Coding Progress

Aaryan Gupta

July 2022

1 11 July - 15 July

The tasks I worked on this week were:

1.1 Integration of RoughMC and LogSATsearch code and division of probability

I have successfully integrated the RoughMC code with the LogSATsearch code and have uploaded it on the main branch. As of now, the probability value δ is divided as 0.02 with roughMC and 0.18 with the number of measurements or main procedure.

This partition is not optimal and here are a couple of ways this can be resolved-

- **Analytically Dependent on n:**
  Runtime for old ApproxMC: $O\left(\frac{log(n)log(\frac{1}{\delta})}{\epsilon^2}\right)$
  Runtime for new ApproxMC: $O\left(\frac{log(n)}{\epsilon^2} + \frac{log(m_1+m_2)log(\frac{1}{\delta_{main}})}{\epsilon^2}\right) = O\left(\frac{log(n)}{\epsilon^2} + \frac{log(\frac{1}{\delta_{rough}})\cdot log(\frac{1}{\delta_{main}})}{\epsilon^2}\right)$
  We use the relation $\delta_{rough} + \delta_{main} = \delta$ to get:-
  Runtime for new ApproxMC $= O\left(\frac{log(n)}{\epsilon^2} + \frac{log(\frac{1}{\delta_{rough}})\cdot log(\frac{1}{\delta-\delta_{rough}})}{\epsilon^2}\right)$
  Differentiating with respect to δ_{rough}, we see that the maxima occurs at $\delta_{rough} = \frac{\delta}{2}$. However, we need the minimum so it is safe to assume that $\delta_{rough} << \delta$. Using this approximation we obtain that $\delta_{rough} \geq \frac{1}{n}$. This makes sense as we need the new binary search to be smaller than $0 \implies n$. A value of δ_{rough} dependent on n would make sense. I have tried out a couple of such approaches. In addition, I will impose the constraint we derived on δ_{rough} in the code.

- **Using the fact that measurements are an integer:**
  We notice that in the old code for $\delta = 0.2$ and $\delta = 0.17$, the number of measurements stay the same as 8. This means that we are wasting the 0.03 of the probabilistic guarantee given to us. We can use this "fractional part" probability for our purposes in roughMC. I am unsure on how to implement this in code efficiently as of now, but I plan to work on it by the end of the week.

- **A proportionality with δ:**
  Something as simple as $\delta_{rough} = \frac{\delta}{20}$.

  I will try out all three approaches by the end of the week. Hopefully they will work better that the $0.02 + 0.18$ approach currently implemented.

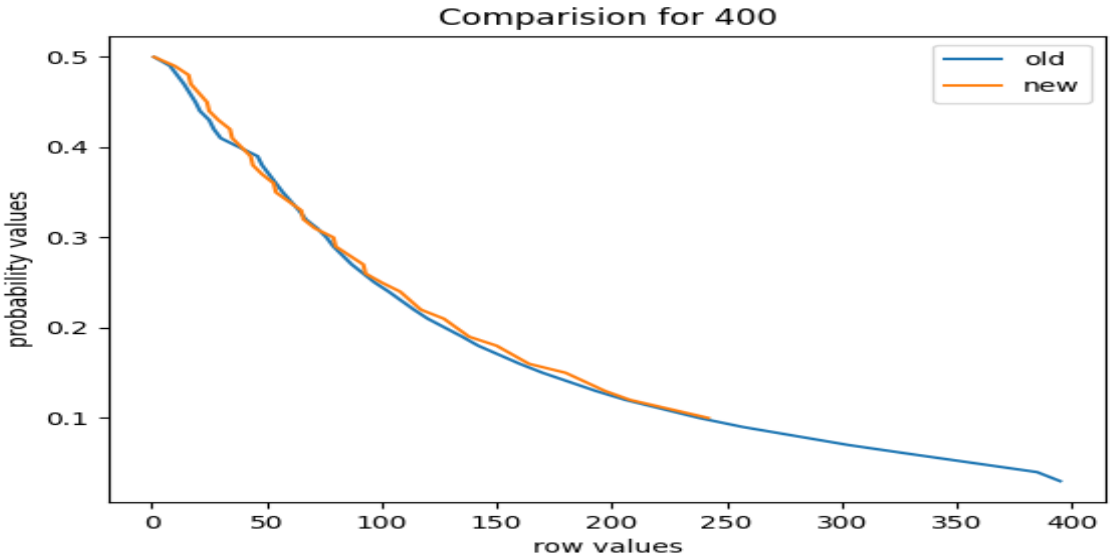
1.2 Comparison of Equal Probability Hashing Values with Old Ones

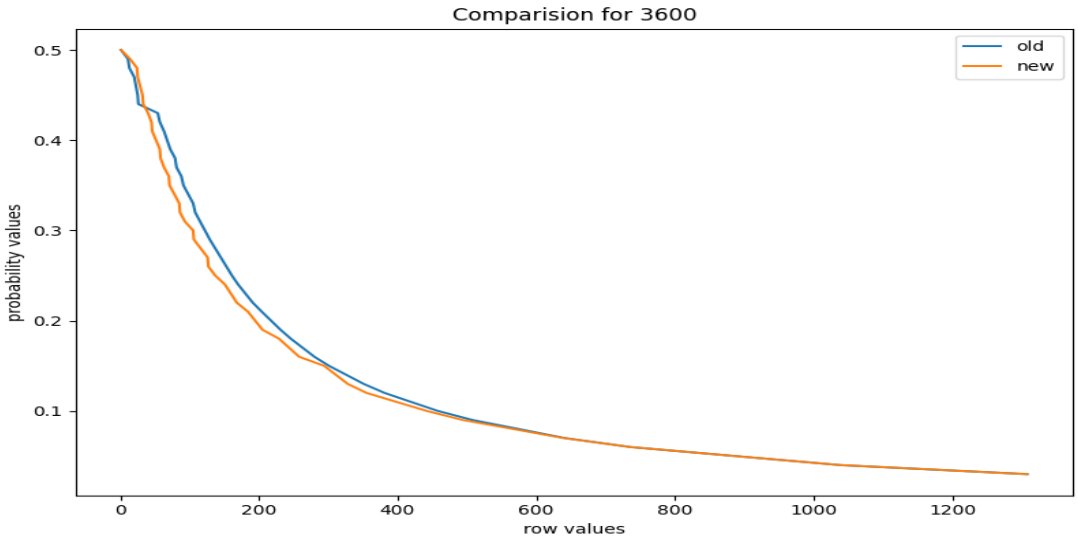
As noticed before, we need not use the hashing matrices from the H_{Rennes} hashing family. Instead we can just use equally sparse hashing matrices and the theoretical guarantees will still hold. we set the current equal hashing probability to be the probability from the last row of the hashing matrix. This works out because we make the assumption $p_i \geq p_m$ and replace p_i by p_m everywhere so we can simply just substitute everything as p_i. This reduction happens at this step:-

$$q(w,m) = \prod_{j=1}^{m}\left(\frac{1}{2} + \frac{1}{2}(1-2p_j)^w\right) \leq \left(\frac{1}{2} + \frac{1}{2}(1-2p_m)^w\right)^m$$

I have not yet implemented and tested this. We expect much better results from this than the current implementation of approxMC5.

1.3 Comparison of new Hashing Probability Values with the Old Ones





2 4 July - 8 July

The updates on the three tasks from last week are:

2.1 RoughMC

I first implemented the roughMC as a linear search. After getting correct values on test cases, I implemented a binary search. The bug-free correct code is pushed on roughmc branch of approxmc_private_improvements. After running the code on a few test cases used in the LICS paper, I noticed that roughMC seems to output a value slightly higher than m_{true}. I will have to look into why this is happening. Otherwise, it seems to run perfectly fine and i will merge this with the main branch once I am done with the task below.

One thing to keep in mind is that I am not returning the power of two. I am returning the log(SolCount).

2.2 Reduced binary search values

Implemented the reduced binary search and pushed working code on GitHub on the main branch. The roughmc value is stored in the conf file. Seems to work well on a few test cases.

2.3 Hashing Probability values higher than 3600

Successfully ran the code and have sent both approach 2 and approach 3 values and code to Yash. I plan to push them to the Hash_Rennes repository after running a few tests.

3 27 June - 1 July

This week I mainly focused on these three tasks:

3.1 RoughMC

This program returns the minimum number of XORs hashes m_{rough} that make our CNF formula UNSAT from SAT. Keep in mind that this is a randomised algorithm so it might return a different answer every time. I have uploaded an implementation with a few bugs in the roughmc branch of the approxmc_private_improvements fork. I will try and debug it and run tests by next week.

3.2 Reduced binary search values

Given the value of m_{rough} from the previous part (hardcoded in the configuration file for now as the RoughMC part is under construction), we output m_1 and m_2 such that the value of m found earlier by logSATsearch, m_{true}, is such that $m_{rough} - m_1 \leq m_{true} \leq m_{rough} + m_2$. The values of m_1 and m_2 were found theoretically in an earlier report, depending on only δ. I have changed the binary search for m in logSATsearch from $0 \implies n$ to $m_{rough} - m_1 \implies m_{rough} + m_2$. This code (again with a few minor bugs due to the dependencies) has been implemented in the main branch of the approxmc_private_improvements fork.

3.3 Hashing Probability values higher than 3600

The approach 2 I worked on last week led to probability values slightly higher than the ones that are currently implemented (probably due to an optimisation that I am missing out on). However, the code runs much faster than the old opt.py code. So we can calculate probability values for rows higher than 3600. I suggest using the old values for rows 0-3600 and using the approach 2 values for rows 3600-6000(roughly) and approach 1 values after that as it is extremely fast. I have implemented the code which converts the probability values into the table format in constants.cpp in ApproxMC. I am running the code for 24 hours and will create a push request to the main public ApproxMC code.