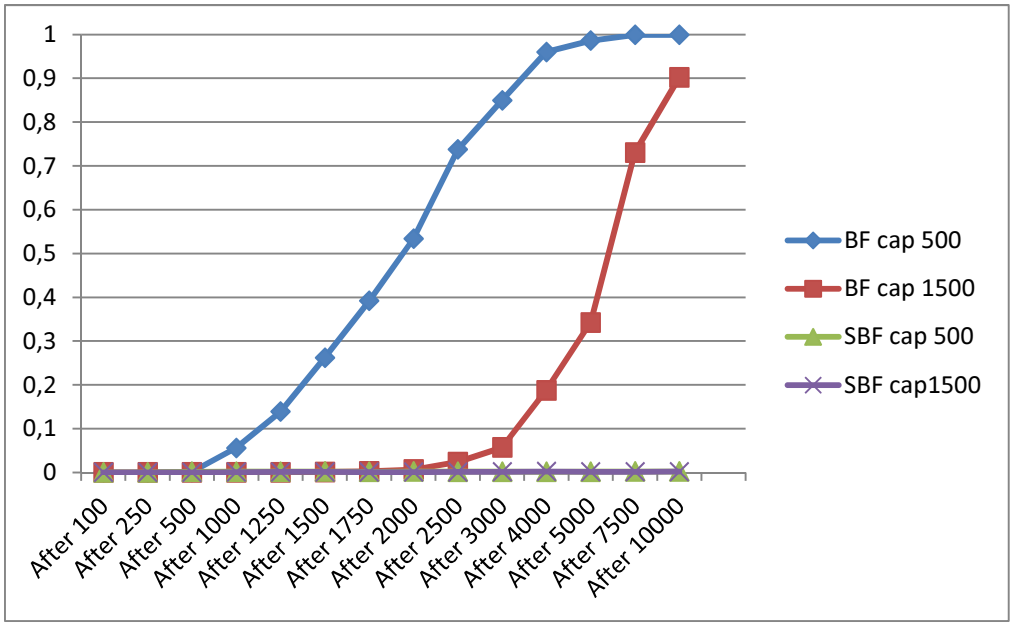
ANEXO

Gráfico 1. Probabilidade de erro após inserção de X elementos, com o Filtro de Bloom e o Filtro de Bloom Escalável.



A experiência feita consiste em adicionar um número de elementos e depois verificar a probabilidade de erro da estrutura. Assim, por exemplo, em 'After 2000' temos as probabilidades de erro após serem inseridos 2000 elementos. Os testes foram feitos inicializando as estruturas de dados com probabilidade de erro inicial igual a 0.1% (0.001). BF cap Y é um Bloom Filter[1] inicializado com capacidade esperada de Y elementos, SBF é o ScalableBloomFilter implementado. Neste gráfico conseguimos ver facilmente que quando o número de elementos inseridos é superior ao número esperado a probabilidade de erro aumenta rapidamente, chegando a um ponto em que o filtro dá sempre um resultado errado. Os filtros de bloom escaláveis pelo contrário, o erro ao longo da execução apoxima-se ao erro requerido.

Comparação do tempo de execução de Java e C++.

	Java	C++
Inserir 10000 elementos.	210 ms.	50 ms.
Verificar 10000 elementos.	82 ms.	20 ms.
Traduzir para JSON (com 10.000 elementos).	15 ms.	20 ms.
Traduzir de JSON para objecto (com 10.000 elementos).	2 ms.	20 ms.
Inserir 50000 elementos.	375 ms.	189 ms.
Verificar 50000 elementos.	180 ms.	124 ms.
Traduzir para JSON (com 50.000 elementos).	20 ms.	274 ms.
Traduzir de JSON para objecto (com 50.000 elementos).	10 ms.	231 ms.

Com os resultados podemos ver que o tempo de execução é melhor na versão em C++. A excepção é quando estamos a criar a string json e a devolver um objecto a partir do json, ai a biblioteca em Java usada escala melhor que a em C++.

[1] `http://github.com/magnuss/java-bloomfilter`