Satisfiability-based Set Membership Filters

Sean Weaver

Information Assurance Research Group
U.S. National Security Agency

May 7, 2018

Outline

The Set Membership Problem

- Efficiently test whether a large set contains a given element
- Some Examples
  - Spell Checking
  - Safe Browsing
- Formally

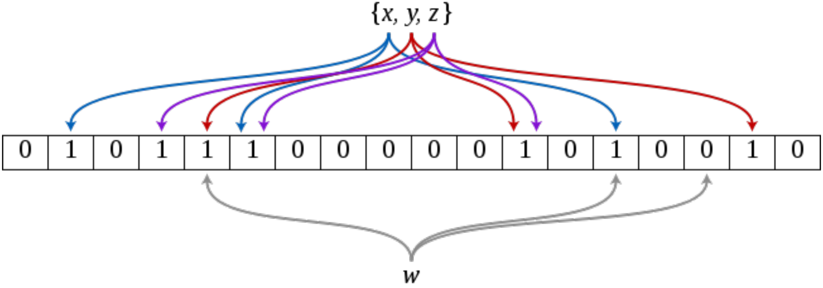
  Given an element $x \in D$ and $Y \subseteq D$, determine if $x \in Y$

Bloom Filter [Blo70]

- Constant time querying
- Probabilistic — can give false positives
- Efficient primary test for set membership
- Algorithm has two stages, building and querying

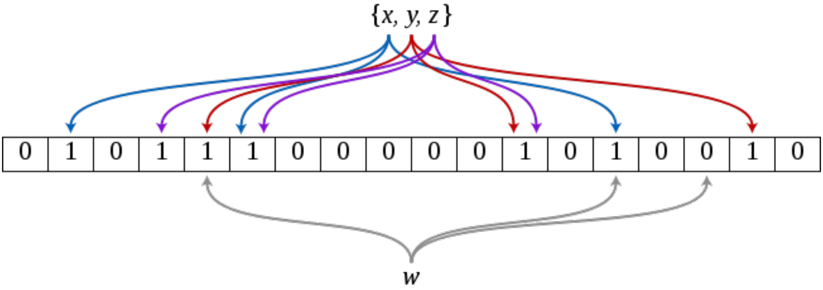
Bloom Filter — Building

- Bloom filter is an array of n bits, initially all 0
- Map each element $y \in Y$ to k indices, $H(y,1), H(y,2), \ldots, H(y,k)$
- For all indices, set the corresponding bits in the Bloom filter



Bloom Filter — Querying

- Map element x to k indices, $H(x,1), H(x,2), \ldots, H(x,k)$
- If the filter's k corresponding indices are all set, x is maybe in Y
- If some bit is not set, then x is definitely not in Y



Bloom Filter — More Information

For more information such as how to calculate the false positive rate, or how to determine the optimal size of the filter, see Wikipedia: `https://en.wikipedia.org/wiki/Bloom_filter`

SAT Filter

- Many different filter constructions since 1970
- Most pertinent to this talk is a filter based on Satisfiability

Satisfiability (SAT)

- Given a set of constraints, determine if a solution exists
- Usually Boolean constraints (clauses) expressed in Conjunctive Normal Form (CNF)
- Example CNF Formula: $(x_0 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_0} \vee \overline{x_1})$
- A solution: $\{x_0, \overline{x_1}, \overline{x_2}, x_3\}$
- Finding a solution is NP-Complete
- Many open-source SAT solvers exist and are available for download here: `http://www.satcompetition.org/`

SAT Filter [WRM+14]

- A filter based on SAT
- Building
  - Hash elements to CNF clauses, rather than array indices
  - Treat the set of clauses as a SAT problem
  - A solution to the SAT problem acts as a filter for the original set
- Querying
  - Hash an element into a CNF clause
  - If the clause is satisfied by the stored solution it passes the filter
  - If the clause is not satisfied, the element doesn't pass the filter

SAT Filter

Build

$$\{x, \; y, \; z\}$$

$$x \xrightarrow{H(x)} \overline{x_0} \vee x_1 \vee \overline{x_3} \qquad y \xrightarrow{H(y)} x_0 \vee x_2 \vee \overline{x_3} \qquad z \xrightarrow{H(z)} \overline{x_0} \vee \overline{x_2} \vee x_3$$

A solution: $\{\overline{x_0}, x_1, x_2, \overline{x_3}\}$

Query

$$w \xrightarrow{H(w)} \overline{x_0} \vee \overline{x_2} \vee x_3$$

Passes

$$v \xrightarrow{H(v)} x_0 \vee \overline{x_2} \vee x_3$$

Doesn't pass

SAT Filter Parameters

- Y is the set of interest
- $m = |Y|$ is the number of clauses
- n is the total number of variables (also the size of the filter)
- k is the number of variables per clause
- $p = 1 - \frac{1}{2^k}$ is the false positive rate

How many variables (k per clause, and n total) should there be?

How can the false positive rate be decreased?

Number of Variables, n

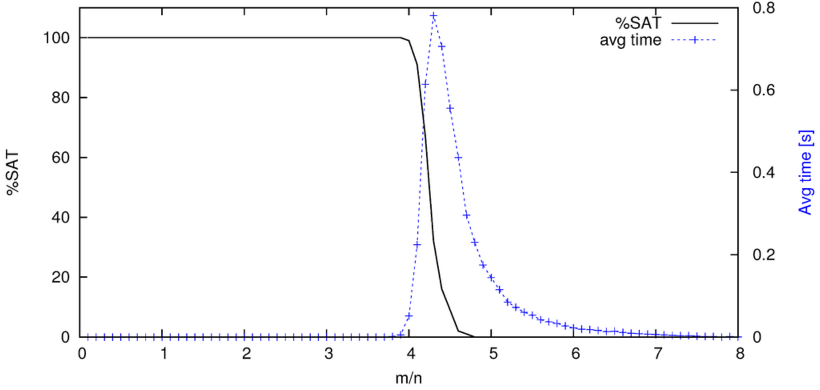
- Amount of long-term storage
- Desire to be as small as possible
- Why not just make n tiny?

- What kind of SAT problems are being generated?
- Random k-SAT!

Random k-SAT

- Clauses drawn uniformly, independently, and with replacement from the set of all width k clauses [FP83]
- The clauses-to-variables ratio $\alpha_k = m/n$ determines almost certainly the satisfiability of the set of clauses drawn [Ach09]

Random k-SAT Threshold



[FR13]

Random k-SAT Threshold

- The threshold for random k-SAT is $\alpha_k = 2^k \ln 2 - O(k)$ [AP04]
- Some values determined experimentally

k	1	2	3	4	5	6	7	8
α_k	0	1	4.26	9.93	21.11	43.37	87.79	176.54

- Why does this threshold exist?

Efficiency

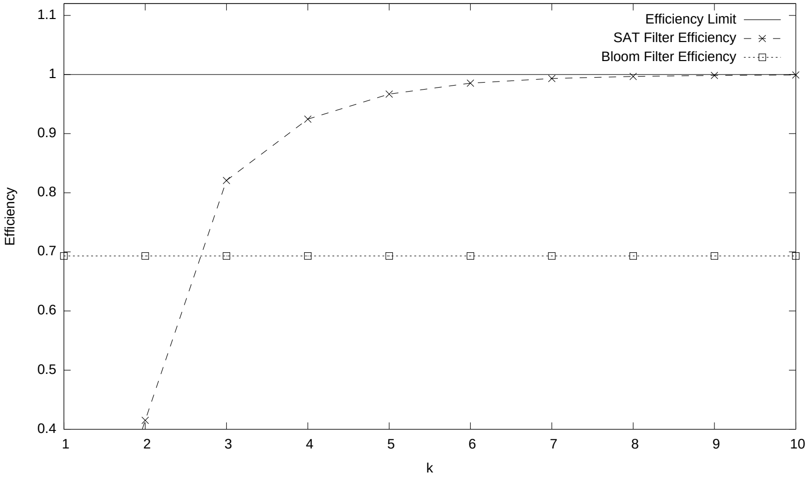
- Only so much information can fit into a filter with a fixed amount of memory, i.e. the *information-theoretic limit*
- Efficiency is a measure of how well a filter uses its memory [Wal07]

$$\mathcal{E} = \frac{-\log_2 p}{n/m} \leq 1$$

- False positive rate p. For SAT filters $p = 1 - 2^{-k}$
- The filter is n bits of memory
- The filter is derived from m elements
- k is the width of a clause

- Plugging $m/n = 2^k \ln 2 - O(k)$ into $\mathcal{E}$ and performing an asymptotic analysis shows that SAT filter efficiency $\mathcal{E} = 1$
- Bloom filter efficiency $\mathcal{E} = \ln 2 \approx 0.69$

Efficiency


Efficiency Limit
SAT Filter Efficiency
Bloom Filter Efficiency
Efficiency
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
1
2
3
4
5
6
7
8
9
10
k

Efficiency

0b10011110

- false positive rate $p = 2^{-3}$, or $\frac{1}{8}$

$$\mathcal{E} = \frac{-\log_2 p}{n/m}$$

$$\mathcal{E} = \frac{-\log_2 2^{-3}}{3/1}$$

$$\mathcal{E} = \frac{3}{3}$$

$$\mathcal{E} = 1$$

False Positive Rate

- The false positive rate ($p = 1 - 2^{-k}$) needs to be improved
- One way is to find multiple solutions, say s
- Now the false positive rate is increased to $p = (1 - 2^{-k})^s$
- One catch — the solutions must be uncorrelated
- Could build s different SAT instances (slow to query)
- Or, could very carefully find s different solutions to the same instance (slow to build)
- Solutions in random k-SAT naturally cluster right before the threshold
- However, this is not so for some other random SAT paradigms (NAESAT, XORSAT, ...).

XORSAT

- A lot like random k-SAT, but the clause operand is XOR($\oplus$), not OR($\vee$)
- Example XORSAT Formula: $(x_0 \oplus x_1 \oplus x_3 \equiv 1) \wedge (x_2 \oplus x_3 \equiv 0)$
- A solution: $\{x_0, x_1, x_2, x_3\}$

- The phase transition is sharp, and tends to 1

k	2	3	4	5	6	7
α_k	0.5	0.917935	0.976770	0.992438	0.997379	0.999063

XORSAT Filter

Build

$$\{x, \; y, \; z\}$$

$$x \xrightarrow{H(x)} x_0 \oplus x_1 \oplus x_3 \equiv 0 \qquad y \xrightarrow{H(y)} x_0 \oplus x_2 \oplus x_3 \equiv 1 \qquad z \xrightarrow{H(z)} x_0 \oplus x_1 \oplus x_2 \equiv 0$$

A solution: $\{x_0, x_1, \overline{x_2}, \overline{x_3}\}$

Query

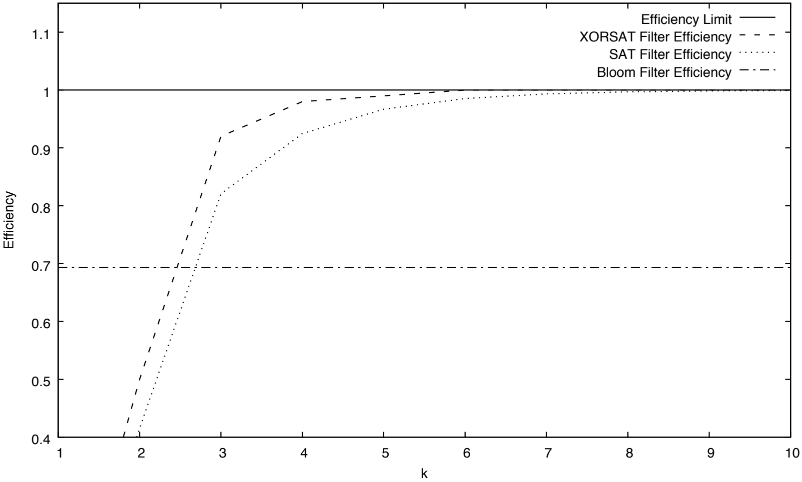
$w \xrightarrow{H(w)} x_0 \oplus x_2 \oplus x_3 \equiv 1$ Passes

$v \xrightarrow{H(v)} x_0 \oplus x_2 \oplus x_3 \equiv 0$ Doesn't pass

XORSAT Filter vs SAT Filter

- XORSAT $p = 2^{-s}$ vs. SAT $p = (1 - 2^{-k})^s$
- Sharper threshold, so the reach information theoretic limit can be achieved with smaller k, and the filter can be smaller
- Easy to find uncorrelated solutions to an XORSAT instance
- Solving linear equations in GF(2) is in P vs. NP for SAT
- XORSAT filters can store and retrieve meta-data

Efficiency



Efficiency Limit
XORSAT Filter Efficiency
SAT Filter Efficiency
Bloom Filter Efficiency
Efficiency
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
1
2
3
4
5
6
7
8
9
10
k

XORSAT Filter Example

$$H = [\text{xxHash(“cat”), xxHash(“fish”), xxHash(“dog”)}]$$
$$= [\texttt{0xb85c341a}, \texttt{0x87024bb7}, \texttt{0x3fa6d2df}] \, .$$

$$SH = [[\texttt{0xb}, \texttt{0x8}, \texttt{0x5}, \texttt{0xc}, \texttt{0x3}, \texttt{0x4}, \texttt{0x1}, \texttt{0xa}],$$
$$[\texttt{0x8}, \texttt{0x7}, \texttt{0x0}, \texttt{0x2}, \texttt{0x4}, \texttt{0xb}, \texttt{0xb}, \texttt{0x7}],$$
$$[\texttt{0x3}, \texttt{0xf}, \texttt{0xa}, \texttt{0x6}, \texttt{0xd}, \texttt{0x2}, \texttt{0xd}, \texttt{0xf}]] \, .$$

XORSAT Filter Example

$$SH = [[\texttt{0xb}, \texttt{0x8}, \texttt{0x5}, \texttt{0xc}, \texttt{0x3}, \texttt{0x4}, \texttt{0x1}, \texttt{0xa}], \\ [\texttt{0x8}, \texttt{0x7}, \texttt{0x0}, \texttt{0x2}, \texttt{0x4}, \texttt{0xb}, \texttt{0xb}, \texttt{0x7}], \\ [\texttt{0x3}, \texttt{0xf}, \texttt{0xa}, \texttt{0x6}, \texttt{0xd}, \texttt{0x2}, \texttt{0xd}, \texttt{0xf}]]\ .$$

$$\begin{aligned}
\mathcal{I}_{Y.0} &= [[SH_{00} (\text{mod } n), SH_{01} (\text{mod } n), SH_{02} (\text{mod } n), SH_{03} (\text{mod } 2^s)], \\
&\quad [SH_{10} (\text{mod } n), SH_{11} (\text{mod } n), SH_{12} (\text{mod } n), SH_{13} (\text{mod } 2^s)], \\
&\quad [SH_{20} (\text{mod } n), SH_{21} (\text{mod } n), SH_{22} (\text{mod } n), SH_{23} (\text{mod } 2^s)]] \\
&= [[\texttt{0xb} (\text{mod } 4), \texttt{0x8} (\text{mod } 4), \texttt{0x5} (\text{mod } 4), \texttt{0xc} (\text{mod } 8)], \\
&\quad [\texttt{0x8} (\text{mod } 4), \texttt{0x7} (\text{mod } 4), \texttt{0x0} (\text{mod } 4), \texttt{0x2} (\text{mod } 8)], \\
&\quad [\texttt{0x3} (\text{mod } 4), \texttt{0xf} (\text{mod } 4), \texttt{0xa} (\text{mod } 4), \texttt{0x6} (\text{mod } 8)]] \\
&= [[3, 0, 1, 4], \\
&\quad [0, 3, 0, 2], \\
&\quad [3, 3, 2, 6]]\ .
\end{aligned}$$

XORSAT Filter Example

$$\mathcal{I}_{Y.0} = [[3,0,1,4],\\ [0,3,0,2],\\ [3,3,2,6]]\ .$$

$$\begin{aligned}
\mathcal{X}_{Y.0} &= [x_3 \oplus x_0 \oplus x_1 \equiv [1,0,0],\\
&\quad x_0 \oplus x_3 \oplus x_0 \equiv [0,1,0],\\
&\quad x_3 \oplus x_3 \oplus x_2 \equiv [1,1,0]]\\
&= [x_0 \oplus x_1 \oplus x_3 \equiv [1,0,0],\\
&\quad x_3 \equiv [0,1,0],\\
&\quad x_2 \equiv [1,1,0]]\ .
\end{aligned}$$

XORSAT Filter Example

$$\begin{aligned}\mathcal{X}_{Y.0} = [x_0 \oplus x_1 \oplus x_3 &\equiv [1,0,0],\\ x_3 &\equiv [0,1,0],\\ x_2 &\equiv [1,1,0]]\ .\end{aligned}$$

$$\begin{aligned}\mathcal{X}_{Y} = [x_0 \oplus x_1 \oplus x_3 &\equiv [1,0,0] \mathbin{||} [0,0],\\ x_3 &\equiv [0,1,0] \mathbin{||} [0,1],\\ x_2 &\equiv [1,1,0] \mathbin{||} [1,0]]\\ = [x_0 \oplus x_1 \oplus x_3 &\equiv [1,0,0,0,0],\\ x_3 &\equiv [0,1,0,0,1],\\ x_2 &\equiv [1,1,0,1,0]]\ .\end{aligned}$$

XORSAT Filter Example

$$\mathcal{X}_Y = [x_0 \oplus x_1 \oplus x_3 \equiv [1,0,0,0,0], \\ x_3 \equiv [0,1,0,0,1], \\ x_2 \equiv [1,1,0,1,0]] \; .$$

$$S_Y = [[x_0 = 1, x_1 = 1, x_2 = 0, x_3 = 1], \\ [x_0 = 0, x_1 = 1, x_2 = 1, x_3 = 1], \\ [x_0 = 0, x_1 = 0, x_2 = 0, x_3 = 0], \\ [x_0 = 1, x_1 = 1, x_2 = 1, x_3 = 0], \\ [x_0 = 1, x_1 = 0, x_2 = 0, x_3 = 1]] \; .$$

XORSAT Filter Example

$$
\begin{aligned}
S_Y = [[&x_0 = 1, x_1 = 1, x_2 = 0, x_3 = 1],\\
&[x_0 = 0, x_1 = 1, x_2 = 1, x_3 = 1],\\
&[x_0 = 0, x_1 = 0, x_2 = 0, x_3 = 0],\\
&[x_0 = 1, x_1 = 1, x_2 = 1, x_3 = 0],\\
&[x_0 = 1, x_1 = 0, x_2 = 0, x_3 = 1]]\ .
\end{aligned}
$$

$$
\begin{aligned}
F_Y = ([&[1, 0, 0, 1, 1],\\
&[1, 1, 0, 1, 0],\\
&[0, 1, 0, 1, 0],\\
&[1, 1, 0, 0, 1]],\\
&n = 3, k = 3, s = 3, r = 2)\ .
\end{aligned}
$$

XORSAT Filter Query Example

$$
\begin{aligned}
H &= \text{xxHash(“horse”)} \\
&= \texttt{0x3f37a1a7}\,.
\end{aligned}
$$

$$
SH = [\texttt{0x3}, \texttt{0xf}, \texttt{0x3}, \texttt{0x7}, \texttt{0xa}, \texttt{0x1}, \texttt{0xa}, \texttt{0x7}]\,.
$$

$$
\begin{aligned}
\mathcal{I} &= [SH_0 (\text{mod } n), SH_1 (\text{mod } n), SH_2 (\text{mod } n), SH_3 (\text{mod } 2^s)] \\
&= [\texttt{0x3} (\text{mod } 4), \texttt{0xf} (\text{mod } 4), \texttt{0x3} (\text{mod } 4), \texttt{0x7} (\text{mod } 8)] \\
&= [3, 3, 3, 7]\,.
\end{aligned}
$$

$$
\begin{aligned}
C &= x_3 \oplus x_3 \oplus x_3 \equiv [1, 1, 1] \mid\mid [1, 1] \\
&= \qquad\qquad x_3 \equiv [1, 1, 1, 1, 1]\,.
\end{aligned}
$$

XORSAT Filter Query Example

$$\begin{aligned} C &= x_3 \oplus x_3 \oplus x_3 \equiv [1, 1, 1] \mid\mid [1, 1] \\ &= \qquad\qquad x_3 \equiv [1, 1, 1, 1, 1] \, . \end{aligned}$$

$$\begin{aligned} C_{F_Y} &= F_Y(3) \equiv [1, 1, 1, 1, 1] \\ &= [1, 1, 0, 0, 1] \equiv [1, 1, 1, 1, 1] \\ &= [1, 1, 0, 0, 1] \, . \end{aligned}$$

XORSAT Filter Query Example

$$x = \text{“cat”}$$

$$C = x_0 \oplus x_1 \oplus x_3 \equiv [1, 0, 0, 1, 1]\ .$$

Evaluating C against F_Y produces

$$\begin{aligned} C_{F_Y} &= F_Y(0) \oplus F_Y(1) \oplus F_Y(3) \equiv [1, 0, 0, 1, 1] \\ &= [1, 0, 0, 1, 1] \oplus [1, 1, 0, 1, 0] \oplus [1, 1, 0, 0, 1] \equiv [1, 0, 0, 1, 1] \\ &= [1, 0, 0, 0, 0] \equiv [1, 0, 0, 1, 1] \\ &= [1, 1, 1, 0, 0]\ . \end{aligned}$$

Since the first three bits of C_{F_Y} are all True, the element passes the filter. Hence, “cat” is in Y with a $\frac{1}{2^3}$ chance of being a false positive. The last two bits of C_{F_Y}, $[0, 0]$, represent the stored meta-data, namely, the index 0.

Results

Table: Achieved efficiency and seconds taken to build *non-blocked* XORSAT filters with $p = \frac{1}{2^{10}}$. m is the number of elements in the data set being stored and k is the number of variables per XOR clause.

m	$k = 3$	$k = 4$	$k = 5$	$k = 6$
2^{10}	(88%, < 1)	(93%, < 1)	(93%, < 1)	(93%, < 1)
2^{11}	(89%, < 1)	(97%, < 1)	(97%, < 1)	(97%, < 1)
2^{12}	(90%, < 1)	(97%, < 1)	(98%, < 1)	(98%, < 1)
2^{13}	(91%, 1)	(97%, 1)	(98%, 1)	(99%, 1)
2^{14}	(91%, 2)	(97%, 3)	(99%, 4)	(99%, 5)
2^{15}	(89%, 17)	(97%, 21)	(98%, 28)	(98%, 36)

Results

Table 4. Achieved efficiency, size (in KB), and seconds taken to build *blocked* XORSAT and SAT filters with an expected 3072 elements per block, variables per clause $k = 5$, and desired false positive rate $p = \frac{1}{2^{10}}$. Desired SAT filter efficiency was set to 75% and desired XORSAT filter efficiency was set to 98%. The SAT filter hamming weight metric [31] was set to 48%. Timeout ('-') was set at one hour. Query speed (in millions of queries per second) is also given for XORSAT, SAT, and Bloom filters.

	XORSAT Filter				SAT Filter				Query Speed		
	Build Time				Build Time						
m	1 Core	8 Cores	$\mathcal{E}$	Size	1 Core	8 Cores	$\mathcal{E}$	Size	XORSAT	SAT	Bloom
2^{15}	< 1	< 1	98%	41	336	105	43%	56	18	4	23
2^{16}	1	< 1	98%	81	883	183	43%	111	18	4	23
2^{17}	2	< 1	98%	163	1768	394	43%	222	18	4	23
2^{18}	5	1	98%	326	3441	723	44%	444	18	4	23
2^{19}	8	1	97%	659	-	1724	44%	887	18	4	23
2^{20}	17	2	97%	1321	-	-	-	-	18	-	22
2^{21}	33	4	97%	2646	-	-	-	-	17	-	22
2^{22}	92	12	97%	5298	-	-	-	-	13	-	20
2^{23}	186	26	97%	10601	-	-	-	-	9	-	20
2^{24}	372	52	97%	21204	-	-	-	-	11	-	20
2^{25}	751	104	96%	42416	-	-	-	-	10	-	17
2^{26}	1515	208	96%	84958	-	-	-	-	7	-	12

Questions?

References I

- Dimitris Achlioptas, *Random satisfiability*, Handbook of Satisfiability (Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, eds.), Frontiers in Artificial Intelligence and Applications **185**, IOS Press, 2009, pp. 245–270.
- Dimitris Achlioptas and Yuval Peres, *The threshold for random k-SAT is $2^k \log 2 - O(k)$*, Journal of the American Mathematical Society **17** (2004), no. 4, 947–973.
- Burton H. Bloom, *Space/time trade-offs in hash coding with allowable errors*, Communications of the ACM **13** (1970), no. 7, 422–426.
- John Franco and Marvin Paull, *Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem*, Discrete Applied Mathematics **5** (1983), no. 1, 77–87.

References II

- Marcelo Finger and Poliana M. Reis, *On the predictability of classical propositional logic*, Information **4** (2013), no. 1, 60–74.
- Alden Walker, *Filters*, Master's thesis, Haverford College, 2007.
- Sean A. Weaver, Katrina J. Ray, Victor W. Marek, Andrew J. Mayer, and Alden K. Walker, *Satisfiability-based set membership filters*, Journal on Satisfiability, Boolean Modeling and Computation **8** (2014), 129–148.