Data-aware Algorithm to Solve Discrete Integration by Parity Constraints and Optimization

Drafting

Yupeng Han

January 26, 2019

1. ABSTRACT

I am here to introduce an approximate algorithm aim at solving discrete integration in very high dimension which is a fundamental and largely unsolved problem of scientific computation with numerous applications, i.e., the partition function of discrete graphical models. It estimates the exponentially large set discrete integrals by solving a small number of instances of a discrete combinatorial optimization problem subject to randomly generated parity constraints. The new algorithm extends the WISH algorithm by using data-aware strategy to reduce the number MAP query without loss of guarantee to give a constant-factor approximation.

2. INTRODUCTION

The same idea as the WISH algorithm New algorithm estimate the discrete integral by estimate points in the tail distribution of discrete integral. New algorithm divide the interval by dichotomy. Calculation the difference between the maximum and minimum of the area of the tail distribution in current interval in the case of query no point in the current interval. Get the ratio between the calculate difference with global lower bound. If the ratio is smaller than comparison factor, namely if query no points within this interval, The overall estimate will not be seriously affected, New algorithm will decide to not query any points within this interval. The comparison factor is not a constant, it will shrink proportion to the length of the interval in logarithmic scale. One can understand this for the longer the interval, the more times the MAP query needs to be called. The more MAP query that can be saved by omitting this interval, the larger the comparison parameter is used. Conversely, when the interval is shorter, the calculation amount saved by this interval is relatively small. , with relatively small comparison parameters, I can keep the guarantee to give a constant-factor approximation.

The amount of computation that the NEW algorithm can reduce comparing to WISH

algorithm depends to great extend on the discrete integrals itself. However, I provide a regret bound to indicate that if exist a “optimal” algorithm, have pre-knowledge to know the shape of tail distribution of discrete integral, New algorithm have a log distance to “optimal” algorithm comparing to WISH algorithm. Where N stands number of discrete items for the original integral problem.

$$|NEW\ Algorithm - "Optimal"| \leq log(log(N)) \times MAPquery \tag{2.1}$$

$$|WISH\ Algorithm - "Optimal"| \leq log(N) \times MAPquery \tag{2.2}$$

Another advantage of this algorithm is that it reduces query without loss of constant-factor approximation guarantee. Assume that we can use MAP query to find the exact value of a point in the tail distribution of the original discrete integral problem (This assumption does not need to hold, here use it to simplify the description). The way I designed how to shrink the comparison factor, under this assumption, I can prove that the upper bound is less than three times of lower bound New algorithm provided. While WISH algorithm showed that the upper bound is less than two times of lower bound WISH can give.

3. PROBLEM DEFINITION AND ASSUMPTIONS

3.1 Problem Definition

Let Σ be the set of elements, ω is a mapping method. For each element x in set Σ, there exist and only exist one specific $\omega(x)$, note that the mapping is not necessary to be bijection nor injection. In other words, a couple of x may point to the same $\omega(x)$. This paper mainly attempts to estimate the W defined below.

$$W = \sum_{x \in \sum} \omega(x) \tag{3.3}$$

3.2 Assumption

Assume that we have access to an NP oracle which can find the maximum a posterior (MAP) queries, namely solving the following constrainted optimization puzzle.

$$arg \quad max_{x \in \sum} p(x|C) \tag{3.4}$$

That is, we can find the most likely state (and its weight) given some evidence C. This is a strong assumption because MAP inference is known to be an NP-hard problem in general. Notice that counting problem we try to estimate is a #P-complete problem, a complexity class believed even harder than NP. It can be a natural assumption that we have access to an NP-oracle.

Note that we can use this MAP inference to solve the following problem.

$$arg \quad max_{x \in \sum'} w(x) \tag{3.5}$$

where $\sum'$ is a subset of the $\sum$ (a set introduced earlier) and $\sum'$ contains all the elements fits the given evidence C. And $w(x)$ can be considered as $p(x)$ in the MAP query.

MAP_Oracle(x)

We will use MAP_Oracle(x) to represent this NP oracle. MAP_Oracle(x) is a solver where input x is independent variable in a non-increasing step function C(x), the output is an "close" estimation of C(x). Note that C(x) can be considered as an unknown mapping and we can only access the values of C(x) by calling this NP-oracle.

4. PRELIMINARIES

4.1 Definition 1.

A family of functions $H = h : \{0,1\}^n \rightarrow \{0,1\}^m$ is pairwise independent if the following two conditions hold when H is a function chosen uniformly at random from H. 1) $\forall x \in \{0,1\}^n$, the random variable $H(x)$ is uniformly distributed in $\{0,1\}^m$. 2) $\forall x_1, x_2 \in \{0,1\}^n \quad x_1 \neq x_2$, the random variables $H(x_1)$ and $H(x_2)$ are independent.

4.2 Proposition 1.

Let $A \in \{0,1\}^{m\times n}, b \in \{0,1\}^m$. The family $H = h_{A,b}(x) : \{0,1\}^n \to \{0,1\}^m$ where $h_{A,b}(x) = Ax + b \, mod \, 2$ is a family of pairwise independent hash functions.

The space $C = \{x : h_{A,b}(x) = p\}$ has a nice geometric interpretation as the translated nullspace of the random matrix A, which is a finite dimensional vector space, with operations defined on the field F(2) (arithmetic modulo 2). We will refer to constraints of the form $Ax = b \, mod \, 2$ as parity constraints, as they can be rewritten in terms of logical XOR operations as $A_{i1}x_1 \oplus A_{i2}x_2 \oplus \cdots \oplus A_{in}x_n = b_i$.

4.3 Counting by Hashing and Optimization

Recent theoretical result that transforms a counting problem to a series of optimization problems. The key idea in this paper is design a practicable method to reduce the number of optimization needed. In order to compute W as defined in Equation (2.1), we define the tail distribution of weights as $G(u) \triangleq |\sigma| w(\sigma) \geq u|$. Then W can be rewritten as $\int_{R+} G(u)$, in other words, we can calculate W by estimating the area under the curve of $G(u)$ vs u, see Figure 1.

Former paper already proofed that one can approach area W under the $G(u)$ by horizontal

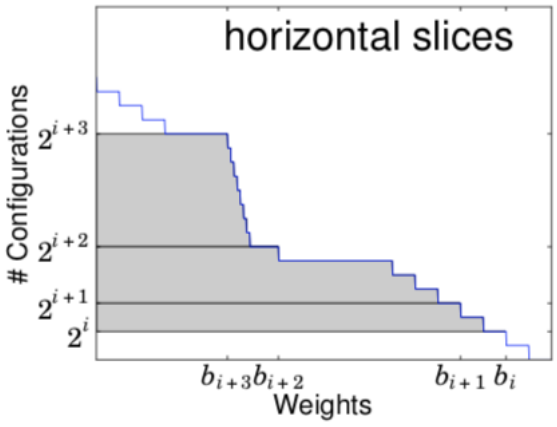

Figure 1: Use tail distribution to estimate discrete integral

slices(split the area based on y-axis). Let $b_0 \geq b_1 \geq b_2 \geq ... \geq b_n$ be the weights of the configuration at the split points. If one can get exact value of b_i, we can approach W with a 2-approximation, namely the upper bound we can give is less than 2 times of the lower bound we can provide. (Detail proof can be found in Ermon's paper "Taming the Curse of Dimensionality: Discrete Integration by Hashing and Optimization"). What this paper want to introduce is a method to carefully select which b_i to solve, without loss of constant guarantee. The merits of reduce the number of instances to solve is that this can reduce the total running time, to some special cases, the number of instances to solve can be reduce from n to log(n). (When the original problem is to integral over 2^n discrete items.)

4.4 Data-aware strategy

4.4.1 Definition of clb, cub and AoI

Let use $C(x)$ represent the inverse function of $G(u)$, where x corresponds to the series number of configurations based on the descending order of their weights and $C(x)$ is the x_{th} most massive weights. Assume that we are dealing with a problem that n binary bits can accurate describe a certain configuration, in this case, we can know x in the range $[2^0, 2^n]$. Assume $C(x_1)$, $C(x_2)$,$C(x_3)$,...,$C(x_k)$, is known at point x_1, x_2, x_3, ..., x_k where $x_1 \leq x_2 \leq x_3 \leq ... \leq x_k$ and $C(x_1) \geq C(x_2) \geq C(x_3) \geq ... \geq C(x_k)$.

Let C, x_1, x_2, ... , x_k, $C(x_1)$, $C(x_2)$, ... , $C(x_k)$ be as above. Use vector $\hat{x}$ and $C(\hat{x})$ to represent those k Points. $\hat{x} =< x_1, x_2, ..., x_k >$ and $C(\hat{x}) =< C(x_1), C(x_2), ..., C(x_k) >$. Define clb, cub and AoI, each parameterized by $\hat{x}, C(\hat{x})$.

Current Lower Bound(clb) & Area under the Lower Bound(alb)

$$clb(\hat{x}, C(\hat{x}), \tilde{x}) = \begin{cases} 0, & \tilde{x} < x_1 \quad or \quad \tilde{x} > x_k \\ C(x_{i+1}), & \tilde{x} > x_i \quad and \quad \tilde{x} \leq x_{i+1} \end{cases} \tag{4.6}$$

clb stands for tight lower bound for every point $C(\tilde{x})$ at $\tilde{x}$ if we know the value of set points $\hat{x}$ and there corresponding $C(\hat{x})$. In the rest of this paper, $clb(\tilde{x})$ is used to represent

$clb(\hat{x}, C(\hat{x}), \tilde{x})$ in short.

For convenience, the rest of paper will use alb to represent the integral of the clb, more specific definitions showed below.

$$alb(\hat{x}, C(\hat{x})) = \int_{x_1}^{x_k} clb(\hat{x}, C(\hat{x}), x)dx = \tag{4.7}$$

$$C(x_2) \times (x_2 - x_1) + C(x_3) \times (x_3 - x_2) + ... + C(x_k) \times (x_k - x_{k-1}) \tag{4.8}$$

$$= \sum_{i=1}^{k-1} C(x_{i+1}) \times (x_{i+1} - x_i) \tag{4.9}$$

This is a variable that will be monitored and updated during the calculation process. The alb stands for the area under the current lower bound(clb) of the step curve $C(x)$ within a certain range of x-axis(as $\hat{x}$ defined above, from x_1 to x_k). In other words, alb is integral the lower bound based on the information of one have at current algorithm step.

For instance, $alb(< x_2, x_4 >, < C(x_2), C(x_4) >)$ stands lower bound of the area under the step curve $C(x)$ where x range from x_2 to x_4. When the scale is not mentioned, alb stands for the lower bound of the area under the step curve in all x ranges. Note that the alb is a non-decreasing variable by adding more points to the vector within a fixed boundary. Further details about the alb can be found in the example below.

Current Upper Bound(cub) & Area under the Upper Bound(aub)

$$cub(\hat{x}, C(\hat{x}), \tilde{x}) = \begin{cases} 0, & \tilde{x} < x_1 \quad or \quad \tilde{x} > x_k \\ C(x_i), & \tilde{x} \geq x_i \quad and \quad \tilde{x} < x_{i+1} \end{cases} \tag{4.10}$$

cub stands for tight upper bound for every point $C(\tilde{x})$ at $\tilde{x}$ if we know the value of set points $\hat{x}$ and there corresponding $C(\hat{x})$. In the rest of this paper, $cub(\tilde{x})$ is used to represent $clb(\hat{x}, C(\hat{x}), \tilde{x})$ in short.

$$aub(\hat{x}, C(\hat{x})) = \int_{x_1}^{x_k} cub(\hat{x}, C(\hat{x}), x)dx = \tag{4.11}$$

$$C(x_1) \times (x_2 - x_1) + C(x_2) \times (x_3 - x_2) + ... + C(x_{k-1}) \times (x_k - x_{k-1}) \tag{4.12}$$

$$= \sum_{i=1}^{k-1} C(x_i) \times (x_{i+1} - x_i) \tag{4.13}$$

aub stands for the upper bound of the area under the step curve within a certain range of x-axis(as $\hat{x}$ defined above, from x_1 to x_k) in the current stage. When the scale is not mentioned, aub stands for the lower bound of the area under the step curve in all x ranges. Note that the aub is a non-increasing variableby adding more points to the vector within a fixed boundary.

Amplitude of an Interval(AoI)

$$AoI(\hat{x}, C(\hat{x})) = aub(\hat{x}, C(\hat{x})) - alb(\hat{x}, C(\hat{x})) \tag{4.14}$$

$$= \sum_{i=1}^{k-1} \{[C(x_i) - C(x_{i+1})] \times (x_{i+1} - x_i)\} \tag{4.15}$$

This is a variable to describe the difference between the upper bound and the lower bound for a specified interval(as $\hat{x}$ defined above, from x_1 to x_k). The interval is based on the x-axis. Define the Amplitude of an Interval for the interval (x_1, x_2) as $AoI(x_1, x_2)$

4.5 Theorem 1

Let G be any non-increasing step function. Given a set of $\hat{x}$ and the corresponding $G(\hat{x})$. Then, for any new $\tilde{x}$ within the range of $\hat{x}$, we have:

$$clb(\hat{x}, C(\hat{x}), \tilde{x}) \leq G(\tilde{x}) \leq cub(\hat{x}, C(\hat{x}), \tilde{x}) \tag{4.16}$$

And we have:

$$alb(\hat{x}, C(\hat{x}), \tilde{x}) \leq \int_{x_1}^{x_k} G(x)dx \leq aub(\hat{x}, C(\hat{x}), \tilde{x}) \tag{4.17}$$

$$alb(\hat{x}, C(\hat{x}), \tilde{x}) \leq \int_{x_1}^{x_k} G(x)dx \leq alb(\hat{x}, C(\hat{x}), \tilde{x}) + AoI(x_1, x_2) \tag{4.18}$$

5. New Algorithm

Armed with the notion of AoI characterizing all non-increasing step functions(C(x)) that could possibly pass through observed points, we describe New Algorithm.

The main idea is to use dichotomy sequence to query the $C(2^i)$, from the boundary to the middle, i.e. First to query the beginning($C(2^0)$), the end($C(2^n)$) and then the mid-point($C(2^{\frac{0+n}{2}})$). After getting the first few points, we can update the alb, and AoI. Before next query, we first compare AoI with $x \times alb$, x here is a factor to determine the significance of the current interval. if the AoI is less than $x \times alb$, we can skip all points in current interval due to without the exact value of those points, the final will not impact a lot. Here x is getting smaller and smaller with the interval getting smaller and smaller. This is due to, when we skip a relatively large interval, we can skip more points to query, so I use a relatively larger x to make it more easy to skip current interval. The strict definition of x and how this method works is shown bellow, another thing to mention is that how to shrink x is carefully designed. Based on my methods, I can give a 3-approximation methods, namely the upper bound is less than 3 times of lower bound I can give, when all the points are exact estimated. However, since we can only query the points by using parity constraints with NP Oracles, one cannot get the exact value of points. I am currently working on this problem.

The idea is first to query the beginning($C(2^0)$), the end($C(2^n)$) and the mid-point($C(2^{\frac{0+n}{2}})$), at this point we separate original range of x into two parts: $(2^0, 2^{\frac{0+n}{2}})$ and $(2^{\frac{0+n}{2}}, 2^n)$.

Then making further queries in the middle for each interval(e.g. query $2^{\frac{3n}{4}}$ for interval $(2^{\frac{0+n}{2}}, 2^n)$) and compute the AoI for each interval induced by current observations(e.g. $AoI(2^{\frac{0+n}{2}}, 2^n) = [C(2^{\frac{n}{2}}) - C(2^{\frac{3n}{4}})] \times (2^{\frac{3n}{4}} - 2^{\frac{n}{2}}) + [C(2^{\frac{3n}{4}}) - C(2^n)] \times (2^n - 2^{\frac{3n}{4}})$), and compare each AoI with the $x \times alb$(area under the lower bound) to decide whether to continue making further queries within this interval. x here is a parameter determined by the stage of the algorithm or say the number of queries. It is obvious that the deeper the algorithm goes, more queries we do, tighter the AoI we can get. So we use a parameter x to manipulate the constrain AoI compare with. At the beginning, we have large x, to relax the constrain. With the algorithm goes deeper, we have more and more tighter AoI, the x become small. This is also beneficial for our ultimate goal, reduce the computation power. At the very beginning, if a interval meets the constrain, we can save neglect half of the MAP_Oracle calls, with the program going on, we are looking at more and more detail of this problem, the amount of computation power we can save is smaller and smaller, so we use more and more strict constrains.

Definition 4

Following the former definition, C(x) is a non-increasing step function which stands for a tail distribution of another discrete function. For the convenience of the paper, assume we have at most 2^n. $C(0)$ is the maximum value of $C(x)$ and $C(2^n)$ is the minimum. Define C_i as follows:

$$C_i = C(2^i) \tag{5.19}$$

C_i is the 2^ith largest value in $C(x)$.

Mark Constraint

This is the constraint that marks an interval "finished" introduced in the pseudo code. The math detail of mark constraint can be represented as follows:

for interval (x_1, x_2), $x_1 = 2^j$, $x_2 = 2^k$, if $AoI(x_1, x_2) \leq \frac{1}{2}^{\frac{n}{k-i}} \times alb$, then this interval (x_1, x_2) fits the mark constraint, mark interval (x_1, x_2) as "finished".

Stopping Criterion

The process stops either all the points that can be described in form 2^t(t is an integer) is estimated by calling MAP_Oracle, or there is no more intervals need further estimation (either fully estimated or marked by "finished").

Upper Bound on the Number of Query Calls

The above stopping conditions imply that New Algorithm never makes more calls than WISH algorithm. In the worst case, when no interval skip, the New Algorithm call at most n times of NP Oracles, which is the same number of calls to the WISH algorithm, assuming the the original weighted sum problem have 2^n items to sum over.

5.1 Peudo Code

for $t = 1, ..., T$ **do**

  w_0^t =Map_Oracle(2^0);

  w_n^t =Map_Oracle(2^n);

end

$M_0 = Median(w_0^1, ..., w_0^T)$;

$M_n = Median(w_n^1, ..., w_n^T)$;

for i = *1,2,...,n-1* **do**

  $M_i = M_0$

end

for i=*1,2,...,*$log_2(n)$ **do**

  for *Every interval* (x_1, x_2), $x_1 = 2^j$, $x_2 = 2^k$, *not marked by "finished"* **do**

    % Use x_m denote the point under the Half-Exponential Condition of this

    % interval, i.e.,when $x_1 = 2^j$, $x_2 = 2^k$, $x_m = 2^{\frac{j+k}{2}}$.

    for $t = 1, ..., T$ **do**

      w_i^t =Map_Oracle(2^i);

    end

    $M_i = Median(w_i^1, ..., w_i^T)$; **for** $j = i, ...n$ **do**

      $M_j = M_i$;

    end

  end

  Update the clb, alb;

  for *Every interval* (x_1, x_2), $x_1 = 2^j$, $x_2 = 2^k$, *not marked by "finished"* **do**

    if $AoI(x_1 x_2) \leq \frac{1}{2}^{\frac{n}{k-i}} \times alb$ **then**

      Mark interval (x_1, x_2) as "finished";

  end

  if *every interval*(x_1, x_2), *already marked by "finished"* **then**

    break;

end

Return $M_0 + \sum_{i=0}^{n-1} M_{i+1} 2^i$

Algorithm 1: Pseudocode

6. Analysis

6.1 Theorem 2

If we looks closer to the Mark Constrain, we will find that there is only one of intervals can be mark by "finish" when we estimate the mid-point of a interval and separate the original interval into two.

Definition 5

Following the former definition, following the New Algorithm, there are some of the intervals not fully estimated. Recall the WISH algorithm, they estimate all the points can be written as $C(2^i)$ $\forall 0 \leq i \leq n$, we only estimate part of those points. This is due to we neglect some queries of some points which fits the Mark Constraint(defined in Chapter 5). Here we define a_i below:

We have a_i numbers of intervals (x_1, x_2) that can be written as $x_1 = 2^{j*\frac{n}{2^i}}$, $x_2 = 2^{(j+1)*\frac{n}{2^i}}$

For instance, a_1 is the number of intervals can be written as $(2^{j*\frac{n}{2}}, 2^{(j+1)*\frac{n}{2}})$.

Note that $a_1 \leq 2$, because there is only 2 intervals can be written as that. The two intervals can be presented in that form is $(0, 2^{n/2})$ and $(2^{n/2}, 2^n)$.

Moreover, based on theorem 2, $a_1 \leq 1$.

6.2 Theorem 3

Following the former definitions, armed with notation of the number of intervals meets the mark constrain and Theorem 2, now I say we have an in-equilibrium for number of intervals a_i $\forall 1 \leq i \leq n$

$$a_i \leq (2^i - 2^{i-1} \times a_1 - 2^{i-2} \times a_2 - ... - 2 \times a_{i-1})/2 \tag{6.20}$$

Proof: This is because, when we calculating a_i, due to definition 5, we at most have 2^i intervals(so 2^i is the first item in left hand side). If we have a a_1 numbers of intervals like $(2^{j\times\frac{n}{2}}, 2^{(j+1)\times\frac{n}{2}})$. Note that a interval looks like $(2^{j\times\frac{n}{2}}, 2^{(j+1)\times\frac{n}{2}})$ contains 2^{i-1} intervals

looks like $(2^{j\times\frac{n}{2^i}}, 2^{(j+1)\times\frac{n}{2^i}})$. So we need to subtract this part that is double-counted(which is second item on the left hand side), and so on.

In the end of the left hand side, we divide the total number of rest intervals into two, this is due to the Theorem 2.

6.3 Theorem 4

Armed with Theorem 3, we have:

$$AoI \leq alb \tag{6.21}$$

AoI defined in 4.1. This is a variable to describe the difference between the upper bound and the lower bound. It can be written as:

$AoI(\hat{x}, C(\hat{x})) = aub(\hat{x}, C(\hat{x})) - alb(\hat{x}, C(\hat{x})) = \sum_{i=1}^{k-1}\{[C(x_i) - C(x_{i+1})] \times (x_{i+1} - x_i)\}$

alb defined in 4.1. The alb stands for the area under the current lower bound(clb) of the step curve $C(x)$

Proof: With Theorem 3, we have

$a_i \leq (2^i - 2^{i-1} \times a_1 - 2^{i-2} \times a_2 - ... - 2 \times a_{i-1})/2$

Modify this in-equilibrium(Dividing both sides simultaneously by 2^{i-1}), we can get:

$\frac{1}{2} \times a_1 + \frac{1}{2^2} \times a_2 + ... + \frac{1}{2^i} \times a_i \leq 1$

Multiply both sides simultaneously by alb, we can get:

$(\frac{1}{2} \times alb) \times a_1 + (\frac{1}{2^2} \times alb) \times a_2 + ... + (\frac{1}{2^i} \times alb) \times a_i \leq 1 \times alb$

Looking at the in-equilibrium more closely, we can find that $(\frac{1}{2} \times alb)$ is the mark constrain for a_1, $(\frac{1}{2^2} \times alb)$ is the mark constrain for a_2 and so on. Since we following in-equilibrium due to Mark Constrain:

$$AoI_{"finish"} \leq (\frac{1}{2} \times alb) \times a_1 + (\frac{1}{2^2} \times alb) \times a_2 + ... + (\frac{1}{2^i} \times alb) \times a_i \tag{6.22}$$

Here $AoI_{"finish"}$ stands for the AoI of intervals that be marked as "finish".

So we have:

$AoI_{"finish"} \leq alb$

Definition 6

For ease of explanation, let use l_w to denote the sum of area under the the lower bounds of the intervals that are not marked by "finished".

let use u_w to denote the sum of area under the the upper bounds of the intervals that are not marked by "finished".

Let use l_j to denote the sum of area under the the lower bounds of the intervals that are not marked by "finished".

let use u_j to denote the sum of area under the the upper bounds of the intervals that are not marked by "finished".

From the details in the "Taming the Curse of Dimensionality: Discrete Integration by Hashing and Optimization". we can know that $u_w \leq 2 \times l_w$. As defined in 4.1 alb denote the sum of area under lower bounds for all the intervals. In other words $alb = l_w + l_j$ same with aub denote the sum of area under upper bounds for all the intervals in the final step. $aub = u_w + u_j$. $AoI_{"finish"} = u_j - l_j$

6.4 Estimate the Total Weight

With the definitions above, (if we have access to the true values of all C_i), my algorithm could obtain a 3-approximation to W, which is equals to:

$$3 \times alb \geq aub \tag{6.23}$$

Proof:

$$l_w + 2 \times l_j \geq l_w + l_j \tag{6.24}$$

This is due to right hand side have one more non negative item.

$$l_w + 2 \times l_j \geq alb \tag{6.25}$$

(6.26)

This is due to $alb = l_w + l_j$.

$$l_w + 2 \times l_j \geq AoI_{"finish"} \tag{6.27}$$

(6.28)

This is due to $AoI_{"finish"} \leq alb$.

$$l_w + 2 \times l_j \geq u_j - l_j \tag{6.29}$$

(6.30)

This is due to $AoI_{"finish"} = u_j - l_j$

$$l_w + 3 \times l_j \geq u_j \tag{6.31}$$

$$2l_w \geq u_w \tag{6.32}$$

Details for this can be found in "Taming the Curse of Dimensionality: Discrete Integration by Hashing and Optimization".

Add (6.26) and (6.27) we can get:

$$3 \times l_w + 3 \times l_j \geq u_j + u_w \tag{6.33}$$

$$3 \times LB \geq UB \tag{6.34}$$

End of proof.

6.4.1 Lemma 1.

Let $M_i = Median(w_i^1, ..., w_i^T)$ be defined as in Algorithm 1 provides and C_i as in Definition 4. Then, for any $d \geq 2$, there exists $\alpha^*(d) > 0$, such that for $0 \leq \alpha \leq \alpha^*(d)$,

$$Pr[M_i \in [C_{min\{i+d,n\}}, C_{max\{i-d,0\}}]] \geq 1 - exp(\alpha T) \tag{6.35}$$

Proof Sketch. This is the same as "High Dimensional Discrete Integration by Hashing and Optimization" Lemma 1.

6.4.2 Definition 7

$\Phi(C_i)$ is a function will pick the nearest max C_j that queried in my Algorithm 1.

$\Omega(C_i)$ is a function will pick the nearest min C_j that queried in my Algorithm 1. For instance

6.4.3 Lemma 2.

Let $L'' \triangleq C_0 + \sum_{i=0}^{n-1} \Omega(C_{min\{i+d+1,n\}})2^i$ and $U'' \triangleq C_0 + \sum_{i=0}^{n-1} Phi(C_{max\{i+1-d,0\}})2^i$. Then $U'' < (2^{2d+1} + 2^d)L''$

Proof Sketch. The proof with details can be found in Appendix. It is clear that if we follow the algorithm 1. For accurate analysis, we can write W as:

$$W \triangleq \sum_{j=1}^{2^n} w(\sigma_j) \triangleq [L, U] \tag{6.36}$$

Note that $U \leq 3L$, we just prove this in 6.4. Hence, if we had access to the true values of all C_i, we could obtain a 3-approximation to W. We do not know true C_i values, but Lemma 1 shows that the M_i values computed by Algorithm 1 are sufficiently close to C_i with high probability. Specifically, applying Lemma 1 with $T = [\frac{ln(n/\delta)}{\alpha}]$, we can show that with probability at least $(1 - \delta)$, the out put of Algorithm 1 lies in [L", U"] as de fined in Lemma 2. Observinng that [L, U] is contained in [L", U"] and applying Lemma 2, we have a $(2^{2d+1} + 2^d)$- approximation of W. Fixing d = 2 and noting that $\alpha^*(2) \geq 0.0042$ finishes the proof.

6.5 Regret Bound

Imagine we have a "optimal" algorithm, have pre-knowledge to know the shape of inverse tail distribution of discrete integral, do minimum number of MAP query at points (which can be written as $x = 2^n$), to estimate the of discrete integral with a same constant guarantee. The "optimal" algorithm will know where in the inverse tail distribution exist a dramatically drop, the strict definition is written in Data-aware Algorithm Working Condition. By saying dramatically drop, what I mean is, given an interval, the inverse tail distribution within this interval a cliff between platforms. A graph like following graph, see in example Figure 2.

The optimal algorithm know when the tail distribution will drop so when "optimal" algorithm try to estimate A, the area under this inverse tail distribution graph, it only need to query at most 2 points(Given the start point and the end point).

For interval($2^{start}, 2^{end}$), the "optimal" algorithm know point $x_{critical}$ will dramatically drop. So the strategy of querying for "optimal" algorithm is:

$$"optimal" \quad algorithm \quad query \quad \begin{cases} C(2^i) \; and \; C(2^{i+1}), & 2^i \le x_{critical} \le 2^{i+1} \\ C(2^i), & x_{critical} = 2^i \end{cases} \tag{6.37}$$

If the entire inverse tail distribution for any spliced interval, there is no such situation(Means do not have a interval that have two platform and a cliff, this is the case where data-aware algorithm can work to skip some points to reduce the total workload). Then, if the original problem is to integral over 2^n discrete items, even the optimal algorithm need $n+1$ times of calls, WISH algorithm need $n+1$ times of calls, my algorithm need $n+1$ times of calls. In this case, my algorithm is as good as "optimal" algorithm.

For situation that my algorithm is not as good as opt algorithms. If there exist a interval $(2^j, 2^k)$ that satisfied the condition that data-aware algorithm can work. In this case, "optimal" algorithm at most need to query 4 points. Assuming $2^i \le x_{critical} \le 2^{i+1}$, "optimal" solution need to query $C(2^j)$, $C(2^k)$, $C(2^i)$ and $C(2^{i+1})$, see example in Figure 2

Meanwhile my algorithm need at most need $log_2(k-j)+2$ queries. WISH algorithm need

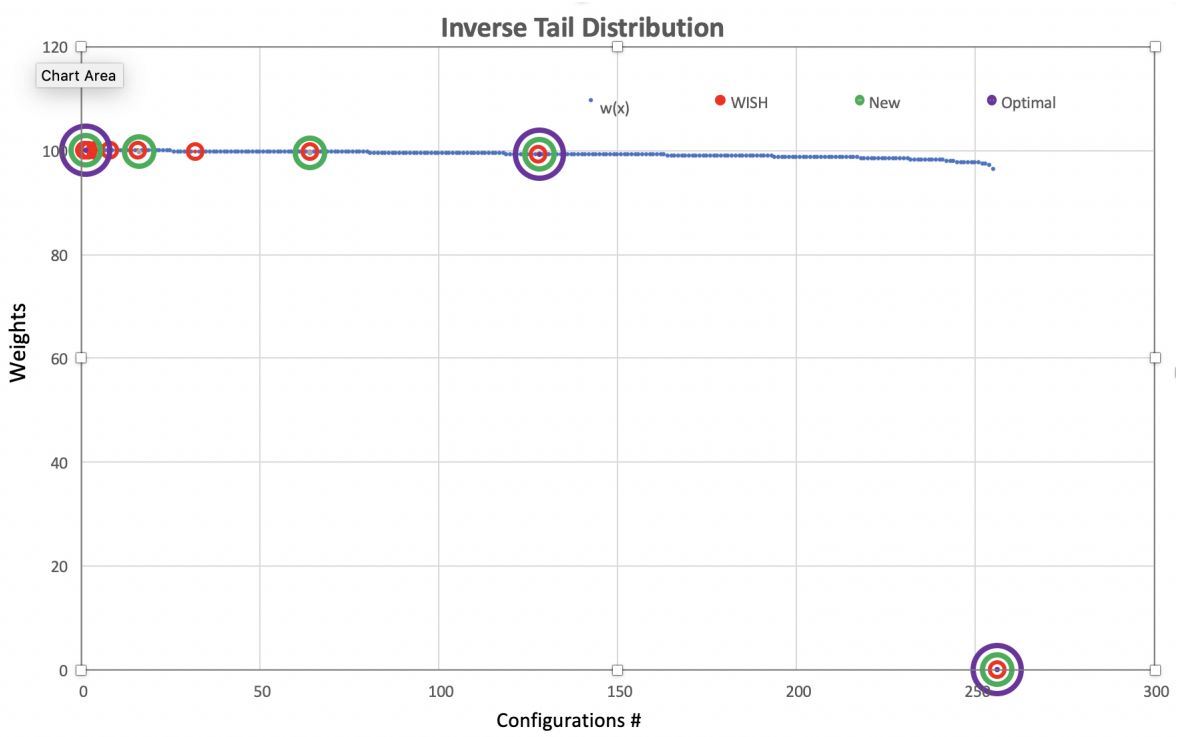

Figure 2: Example of Regret Bound

$k-j+1$ queries. In the extreme condition, the entire inverse tail distribution is satisfied the Data-aware Algorithm Working Condition, the "optimal algorithm" need 4 queries, New Algorithm need $log_2(n)+2$ queries and WISH algorithm need $n+1$ queries.

$$|NEW\ Algorithm - "Optimal"| \leq log(n) \times MAPquery \tag{6.38}$$

$$|WISH\ Algorithm - "Optimal"| \leq n \times MAPquery \tag{6.39}$$

Data-aware Algorithm Working Condition

Here I define the condition that data-aware strategy can works to reduce workload. For interval $(2^j, 2^k)$, exist point $x_{critical}$, assume $2^i \leq x_{critical} \leq 2^{i+1}$. Based on the how New Algorithm shrinking the comparison factor x, defined in Chapter 4. If:

$$(C(2^j) - C(2^i)) \times (2^i - 2^j) + (C(2^i+1) - C(2^k)) \times (2^k - 2^{i+1}) \leq \frac{j-k}{n} \times alb \tag{6.40}$$

We say this is a interval under Data-aware Algorithm Working Condition.

7. Appendix: Proofs

Lemma 2 First of all, I would like to define L' and U'. $L' \triangleq C_0 + \sum_{i=0}^{n-1} C_{min\{i+d+1,n\}} 2^i$ and $U' \triangleq C_0 + \sum_{i=0}^{n-1} C_{min\{i+1-d,n\}} 2^i$. From paper "High Dimensional Discrete Integration by Hashing and Optimization" we know that $U' \leq 2^{2d} L'$

Imagine that if my algorithm is the same as WISH, then we would have $L' = L''$. So the difference between L" and L' is the points (C_i) we ignored. If, in on step of Algorithm 1, we didn't query x points, then we need to use Φ function to substitute the nearest min of those points when calculating the lower bound.

$$L' - L'' = \sum (C_{m+d} - C_{m+d+\frac{n}{x}})(2^{m+n/x-1} - 2^m) \leq \frac{1}{2^{d+1}} AoI \leq 1/2L' \tag{7.41}$$

$$L' - L'' = \sum (C_{m+d} - C_{m+d+\frac{n}{x}})(2^{m+n/x-1} - 2^m) \tag{7.42}$$

This is because we rewrite the lower bound and use the points we queried in Algorithm 1 to substitue the lower bound.

$$\frac{1}{2^{d+1}} AoI \leq 1/2L' \tag{7.43}$$

This is because $\sum AoI = 2^d L'$. Similarly, we have

$$U'' - U' = \sum (C_{m+d} - C_{m+d+\frac{n}{x}})(2^{m+n/x-1} - 2^m) \leq \frac{1}{2^1} AoI \leq 2^{d-1} L' \tag{7.44}$$

So we have, $U'' \leq (2^{2d} + 2^{d-1})L'$ and $L'' \geq \frac{1}{2} L'$, so $U'' < (2^{2d+1} + 2^d)L''$. This finish the proof.