An Improved Evolutionary Algorithm with Data Mining for a Vehicle Routing Problem

L. H. C. Merschmann, E. H. Marinho, H. G. Santos, L. M. A. Drummond, L. S. Ochi, F. Dalboni
Computing Institute – Fluminense Federal University
Niterói, Rio de Janeiro, Brazil
{lmerschmann, emarinho, hsantos, lucia, satoru}@ic.uff.br, *dalboni@openlink.com.br*

Abstract

The aim of this work is to present some alternatives to improve the performance of an Evolutionary Algorithm applied to the problem known as Oil Collecting Vehicle Routing Problem. Some proposals based on the insertion of Local Search and Data Mining modules in a Genetic Algorithm are presented. Four algorithms were developed: a Genetic Algorithm, a Genetic Algorithm with a Local Search procedure, a Genetic Algorithm including a Data Mining module and a Genetic Algorithm including Local Search and Data Mining. The results demonstrate that the incorporation of Data Mining and Local Search modules in GA improved the solution quality produced by this method.

1. Introduction

Concerning oil exploitation, there is a class of onshore wells called artificial lift wells where the use of auxiliary methods for the elevation of fluids (oil and water) is necessary. In this case, a fixed system of beam pump is used when the well has a high productivity. Because oil is not a renewable product, the production of such wells will diminish until the utilization of equipments permanently allocated to them will become economically unfeasible. The exploitation of low productivity wells can be done by mobile equipment coupled to a truck. Usually the collector mobile is not able to visit all wells in a single day. In this context, arises the problem called Oil Collecting Vehicle Routing Problem (OCVRP).

The OCVRP can be modeled by a graph $G = (X,U)$, where the vertices correspond to the wells and each edge represents the road linking two wells. Each vertex $i \in X$ has an associated value $v_i \in R^+$ which corresponds to the level of oil in the well i and each edge $\{i, j\} \in U$ has a value $t_{ij} \in R^+$, which represents the time to travel between wells i and j.

We describe OCVRP as the problem of routing a vehicle starting and finishing at the same location, denominated Oil Treatment Station (OTS), where the separation of oil from water occurs. Visiting all the wells from a subset $J \subseteq X$, the aim is to maximize the amount of collected oil without violating the constraints set which include daily time limit of work [3], besides the non-formation of subtours disconnected from origin as in the Traveling Salesman Problem (TSP) [17]. The OCVRP can be considered as a generalization of TSP which is classified as NP-hard. In this way, the application of exact methods becomes limited, justifying the use of heuristics techniques, such as Evolutionary Algorithms (EAs).

The literature presents some similar problems such as the Traveling Purchaser Problem [11], the Prize Collecting Problem [4] and the Orienteering Problem [14]. It has been shown that this kind of problem, where only a subset of vertices are included in the solution, can be efficiently solved by evolutionary metaheuristics [10,19]. In this work we propose alternatives to improve the performance of EAs. For this purpose, we implemented four versions of Genetic Algorithms (GAs) to solve the OCVRP. Initially we describe a GA and afterwards, three improved versions, GA with Local Search, GA with Data Mining and GA with Data Mining and Local Search.

2. Evolutionary Algorithms

Evolutionary algorithms and the GAs, its most popular representative, are part of the research area of Artificial Intelligence inspired by the Natural Evolution Theory and Genetics, known as

Evolutionary Computation. Those algorithms try to simulate some stages of Darwin's Natural Selection and have been used in several areas to solve problems considered intractables (NP-complete and NP-hard), although its traditional versions do not demonstrate too much efficiency in the resolution of high complexity combinatorial optimization problems [9,13,16].

GA is an iterative heuristic procedure in which selection, crossover and mutation operators are systematically applied to a population of individuals. These individuals, also called chromosomes, often encode different solutions to the problem. Each position in the chromosome is called gene. The solution quality is measured by means of a function, called fitness function. Through the application of genetic operators, the search space is explored to find good solutions for the problem in question.

In order to improve the performance of GA, researchers have proposed variants such as Memetic Algorithms [18], Scatter Search [12] and Population Heuristics [7].

2.1. Genetic Algorithm

This section describes an overview of the GA proposed to solve the OCVRP.

In order to represent a solution, each individual is defined by a variable size list of integer numbers, where the genes correspond to wells pertaining to route coded by an individual. The last gene of each individual always represents the Oil Treatment Station, using 0 to code it. The position of a well in this list represents its order in the route. An example of genetic representation is showed in Figure 1. In this case, the following route is represented: OTS – well 16 – well 34 – well 21 – well 17 – well 76 – well 03 – well 58 – well 44 – OTS. Note that the route always starts at OTS, in spite of the first gene not representing it.

16	34	21	17	76	3	58	44	0

Figure 1. Sample individual

To generate the initial population, individuals are constructed using an iterative procedure. They are generated considering a greedy criterion which is used to define the well that will be included in the solution being constructed. This criterion is based on $v_i/t_{j,i}$ ratio, where v_i is the production rate of the candidate well i and $t_{j,i}$ is the time spent to travel from the last well j included in the partial solution to the well i. Thus, at each iteration, a well is selected from a list that contains all wells not yet included in the solution, with $v_i /t_{j,i}$ ratio in the range $[r_{min}, r_{max}]$. $\bar{r}$ being the greatest $v_i /t_{j,i}$ ratio among all wells not yet included in the solution and $\underline{r}$ being the smallest ratio among these same wells then $r_{min}= \bar{r} - \alpha (\bar{r} - \underline{r})$ and $r_{max}= \bar{r}$. The α parameter should be selected in the range [0,1], allowing the choice of the randomization degree of individual generation. This process is executed while the total time consumed on the route being constructed has not reached the time limit (OCVRP constraint).

Our GA uses a genetic operator different from traditional crossover operator. A new individual (offspring) is generated from *np* solutions (parents) of the current population, where *np* varies from one up to the size of the population. Each parent solution is chosen by a tournament procedure that selects the fittest individual from a set of *k* individuals randomly selected from the population.

The crossover operator proposed here combines information from parent solutions and the heuristic criterion employed in the constructive phase, as follows: during each iteration of offspring generation, a well is chosen to take part in the route. This well is randomly chosen from a list of remaining wells taking into account a probability distribution that favors the addition of wells with the highest $\gamma_i = (1+\omega_{j,i})*(v_i / t_{j,i})$, where $\omega_{j,i}$ is the frequency which well i (candidate well) is the successor of well j (last well added to the partial solution) in parent solutions. The probability of chosing a given well is proportional to the position of this well in a list sorted by non-ascending order of the greedy criterion γ_i, as suggested in [8]. In this work, polynomial distribution probability was chosen. This process is executed while the total time consumed on the route being constructed has not reached the time limit. It is important to note that our crossover operator does not rely only on information from parent solutions, allowing the occurrence of offsprings with genetic code not available on parents. Thus, there is no need of incorporating a mutation operator.

During the evolutionary process, the population remains a fixed size, i.e., the insertion of β new individuals substitutes the β worst individuals in the population.

2.2. Genetic Algorithm with Local Search

Local search is a post-optimization procedure that allows a systematic search in a solution space by using

a neighborhood structure [5].

In this work two neighborhood structures were proposed, aimed at increasing the amount of collected oil for a feasible solution for the problem (base solution). The pseudo-code shown in Figure 2 depicts how both neighborhood structures are systematically applied. The process starts with the first neighborhood (k=1) and whenever an improvement movement is found, it restarts the search in the first neighborhood. When no improvement movement is found, it proceeds to the next neighborhood or finishes the search (if k=2). In both neighborhoods, a list L_r of wells not yet included in the current solution is created. In the first neighborhood, we try to add in the current solution each well pertaining to list L_r, searching for the first possible position for insertion. In the second neighborhood, we try to exchange each well from the current solution with one from list L_r.

```
1.  Procedure LocalSearch
2.  Input: individual
3.  Select the neighborhood structures N_k, k = {1, 2};
4.  s ← individual; k ← 1;
5.  while k ≤ 2 do
6.      s' ← FirstImprovement(N_k(s));
7.      if f(s') > f(s) then
8.          s ← s';
9.          k ← 1;
10.     else
11.         k ← k + 1;
12.     end if
13. end while
14. return s.
```

Figure 2. Pseudo-code of local search procedure

2.3. Genetic Algorithm with Data Mining

With the aim of accelerating the occurrence of high quality solutions in the population, we proposed the incorporation of a Data Mining module in the GA. This module aims to discover patterns (subroutes) which are commonly found in the best solutions of the population. This approach significantly differs from current applications that combine genetic algorithms and data mining, because until now, most of the efforts deal with the development of GAs as optimization methods to solve Data Mining problems [15], such as the discovery of association and classification rules and clustering, which is not our case.

The process starts with the creation of an Elite Set of solutions (ES). This subset of population is formed by the s best solutions found so far and is updated whenever an individual that is better than the worst solution of the ES and different from all ES solutions is generated. The ES will be the database in which we will try to discover relevant patterns. To do this, we developed a slight modified version of Apriori algorithm [1], which discovers frequent sequences (instead of association rules) in the ES. The algorithm receives the minimum support as an input parameter, i.e., the minimal statistical significance that one sequence must satisfy to be considered frequent. Thus, the algorithm will discover all sequences (subroutes in the ES) of all sizes, which satisfy the minimal support. For instance, a support of 0.5, indicates that at least half of ES solutions must incorporate the considered sequence. Once a set of frequent subsequences is available, it is used to guide the construction of new individuals, in the following way: new individuals are built using a constructive algorithm like stated in section 2.1, except that whenever a well is selected to be added to the partial solution, a set of valid subroutes is built. This set contains only frequent subsequences whose wells do not appear in the partial solution, starting with the selected well. If more than one subroute is found, we randomly choose one from available subsequences. The subsequence chosen is incorporated in partial solution. Since time limit constraint can easily be violated through the addition of a subsequence of wells, a reparation operator is applied to remove the last wells, if necessary. If no valid subroute is found in data mining results, only the selected well is added.

Each time that the Data Mining module is triggered, β new individuals are generated using data mining information. As in crossover operator, population remains a fixed size. This process is applied at a fixed interval of generations μ.

2.4. Genetic Algorithm with Data Mining and Local Search

The application of Data Mining can be used together with Local Search procedures, through the application of Local Search in new individuals from crossover operator and from Data Mining procedure. This configures the last version of GA proposed here, named Genetic Algorithm with Data Mining and Local Search. The pseudo-code for this algorithm is presented in Figure 3. Initial population is generated using the constructive procedure (function GRC, line 3), which was described in section 2.1. Function

Offspring (line 7) indicates the application of crossover operator using *np* parent solutions from population *P* to generate each one of the β new individuals. To keep a fixed population size through generations, β worst individuals of population (function Worst(*P*, β)) are removed whenever β new individuals are included. In the application of Data Mining (lines 15 and 16), β new individuals are generated using the discovered frequent sequences *SEQ* by the GRCDM(*SEQ*, β) procedure (Greedy Randomized Constructive with Data Mining), whose functioning was described in section 2.3. The best solution of all generations (with the highest value of fitness function $f(x)$, where x is the evaluated solution) is kept and returned by the algorithm. Simpler versions of this algorithm (GA without Local Search and/or Data Mining) are just as the algorithm in Figure 3, except by the removal of modules of Data Mining (lines 15-21) and/or Local Search (function LocalSearch).

```
1.  Procedure GADMLS
2.  Input: popSize, np, β, α, sup, μ
3.  P = GRC(α, popSize);
4.  gen ← 1; NewIndividuals ← ∅; EliteSet ← ∅;
5.  bestSolution ← ∅; bestOfGen ← ∅;
6.  while not StoppingCriterionReached() do
7.    NewIndividuals ← Offspring(P, β, np);
8.    for each ind ∈ NewIndividuals do
9.      ind ← LocalSearch(ind);
10.   end for
11.   P ← P ∪ NewIndividuals;
12.   P ← P - Worst (P, β);
13.   Update EliteSet using P;
14.   if ( (gen mod μ)= 0 ) and (gen>0) then
15.     Discover sequences SEQ with minimum support
        sup in EliteSet;
16.     NewIndividuals ← GRCDM(SEQ, β);
17.     for each ind ∈ NewIndividuals do
18.       ind ← LocalSearch(ind);
19.     end for
20.     P ← P ∪ NewIndividuals;
21.     P ← P - Worst (P, β);
22.   end if
23.   bestOfGen ← ind ∈ P | f(ind) > f(x) ∀ x ∈ P;
24.   if f(bestOfGen) > f(bestSolution) then
25.     bestSolution ← bestOfGen;
26.   end if
27. end while
28. return bestSolution.
```

Figure 3. Pseudo-code for GADMLS algorithm

3. Computational Results

To our best knowledge no set of instances were made publicly available to the OCVRP. Thus a set of instances (Table 1) with different characteristics were generated. Symmetric instances incorporate distances from instances from TSP-Library problems [6]. The source problems for symmetric instances are shown in Table 1. For this kind of instance, well production was randomly generated in the range [1,100000]. Asymmetric instances were randomly generated with distances in the range [1,1000]. Well production for asymmetric instances varies from 1 to 100. The vertices column corresponds to the number of wells including OTS (the first vertex) for each instance. The time limit constraint for all instances was defined in a way that no trivial solution including all wells could be found.

Two sets of experiments were done to evaluate the proposed algorithms. Different algorithms were built by incorporating into our GA algorithm modules of Local Search (GALS), Data Mining (GADM) or both (GADMLS).

Table 1. Problem instances

Problem	Type	Source	Vertices
1	Symmetric	rat99	99
2	Asymmetric	--	300
3	Symmetric	pr439	439
4	Symmetric	d493	493
5	Asymmetric	--	500
6	Asymmetric	--	500
7	Symmetric	d657	657
8	Asymmetric	--	1,000
9	Asymmetric	--	1,000
10	Symmetric	pr1002	1,002

The execution parameters (Table 2) were defined through previous experiments not reported here.

Table 2. Execution parameters

Parameter	Value
Population size	500
Parents for crossover (*np*)	2
New individuals by generation (β)	50
Randomization degree (α)	0.5
Minimal support	0.5
Tournament size	2
Data Mining application interval (μ)	20

In the first set of experiments, the objective was to verify the average solution quality provided by each algorithm when executed in fixed time. The following

time limits where imposed: 1,800 seconds for instances 1 and 2; 2,700 seconds for instances 3-7 and 3600 for instances 8-10.

Average results from 4 executions for each instance are presented in Table 3. As can be seen, the GA algorithm often presents very poor results. Although GALS and GADM significantly improve the solution quality, there is not a clear dominance between them. The best results for most of instances occur in the GADMLS algorithm.

Table 3. Average percentage distance from best know solution

Problem	GA	GALS	GADM	GADMLS
1	14.44	7.19	5.14	**0.00**
2	6.78	2.71	1.59	**0.00**
3	4.12	1.93	**0.00**	0.91
4	5.09	2.23	1.92	**0.00**
5	6.90	1.69	3.17	**0.00**
6	6.21	0.37	3.26	**0.00**
7	4.09	3.26	1.08	**0.00**
8	5.93	0.25	0.98	**0.00**
9	6.23	**0.00**	2.13	0.41
10	1.16	2.59	**0.00**	1.85

In another set of experiments, the objective was to verify the empirical probability distribution of reaching a given sub-optimal target solution value (i.e. find a solution with value as well as the target solution value or better) in function of time in different instances. The sub-optimal values were chosen in a way that the slowest algorithm could terminate in a reasonable amount of time. The execution times of 100 independent runs for each instance were computed. The experiment design follows the proposal of [2]. The results of each algorithm were plotted by associating the i-th smallest running time t_i with a probability $p_i=(i\text{-}0.5)/100$, which generates points $z_i=(t_i,p_i)$, for $i=1, \ldots,100$. Although we ran the experiment for all instances, we included only the results of tests (Figure 4) for one problem, whose results illustrate the typical result obtained for all problems. A simple analysis of the results can be done considering the alignment of curves: leftmost aligned curves indicate faster convergence of the algorithm, while rightmost aligned curves indicate an algorithm with slower convergence. The results show that the simplest version (GA) takes considerably more time to achieve high cumulative probability values (>0.5). By means of an example, there is a probability of 50% of GA to reach the target at 1,250 seconds, while for other algorithms it takes approximately 850 seconds. As can be seen in Figure 4, the superiority of GADMLS algorithm upon other versions is not explicit as the results showed in Table 3. We believe that this result occurred due to the fact that the target solution value was easier to find that the average solution quality produced by GADMLS in experiments with fixed time. The choice for an easy target solution value was done because harder targets would consume too much time in experiments with GA.

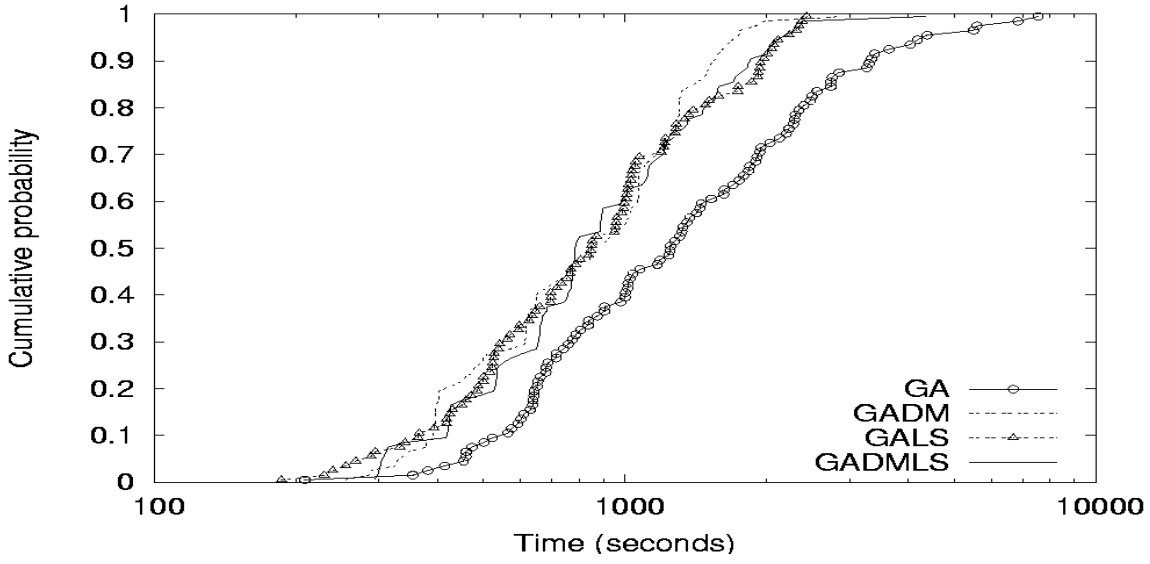

Figure 4. Cumulative probability distribution of reaching target 18,300,000 for problem 7

4. Conclusions and Future Works

In this work we presented three improved versions of an evolutionary algorithm. Versions which include Local Search and/or Data Mining were presented. Although applications of genetic algorithms with local search are abundant in the literature, the application of Data Mining to improve the results of evolutionary algorithms is still scarce. The Data Mining module proposed corresponds to an intensification strategy, since it tries to discover good features in the best solutions found so far and to apply them in the generation of new solutions. The results show that hybrid versions provide better results than the pure genetic algorithm version. In most cases, the best results were found with the application of Data Mining joined with a Local Search procedure.

Since these hybrid versions consume considerable computational resources, a promissing research area is the development of parallel versions.

Acknowledgments

The authors are grateful to CNPq and CAPES that partially funded this research.

References

[1] R. Agrawal, and R. Srikant, "Fast algorithms for mining association rules", *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*. Morgan Kaufmann, 1994, pp. 487-499.

[2] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro, "Probability distribuition of solution time in GRASP: an experimental investigation", *Journal of Heuristics*, 8, 2002, pp. 343-373.

[3] D.J. Aloise, J.A. Barros, and M. Souza, "A genetic algorithm for a Oil Retrieval System" (In Portuguese), *Proc. of the XXXII Brazilian Symposium on Operations Research*, São Paulo, 2000.

[4] E. Balas, "The Prize Collecting Traveling Salesman Problem", *Networks*, 19, 1989, pp. 621-636.

[5] E.H.L. Aarts, and M. Verhoeven, "Local search", *Annotated Bibliographies in Combinatorial Optimization,* (M. Dell'Amico, F. Maffioli, and S. Martello, eds.), Wiley, New York, 1997, pp. 163-180.

[6] G. Reinelt, "TSPLIB - A traveling salesman problem library", *ORSA J. Comput.*, 3, 1991, pp. 376-384.

[7] J. Beasley, "Population Heuristics", *Handbook of Applied Optimization*. Oxford University Press, Oxford, 2002, pp. 138-157.

[8] J.L Bresina, "Heuristic-biased stochastic sampling", *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, 1996, pp. 271-278.

[9] R.R. Colin, "Genetic algorithm", *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill book company, 1995, pp. 151-196.

[10] L.M.A. Drummond, D.S Vianna, and L.S. Ochi, "An evolutionary hybrid metaheuristic for solving the vehicle routing problem with heterogeneous fleet", *Lecture Notes in Computer Science*, 1391, 1998, pp. 187-195.

[11] L.M.A. Drummond, L.S. Vianna, M.B. Silva, and L.S. Ochi. "Distributed Parallel Metaheuristic based on GRASP and VNS for solving The Traveling Purchaser Problem", *Proc. of the 2002 Int. Conf. on Parallel and Distributed Systems*, Taiwan, 2002, pp. 257-263.

[12] F. Glover, M. Laguna, and R. Marti, "Fundamentals of Scatter Search and Path Relinking", *Control and Cybernetics,* 39 (3), 2000, pp.653-684.

[13] D.E. Goldberg. *Genetic Algorithms in Search Optimization & Machine Learning*. Addison-Wesley, Menlo Park, 1989.

[14] B.L. Golden, L. Levy, and R. Vohra, "The Orienteering Problem", *Naval Research Logistics*, 24, 1987, pp. 307-318.

[15] J. Han, and K. Micheline, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2000.

[16] J.H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, 1975.

[17] E.L Lawler, J.K Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, "The Traveling Salesman Problem", John Willey & Sons, 1985.

[18] P. Moscato, "On Evolution, Search, Optimization Algorithms and Martial Arts: Towards Memetic Algorithms", Report 826, Caltech Concurrent Computation Program, California Institute of Technology, 1989.

[19] L. S. Ochi, D.S. Vianna, and L.M.A. Drummond, "An Asynchronous Parallel Metaheuristic for the Period Vehicle Routing Problem", *Future Generation Computer Systems*, 17, 2001, pp. 379-386.