Assignment 1

CS533 ML for EDA

By
Ashwin Iyengar (244101007)
Devansh Upadhyay (244101015)
Saurav Chaudhary (244101048)
Vaibhav Wankar (244101061)

23 March 2025

Github Repo: QoR-predictor

Table of Contents

1. Assignment Overview

The assignment is a part of the CS533 ML for EDA course. This assignment acts as the first step in the course's final project. The work to be done for the assignment is as follows,

a. Running hand-crafted recipes (e.g., “abc.rc” contains few of them) and a few random recipes (length >= 15) to optimise the design and tabulate the results.

b. Design a recipe to minimize the area, delay, and QoR (area-delay product). We need to justify why we are choosing a particular recipe in light of the insight gained while running different hand-crafted and random recipes.

We are given three designs (one for each small, mid, and large design). Following are the designs assigned to our group.

SNo.	**Design Name**	**And Nodes**	**# I/O**	**Level**	**Size**
1	simple_spi	930	164/132	11	Small
2	Tv80	11328	636/361	53	Mid
3	jpeg	114771	4962/4789	39	Big

Table 1.1 Designs assigned to our group

The next few sections will discuss about the various steps done to create random recipes. In the final section, we will conclude the findings.

1.1 Important links

The prominent source for our assignment was the official abc website.

Website link: https://people.eecs.berkeley.edu/~alanmi/abc/

GitHub Repo: https://github.com/berkeley-abc/abc

Our GitHub Repo: https://github.com/Ash-intosh/QoR-predictor

1.2 Files used

For mapping, we used nangate_45.lib file.

The bench files for the three designs were downloaded from the course teams group.

2. Running the predefined recipes

We started the process by examining some existing standard recipes. These recipes can be found in the official " abc " repo in the file abc.rc. Some of the standard recipes used are as mentioned in the following table,

resyn	compress2	share	resyn3
resyn2	choice	addinit	compress
resyn2a	choice2	rwsat	drwsat2

Table 2.1 Shows the standard recipes used taken from abc.rc file

A shell script named "predefined_recipe_runner.sh" file was created to run all the above standard recipes for a design, and the result was tabulated. This script can be found in our project repo. The following tables 1.2, 1.3 and 1.4, show the results for standard recipes. Full results can be found in our repo in a pdf file named "Predefined Recipe Results"

Command	Delay	Area	QoR
choice	181.22	540.14	97884.1708
choice2	180.87	541.73	97982.7051
resyn2rs	185.97	533.94	99296.8218
r2rs	185.97	533.94	99296.8218
resyn2	185.97	538.99	100235.9703
resyn	185.99	547.51	101831.3849

Table 2.2 Result for simple_spi design

Command	Delay	Area	QoR
choice2	748.10	5728.53	4285513.293
choice	752.45	5882.42	4426226.929
resyn2rs	839.04	6243.81	5238806.3424
r2rs	839.04	6243.81	5238806.3424
resyn2	830.58	6309.95	5240918.271
resyn	834.90	6352.12	5303384.988

Table 2.3 Result for tv80 design

Command	Delay	Area	QoR
resyn2rs	674.47	97621.44	65842732.6368
r2rs	674.47	97621.44	65842732.6368
choice2	656.59	102719.79	67444786.9161
choice	645.79	104475.93	67469510.8347
resyn2	666.32	102229.94	68117853.6208
resyn	690.97	98824.05	68284453.8285

Table 2.4 Result for jpeg design

Insight 1

For all our designs, the first observation we made was choice, choice2, resyn2rs, and r2rs were the best-performing recipes. This helped us finalise a few operations that we will consider for the creation of random recipes.

The next step we to create a few random recipes to start with. Based on insight 1, we finalized a few operations which are listed below.

- b -> balance
- rw -> rewrite
- rwz -> rewrite -z
- rf -> refactor
- rfz -> refactor -z
- fsto -> fraig_store
- fres -> fraig_restore
- rs -> resub with cut size between 6 to 16 and -N as 2

3. Recipe Synthesis

3.1 Random Recipe Synthesis

We created another shell script to help us create random recipes. This file is named "fully_random_recipe_generator.cpp" and can be found in our repo.

———————————————————————————

Please Note:

1. The recipe length was kept at 24. The requirement was to keep it greater than 15.
2. Strash "st" is run initially before running the recipe. This is consistently followed for the standard, random and hand-crafted recipes.

———————————————————————————

This script creates fully random recipes using the above-mentioned operations. The results were not good as compared to the standard recipes. But some of them were performing better than others, and here we noticed another trend.

Insight 2 When balance and fraig store were done every few operations, the results were improving. This pair was also used in choice and choice2, which was the best-performing standard recipe.

So, while creating our random recipe this time, we added this conditioning after every five operations. We consistently concluded all the recipes with fraig_restore.

As per the official abc site, fraig_store and fraig_restore are part of the FRAIG package and are defined as follows

fraig_store – Stores the current network as one "synthesis snapshot" in the internal AIG database to be restored and used for technology mapping later [1].

fraig_restore – Converts the currently stored AIG snapshots into a FRAIG and sets it to be the current network, to which technology mapping can now be applied. The AIG database is reset by calling this command [1].

In the classical synthesis, the network is subjected to a sequence of optimization steps. Each step leads to a new network while the previous networks are no longer considered; in other words, they are lost for optimization. Meanwhile, it is possible that an intermediate network was better, in whole or in part, compared to the final one. The idea of lossless synthesis is to accumulate all the intermediate logic representations and postpone the final selection until later in the design process. This is termed as "Lossless" Logic Synthesis[2].

Insight 3

We also observed that balance, along with the rewrite, worked better in conjunction. The reason may be that the balance operation produced a balanced AIG, which has a shorter minimum delay. Subsequently, the rewrite or rewrite Z (with zero cost) modifies a portion of the AIG with a smaller or more efficient AIG, thereby reducing the area.

These two insights helped us design a better random recipe generator. The random recipe generator has the following features.

- Always starts with b (balance)
- Contains a pair of fsto, b and rw after a few other operations in between. And ended the recipe with fres.
- In between we used a series of operations such as rfz, rwz and rs.
- We also saw that rs with cut worked better than just rs. So, we are using rs with various cut sizes. We also have made a provision to have these operations in increasing order of cut size.

This way, we semi-random hand-crafted a few recipes of our own. A CPP file named "hand-crafted_recipe_generator.cpp" was created.

3.2 Semi-Random Handcrafted Recipe Synthesis

Some of the recipes generated using our new strategy worked better than the standard recipe. The results are tabulated below show a few of the semi-random handcrafted recipes. The complete results for the simple_spi design, along with the results for the other two designs, can be found on the GitHub repo. It has 100 such recipes (for each design), with some performing better than the best of the standard recipe and most of them performing better than most of the standard recipes.

Command	Delay	Area	QoR
b; rf; rf; rfz; rfz; rs -K 6; fsto; b; rw; rwz; rfz; rfz; rs -K 6 -N 2; rs -K 8; fsto; b; rw; rwz; rfz; rf; rwz; rs -K 8 -N 2;	180.25	529.81	95498.2525
b; rwz; rf; rs -K 6; rf; rwz; fsto; b; rw; rfz; rfz; rfz; rwz; rs -K 6 -N 2; fsto; b; rw; rfz; rwz; rfz; rfz; rfz; fsto; fres;	174.58	547.63	95605.2454
b; rfz; rf; rs -K 6; rs -K 6 -N 2; rs -K 8; fsto; b; rw; rs -K 8 -N 2; rf; rs -K 12; rwz; rfz; fsto; b; rw; rwz; rwz; rwz; rf; rs -K 12 -N 2; fsto; fres;	180.25	533.03	96078.6575
b; rwz; rf; rwz; rs -K 6; rfz; fsto; b; rw; rs -K 6 -N 2; rfz; rs -K 8; rf; rwz; fsto; b; rw; rf; rwz; rfz; rs -K 8 -N 2;	175.28	548.46	96134.0688
b; rfz; rfz; rwz; rfz; rwz; fsto; b; rw; rf; rfz; rs -K 6; rfz; rf; fsto; b; rw; rf; rf; rfz; rs -K 6 -N 2; rfz; fsto; fres;	182.02	529.27	96337.7254
b; rwz; rf; rfz; rwz; rf; fsto; b; rw; rf; rfz; rs -K 6; rf; rfz; fsto; b; rw; rwz; rfz; rs -K 6 -N 2; rwz; rs -K 8; fsto;	176.65	545.77	96410.2705
b; rfz; rfz; rfz; rf; rwz; fsto; b; rw; rf; rf; rfz; rfz; rs -K 6; fsto; b; rw; rfz; rf; rf; rf; rfz; fsto; fres;	182.02	529.77	96428.7354

Table 3.1 Results for few semi-random handcrafted recipes for simple_spi design

The QoR of the above recipes are better than “choice”. This meant that the strategy to a) include fsto, b and rw in between every few operations and ending it with fres and b) use of rs, rwz and rfz as intermediate operations improved the QoR score.

To further improve our recipe, we decided to make use of the “simulated annealing technique”.

3.3 Simulated Annealing

The basic idea of simulated annealing algorithms comes from annealing in metallurgy. Heating and re-cooling the material during the metallurgical process allows the atom to remain in a position where the energy has a local minimum.

The simulated annealing method starts with an initial solution and tries to find a better solution. We make use of an objective function; this can be seen as a feature that we want to optimise. In our code for the assignment, we decided to optimise QoR, which is the product of Area and Delay. This enables us to keep both delay and area optimised rather than optimising just area or delay. There's a tradeoff between Area and Delay, this is discussed towards the end of this section.

The algorithm used for simulated annealing is as follows,

Algorithm 1: Delay-Area Optimization Using Simulated Annealing

```
Input: Initial recipe $R$
Output: Optimized recipe $R_{opt}$ with improved delay-area product.
Initialise the solution space with a given recipe $R$.
Calculate the delay-area product using ABC and store it in $QoR$.
Store the current recipe $R_{prev} = R$
**for** iteration $i = 1$ to 300 **do**
    Randomly select two indexes $idx_1$ and $idx_2$ in $R$.
    Apply a randomly chosen operation to these indexes to generate a new
    recipe $R'$.
    Evaluate $R'$ using ABC to compute its delay-area product $QoR'$.
    **if** $QoR' < QoR$ **then**
        Update $R = R'$.
        Update $QoR = QoR'$
    **end if**
**end for**
Return the optimised recipe $R_{opt} = R$.
```

We decided to run the algorithm for 300 iterations and we stored all the better recipes instead of storing the most optimal one. The initial recipes used for simulated annealing were the best ones from the semi-random handcrafted recipes discussed in section 3.2

We have tabulated a few recipes below, the complete table has been stored in our repo.

Command	Delay	Area	QoR
rw; rs -K 12; rs -K 12; rfz; fsto; rwz; fsto; rs -K 8 -N 2; rs -K 12 -N 2; rs -K 12 -N 2; rs -K 14; rwz; rs -K 16; rfz; rs -K 12 -N 2; b; fsto; rf; rwz; rf; rs -K 16; rs -K 8; rs -K 8; fres;	173.52	537.58	93280.9
rw; rs -K 12; rs -K 12; rfz; fsto; rwz; fsto; rs -K 8 -N 2; rw; rfz; rs -K 14; rwz; rs -K 16; rfz; rs -K 12 -N 2; b; fsto; rf; rwz; rf; fsto; rs -K 8; rs -K 8; fres;	173.52	537.59	93282.6
b; fsto; rfz; rwz; rfz; rs -K 14 -N 2; fsto; rs -K 12; rf; rfz; fsto; rs -K 6 -N 2; rs -K 10 -N 2; rs -K 10; b; b; rs -K 8 -N 2; rwz; rfz; rwz; rs -K 12; rf; fsto; fres	175.01	533.31	93334.6
rw; rs -K 12; rs -K 12; rfz; fsto; rwz; fsto; rs -K 8 -N 2; rw; rfz; rfz; rfz; rs -K 16; rs -K 16 -N 2; rs -K 12 -N 2; b; fsto; rf; rwz; rf; fsto; rs -K 8; rs -K 8; fres;	172.24	542.07	93366.1
b; fsto; rfz; rwz; rfz; rs -K 14 -N 2; fsto; rs -K 12; rs; rfz; fsto; rs -K 6 -N 2; rs -K 10 -N 2; rs -K 10; fsto; b; rs -K 8 -N 2; rwz; rfz; rwz; rs -K 12; rf; fsto; fres	175.01	534.1	93472.8
rs -K 6; rwz; rs -K 8; rf; rs -K 10; rwz; rs -K 14 -N 2; b; rfz; rfz; rs; rfz; rwz; rs; fsto; b; rw; rfz; rf; fsto; rwz; rfz; fsto; fres;	172.06	543.47	93509.4
rs -K 6; rwz; rs -K 8; rf; rs -K 10; rwz; rs -K 14 -N 2; b; rfz; rfz; rs; rfz; rwz; rs -K 8; fsto; b; rw; rfz; rwz; fsto; rwz; rfz; fsto; fres;	172.41	542.92	93604.8
b; fsto; rfz; rwz; rfz; rs -K 14 -N 2; fsto; rs -K 12; rs; rfz; rs -K 6; rs -K 6 -N 2; rs -K 10 -N 2; rs -K 10; fsto; b; rs -K 12 -N 2; rwz; rfz; rwz; rs -K 12; rf; fsto; fres	175.01	535.15	93656.6

Table 3.2 Shows the results of Simulated annealing for simple_spi

Similarly for tv80,

Command	Command	Area	QoR
fsto; rs -K 16 -N 2; rf; rs -K 6 -N 2; b; rs -K 10 -N 2; fsto; b; rw; rs -K 10; rfz; rwz; rfz; rs -K 12 -N 2; fsto; b; rw; rwz; rs -K 8 -N 2; rfz; rwz; b; fsto; fres;	711.04	5603.52	3984330
rfz; rs -K 8 -N 2; rwz; rs -K 6; rfz; rs -K 8; fsto; b; rw; fsto; rwz; rfz; fsto; rs -K 12 -N 2; rs -K 8; b; rs -K 14 -N 2; rs -K 10 -N 2; rfz; rs -K 8 -N 2; rwz; rs -K 14 -N 2; fsto; fres;	708.44	5648.00	4001270
rfz; rs -K 8 -N 2; rwz; rs -K 6; rfz; rs -K 8; fsto; b; rw; fsto; rwz; rfz; fsto; rs -K 12 -N 2; rs -K 8; b; rs -K 14 -N 2; rs -K 10 -N 2; rfz; rs -K 8 -N 2; rwz; b; fsto; fres;	711.01	5632.23	4004570
rs -K 10 -N 2; rwz; fsto; rfz; rs -K 8; rfz; fsto; b; rw; rfz; rf; rwz; rs -K 12; rfz; rs -K 10 -N 2; rwz; rs -K 8; rwz; b; rs -K 12; fsto; rwz; fsto; fres;	731.91	5475.85	4007830
rs -K 10 -N 2; rwz; fsto; rfz; rs -K 6 -N 2; rfz; fsto; b; rw; rfz; rf; rwz; rs -K 12; rfz; rs -K 16 -N 2; rwz; rs -K 8; rwz; b; rs -K 12; fsto; rwz; fsto; fres;	733.47	5471.38	4013090
rs -K 10 -N 2; rwz; fsto; rfz; rs -K 8; rfz; fsto; b; rw; rfz; rf; rwz; rs -K 12; rfz; rs -K 16 -N 2; rwz; rw; rwz; b; rs -K 12; fsto; rwz; fsto; fres;	733.15	5486.23	4022230
rfz; rs -K 8 -N 2; rwz; rs -K 6; rfz; rs -K 8; fsto; b; rw; fsto; rwz; rfz; fsto; rs -K 12 -N 2; rs -K 8; b; rs -K 14 -N 2; rf; rfz; rs -K 8 -N 2; rwz; rs -K 6 -N 2; fsto; fres;	708.44	5679.46	4023560

Table 3.3 Shows the result of Simulated annealing for tv80

Command	Delay	Area	QoR
b; rfz; rfz; rwz; rs -K 12 -N 2; rs -K 6; fsto; rs -K 14 -N 2; rw; rs -K 14; rw; rs -K 12 -N 2; rs -K 12; rfz; rs -K 8 -N 2; b; rw; rs -K 12 -N 2; rf; rs -K 16; rf; rs; fsto; fres;	652.35	97176.05	63392796.2175
b; b; rfz; rs -K 16; rwz; rfz; fsto; rs -K 10; rw; rs -K 6; rs -K 6 -N 2; rfz; rs -K 12 -N 2; rwz; fsto; rf; rs -K 8; rs -K 12 -N 2; rf; rf; rs -K 6 -N 2; rfz; fsto; fres;	662.55	96051.16	63638696.058
b; b; rfz; rs -K 16; rwz; rfz; fsto; rs -K 10; rw; rs -K 6; rs -K 6 -N 2; rfz; rs -K 12 -N 2; rs -K 8; fsto; rf; rs -K 8; rwz; rf; rf; rs -K 6 -N 2; rfz; fsto; fres;	662.55	96075.24	63654650.262
b; rfz; rfz; rs -K 16; rwz; rfz; fsto; b; rw; rs -K 6; rs -K 6 -N 2; rfz; rs -K 12 -N 2; rs -K 8; fsto; rf; rs -K 8; rwz; rf; rf; rs -K 6 -N 2; rfz; fsto; fres;	648.51	98321.85	63762702.9435
b; rfz; rwz; rs -K 12; rf; rs -K 6 -N 2; fsto; b; rw; rfz; rs -K 10 -N 2; rs -K 8 -N 2; rfz; rs -K 6 -N 2; fsto; b; rw; rfz; rwz; rs -K 12; rwz; rs -K 8; fsto; fres;	648.08	98425.88	63787844.3104
b; rfz; rfz; rwz; rf; rs -K 6; fsto; rs -K 14 -N 2; rw; rs -K 6 -N 2; rw; rs -K 12 -N 2; rs -K 12; rfz; rs -K 8 -N 2; b; rw; rs -K 12 -N 2; rf; rs -K 16; rf; rs; fsto; fres;	652.35	97957.16	63902353.326

Table 3.4 Shows the result of Simulated annealing for jpeg

All the results performed better than the predefined recipes. We also observed that Lossless Logic Synthesis remained true even after performing simulated annealing.

3.3.1 Area Delay Tradeoff

We also performed Simulated annealing to improve just the Delay and, similarly, just the Area. The results were as we expected. The graph presents the trade-off between Area and Delay for three different optimisation strategies:

- Best QoR Best (Quality of Results)
- Best Area
- Best Delay

We gathered some data on this and summarised it in the following scatter plot.

The graph clearly shows the inverse relationship between Area and Delay.

- The Best QoR approach provides a balanced compromise, making it ideal for general applications.

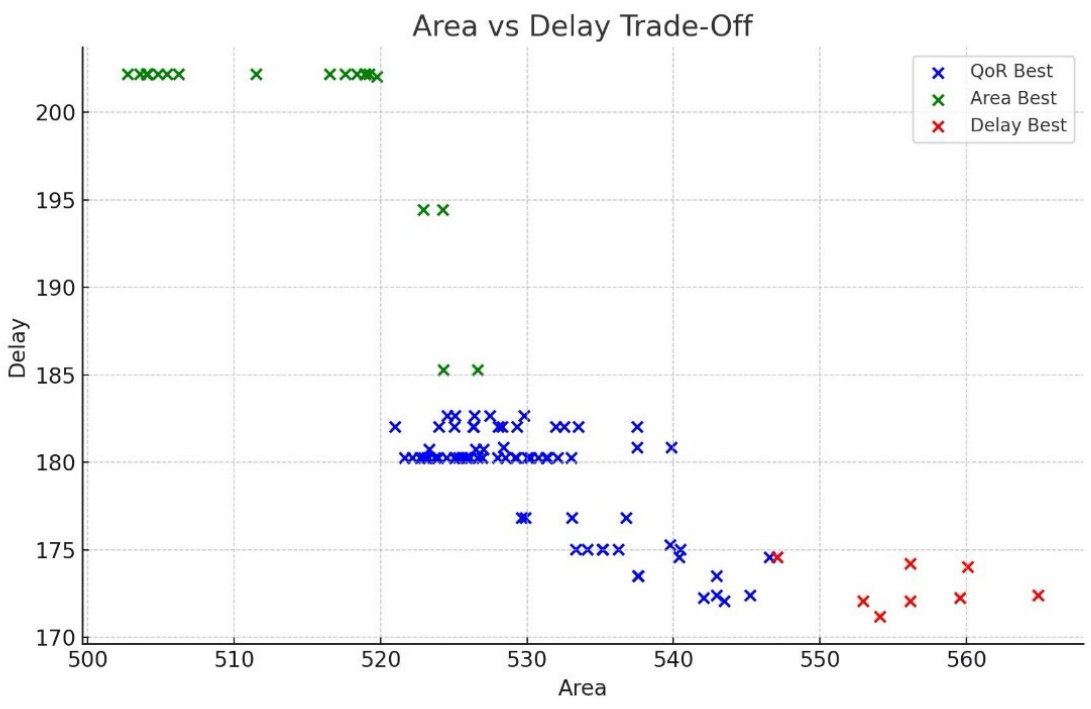

Figure 3.1 shows the Area Delay tradeoff for a few recipes in simple_spi design

- For performance-critical systems, the Best Delay approach would be preferable despite the area increase.
- Conversely, the Best Area approach suits applications requiring minimal area, such as embedded systems with limited space.

The data points used for this scatter plot are available in our repo.

3.4 Section conclusion

We started by creating random recipes using the operations we finalised in section 2. The random recipes performed badly compared to the standard recipes. But this helped us draw a few insights, which helped us create a better algorithm to create better random recipes. We call them semi-random handcrafted recipes. Key insights were to use b, fraig store and rw after every few operations and end it with fraig restore.

To further improve the recipes we decided to make use of a simulated annealing technique. We have added a few data points in the report whereas the complete results will be available in our GitHub repo.

4. Final words

The following line graph shows the difference between the standard recipes, semi-random handcrafted recipes, and recipes synthesised through simulated annealing.

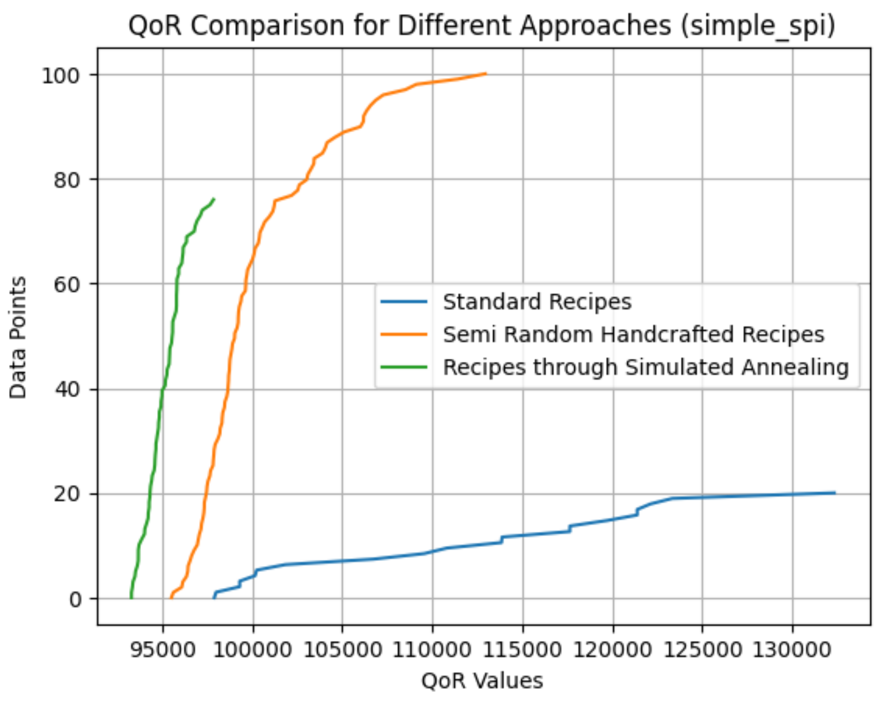

Figure 4.1 Comparison for simple_spi design

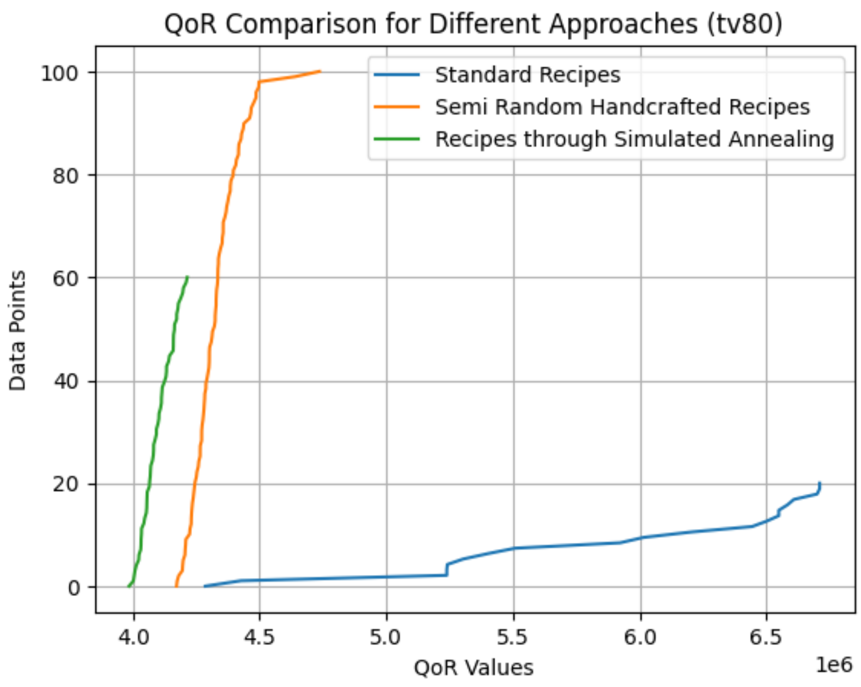

Figure 4.2 Comparison for tv80 design

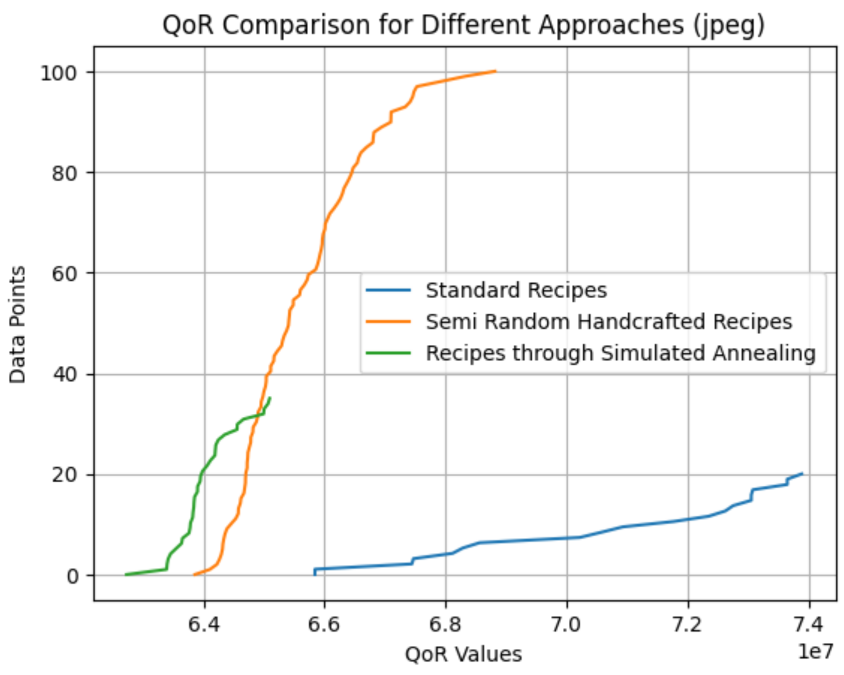

Figure 4.3 Comparison for jpeg design

The recipes generated through simulated annealing are performing better than the standard recipes.

It was observed that the annealing primarily altered operations that did not contribute to improving the Quality of Results (QoR). Notably, even after numerous iterations, balance and fraiging operations remained consistently positioned at regular intervals, as intended by our semi-random recipe generation algorithm.

In all three models, we have observed that the top recipes of annealed one contain a lot of rs (resubstitutes with cuts). So, we have tried a few variations with it such as we removed the cuts in one recipe while we removed the whole rs operation from recipes.

Among all the three recipes we observed that the first one (rs with cuts) performed the best and recipes without rs were close to it, there was not much difference. But in recipes with rs (without cuts), the area and delay were significantly increased.

According to us, the recipes without rs were, obviously, smaller than the other two. So, we have observed that a smaller recipe and a comparatively large recipe yield almost the same area and delay. Thus, we concluded that in large recipes, redundant operations may increase the area and delay, while resubstitute with cuts does increase the area and delay on redundant operations, it may even decrease them a bit. Also, re-subsitute (without cuts) acts the same as other operations and increases area and delays redundancy.

The top recipes from each design are performing at par with the standard recipes for the other two designs as well.

Recipe
rw; rs -K 12; rs -K 12; rfz; fsto; rwz; fsto; rs -K 8 -N 2; rs -K 12 -N 2; rs -K 12 -N 2; rs -K 14; rwz; rs -K 16; rfz; rs -K 12 -N 2; b; fsto; rf; rwz; rf; rs -K 16; rs -K 8; rs -K 8; fres;
rw; rs -K 12; rs -K 12; rfz; fsto; rwz; fsto; rs -K 8 -N 2; rw; rfz; rs -K 14; rwz; rs -K 16; rfz; rs -K 12 -N 2; b; fsto; rf; rwz; rf; fsto; rs -K 8; rs -K 8; fres;
fsto; rs -K 16 -N 2; rf; rs -K 6 -N 2; b; rs -K 10 -N 2; fsto; b; rw; rs -K 10; rfz; rwz; rfz; rs -K 12 -N 2; fsto; b; rw; rwz; rs -K 8 -N 2; rfz; rwz; b; fsto; fres;
rfz; rs -K 8 -N 2; rwz; rs -K 6; rfz; rs -K 8; fsto; b; rw; fsto; rwz; rfz; fsto; rs -K 12 -N 2; rs -K 8; b; rs -K 14 -N 2; rs -K 10 -N 2; rfz; rs -K 8 -N 2; rwz; rs -K 14 -N 2; fsto; fres;
b; rfz; rfz; rwz; rs -K 12 -N 2; rs -K 6; fsto; rs -K 14 -N 2; rw; rs -K 14; rw; rs -K 12 -N 2; rs -K 12; rfz; rs -K 8 -N 2; b; rw; rs -K 12 -N 2; rf; rs -K 16; rf; rs; fsto; fres;
b; b; rfz; rs -K 16; rwz; rfz; fsto; rs -K 10; rw; rs -K 6; rs -K 6 -N 2; rfz; rs -K 12 -N 2; rwz; fsto; rf; rs -K 8; rs -K 12 -N 2; rf; rf; rs -K 6 -N 2; rfz; fsto; fres;

Table 4.1 Shows the best recipes synthesised

5. References

[1] “ABC: A System for Sequential Synthesis and Verification.” [Online]. Available: https://people.eecs.berkeley.edu/~alanmi/abc/.

[2] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," ERL Technical Report, EECS Dept., UC Berkeley, 2005.