Advanced Algorithms Coursework

Oliver Goldstein (og14775)

April 2017

1 Problem 1

1.1 Part 1

Let $A[1, ..., n]$ and $B[1, ..., n]$ be the number of occurrences of item i in two streams respectively. The algorithm should achieve a (ϵ, δ) approximation $s.t.$

$$\Pr[Output \in (1 - \epsilon, 1 + \epsilon) * Exact] \geq 1 - \delta$$

$N.B.$ Whilst the Count-Min Sketch procedure includes turnstile type functionality, by only using the insert operation, we consider the cash register model only.

Proposed Algorithm

1. Initialise two Count-Min Sketch Tables At, Bt.
2. While item i from stream A or B arrives:
3. Insert i into table A or B, respective to the stream.
4. Return X = $\sum_{i=1}^{n} queryA(i) * queryB(i)$

$N.B.$ n is defined above and $queryA(i)$ translates as query table A for item i.

Brief Analysis

The Count-Min Sketch provides us with the following guarantees, such that X_1 is the first moment of the set X and mx is the true number of instances of a particular item counted.

$$1. \Pr[m'x \geq mx + \epsilon * A_1] \leq e^{-d} \leq \delta$$
$$2. \Pr[m'x \geq mx + \epsilon * B_1] \leq e^{-d} \leq \delta$$

Assuming an (ϵ, δ) approximation is achieved such that $\delta \leq 1/3$ then

$$\Pr[Error] = \delta^n = (1/3)^n$$

$$\Pr[NoError] = (1-\delta)^n = (2/3)^n$$

Performing $\Pr[Error]/\Pr[NoError]$ shows in general, for $i \in 1...n$ $Pr[Error] = 2^{-i}$

Summing $\forall i$ yields $\sum_{i=1}^{n} 2^{-i} = \sum_{i=1}^{n} (1/2)^{-i}$ a simple geometric series.

This sum is defined as:

$$s_{\mathrm{n}} = 1/2 + 1/4 + 1/8 + ... + 1/2^{n-1} + 1/2^n$$

Multiplying the sum by 2 gives:

$$s_{\mathrm{n}} = 1 + [1/2 + 1/4...1/2^{n-1}] = 1 + [s_{\mathrm{n}} - 1/2^n]$$

Subtracting s_{n} from both sides gives

$$s_{\mathrm{n}} = 1 - 1/2^n$$

This shows at the limit of infinity, the expected error is at most

$$Pr[Error] = (1 - 1/2^n) * \epsilon * A_1 * B_1 = \epsilon * A_1 * B_1$$

In addition, $\forall \delta \leq 1/3$ the same relationship can be seen.

1.2 Part 2

Space Complexity

In total, two Count Min Sketches are also included, one for each stream. The space complexity depends on ϵ and δ. In addition, for each row in each Count-Min Table, a hash function must be stored. I will denote this as a constant k.

In particular the space complexity is:

$$O(log(1/\delta) * e/\epsilon * k * 2)$$

$log(1/\delta)$ is the number of rows in the table and e/ϵ is the number of columns.

Proof of Correctness

The correct answer is achieved if the algorithm returns a valid (ϵ, δ) approximation.

Proof of Correctness - First attempt

The following bound is the weakest proof. I will follow this by a stronger proof given the time. On each iteration of the summation two query actions are performed, each with error at most: $\epsilon * A_1$. The probability that the result a smaller error than the max is at least: $1 - \delta$ The probability of it maintaining an error within the bound for all iterations is $\sum_{i=1}^{n} NoErrorA \wedge NoErrorB = (1 - 2 * \delta + \delta^2)^n$ as $\delta > \delta^2$ and $0 \leq \delta \leq 1$ it decays to zero exponentially.

This bound exponentially decays to zero for reasonably sized n and therefore does not yield much value.

Proof of Correctness - Strong Bound (Second attempt)

A better bound comes through a different approach.

I attempt to show that for $\delta = 1/3$ each time a query operation occurs it succeeds with less than the upper bound at least 2/3 of the time.

Due to the property of the query operation retrieving the minimum value on each row, I only consider the case where all of the rows exceed the error.

The probability of a collision at each query operation is $\mathbb{E}[\mathcal{Z}_{\mathrm{j,\ x}}] \leq (\epsilon/e)$ The probability that every row queried collides, is ϵ is $(\epsilon/e)^{log(1/\delta)}$

This is the probability $1 - (\epsilon/e)^{log(1/\delta)}$ The minimum value of this equation occurs where epsilon is greatest i.e. 1.

At this point the value is

$$1 - (\epsilon/e)^{-log(\delta)}$$

=

$$1 - (\epsilon^{-log(\delta)}/e^{-log(\delta)})$$

=

$$1 - (\delta/\epsilon^{log(\delta)})$$

= (As ϵ is 1)

$$1 - \delta$$

=

$$2/3$$

Using knowledge from part 1, we can see clearly that an (ϵ, δ) approximation is achieved $\forall \delta \leq 1/3$

2 Problem 2

2.1 Part 1

Proof of Universality

Working from the definition of independence, starting with $x_1, x_2 \in \{0,1\}^m$

$h(x_1) = a(h)x_1$
$h(x_2) = a(h)x_2$

$\Pr \forall h \in \mathcal{H}[h(x_1) = h(x_2)]$
$= \Pr \forall h \in \mathcal{H} \wedge \forall i \in 1...n\ [\sum_{j=1}^{m} a(h)_{\mathrm{i,j}} X_{\mathrm{j}} = \sum_{j=1}^{m} a(h)_{\mathrm{i,j}} X'_{\mathrm{j}}]$
$= \Pr \forall h \in \mathcal{H} \wedge \forall i \in 1...n\ [\sum_{j=1}^{m} h_{\mathrm{i\text{-}j+m}} X_{\mathrm{j}} = \sum_{j=1}^{m} h_{\mathrm{i\text{-}j+m}} X'_{\mathrm{j}}]$

The rows of the matrix have not yet been shown to be independent and so must be analysed further. At this point the rows of the matrix formed by iterating over h are:

$$a(h) = \begin{bmatrix} h_{\mathrm{m\text{-}1}} & h_{\mathrm{m\text{-}2}} & h_{\mathrm{m\text{-}3}} & ... & h_1 & h_0 \\ h_{\mathrm{m}} & h_{\mathrm{m\ -\ 1}} & h_{\mathrm{m\text{-}2}} & ... & h_2 & h_1 \\ ... & ... & ... & ... & ... & ... \\ h_{\mathrm{m+n\text{-}1}} & h_{\mathrm{m+n\text{-}2}} & h_{\mathrm{h+m\text{-}3}} & ... & h_{\mathrm{n}} & h_{\mathrm{n\ -\ 1}} \end{bmatrix}$$

If the inputs x_1, x_2 are uniformly and randomly distributed $\in GF(2)$ then the probability of any pair of bits being equal, a priori, is precisely half. The rows of $a(h)$ are independent as each row contains at least one bit not contained in any other row pair, assuming h itself is randomly distributed amongst the possible binary bit strings. We can convert the summation into a multiply due to the independence property.

$= \prod_{i=1}^{n} \Pr \forall h \in \mathcal{H}[\sum_{j=1}^{m} h_{\mathrm{i\ -\ j\ +\ m}} X_{\mathrm{j}} = \sum_{j=1}^{m} h_{\mathrm{i\ -\ j\ +\ m}} X'_{\mathrm{j}}] = \prod_{i=1}^{n} 1/2 = 2^{-n}$

Pairwise Independence

On each row, the value of h is zero which has the effect of cancelling out differences between x and y. This means instead of the probability of there existing a difference between x and y being 1/4 for each bit of x and y, it is reduced to half. The rows themselves are still independent as the probability that an individual h value is either 0 or 1 is half.

$a(h)_i$	x_i	y_i	$a(h)_i x_i$	$a(h)_i y_i$	difference
0	1	1	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	0	0	0
1	1	1	1	1	0
1	1	0	1	0	1
1	0	1	0	1	1
1	0	0	0	0	0

One can notice that when there are differences which is half of the time, half of the time when h is 0, the differences are cancelled out, for both bits. This means the probability of any one of them being a unique binary digit is limited when h takes on the value of 0. The two are not independent, as knowing $a(h)_i x_i$ is 0 gives us information to suggest that $a(h)_i y_i$ is more likely to be 0 also, and as such it cannot be pairwise independent.

2.2 Part 2

In order to prove $\mathcal{H}$ is pairwise independent, I first note that $h(x) = a(h)x$ remains universal, as the only difference between itself and the matrix in the proof of universality above is just the length of h, which does not affect the universality property with respect to a(h)x as the rows of the matrix are still row-wise independent.

Observing the bits of b(h) are disjoint and independent from a(h), one can see that addition in $GF(2)$ is equivalent to the XOR operator.

a(h)x	b(x)	XOR	$+ \in$ GF(2)
1	1	0	0
1	0	1	1
0	1	1	1
0	0	0	0

I note that $\oplus$ (XOR) is distributive and associative over GF(2), as it is equivalent to addition in

GF(2). One can now represent the output of the computation as follows, given input x and y:

$$\begin{bmatrix} h_{\text{m - 1}} & h_{\text{m-2}} & h_{\text{m-3}} & \dots & h_1 & h_0 \\ h_{\text{m + 1}} & h_{\text{m}} & h_{\text{m - 1}} & \dots & h_2 & h_1 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ h_{\text{m+n-1}} & h_{\text{m+n-2}} & h_{\text{m+n-3}} & \dots & h_{\text{n}} & h_{\text{n - 1}} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ \dots \\ \dots \\ x_{\text{m}} \end{bmatrix} \oplus \begin{bmatrix} h_{\text{m+n-1}} \\ h_{\text{m+n}} \\ \dots \\ \dots \\ \dots \\ h_{\text{2n+m-1}} \end{bmatrix} = \begin{bmatrix} h_{\text{m+n-1}} \oplus (h_{\text{m}} x_1 + & \dots & h_1 x_{\text{m}}) \\ & \dots & \\ h_{\text{m+2n}} \oplus (h_{\text{m+n}} x_1 + & \dots & h_{\text{n}} x_{\text{m}}) \end{bmatrix}$$

Formal analysis

If the probability of each hash function achieving any value, independent from the other, is 1/2, i.e. they are independent, then the joint probability of getting h(x) = u and h(x) = v is 1/4. In the analysis in part 1, I demonstrated that half of the time, h cancels out differences between the bit strings.

I will demonstrate why the XOR operation with random bits restores the property of pairwise independence. I first observe that:

$$\Pr \forall h \in \mathcal{H}[b(h) = 1] = 1/2 \wedge \Pr \forall h \in \mathcal{H}[b(h) = 0] = 1/2$$

$a(h)_i$	x_i	y_i	$a(h)_i x_i$	$a(h)_i y_i$	difference
0	1	1	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	0	0	0
1	1	1	1	1	0
1	1	0	1	0	1
1	0	1	0	1	1
1	0	0	0	0	0

The notion of difference captures the idea of pairwise independence. If the probability of a difference is half, it infers that given the value of either $a(h)_i x_i$ or $a(h)_i y_i$, the mutual information learned about the other is 0, i.e. it is independent.

$$\Pr \forall h \in \mathcal{H}[(a(h)_i x_i - a(h)_i y_i) = 1] = 1/4$$

$$\Pr \forall h \in \mathcal{H}[(a(h)_i x_i - a(h)_i y_i) = 0] = 3/4$$

Introducing b(h) changes the probabilities, as b(h) is half of the time 0 and half of the time 1.

$$\Pr \forall h \in \mathcal{H}[b(h) \oplus (a(h)_{\mathrm{i}} x_{\mathrm{i}} - a(h)_{\mathrm{i}} y_{\mathrm{i}}) = 1] =$$

$$\Pr \forall h \in \mathcal{H}[b(h) = 1] \wedge \Pr \forall h \in \mathcal{H}[(a(h)_{\mathrm{i}} x_{\mathrm{i}} - a(h)_{\mathrm{i}} y_{\mathrm{i}}) = 0] = 1/2 * 3/4 +$$

$$\Pr \forall h \in \mathcal{H}[b(h) = 0] \wedge \Pr \forall h \in \mathcal{H}[(a(h)_{\mathrm{i}} x_{\mathrm{i}} - a(h)_{\mathrm{i}} y_{\mathrm{i}}) = 1] = 1/2 * 1/4$$

$$= 3/8 + 1/8 = 1/2$$

The probability that the final result is 0 is the same, implying there is no information gained by knowing one of the values, about the other, inferring independence. Mutual information is defined as $log(p(x, y)/p(x) * p(y))$

This implies

$$\Pr \forall h \in \mathcal{H}[b(h) \oplus a(h)_{\mathrm{i}} x_{\mathrm{i}} = u \wedge -b(h) \oplus a(h)_{\mathrm{i}} y_{\mathrm{i}} = v] = 1/2 * 1/2 = 1/4 = 2^{-2n}$$

for an input of length n

2.3 Part 3

An algorithm is poly time solvable, if there exists a procedure, such that in polynomial time, it constructs and also verifies a witness which solves the problem at hand.

I first show that for $\alpha(m)$ is a constant k for any m, then it is never poly time solvable. I will show this by a proof by contradiction. If one assumes that it is poly time solvable for some k, then any 3-SAT problem can be reduced to it. If one takes an instance of 3-SAT, one constructs an instance of 2/3-SAT which is 1/k times larger than the original 3-SAT and one can fill the rest of the 2/3-SAT with random variables. As 3-SAT is NP-Hard, 2/3-SAT with constant $\alpha(m)$ is also NP-Hard in the general case.

I will now try to construct an algorithm that solves 2/3-SAT, such that the proportion of 3-SAT is at most log(m) clauses, in POLY time. As 3-SAT in general takes EXP(m) time, if there exist only log(m) clauses then the time will be EXP(log(m)) which is POLY time relative to m. I will now attempt to show that for a proportion of SAT that is log(m) 3-SAT and m - log(m) 2-SAT, there can be a POLY time solution.

First, I will construct an implication graph for the 3-SAT problem. N.B. The algorithm assumes 2-SAT $\in$ P.

Algorithm

1. 2-SAT(CLAUSES) takes as input a set of 2SAT clauses and consists of a linear time algorithm in which the 2-SAT implication graph is constructed and determined in the method of Aspvall, Plass and Tarjan (1979). Returns true if it can be satisfied, false otherwise.
2. CREATE-MODELS(takes the 3 clause to convert and the input and returns a set of 2/3SAT
Given: (x $\vee$ b $\vee$ c)

either

$$(b \vee c) \wedge (\neg x \vee \mathcal{F})$$

or

$$(x \vee c) \wedge (\neg b \vee \mathcal{F})$$

or

$$(x \vee b) \wedge (\neg c \vee \mathcal{F})$$

or

$$(b \vee c) \wedge (x \vee \mathcal{F})$$

or

$$(x \vee c) \wedge (b \vee \mathcal{F})$$

or

$$(x \vee b) \wedge (c \vee \mathcal{F})$$

I denote $\mathcal{F}$ as a variable which is set to false. The usual 2-SAT implication graph is constructed with this.

3. APPEND(x, (X)) simply appends x to (X) and returns (X)

The construction of the 2-SAT implication graph covers the instances where exactly one of the variables out of the three will be true. Whether they are here or not does not affect the resulting analysis.

Input: $((X) \in 2/3SAT)$
ALGO$((X) \in 2/3SAT)$:
    if ($\exists$ 3SAT clause $\in (X)$):
        foreach($x \in$ CREATE-MODELS(clause)) // exactly 6 are created,
            $z =$ APPEND(x, (X)) // constant time conversion.
            ALGO(z)
    else
        y = 2-SAT((X)) // linear time, true if satisfiable.

```
        if y == true:
            return true
return false
```

If a single branch of the computation returns true, this implies there is a possible assignment for each of the variables such that there are no contradictions, as all possible worlds have been explored and converted to 2-SAT.

If none of the branches return true, then it is not satisfiable as contradictions are found in all possible assignments of the 3-SAT to 2-SAT conversion.

Time complexity

If there are at most log(m) clauses, each clause splits the search space into 6 independent tasks. The 2-SAT algorithm takes linear time to determine whether the clause is satisfiable or not i.e. it checks for contradictions ($\neg x \Rightarrow x$) which takes time O(V + E) with a breadth first search implementation.
On each level, the whole input must be scanned to check for, and convert, a single 3-SAT clause which can be done in O(m) time. This yields the recurrence formula T(K) = 6T(K - 1) + S where K is the number of 3 SAT clauses, and S is input length.

The height of the tree is K, and at each level the cost is $6^i * S$. As the height is at most log(n) where n is the total number of clauses, the cost is at most: $\sum_{j=1}^{log(m)} 6^i * S$

This is a geometric series with common ratio 6 and a = S, with formula $S_\text{n} = a_1(1 - r^n)/(1 - r)$

Calculating the output then yields: $S * (1 - 6^{log_6(m)})/(-2) \leq S * ((1 - m)/ - 2) = S * ((m - 1)/2)$
As S is at most the input length km, it takes at most $\mathcal{O}(km^2)$ to solve which is POLY time. If none of the input clauses are 3-SAT then it will terminate in linear time as it just executes the 2-SAT algorithm. □