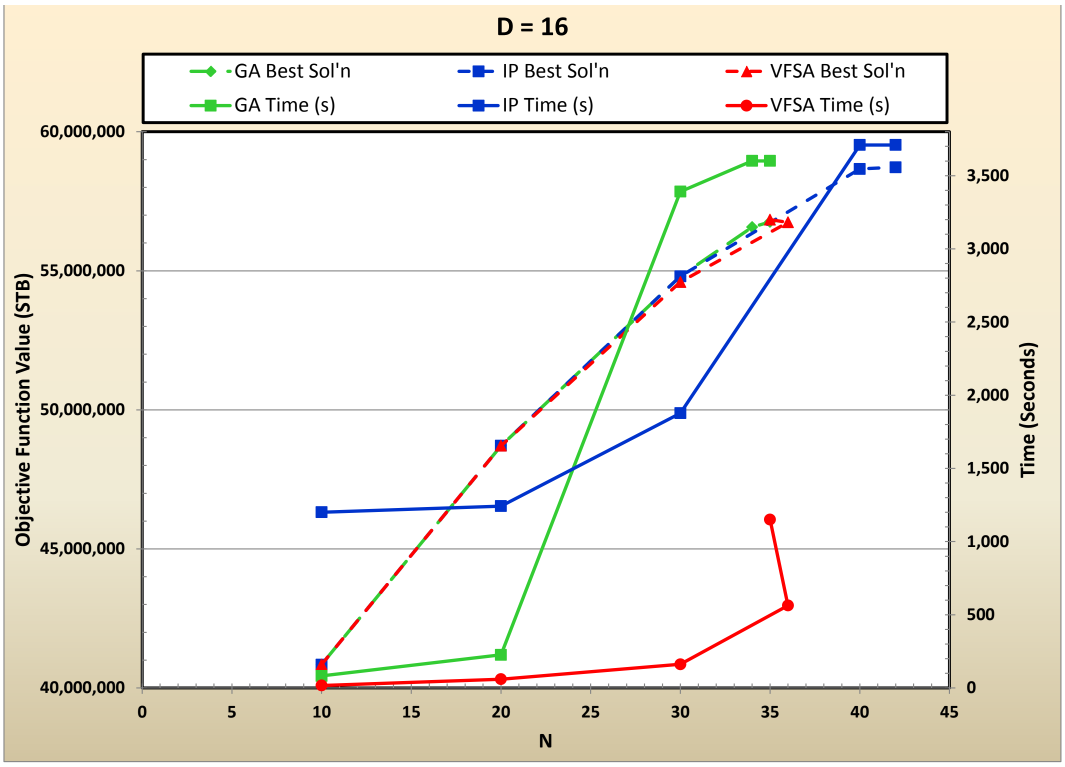


Figure 7. GA, VFSA and IP performance comparison in the subject of time and objective function values for *D* = 16.

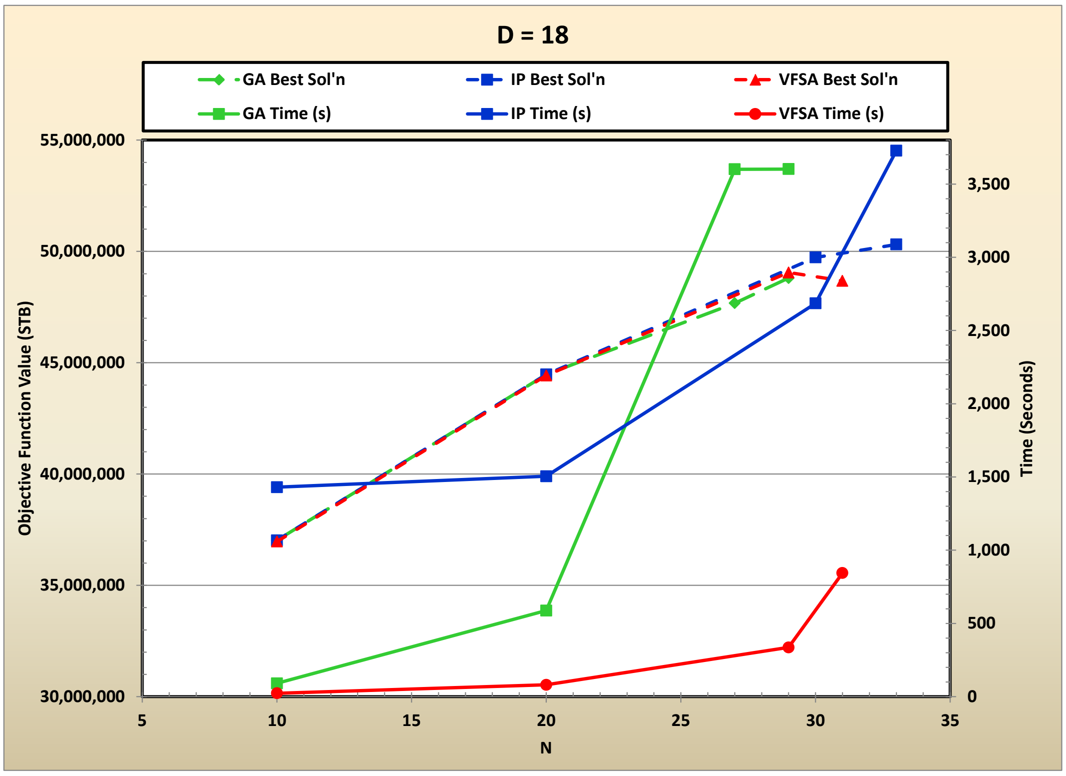


Figure 8. GA, VFSA and IP performance comparison in the subject of time and objective function values for *D* = 18.

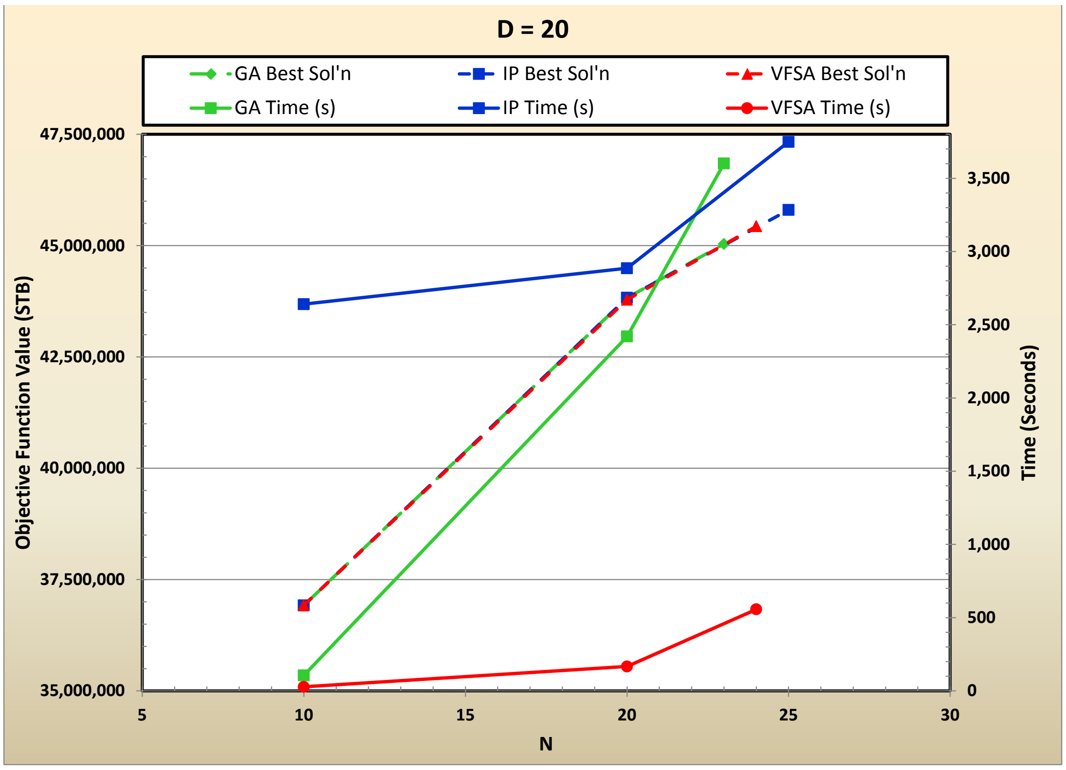
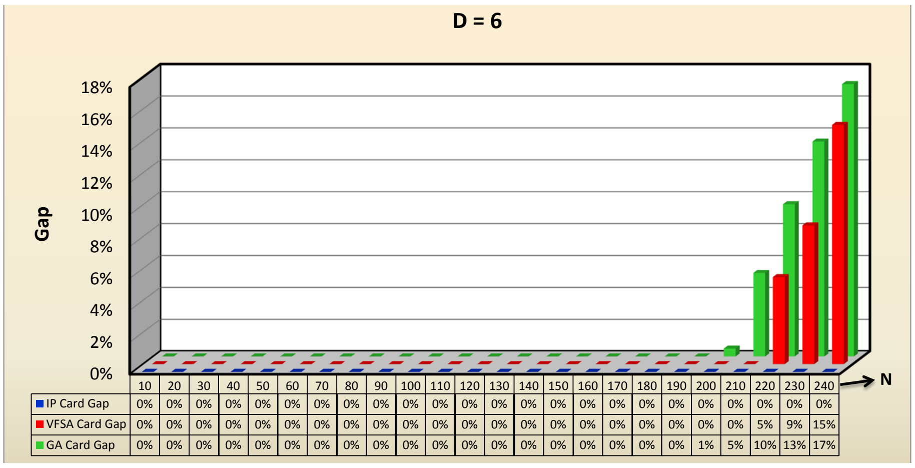
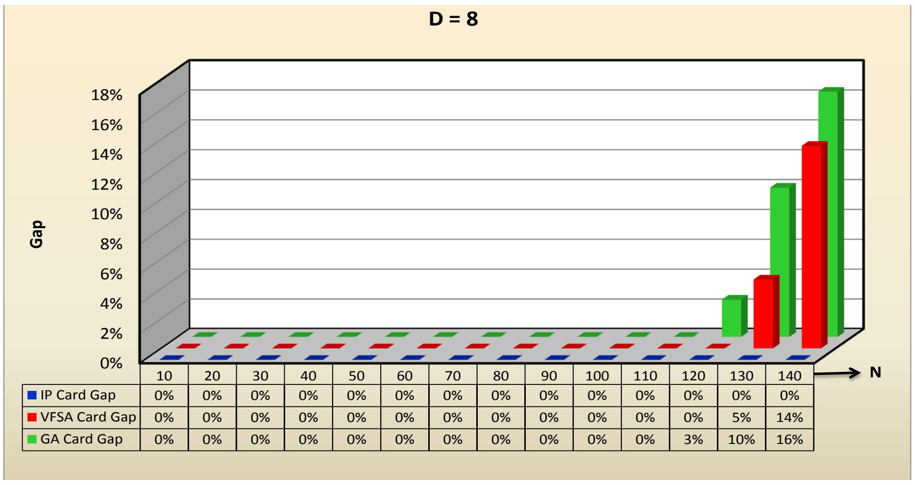


Figure 9. GA, VFSA and IP performance comparison in the subject of time and objective function values for D = 20.



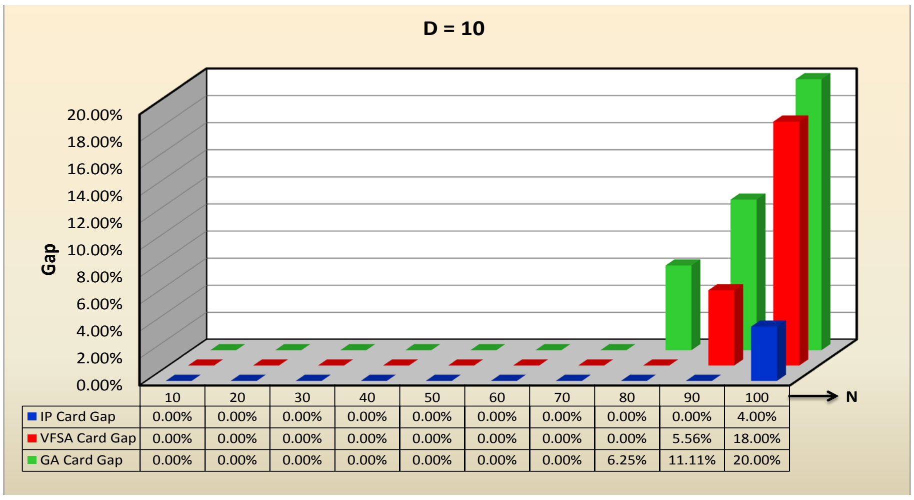
	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200	210	220	230	240
🟦 IP Card Gap	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
🟥 VFSA Card Gap	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	5%	9%	15%
🟩 GA Card Gap	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	5%	10%	13%	17%

Figure 10. Bar chart showing the cardinality gaps. The color blue, red, and green show the cardinality obtained using IP, VFSA and GA, respectively, for *D* = 6.



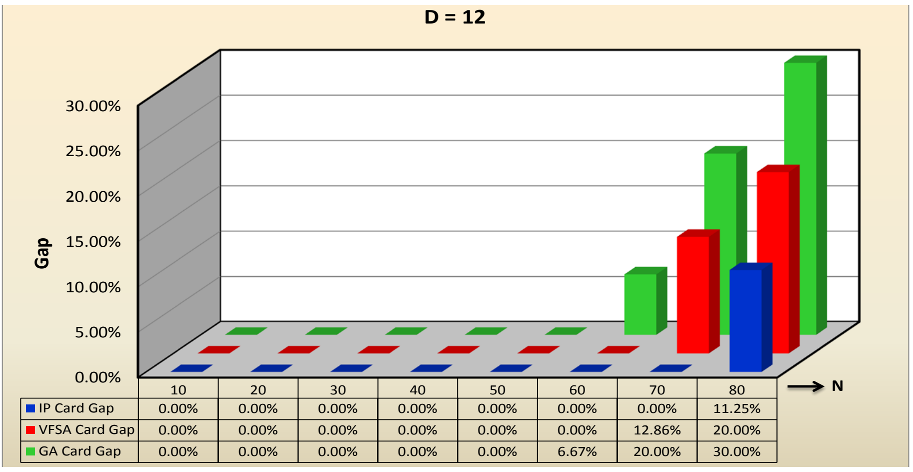
	10	20	30	40	50	60	70	80	90	100	110	120	130	140
🟦 IP Card Gap	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
🟥 VFSA Card Gap	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	5%	14%
🟩 GA Card Gap	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	10%	16%

Figure 11. Bar chart showing the cardinality gaps. The color blue, red, and green show the cardinality obtained using IP, VFSA and GA, respectively, for *D* = 8.



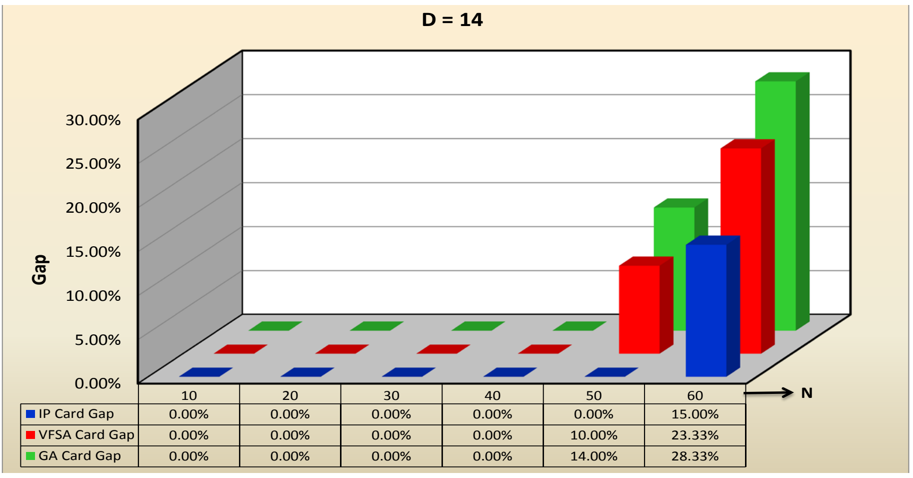
	10	20	30	40	50	60	70	80	90	100
🟦 IP Card Gap	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	4.00%
🟥 VFSA Card Gap	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	5.56%	18.00%
🟩 GA Card Gap	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	6.25%	11.11%	20.00%

Figure 12. Bar chart showing the cardinality gaps. The color blue red, and green show the cardinality obtained using IP, VFSA and GA, respectively, for *D* = 10.



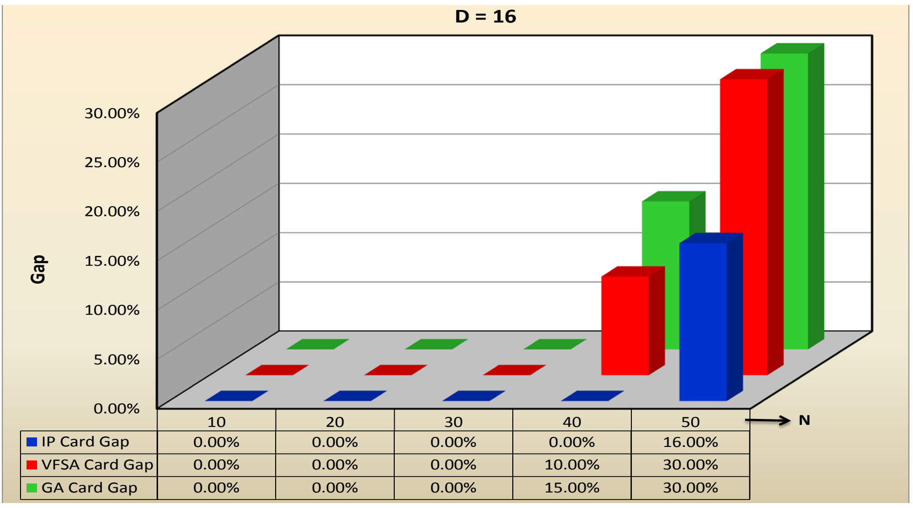
	10	20	30	40	50	60	70	80
🟦 IP Card Gap	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	11.25%
🟥 VFSA Card Gap	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	12.86%	20.00%
🟩 GA Card Gap	0.00%	0.00%	0.00%	0.00%	0.00%	6.67%	20.00%	30.00%

Figure 13. Bar chart showing the cardinality gaps. The color blue, red, and green show the cardinality obtained using IP, VFSA and GA, respectively, for $D = 12$.



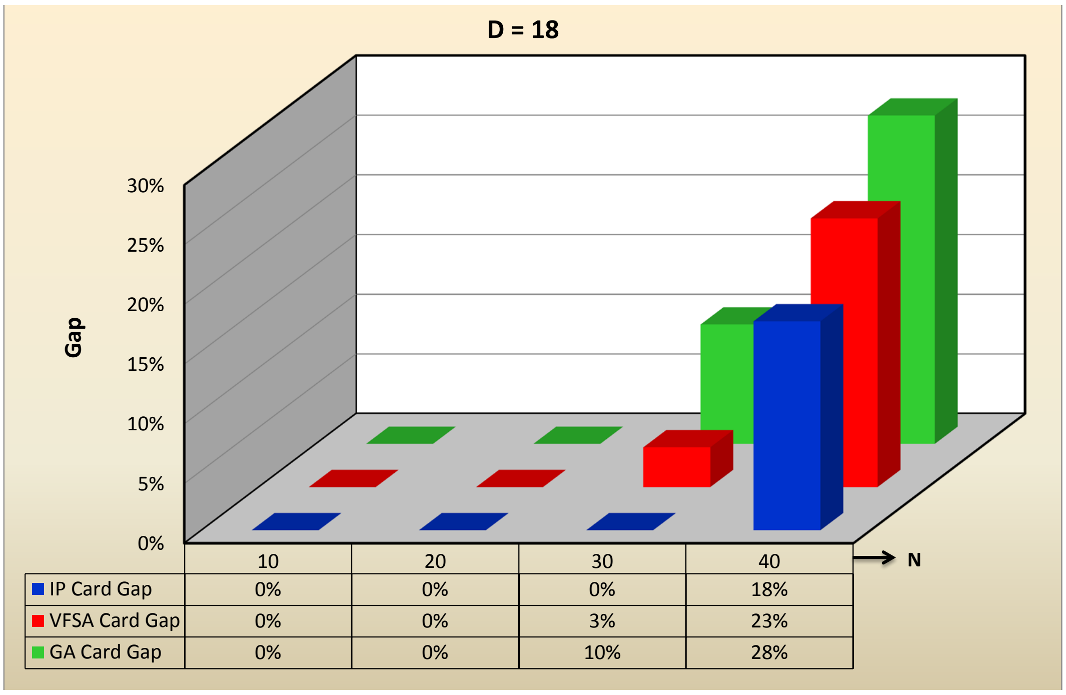
	10	20	30	40	50	60
🟦 IP Card Gap	0.00%	0.00%	0.00%	0.00%	0.00%	15.00%
🟥 VFSA Card Gap	0.00%	0.00%	0.00%	0.00%	10.00%	23.33%
🟩 GA Card Gap	0.00%	0.00%	0.00%	0.00%	14.00%	28.33%

Figure 14. Bar chart showing the cardinality gaps. The color blue , red, and green show the cardinality obtained using IP, VFSA and GA, respectively, for $D = 14$.



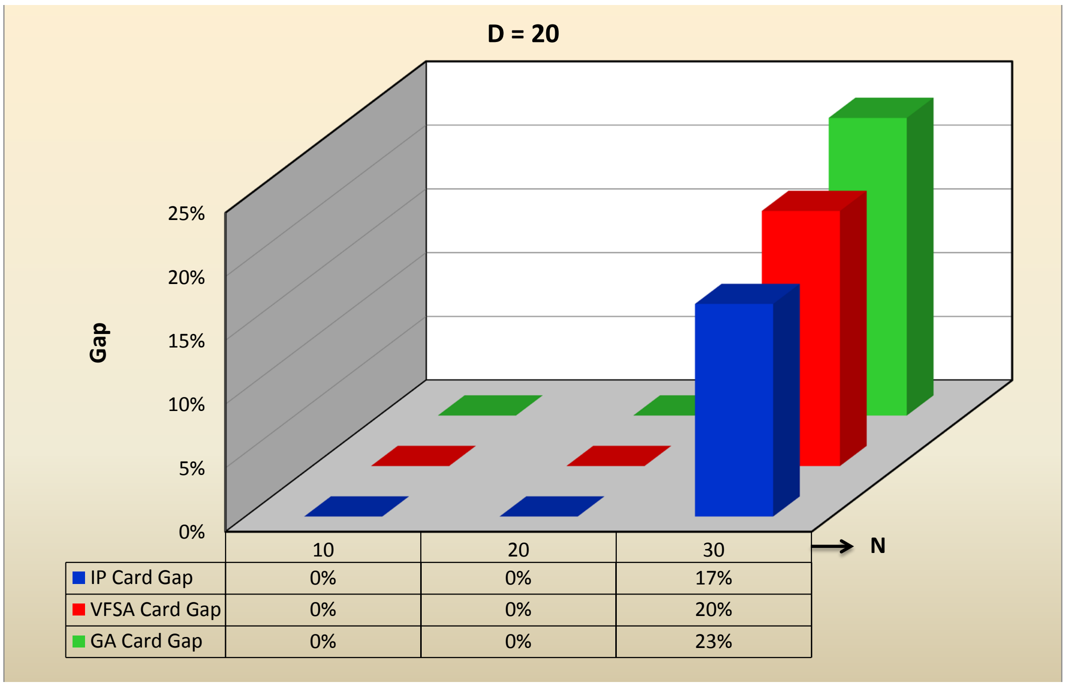
	10	20	30	40	50
🟦 IP Card Gap	0.00%	0.00%	0.00%	0.00%	16.00%
🟥 VFSA Card Gap	0.00%	0.00%	0.00%	10.00%	30.00%
🟩 GA Card Gap	0.00%	0.00%	0.00%	15.00%	30.00%

Figure 15. Bar chart showing the cardinality gaps. The color blue, red, and green show the cardinality obtained using IP, VFSA and GA, respectively, for $D = 16$.



	10	20	30	40
🟦 IP Card Gap	0%	0%	0%	18%
🟥 VFSA Card Gap	0%	0%	3%	23%
🟩 GA Card Gap	0%	0%	10%	28%

Figure 16. Bar chart showing the cardinality gaps. The color blue, red, and green show the cardinality obtained using IP, VFSA and GA, respectively, for *D* = 18.



	10	20	30
🟦 IP Card Gap	0%	0%	17%
🟥 VFSA Card Gap	0%	0%	20%
🟩 GA Card Gap	0%	0%	23%

Figure 17. Bar chart showing the objective function and cardinality gaps. The color blue and red show the cardinality obtained using GA and IP, respectively, while the color green displays the objective function difference between the two methods for D = 20.

From the first set of figures, it is clear that the difficulty of the problem increases as the values of D and N increase, as illustrated in **Figures 2-9**. On the IP side, this is reflected in the time required to solve the instances, which is denoted with solid blue lines. On the GA and VFSA side, both the computational time, denoted with solid green lines for GA and solid red lines for VFSA, and the cardinality gap of the best solution, represented with green bars for GA and red bars for VFSA in **Figures 10-17**, show the increase in the difficulty of the problem. For instance, for $D = 6$ and $N = 210$, the best solution that GA can find has cardinality gap of 5.24% (*i.e.* 199), although solutions with zero cardinality gap (and better objective function values) do exist, as the IP and VFSA results show in **Figure 2** and **Figure 10**. When $N = 240$, for $D = 6$, the cardinality gap for GA and VFSA reaches 17.1% and 15%, respectively, even though zero cardinality gap (and better objective function) is noticed as the IP results indicate in **Figure 2** and **Figure 10**. Similar situations are observed when $D = 8,10,12,14,16,18,$ and 20 for $N \geq 120, 80, 60, 50, 40, 30,$ and 30, respectively.

Overall, IP was capable of finding better solutions than GA and VFSA, especially, when the instances are getting harder. In every test, the best solution found by IP, which is denoted with dashed blue lines in the first set of figures, had an objective function value at least as good as the one found by the GA, which is denoted with dashed green lines in the same graphs, or the one found by VFSA, which is denoted with dashed red lines. In 10 cases, the three methods found an optimal solution to the problem. In 4 other instances, the objective function gap between IP and GA methods was $\leq 1\%$ and in 43 instances, the objective function gap between IP and VFSA methods was $\leq 1\%$. This shows that GA and VFSA can be effective for instances that are less challenging, but, for the harder instances, IP was notably more successful than both.

The merit of being faster and finding better solutions is due not only to the differences in approaching the problem (IP vs. VFSA vs. GA), but also a result of the algorithms used to solve the instances. While the developed GA or VFSA algorithms have hundreds of code lines, Gurobi is state of the art professional solver package developed for years. The apparently dashed lines representing the values of the objective functions from the three methods, as in the first set of figures, can go as high as 11% between IP and GA when $D = 10$ and $N = 100$, and 7% between IP and VFSA when $D = 6$ and $N = 240$ which underscores the contrast between the investigated algorithms. For instance, 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 for the GA or the VFSA methods. Moreover, the fact that Gurobi can guarantee the optimality of a solution is a feature that neither GA nor VFSA have. It is only in comparison studies, such as this one, the effectiveness of GA or VFSA can be visualized in finding optimal solutions. Finally, we note that Gurobi is a multi-purpose solver that can handle a vast number of different problems, while our GA and VFSA were developed specifically to tackle WPP.

5. Conclusions

In this study, we compared the performance of GA, VFSA 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 objective of extracting the maximum amount of oil from the reservoir. Our results indicate that GA and VFSA can be effective for easier instances, but lacks performance for harder ones. In comparison, Gurobi (which takes the IP approach) always found a solution at least as good as the developed GA or VFSA, and faster in many instance. Moreover, IP has the advantage of finding provably optimal solutions, while either GA or VFSA are not able to guarantee that a solution is optimal.

The GA presented here is designed and developed differently than the one used in [12] with minimum alterations for the genetic parameters which can be sensitized and have its performance enhanced. The VFSA, likewise, was developed with minimum edits to the annealing parameters in which common values were used and, thus, can be improved by sensitizing the annealing parameters or, simply, by changing the algorithm. Similarly, the solution via IP can be made faster by considering different formulations of the problem and tuning some parameters on Gurobi. We opted not to complicate the investigated approaches by keeping them simple without revamping genetic, annealing and tuning parameters for the GA, VFSA, and Gurobi, respectively.

References

[1] Wang, B. (2002) Well Site Selection Algorithm Considering Geological, Economical and Engineering Constraints. Thesis, University of Alberta.

[2] Gutteridge, P.A. and Gawith, D.E. (1996) Connected volume calibration for well-path ranking. *EPOC*'96: *European Production Operations Conference*.

[3] Deutsch, C.V. (1998) Fortran Programs for Calculating Connectivity of Three-Dimensional Numerical Models and for Ranking Multiple Realizations. *Computers & Geosciences*, **24**, 69-76. http://dx.doi.org/10.1016/S0098-3004(97)00085-X

[4] Henery, F., Trimbitasu, T. and Johnson J. (2011) An Integrated Workflow for Reservoir Sweet Spot Identification. 2011 *Gussow Geoscience Conference*, Banff, Alberta.

[5] Vasantharajan, S. and Cullick, A.S. (1993) Well Site Selection Using Integer Programming. *Proceedings of IAMG* '97: Volume 1, pages 421-426. CIMNE.

[6] Talbi, E.G. (2009) Metaheuristics: From Design to Implementation (Vol. 74). John Wiley & Sons. http://dx.doi.org/10.1002/9780470496916

[7] Sen, M.K. and Stoffa, P.L. (1995) Global Optimization Methods in Geophysical Inversion. Elsevier.

[8] Emerick, A.A., *et al.* (2009) Well Placement Optimization Using a Genetic Algorithm with Nonlinear Constraints. *SPE reservoir Simulation Symposium.*

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

[10] Ding, D.Y. (2008) Optimization of Well Placement Using Evolutionary Algorithms. *SPE EUROPEC/EAGE Annual Conference and Exhibition*, SPE. Vol. 113525.

[11] Bangerth, W., Klie, H., Wheeler, M.F., Stoffa, P.L. and Sen, M.K. (2006) On Optimization Algorithms for the Reservoir Oil Well Placement Problem. *Computational Geosciences*, **10**, 303-319. http://dx.doi.org/10.1007/s10596-006-9025-7

[12] AlQahatani, G., Alzahabi, A., Kozyreff, E., de Farias Jr., I.R. and Soliman, M. (2013) A Comparison between Evolutionary Metaheuristics and Mathematical Optimization to Solve the Wells Placement Problem. *Advances in Chemical Engineering and Science*, Paper ID 3700353.