Relatório - Aplicação do Memético

Giovanne da Silva Santos

12 de janeiro de 2021

Resumo

O gás lift é uma das técnicas artificiais mais comum de injeção para a produção de óleo. E isto está vinculado com o aumento de velocidade e precisão na injeção de gás neste petróleo. Contudo, a aplicação demasiada de gás em certo poços, prejudicará na produção de óleo por conta da pressão e atrito que o gás causará nos dutos em que o óleo percorre, assim como o desperdício de gás que afetará o produtor de petróleo economicamente. Além disso, caso não aplique-se gás suficiente, então não haverá pressão suficiente para ejetar o óleo presente no poço. Por conta disso, a resolução do problema de otimização do gás lift é essencial na produção de petróleo a fim de minimizar os custos e maximizar a produção.

1 Introdução

Caso um determinado reservatório de petróleo não consegue emergir os fluídos até a superfície, então uma técnica de injeção faz-se necessário, mas tal técnica terá de ser aplicada da melhor forma e otimizada possível para fins econômicos. Uma técnica artificial de injeção reduz a pressão da coluna do fluido causando, assim, uma redução da pressão do fundo poço. Por conta disso, uma grande diferença de pressão entre o reservatório e o fundo do poço é criada, e, consequentemente, os fluidos irão emergir até a superfície. O processo de gás lift é umas das técnicas artificiais de injeção mais utilizadas. No gás lift, o gás é injetado na parte inferior da tubulação, e o óleo é produzido através da impulsão feita pela expansão do gás e na redução de pressão hidrostática da coluna do fluido que está dentro do poço.

O algoritmo desenvolvido neste trabalho é o método de enxame de partícula, ou PSO. No qual é uns dos principais métodos da computação baseado no comportamento da natureza. Tal técnica será embasada e explicada na seção de fundamentação teórica, assim como o pseudo-algoritmo. Além disso, será mostrado o algoritmo implementado, baseado nesse método, na seção de descrição do algoritmo implementado e as análises e conclusões dos resultados deste algoritmo na seção de resultados obtidos.

2 Descrição do problema

No problema de otimização de gás lift, temos dois cenários. O primeiro cenário consiste em maximizar a produção de óleo dado uma quantidade suficiente disponível de gás para injeção, e o segundo cenário tem o objetivo de minimizar a quantidade de gás dado uma quantidade de óleo produzido pré-planejada. A otimização, de um modo geral, leva em conta com a curva de performance do gás lift e fatores econômicos.

Para o cenário em que queremos maximizar a produção de óleo e tendo uma quantidade suficiente de gás, temos a seguinte função objetiva:

$$max \sum_{1}^{n} Q_{o_i} = f(Q_{g_1}, Q_{g_2}, ..., Qg_i), i = 1, 2, ..., n$$

sendo que a quantidade total de gás injetada nos poços deverá ser igual ou menor que a quantidade de gás disponível conforme é representado nesta equação (Hamedi[3]):

$$\sum_{1}^{n} Q_{g_i} \leq A$$

onde Q_{o_i} é quantidade de óleo produzido por poço, n é a quantidade de poços, Q_{g_i} é quantidade de gás injetada em cada poço e A é a quantidade de gás disponível.

No segundo cenário, que é o cenário estudado neste trabalho, queremos minimizar quantidade de gás injetado e tendo uma quantidade de produção de petróleo já pré-determinada, teremos:

$$min \sum_{1}^{n} Q_{g_i}, i = 1, 2, ..., n$$

em que a quantidade total de petróleo será igual a quantidade já pré-determinada, como mostrado a seguir:

$$\sum_{1}^{n} Q_{o_i} = f(Q_{g_1}, Q_{g_2}, ..., Qg_i) = B$$

sendo B a quantidade pré-determinada.

tendo também estas restrições em ambos os cenários:

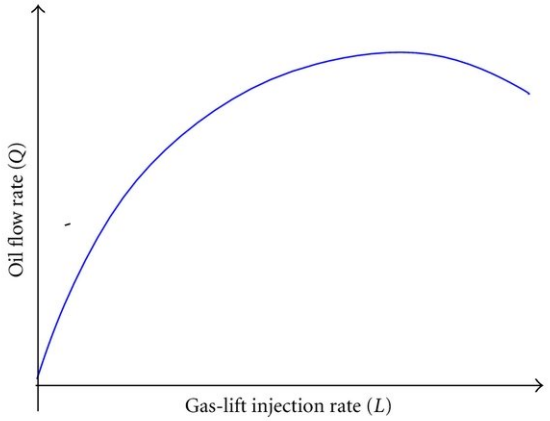
$$Q_{g_i} \geq Q_{g_i min}$$

$$Q_{g_i} \leq Q_{g_i max}$$

Além disso, vale salientar que alguns poços produzem petróleo naturalmente ou que precisam de muito pouco gás injetado, assim como temos poços que não respondem bem à injeção de gás. Ademais, há os poços "mortos"no qual é necessária uma quantidade miníma de gás igual ou maior do que $Q_{g_i min}$ para

poder produzir petróleo. Assim como a quantidade de gás não pode exceder em um poço, ou seja, ser maior do que Q_{g_imax}, a fim de não haver o decaimento da produção de petróleo por conta da percar de pressão que pode ocasionar na injeção do gás.

A quantidade máxima Q_{g_imax} e miníma Q_{g_imin} são obtidas através curva de performance do gás lift(GLPC) é ilustrado na imagem a seguir:



Em que temos no eixo y a produção de petróleo e no eixo x a quantidade de gás injetado. Como podemos ver, a quantidade ideal de gás injetado em um poço, deverá ser no pico da curva caso tenhamos uma quantidade de gás suficiente, senão, teremos que, ao menos, determina uma quantidade miníma de gás de tal modo que produza a maior quantidade possível ou pré-determinada de petróleo.

Para representar a curva GLPC, temos o modelo introduzido por Alarco et al[1] para gerar a equação que calcula a produção de petróleo a partir de uma gás injetado que é:

$$Q_o = a + b \times Q_g + c \times Q_g^2 + d \times \ln(Q_g + 1)$$

, em que a, b, c e d são os coeficientes que cada poço irá ter.

3 Fundamentação teórica

No estudo do algoritmo genético, temos as seleções, recombinações e mutações para a criação de novos indivíduos. Contudo, neste procedimento há a passagem de genes ruins aos indivíduos também. Por conta disso, há o estudo memético em que traz uma alteração não genética para o melhoramento da população. No algoritmo memético, temos a implementação do algoritmo genético e da busca local após a recombinação e após a mutação.

No algoritmo memético no temos que a recombinação corresponde à etapa de troca de informação entre os cromossomos da população. Já o cruzamento, há o cruzamento de informação através do crossover entre os indivíduos.

A seguir teremos o pseudo-algoritmo do algoritmo genético:

Algorithm 1 Memético

```
1:  procedure MEMÉTICO
2:      t ← 0
3:      Inicialize S(t)
4:      Avalie S(t)
5:      while  o critério de parada não for satisfeito do
6:          t ← t+1
7:          Selecione S(t) a partir de S(t-1)
8:          Aplique crosssover sobre S(t)
9:          Aplique a busca local sobre S(t)
10:         Aplique mutação sobre S(t)
11:         Aplique a busca local sobre S(t)
12:         Avalie S(t)
13:     end while
14:     retorne o melhor indivíduo de S(t)
15: end procedure
```

Na linha 2, temos a t a iteração/geração atual que no momento é 0, em seguida, na linha 3, há a inicialização da população S(t) da geração t. Depois, é avaliado o melhor individuo em S(t) de acordo com a função objetiva. No laço, da linha 5 a 11, temos o incremento para a próxima geração, a seleção dos indivíduos para a população S(t) a partir de S(t-1), a partir de algum critério de seleção, aplicação do crossover, busca local, mutação e busca local, novamente, entre a população S(t) e, por fim, a avaliação dos indivíduos de S(t). No final, é retornado o melhor indivíduo.

4 Descrição do algoritmo implementado

Neste seção será explicado as classes e funções mais relevantes para resolução do problema utilizando as técnicas explicadas na seção passada.

Em relação ao tratamento das entradas, criação e consulta dos poços(wells), serão o mesmo dos trabalhos passado.

Primeiramente, falaremos do tipo de dado Indiviual que terá uma solução da classe Solution, uma variável inteira para o índice, uma booleana "selected"para auxiliar na seleção e o construtor que o inicializará uma solução, receberá o índice, e "selected"inicializada como falsa. Esse tipo de dados representará o indivíduo de uma população.

Em seguida, temos a classe Memético que terá como atributo o vetor de poços(wells), quantidade de petróleo pré-planejada, número de indivíduos iniciais(n_initial), número de gerações/iterações(n_generations) e o construtor:

Algorithm 2 Memético

```
1: procedure MEMÉTICO(wells,oilReq,n_initial, n_generations)
2:     population ← Ø
3:     for  i from 0 to n_initial do
4:         Inicialize Inidividual inidividual(wells,oilReq,i)
5:         population ← population ∪ individual
6:     end for
7: end procedure
```

Neste construtor, é inicializado o individuo com os poços, quantidade de petróleo pré-planejada e o índice correspondente. Além disso, quando o individuo é inicializado, é inicializado uma solução com o construtor da solução já mostrado nos trabalhos anteriores, ou seja, o individuo começará, com os valores do seu vetor, aleatoriamente. Em seguida, ocorre a junção dele a um vetor de Individuals chamado population. Além disso, será passado o índice referente ao indivíduo inicializado para efeito de ordenação no algoritmo genético. A complexidade será O(n), sendo n o número de indivíduos inicial.

Agora teremos a função selection que será definido a seguir:

Algorithm 3 selection

```
1: procedure SELECTION
2:     Ordene population baseado na função objetivo
3:     parent1 ← um indivíduo aleatório da melhor metade de population
4:     parent2 ← um indivíduo aleatório de population
5:     Remova parents1 de population
6:     Remova parents2 de population
7:     parents ← parent1,parent2
8:     Retorne parents
9: end procedure
```

No selected, é selecionado o primeiro pai baseado no elitismo sendo a melhor primeira metade. Em seguida, o segundo pai é escolhido aleatoriamente dentre a população. A complexidade será O($n \log$) por conta da ordenação sort().

Em seguida, teremos o crossover:

Algorithm 4 Crossover

```
1:  procedure CROSSOVER(parents)
2:      parent1 ← parents[0]
3:      parent2 ← parents[1]
4:      getIndex ← Um índice aleatório entre 0 e número de poços
5:      child1 [0 até getIndex] ← parent1[0 até getIndex]
6:      child1 [getIndex,numeroDePoco] ← parent2 [getIndex,numeroDePoco]
7:      child2 [0 até getIndex] ← parent2[0 até getIndex]
8:      child2[getIndex,numeroDePoco] ←parent1 [getIndex,numeroDePoco]
9:      offspring ← parent1,parent2,child1,child2
10:     Retorne o melhor individuo de offspring
11: end procedure
```

Na linha 2 e 3 apenas temos a atribuição dos dois pais. Em seguida, na linha 4, temos a escolha aleatória de um índice entre 0 e o número de poço(tamanho da solução também), na linha 5 e 6 temos a atribuição do primeiro filho que terá a primeira parte, que é de 0 até o índice aleatório escolhido, igual ao primeiro pai e a segunda parte, que é do índice aleatório até o final, igual ao segundo pai. No segundo filho será o mesmo princípio, porém invertendo os pais em relação a atribuição dos valores no vetor do segundo filho. Por fim, na linha 9, temos a alocação de toda descendência(offspring) com os dois pais e dois filhos. No final é retornado a descendência. A complexidade será O(n), pois será percorrido toda a solução para a atribuição dos valores.

Agora teremos a função mutation:

Algorithm 5 Mutation

```
1:  procedure MUTATION(offsprings)
2:      off1 ← melhor solução de offsprings
3:      off2 ← segunda melhor solução de offsptrings
4:      mutationParameters ← Escolha aleatoriamente um valor entre -0.2 e 0.2
5:      indice ← Escolha um valor aleatório entre 0 e tamanho da solução
6:      off1 [indice] += mutationParameters
7:      Randomize novamente os valores de mutationParameters e indice
8:      off2 [indice] += mutationParameters
9:      offsprings[0] ← off1
10:     offsprings[1] ← off2
11: end procedure
```

Na linha 2 e 3 temos a escolha das duas melhores soluções da descendência. Na linha 4 temos a escolha aleatória da mutação entre -0.2 e 0.2 baseado em Zerafat et al[5]. Já na linha 5 temos a escolha aleatória de um índice que será para escolher o valor do vetor solução que será alterado. Na linha 8, teremos o mesmo procedimento com o valor do índice e mutação re-randomizados. Por fim, teremos a atualização dos dois melhores resultados. A complexidade será O(1), pois é apenas foi modificado os valores diretamente da soluções.

Por fim, teremos o algoritmo do run que executará de fato o algoritmo genético:

Algorithm 6 run

```
1:  procedure RUN
2:      for i from 0 to nº de gerações do
3:          parents ← selection()
4:          offsprings ← crossover(parents)
5:          offsprings ← busca_local(offsprings)
6:          offsprings ← mutation(offsprings)
7:          offsprings ← busca_local(offpsprings)
8:          population ← population ∪ offspringsMutationed[0]
9:          population ← population ∪ offspringsMutationed[1]
10:         Ordene population
11:     end for
12:     Retorne a melhor solução de population
13: end procedure
```

No algoritmo temos a lógica parecida com a do algorítimo memético explicado na seção de fundamentação teórica. Com a etapa de seleção, crossover, busca local, mutação e busca local, novamente, em seguida, o retorno da melhor solução com base na função objetiva. A busca local implementada é a mesma do trabalho do GRASP.

5 Descrição dos experimentos computacionais

As instâncias utilizadas foram retiradas no Buitrago et al[2], lá temos duas tabelas, um com 6 poços e outras com 56 poços, além de que foi criado uma instância própria com 24 poços. Em run_experiments, é feito a leitura em do txt que tem, na primeira linha o número de poços, e a sequência, em seguida, da linha de gás injetado em u poço e na próxima linha o petróleo produzido no poço com base nos valores do gás injetado na linha anterior e assim sucessivamente com cada dupla de linha representando um poço. A saída do programa grava em um arquivo e imprime no termina o número de poços da instância, quantidade de petróleo pré-planejada, o vetor da solução gerada, petróleo produzido, a quantidade mínima e a máxima de petróleo possível e o gás injetado a partir da solução encontrada. A partir dessas instâncias, utiliza-se um método gaussiano para calcular os coeficientes de cada poço que irão calcular a quantidade de petróleo baseado no gás injetado no poço. Foi utilizado os parâmetros de Zerafat et al.[5]

Primary population	Crossover	Mutation	Iteration
20	0.8, 2-points	0.2, Gaussian	51

O ambiente de teste foi um notebook Lenovo com processador Intel Core i7-7500U CPU 2.70GHz x 4, memória RAM de 3,6 GB e HD de 500 GB e sendo o sistema operacional o Ubuntu 20.0.

Além disso, os resultados, que serão mostrados na próxima seção, foram extraídos a partir da média entre 30 testes. Ademais, para cada instância(6, 24 e 56 poços) temos a primeira tabela sendo resultado do petróleo produzido e na segunda tabela o gás aplicado. Na primeira linha das tabelas de petróleo produzido, temos as quantidade pré-planejadas utilizadas como entrada.

Para a compilação do programa, através do terminal, basta digitar "make"na pasta do programa. Para execução do programa, será necessário entrar na pasta "build"e digitar ./exe.out [nome da instâncias]. Até o momento, temos as instâncias well-six.txt, well-fifty-six.txt e well-twenty-four.txt.

6 Resultados obtidos: análise e discussão

Nesta seção será analisado os experimentos através de tabelas e, para efeitos de comparação, será mostrado a quantidade de gás injetado pela melhor solução encontrada e a produção de petróleo produzido. Para efeito de comparação, será comparado os resultados obtidos com o PSO(enxame de partícula), ACO(Colônia de formiga) e GA(Algoritmo Genético).

Primeiro, será apresentado os resultado com 6 poços que terá uma produção mínima de petróleo de 1772.9, utilizando 1587 de gás, e máxima de 4172.3, logo:

Qtd de petróleo pré-planejada	4772.9	3772.9	1772.9
PSO	3855.78	3773.69	1787.69
ACO	3069.87	3021.75	1819.34
Genético	3527.7	3021.75	1819.34
Memético	2517.83	2478.29	2513.38

Tabela 1: Média de petróleo produzido para 6 poços

PSO	8128.08	6863.55	1592.11
ACO	3401.29	2130.46	823.703
Genético	5111.06	5108.36	5118.73
Memético	1977.32	1894.51	1941.32

Tabela 2: Média de gás produzido para 6 poços

	Tempo(segundos)
Genético	0.586011
Memético	height

Tabela 3: Média de tempo produzido para 6 poços

Em resumo, temos o mesmo problema com o genético e no memético, com o memético encontrando valores de gás de injeção bem mais baixo assim como a quantidade de petróleo produzido. Contudo, o memético não encontrou a quantidade de gás injetado mais otimizado, tendo em vista que o ACO e PSO foram melhores ainda.

Os resultados com 24 poços que tem quantidade mínima de produção de petróleo de 10269, que utiliza 1564 de gás, e máximo de petróleo de 14222:

Qtd de petróleo pré-planejada	13623	12623	10623
PSO	13118.1	12624.1	10629
ACO	13935	13012.8	11433.9
Genético	12526.2	12519.3	12530.1
Memético	12048.3	12048.3	12072.6

Tabela 4: Média de petróleo produzido para 24 poços

PSO	18486.7	11029	2343.77
ACO	8259.29	7129.67	4725.94
Genético	10212.1	10156.7	10242
Memético	7361.82	7564.32	7502.03

Tabela 5: Média de gás produzido para 24 poços

	Tempo(segundos)
Genético	4.53191
Memético	height

Tabela 6: Média de tempo produzido para 24 poços

Já para 24 poços, o memético acho quantidade boas de gás injetado para quantidades razoáveis de petróleo produzido tendo desempenho melhor do que o genético, embora ainda seja inferior ao PSO e ACO em alguma ocasiões.

Já para os resultados da instância com 56 poços que tem quantidade mínima de produção de petróleo de 18163 com injeção de gás de 23425 e máxima de 25114:

Qtd de petróleo pré-planejada	22475	21475	19475
PSO	22460.2	21500.3	19489.8
ACO	16076.5	15611	15640.5
Genético	21845.8	21855.5	21851.1
Memético	14479.1	14626.7	14513.4

Tabela 7: Média de petróleo produzido para 56 poços

PSO	43477	360619.7	26311.1
ACO	21498	21353.9	21043.9
Genético	38830.3	38904.2	38869.8
Memético	28821.6	28882.3	28676.4

Tabela 8: Média de gás produzido para 56 poços

	Tempo(segundos)
Genético	
Memético	height

Tabela 9: Média de tempo produzido para 56 poços

Com 56 poços, tivemos um desempenho semelhante com 6 poços, sendo o memético encontra quantidade de gás injetado menor do que o genético mas não conseguindo ultrapassar a quantidade de petróleo pré-planejada. De todo modo, o PSO foi o que teve melhor desempenho.

7 Conclusão

Os problemas de otimização associados ao gás lift são bastante estudados, principalmente, por conta do valor econômico que traz com a resolução de tais problemas. Por conta disso, temos várias meta-heurísticas e modelagens aplicadas a fim de trazer melhores resultados. Contudo, o segundo cenário ainda não tem muitos estudos envolvendo ele, tornando-se necessário criar e aplicar novos algoritmos para a resolução como Namdar[4] mencionou.

Como podemos ver, o algoritmo memético trouxe quantidade de gás injetado bem menores, mas ainda está muito inferior ao PSO e ACO, tendo em vista que houve o mesmo problema com o genético, sendo necessário mudar certas estratégias como não permite a criação de indivíduos muito parecidos ou algum outro critério para poder resolver o problema do segundo cenário do gás-lift.

Referências

[1] Gabriel A Alarco´ n, Carlos F Torres, and Luis E Go´ mez. Global optimization of gas allocation to a group of wells in artificial lift using nonlinear constrained programming. *J. Energy Resour. Technol.*, 124(4):262–268, 2002.

[2] S Buitrago, E Rodriguez, D Espin, et al. Global optimization techniques in gas allocation for continuous flow gas lift systems. In *SPE gas technology symposium*. Society of Petroleum Engineers, 1996.

[3] H Hamedi, F Rashidi, and E Khamehchi. A novel approach to the gas-lift allocation optimization problem. *Petroleum Science and Technology*, 29(4):418–427, 2011.

[4] Hamed Namdar. Developing an improved approach to solving a new gas lift optimization problem. *Journal of Petroleum Exploration and Production Technology*, 9(4):2965–2978, 2019.

[5] Mohammad M Zerafat, Shahab Ayatollahi, and Ali A Roosta. Genetic algorithms and ant colony approach for gas-lift allocation optimization. *Journal of the Japan Petroleum Institute*, 52(3):102–107, 2009.