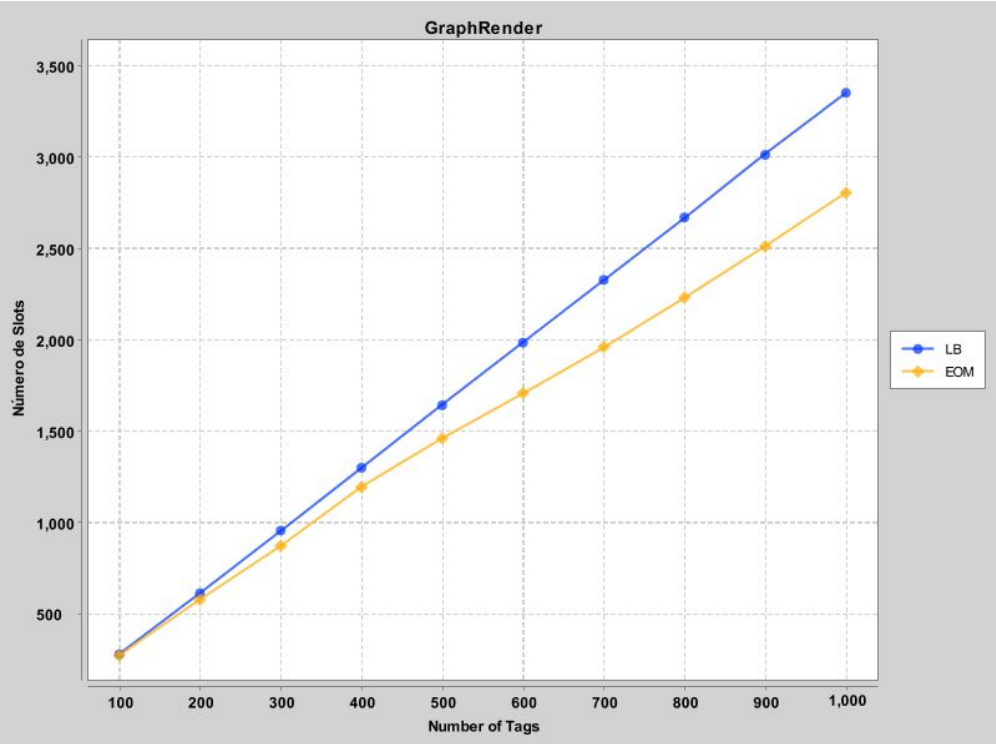
Estimadores para o DFSA

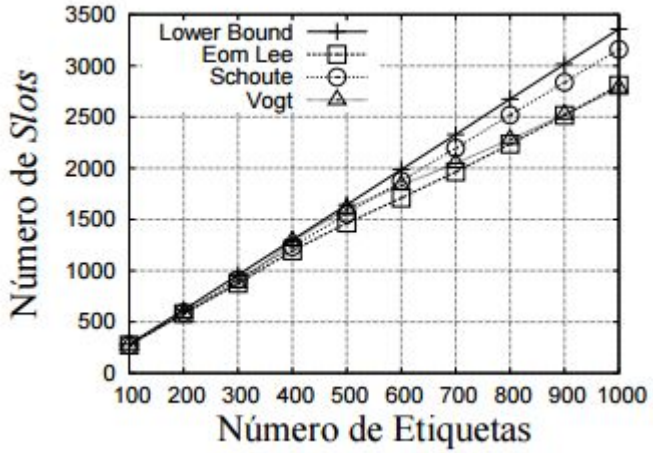
Atividade 5 - Atualizada com Chen, Vahedi, e correções

- Bruno Dutra de Lemos Neto (bdln)
- Danilo Alfredo Marinho de Souza (dams)
- Lucas de Souza Albuquerque (lsa2)
- Victor Nunes de Farias Neves (vnfn)

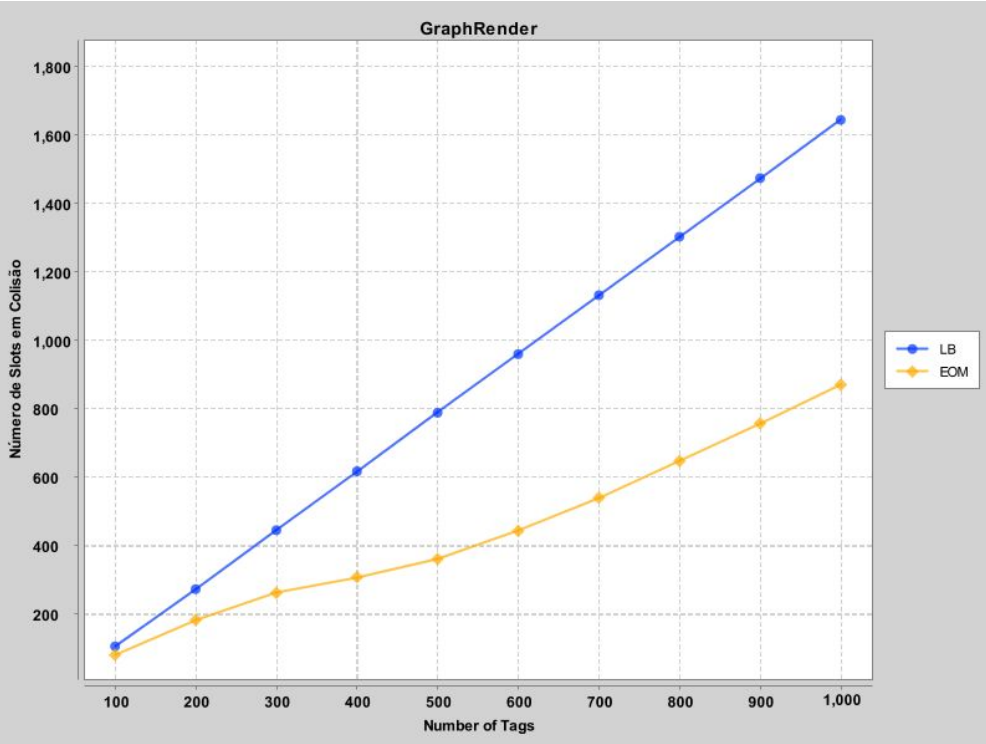
Lower Bound, Eom-Lee (Slots Totais)

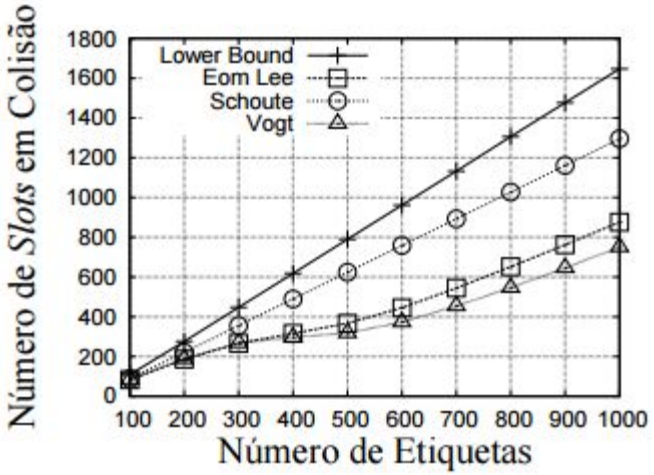


*Escalas foram ajeitadas para comparação com os gráficos

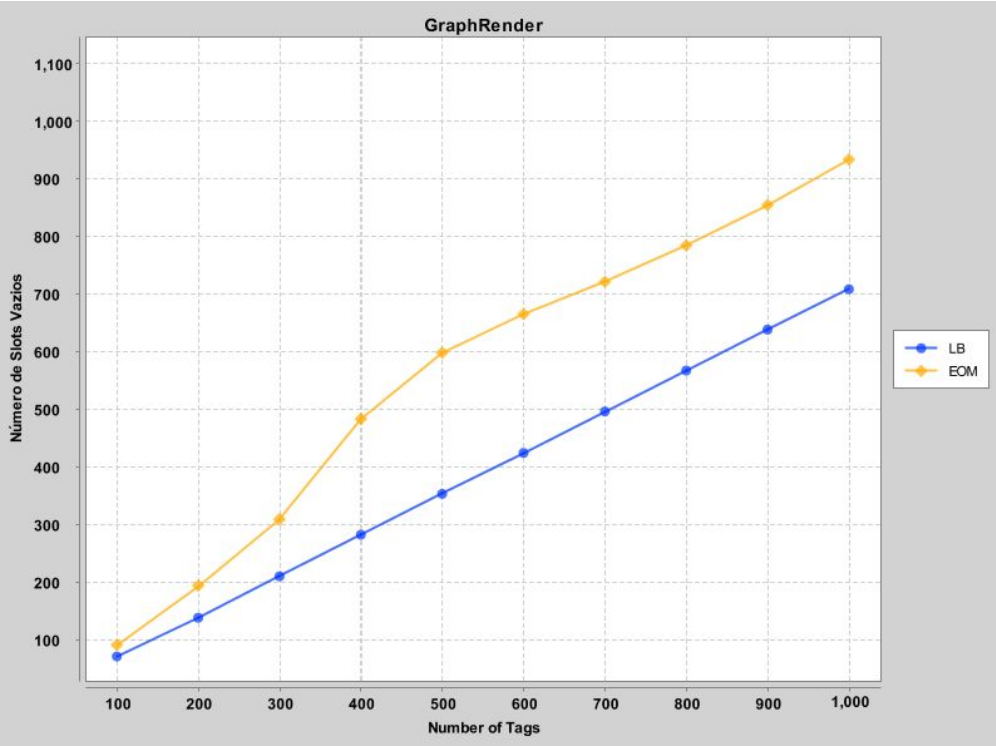


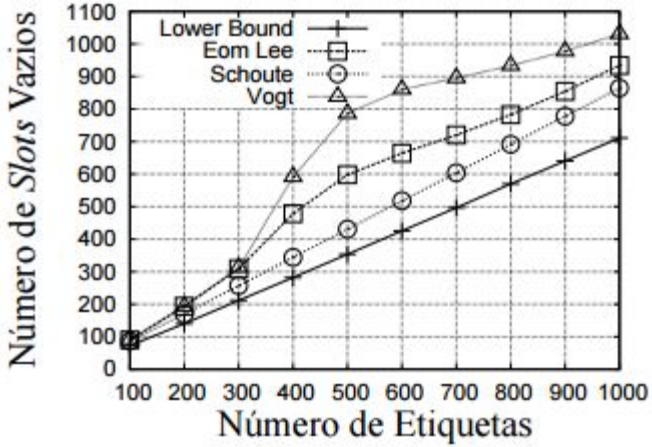
Lower Bound, Eom-Lee (Slots em Colisão)



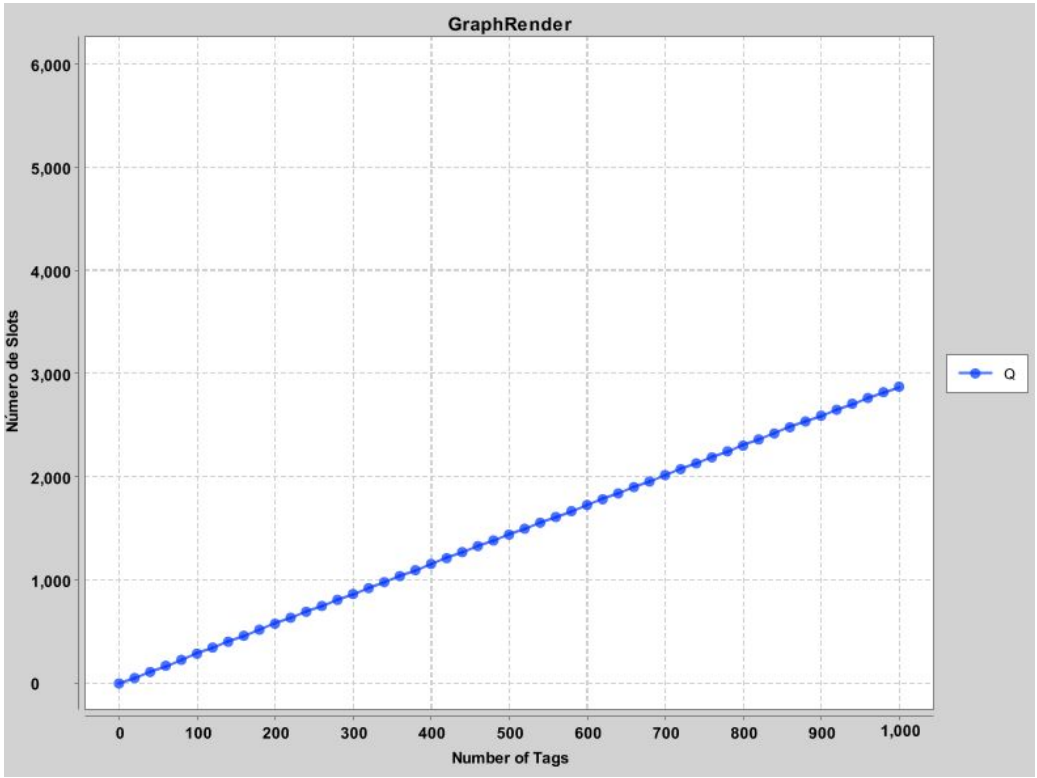


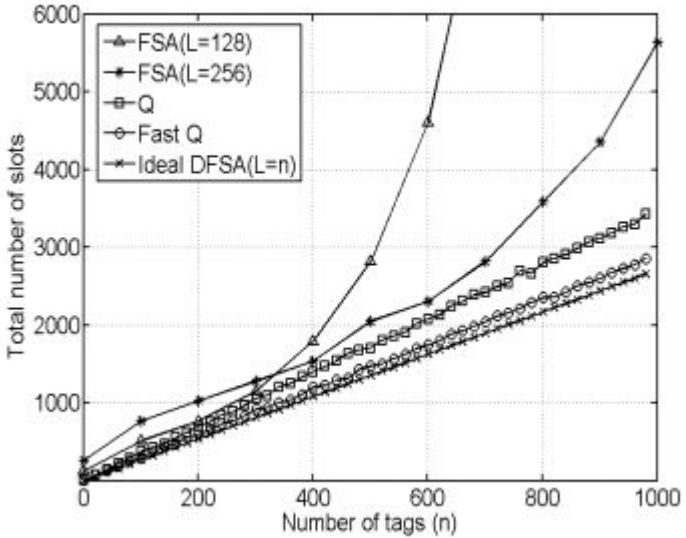
Lower Bound, Eom-Lee (Slots Vazios)





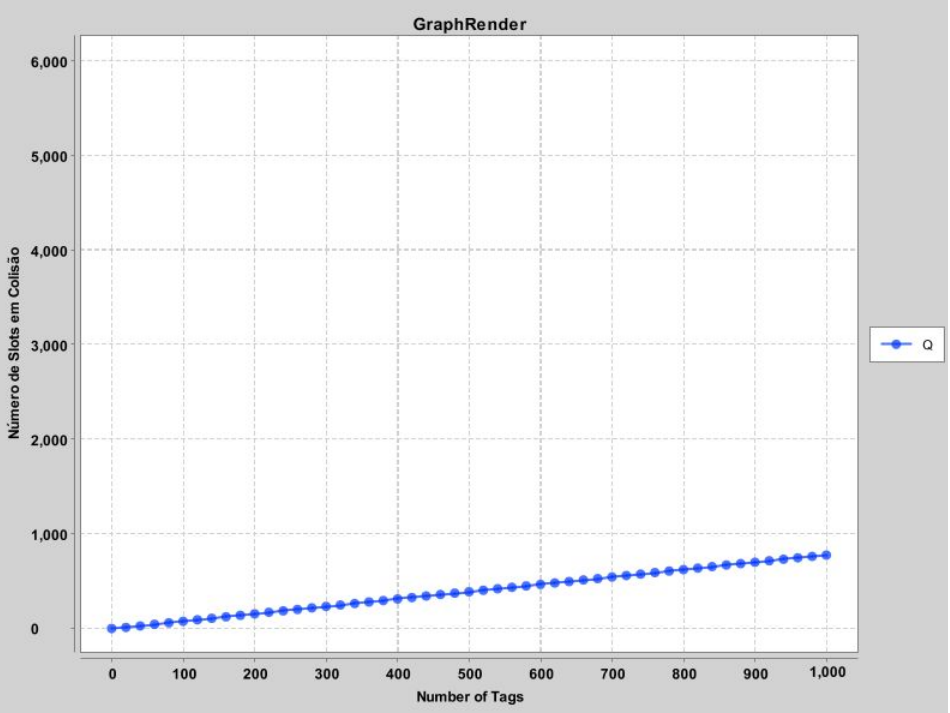
Q (Número de Slots) - **FAST-Q**

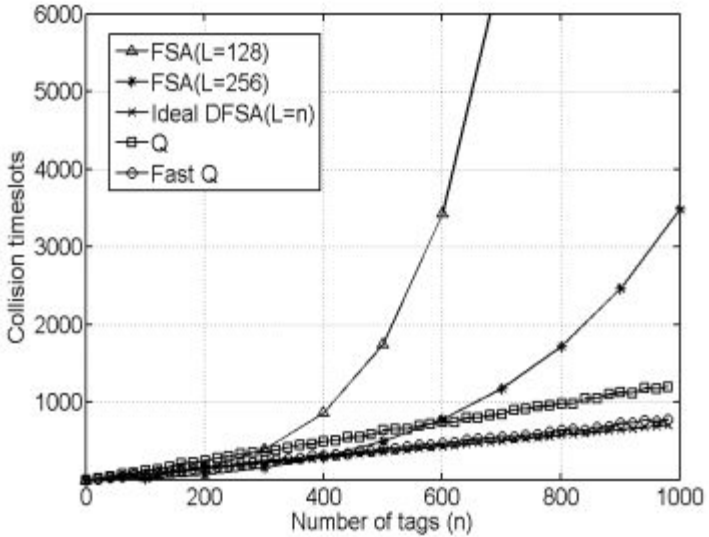




*A implementação do Q foi a mostrada no algoritmo da especificação: o FAST-Q com constantes diferentes para colisão *(Ccol)* e slots vazios *(Cidle)*

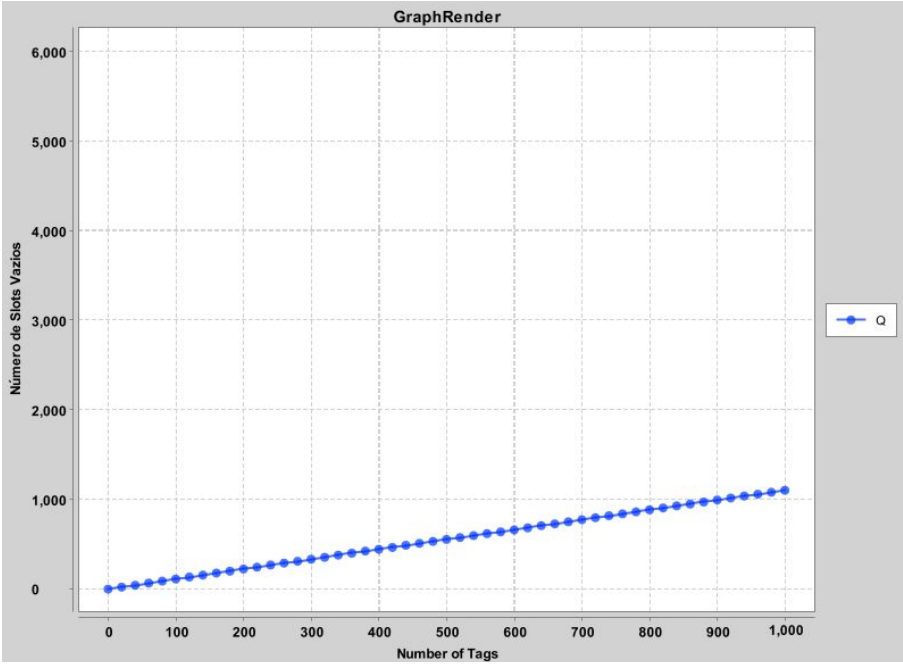
Q (Número de Slots em Colisão)

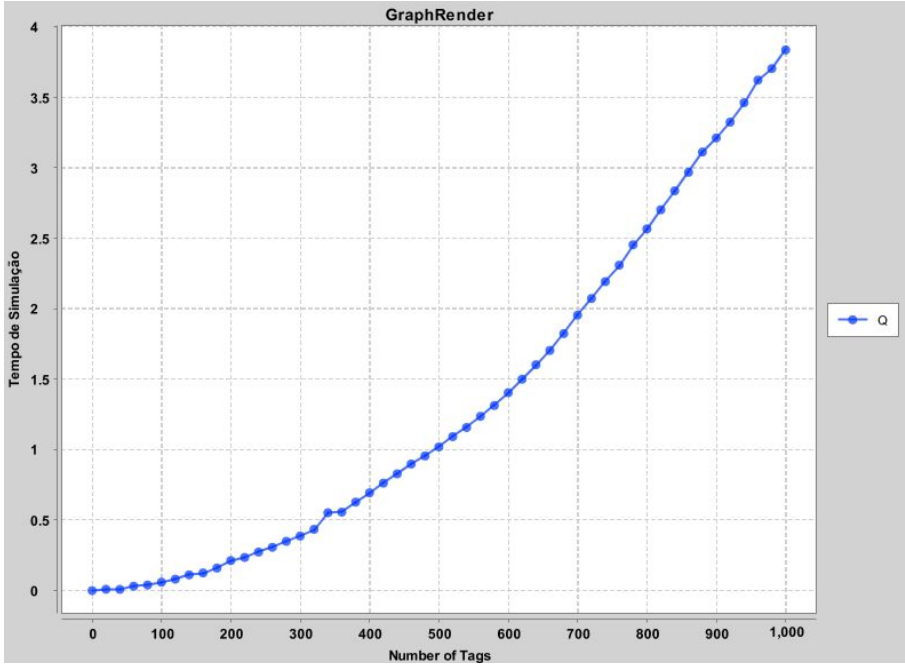




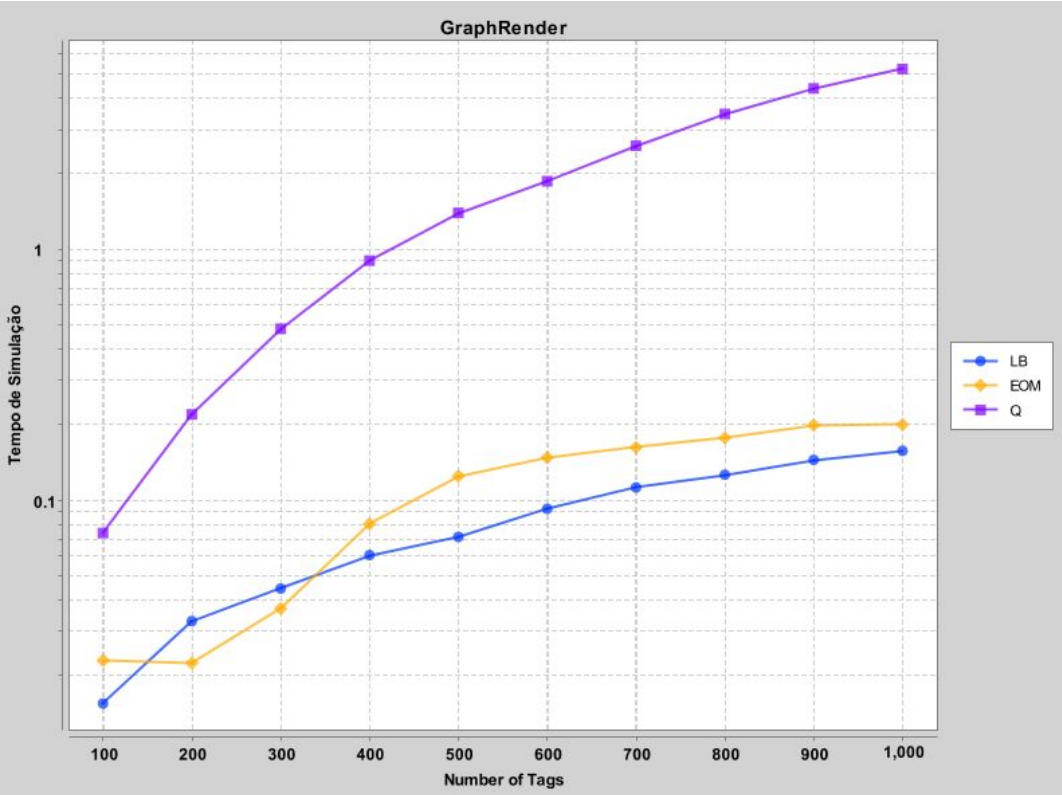
Q (Número de Slots Vazios e Tempo)

(todas as medidas de tempo são em milisegundos)





Lower Bound, Eom-Lee e Q (Tempo - Escala Logarítmica)

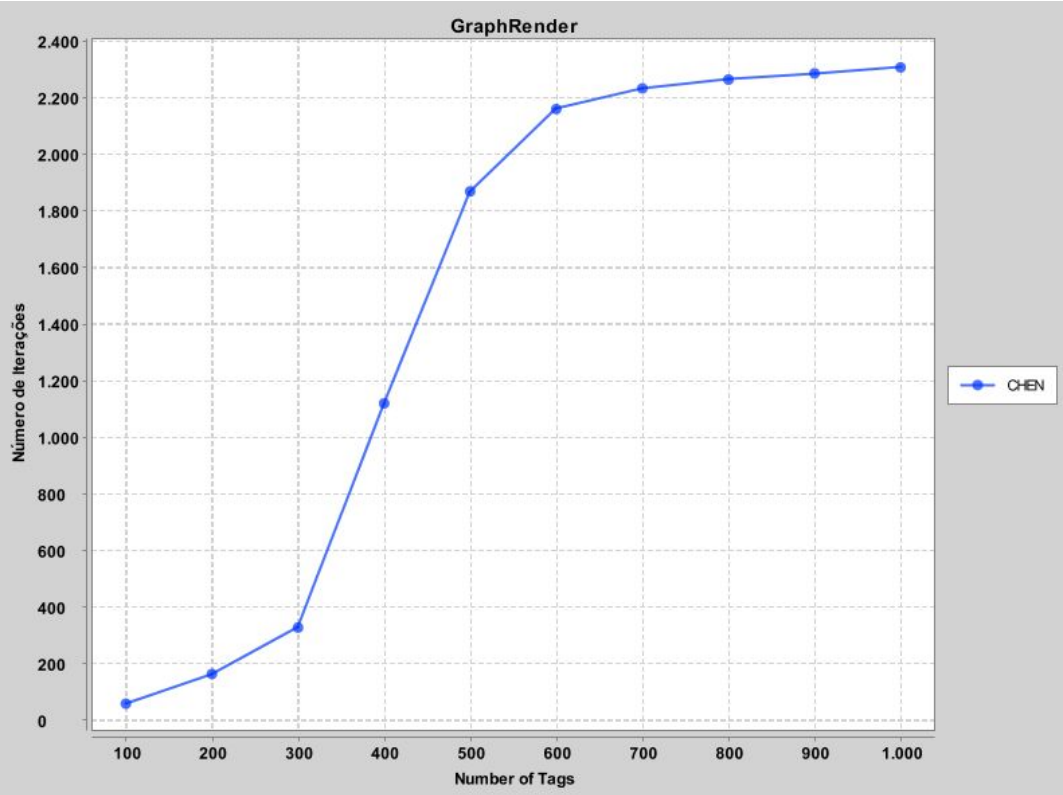


*Tempo agora apresentado em uma escala logarítmica.

*Após a primeira apresentação, foi-se corrigido um erro na implementação que fazia cálculos/loops desnecessários e tornava o tempo de cálculo do Lower Bound e Eom-Lee em um tempo exponencial

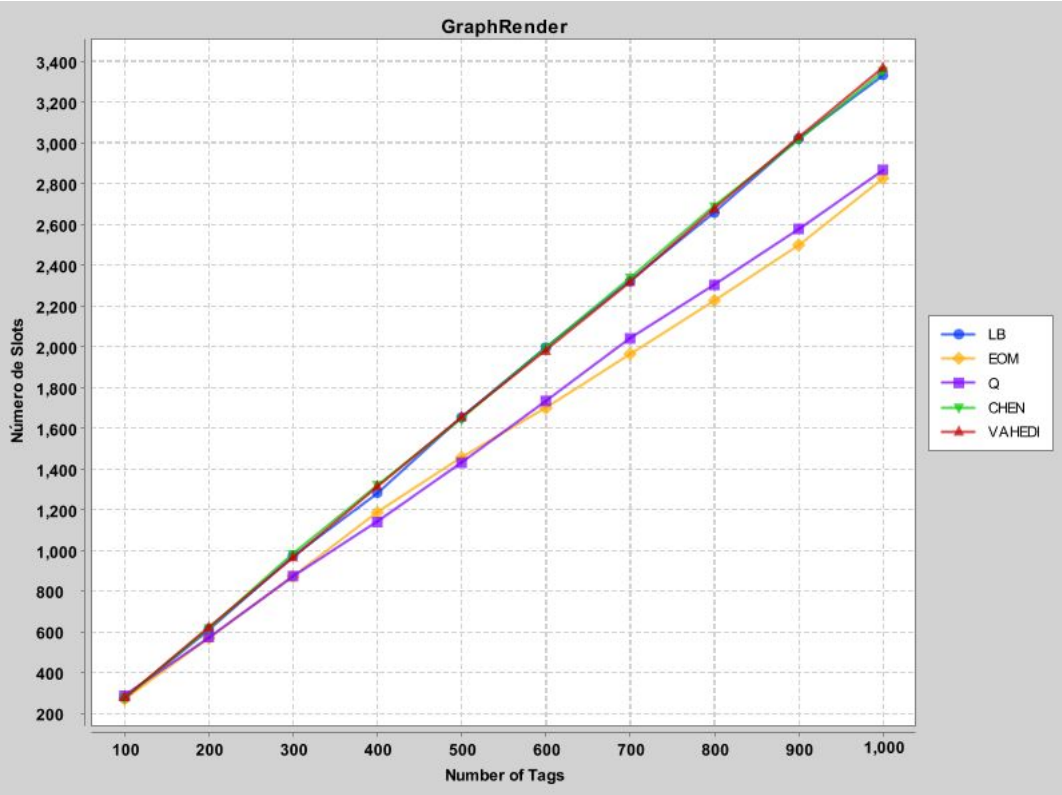
*A mesma mudança melhorou consideravelmente o tempo de Q

Chen (Iterações do algoritmo)



*Iterações do loop principal (prev < next) para o algoritmo Chen

Lower Bound, Eom-Lee, Q, Chen e Vahedi (Slots) (n-2-S)



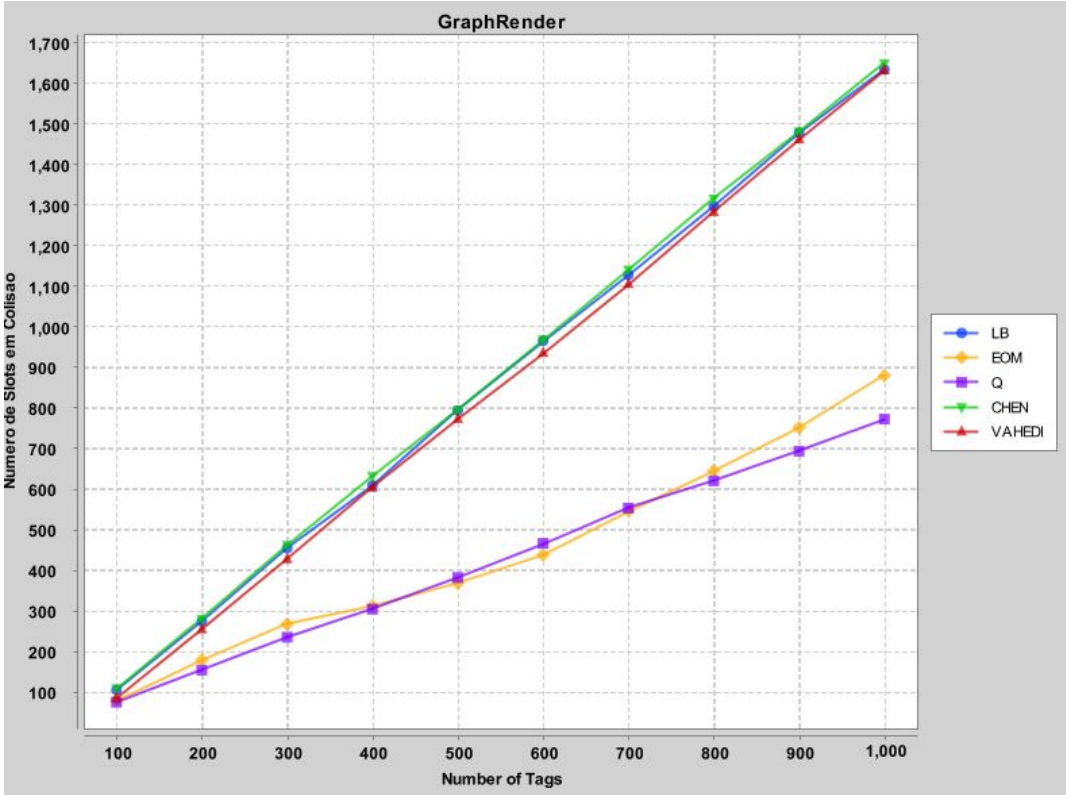
*Devido à demora do Vahedi, estes gráficos foram feitos com 200 repetições cada, em vez de 2000

*Cálculos feitos para retorno do Chen = (n-2-Success), subtraindo-se o número de tags em sucesso. Com este método, o CHEN e VAHEDI ficam bem similares ao Lower Bound (fora o tempo).

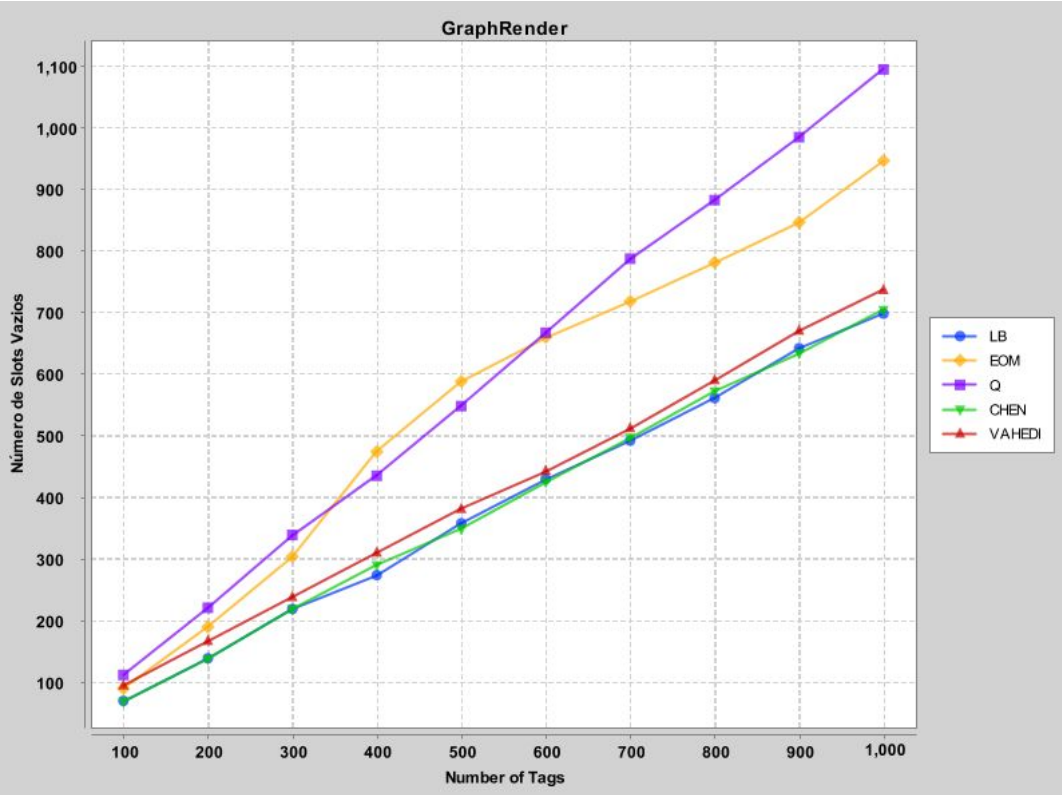
*Resultados calculados com a seguinte operação de fatorial:

```
Sim.arr = new BigInteger[5000];
arr[0] = BigInteger.valueOf(1);
BigInteger result = arr[0];
for(int i = 1; i < arr.length; i++){
    result = result.multiply(BigInteger.valueOf(i));
    arr[i] = result;
}
```

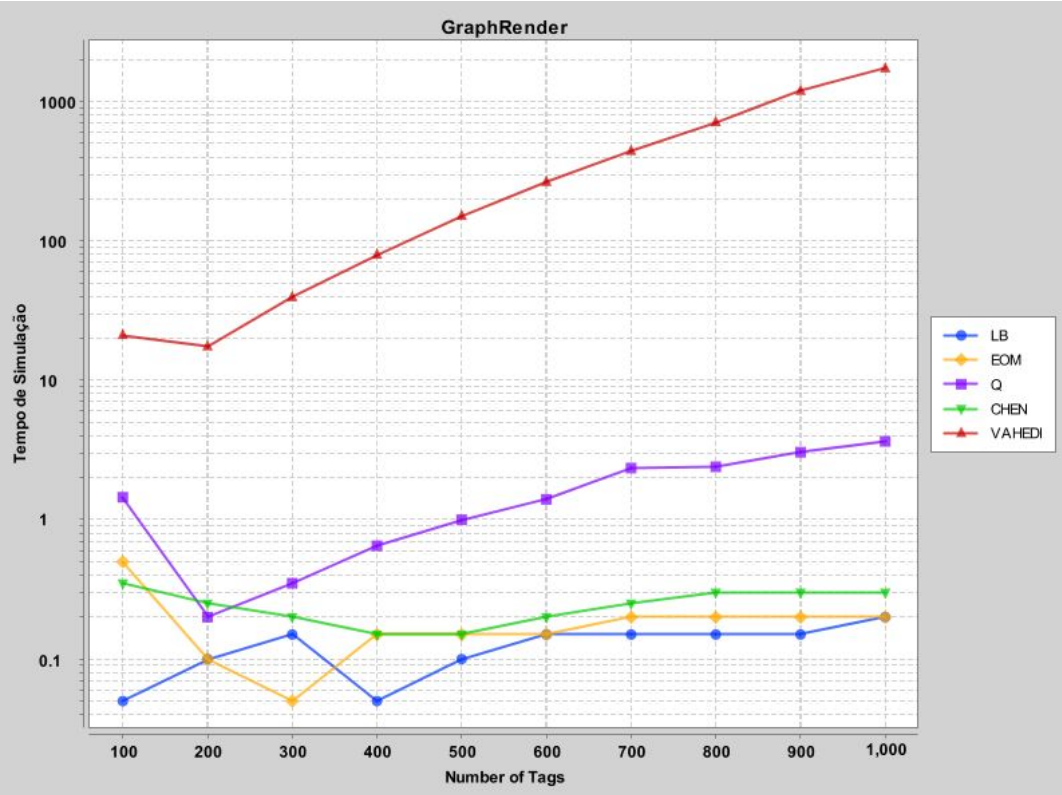
Lower Bound, Eom-Lee, Q, Chen e Vahedi (Slots em Colisão)



Lower Bound, Eom-Lee, Q, Chen e Vahedi (Slots Vazios)



Lower Bound, Eom-Lee, Q, Chen e Vahedi (Tempo)

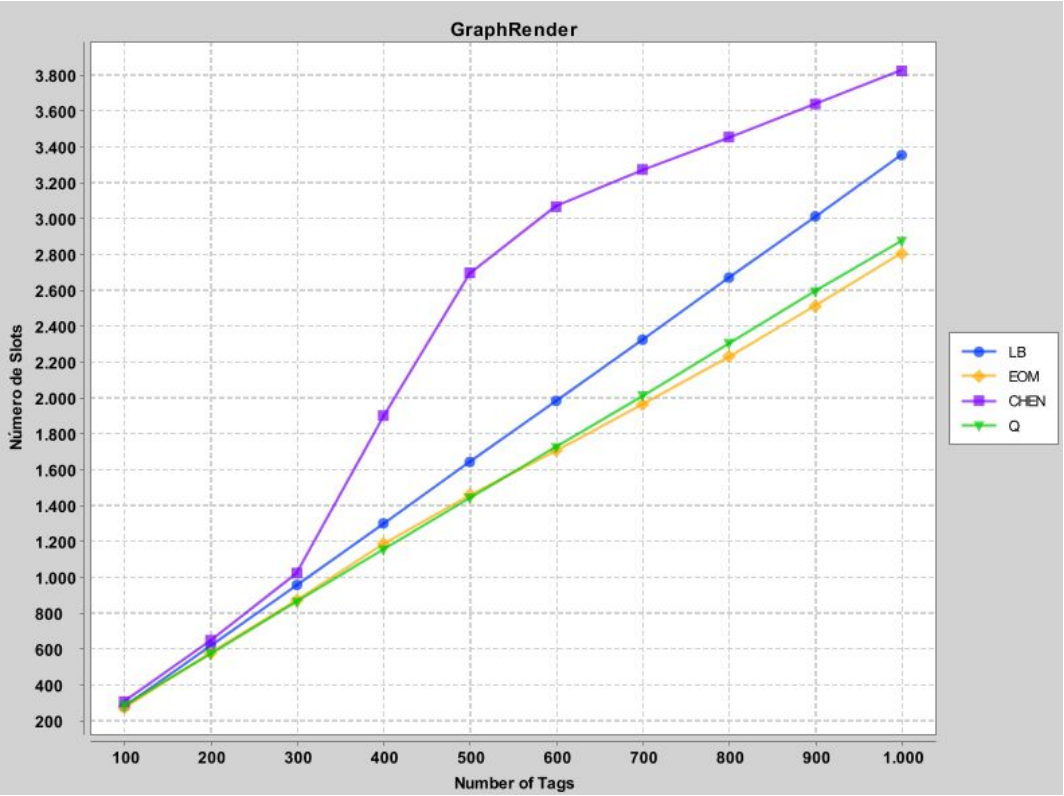


*Os fatoriais de 0 até 5000 foram calculados e armazenados com antecedência, melhorando consideravelmente o tempo de tanto o algoritmo CHEN quanto o VAHEDI

*O tempo ainda grande do VAHEDI se deve aos dois loops para o cálculo da variável P3, que são baseados nos números de slots de colisão: a complexidade, para cada novo cálculo do tamanho do próximo frame, se aproxima de $O(n^3)$

Lower Bound, Eom-Lee, Q, Chen

(n-2)



*Gráficos feitos com 2000 repetições, excluindo o Vahedi

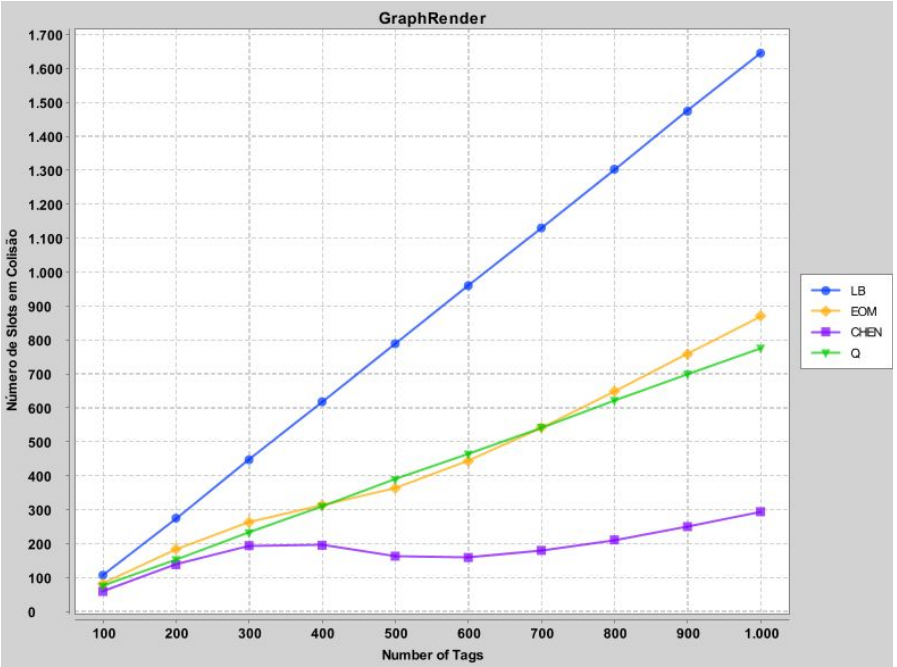
*Cálculos feitos para retorno do Chen = (n-2), como mostrado no algoritmo. A diferença leva a entender que o valor "melhor" é o retornado pelo algoritmo, e ele já mostra o tamanho do próximo quadro.

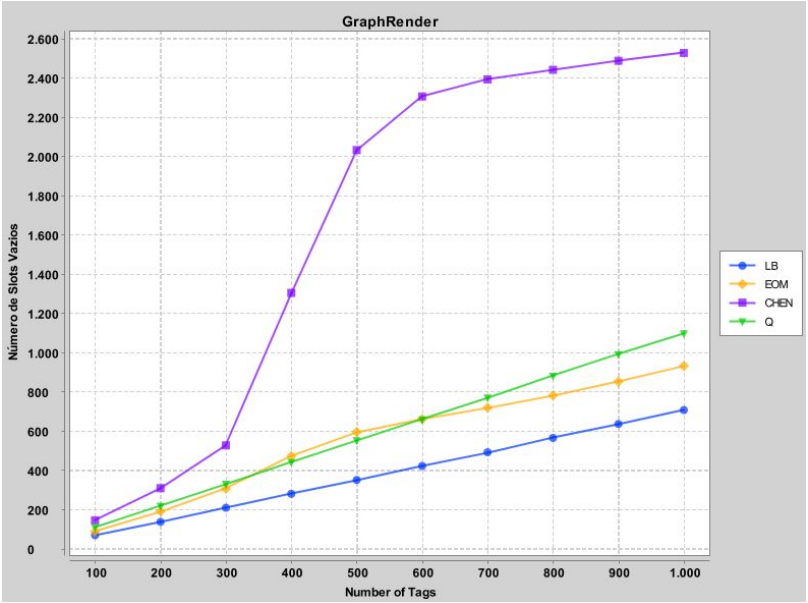
PS: (Experimentos posteriores mostraram que esse resultado não é conseguido 100% das vezes com a fatorial mostrada, e que por mais que a diferença de (n-2) e (n-2-S) seja um fator que muda o resultado, não é o maior)

Lower Bound, Eom-Lee, Q, Chen

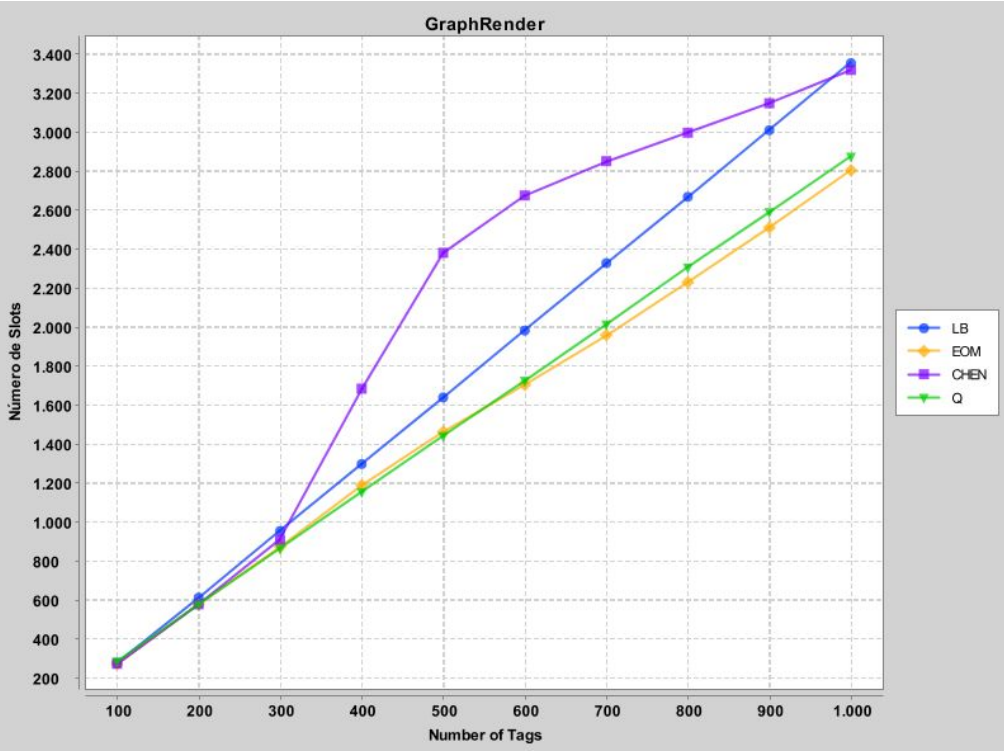
(n-2)

*Porque essa diferença?





Lower Bound, Eom-Lee, Q, Chen (diferente cálculo de fatorial) (n-2-S)



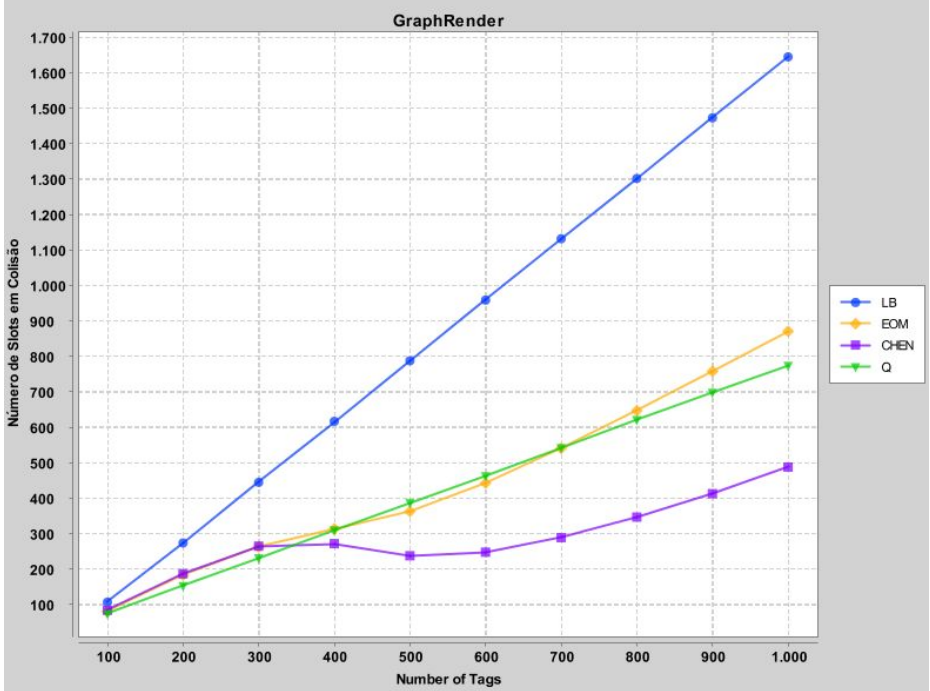
Chen com retorno de (n-2-S) com resultados similares ao (n-2)

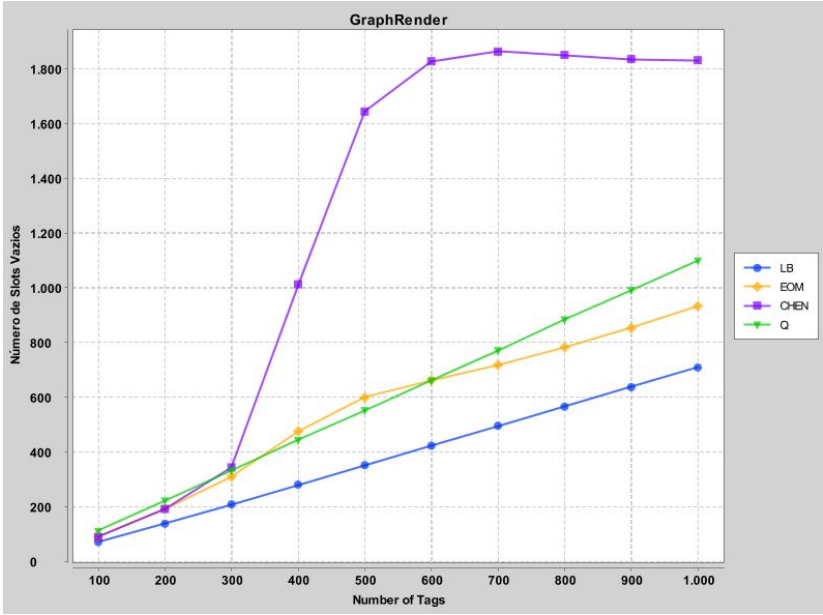
*Soluções diferentes consistentes foram achadas com outro tipo de fatorial para o cálculo de Chen, com as potências calculadas primeiro (o que impede o estouro das fatoriais) e torna desnecessário o uso de BigInt

```
public static double nufactorial(double a, int b, int c, int d, double pe, double ps, double pc) {
    double resultado = pe * ps * pc;

    int n_1 = (b < c && b < d) ? b : ((c < d) ? c : d);
    int n_2 = (n_1 == b) ? ((c < d) ? c : d) : ((n_1 == c) ? d : c);
    int ma = (b > c && b > d) ? b : ((c > d) ? c : d);

    while (a > ma) {
        resultado *= a;
        a--;

        if (n_1 > 1) {
            resultado /= n_1;
            n_1--;
        }

        if (n_2 > 1) {
            resultado /= n_2;
            n_2--;
        }
    }

    return resultado;
}
```

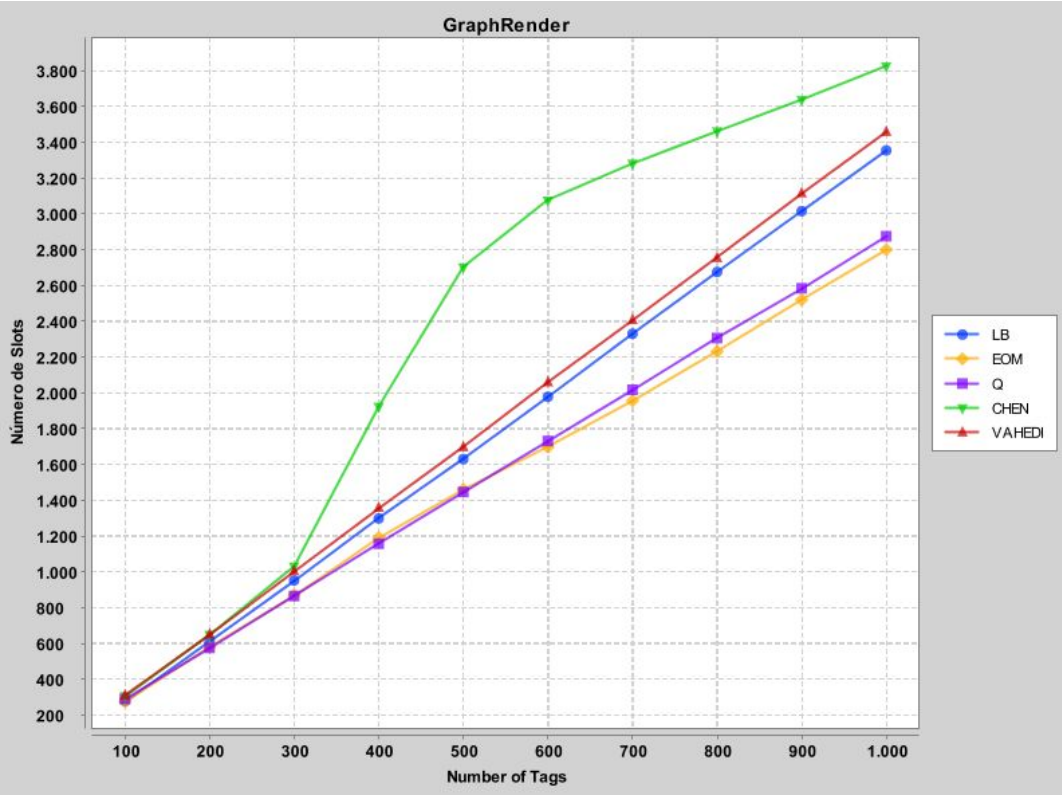
*Possível estouro com BigInteger para alguns casos?

Lower Bound, Eom-Lee, Q, Chen (n-2-S)



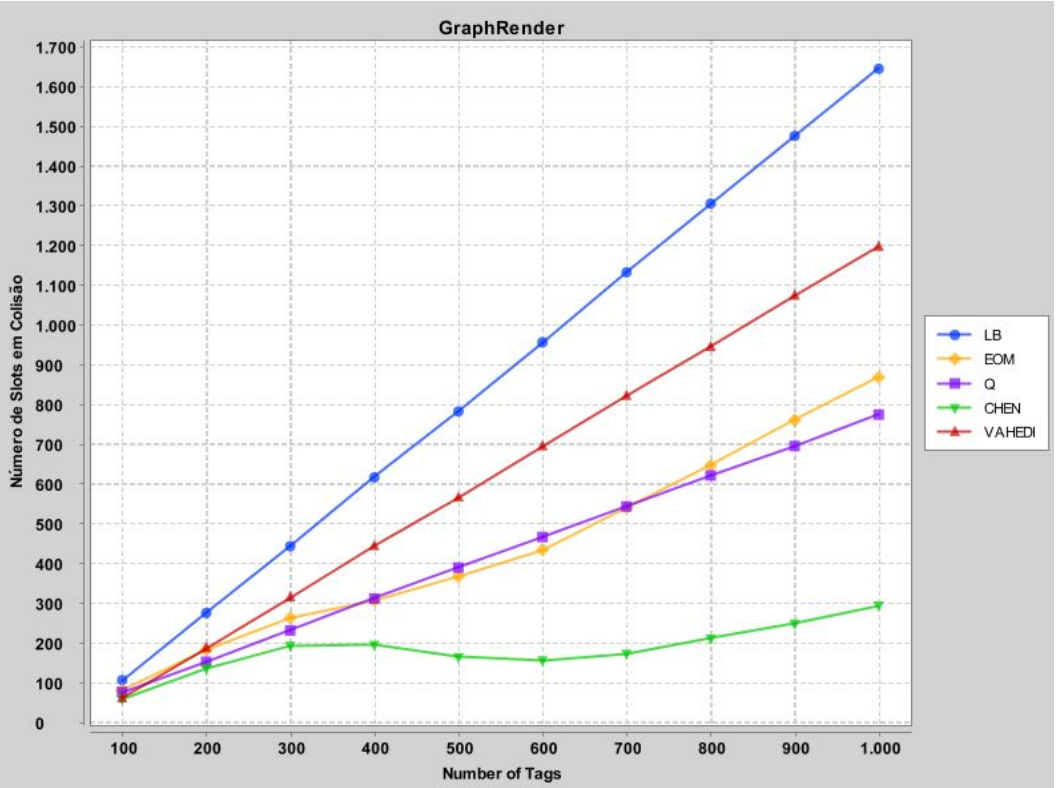


Lower Bound, Eom-Lee, Q, Chen/Vahedi (n-2)



*Usando-se o segundo fatorial, resultados permanecem semelhantes até mesmo com poucas simulações (100), para rodar o vahedi em tempo aceitável.

Lower Bound, Eom-Lee, Q, Chen/Vahedi (n-2)



*Resultados um pouco melhores do que o mesmo fatorial com (n-2-S) para Chen

Conclusões

FATORES VARIANTES
*Subtração do número de tags Sucesso (n-2 ou n-2-S)?
*Número de iterações (baixos com o Vahedi, altos sem)?
*Tipo de Fatorial
    -> Cálculo da fatorial por BigInteger e depois multiplicação pelos fatores pe, ps e pc
    -> Cálculo dos fatores pe, ps e pc primeiro para evitar estouro na fatorial.
    (Aparentemente é o que mais muda a performance do algoritmo!)

PASSOS FUTUROS?

- Se possível, mais experimentos deveriam ser feitos mas não foram viáveis (por exemplo, 2000 iterações para o Vahedi: com a complexidade deste se aproximando de $O(n^3)$, um teste sozinho para as 2000 iterações demoraria aproximadamente uma hora e vinte no meu computador, sem uso de threads)
- Implementar o simulador com threads para cálculos de repetições simultâneos.
- Ver passo a passo as diferenças entre os tipos de cálculo de fatorial, e porque eles retornam valores tão diferentes.