EDA Final Project - ICCAD Problem A

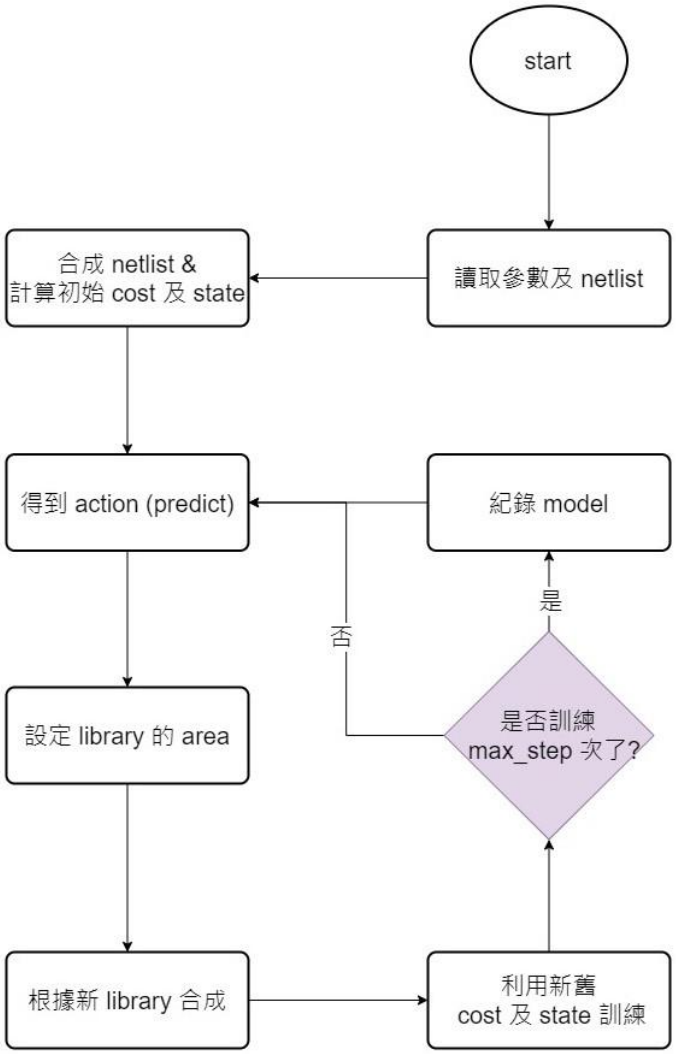
110511195 林奕奇 110511203 王姿惠 110511210 李博安

ICCAD 繳交截圖

```
[a_1014@iccad83 ~]$ cd cada1014_alpha/
[a_1014@iccad83 ~/cada1014_alpha]$ ls
cada1014_alpha.tgz
```

程式流程圖+分工方式

<程式流程圖>



<分工方式>

架設 pyinstaller 及 pytorch 環境、實驗分析：林奕奇

架設 Yosys 環境、包 Yosys 指令及轉檔、RL：王姿惠

I/O、RL、程式流程：李博安

流程詳細介紹

<I/O、轉檔>

將所需輸入和輸出用類似 argv 的方式讀取，執行指令為 "./cada0000_alpha -cost_function cost_function_1 -library low_vt.lib -netlist design.v -output design_optimized.v" ，透過 yosys 做 synthesis (blif 轉回 .v) 時要將 lib.json 轉成特定格式供讀取，yosys 轉出的 .v 檔格式也與 cost estimator 所要讀取的不同，所以也需要將輸出轉成特定格式，如下圖

(左為原格式，右為更改後格式，這是範例格式)

changing library format:

```
{
        "information" : {
                "cell_num" : "8" ,
                "attribute_num" : "6" ,
                "attributes" : [
                        "cell_name" ,
                        "cell_type" ,
                        "delay_f" ,
                        "power_f" ,
                        "attribute_1_i" ,
                        "attribute_2_f"

                ]
        } ,
        "cells" : [
                {
                        "cell_name" : "NAND2X1" ,
                        "cell_type" : "nand" ,
                        "delay_f" : "45.23" ,
                        "power_f" : "910.85" ,
                        "attribute_1_i" : "123" ,
                        "attribute_2_f" : "45.678"
                } ,
                ...
        ]
}
```

```
library(yosys_cells) {
        cell(and_1) {
                cell_type : "and";
                area : 2;
                pin(A) {
                        direction : input;
                }
                pin(B) {
                        direction : input;
                }
                pin(Y) {
                        direction: output;
                        function : "A * B";
                }
        }
        ...
}
```

changing output format:

```
module top_810038711_1598227639(n12, n18, n56);
    wire _000_;
    input n12;
    input n18;
    output n56;
    not_9 _065_ (
        .A(n12),
        .Y(_000_)
    );
    nand_1 _073_ (
        .A(_000_),
        .B(n18),
        .Y(n56)
    );
```

```
module top_810038711_1598227639(n12, n18, n56);
    input n12, n18;
    output n56;
    not_9 _065_ (n12, _000_);
    nand_1 _073_ (_000_, n18, n56);
```

<Yosys>

使用 python 呼叫 yosys 將 .v 檔轉成 blif，再將 blif 轉回 .v 檔，使用 yosys 最主要原因是因為可以透過其進行合成及化簡，須不停運用將 blif 轉回.v 檔的功能將輸出檔丟進 cost_estimator 來得到新的 cost，並搭配 RL 更改 library 裡面的 area 供 blif 轉 .v 時使用。

<RL>

在這次的比賽中我們是使用 RL 的方式來降低 cost。我們採用的是 DQN(Deep Q-Network)，透過對一個深度學習網路輸入我們所定義的 state 來算出應該要做什麼 action。

我們的 state 是由兩兩一組共十六個數字組成，每一組中的兩個數字分別為 use_more 以及 gate_num。gate_num 代表了這一輪的 circuit 所使用 gate 選擇了哪一號(官方所給定的 library 中會有 and_1, and_2 之類的，選到 1 的話就代表這次用的是 and_1)，use_more 則是代表各自的 area 要如何分配(因為我們所使用的 synthesis tool 是以 area 這個參數為調整的標準，越小越容易被用到)，若為 0 則選中的那號 gate area = 1，其他 area = 2 (用這種 gate 的比例不會那麼高)，若為 1 則選中的那號 gate area = 0.9，其他 area = 1 (用這種 gate 的比例較高)。

我們的 action 則是有三個輸出，第一個 output node 的內容會是 0~8 的一個數字，每個數字分別對應到一種 gate (對照表如下)

gate_numbers	0	1	2	3	4	5	6	7
	and	or	nand	nor	xor	xnor	not	buf

第二個 output node 的內容是 gate_num，也就是看這輪的 action 選取到了幾號 gate，方便接下來的程式去做更動。第三個 output node 的內容會是 use_more，我們會利用程式去調整選取到的 gate 的 area。舉例來說，如果今天前面兩個輸出分別是 2 和 5，第三個則是 0，那麼我們會把 nand_5 的 area 設為 1，其他 nand 的 area 改為 2，再寫回 library 檔案中，至於其他 gate 的數據則不做變動。

在架構上，我們將整個程式大致上拆分成 agent，environment 以及 model 三個部分。Model 中有兩個元素，分別為 linear 的 Q-net，也就是一個線性的神經網路，以及要如何訓練我們的 net (train_step)。我們在 train_step 中有使用到 Bellman’ s equation 來更新我們的 Q 值，而我們的 loss funtio n 是 Q_new 與預測值的 MSE。

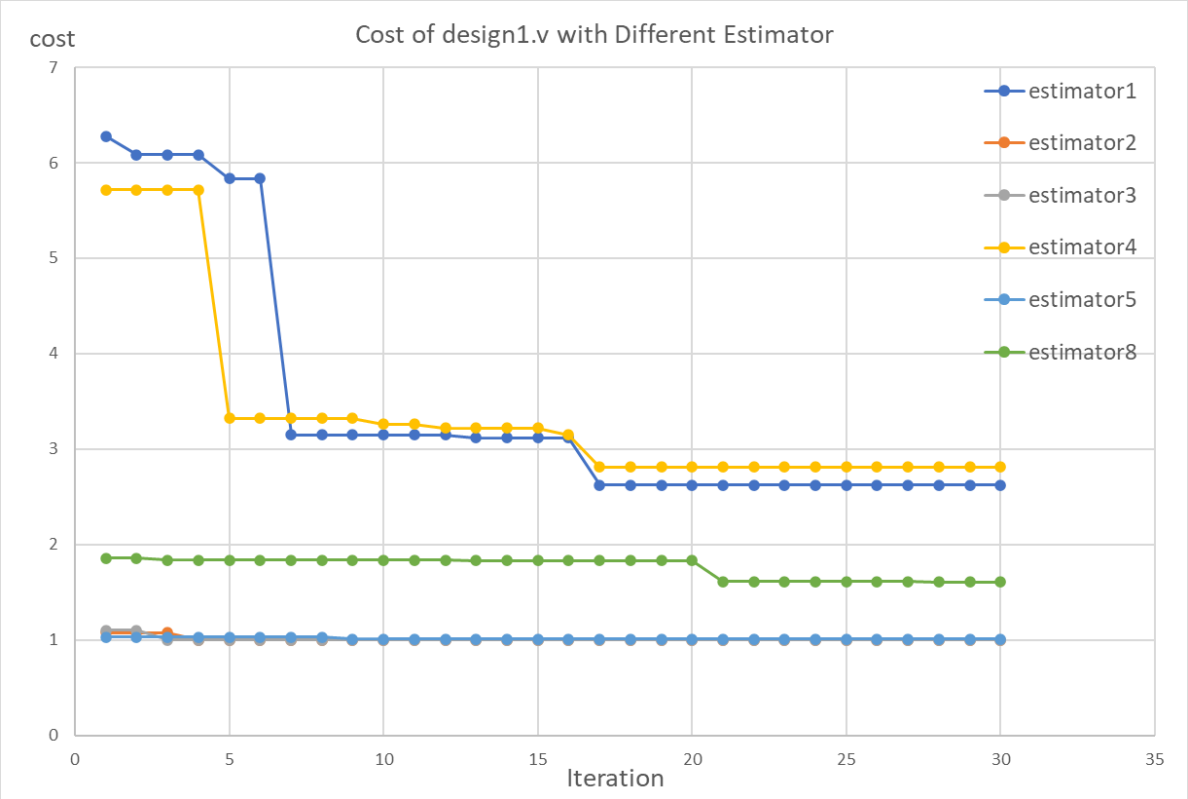
$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \overbrace{\alpha\left(R_{t+1} + \gamma \max_{a'} q(s',a')\right)}^{\text{learned value}}$$

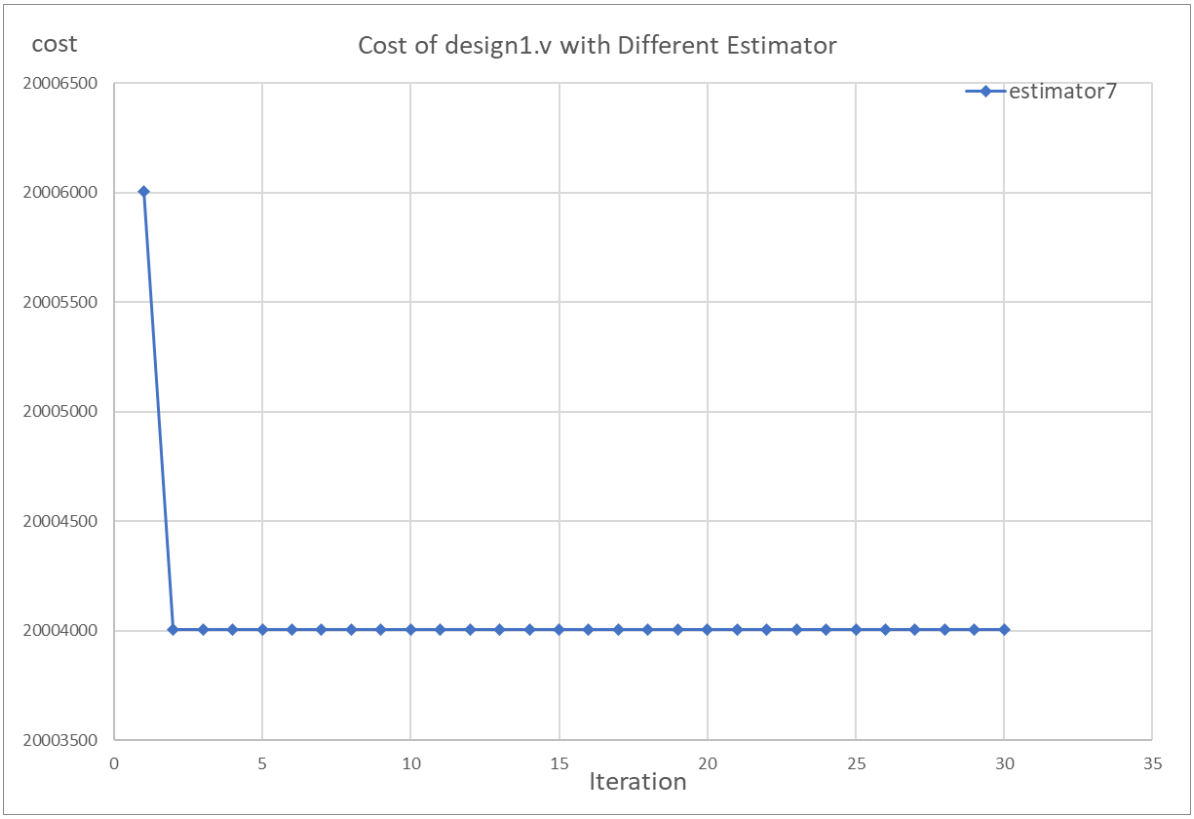
(Bellman’ s equation; s: state, a: action, q: q value, R: reward, max q(s’, a’): max of q of next state s’ and next action a’)

Environment 包含了我們用來將 netlist 合成成 circuit 的程式，也有改變 library 的程式，總而言之就是一切操作的中介，而同時，我們也可以從 environment 中得到 reward 以及 state。Agent 內部則是包含了 model，並從 environment 裡所得到的 state 來判斷要進行哪個 action (exploitation 及 exploration 由一個名為 epsilon 的值控制)，接著得到新的 state 後再改變 library，然後將這一輪的數據拿去做 training。這便是我們 RL 部分的介紹。

實驗結果

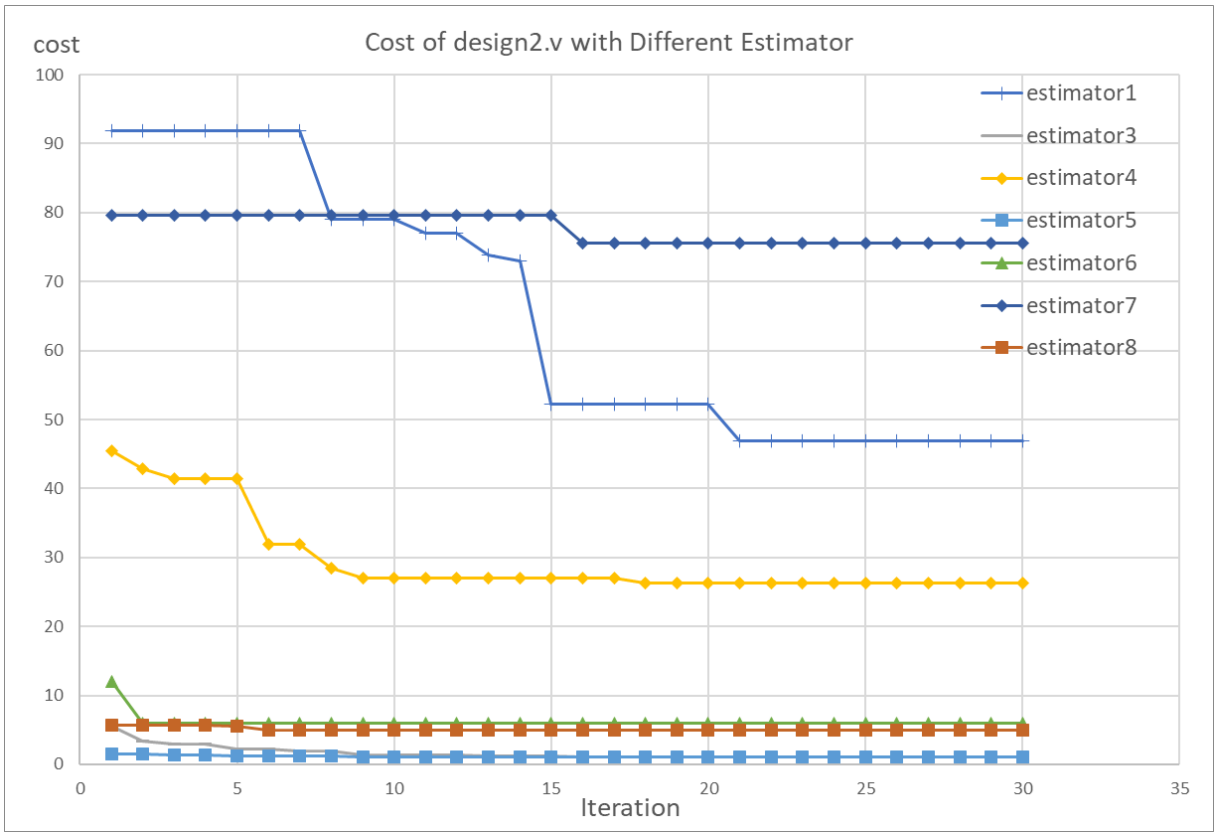
<design1.v> (estimator 2 及 estimator 3 所產生的值皆為 1.0)

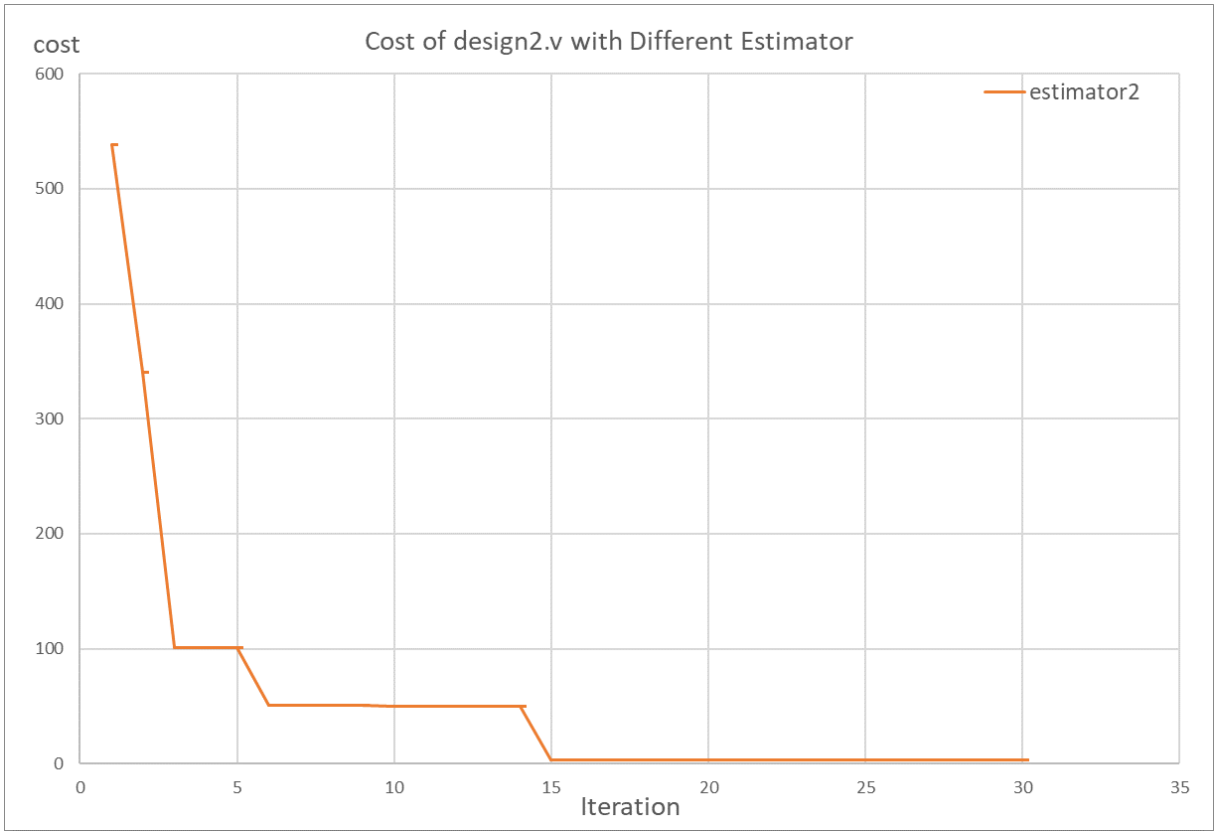




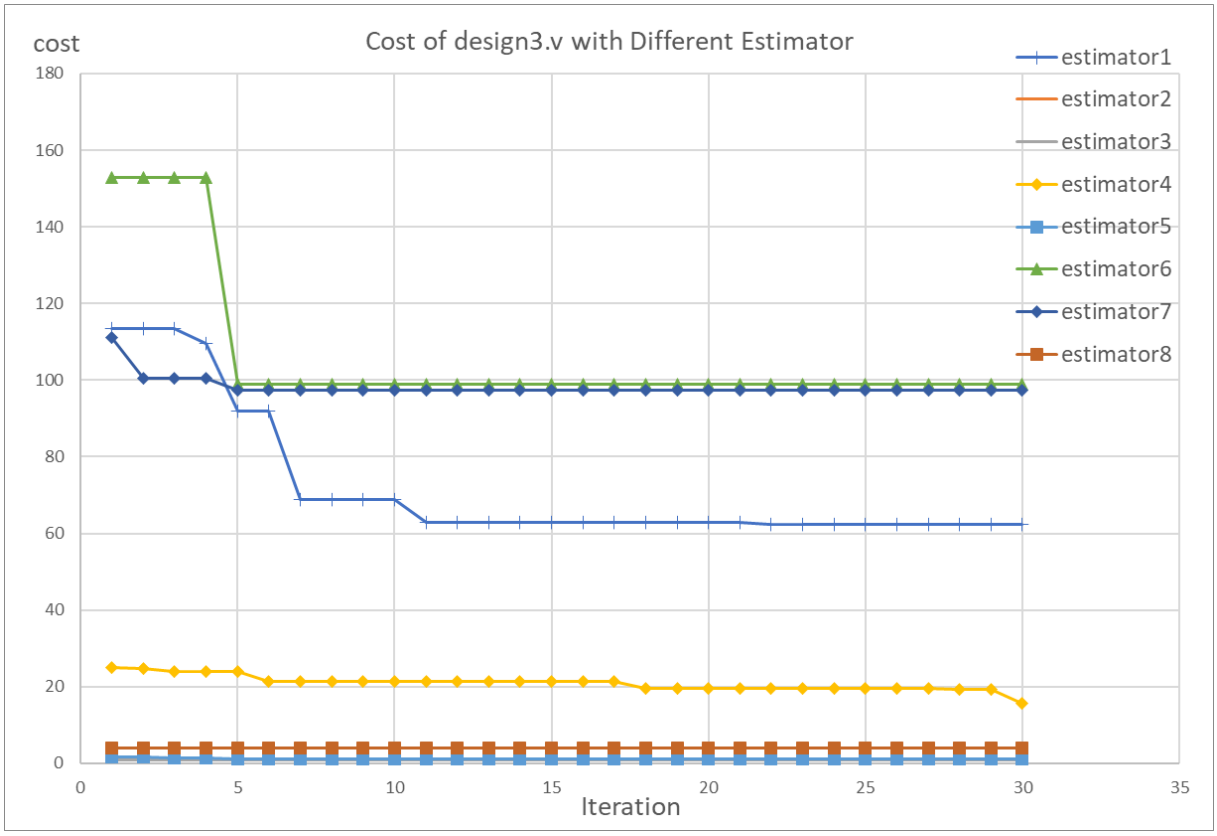
*design1.v 在 cost_estimator6 會出現 cost 為 0 的情形，並不合比賽方的敘述(non-zero positive real number)，因此沒有列入

<design2.v>

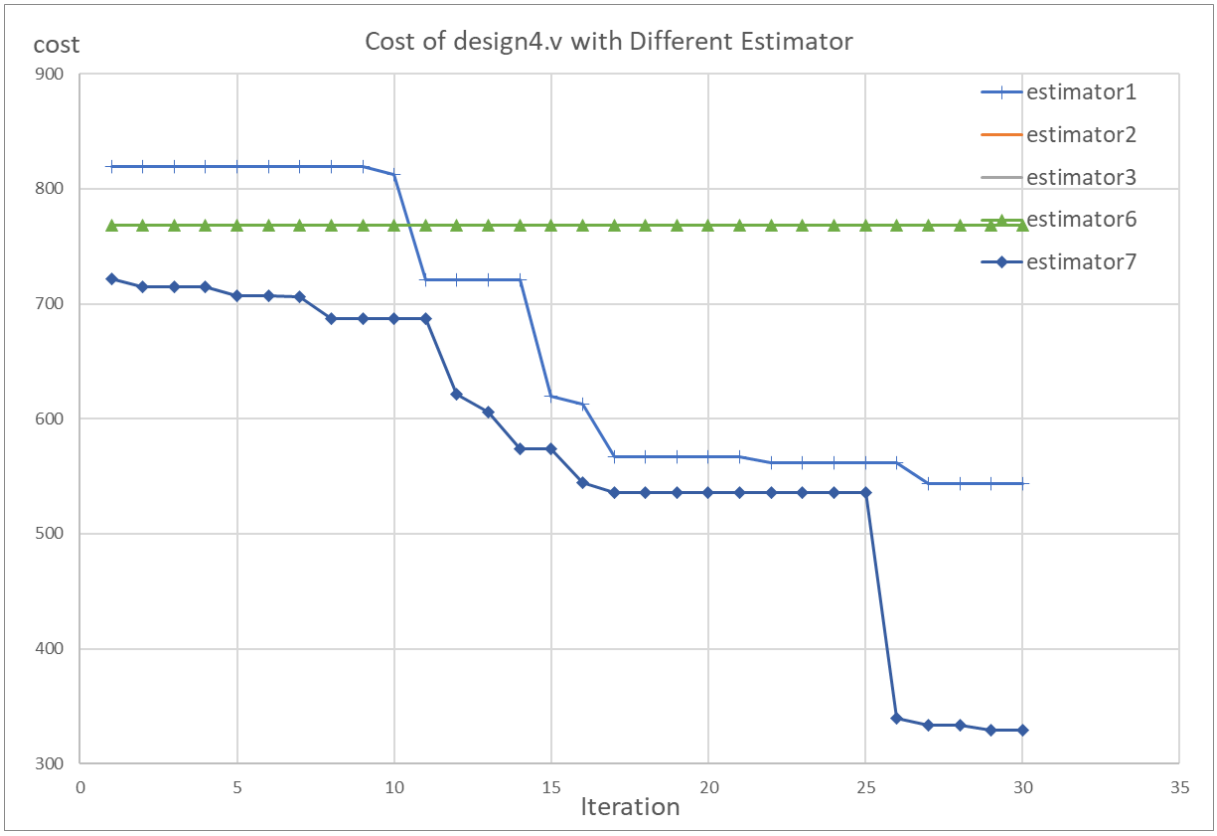


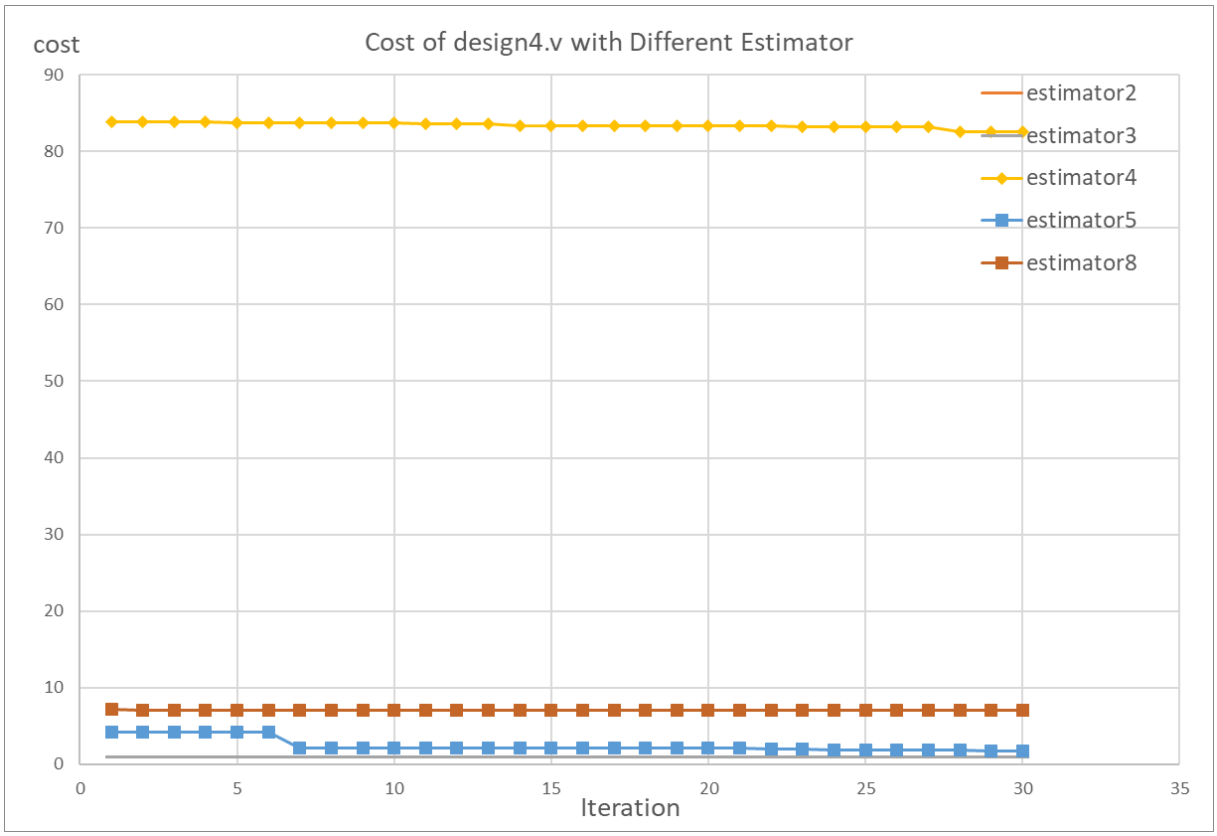


<**design3.v**> (estimator 2 及 estimator 3 所產生的值皆為 1.0)

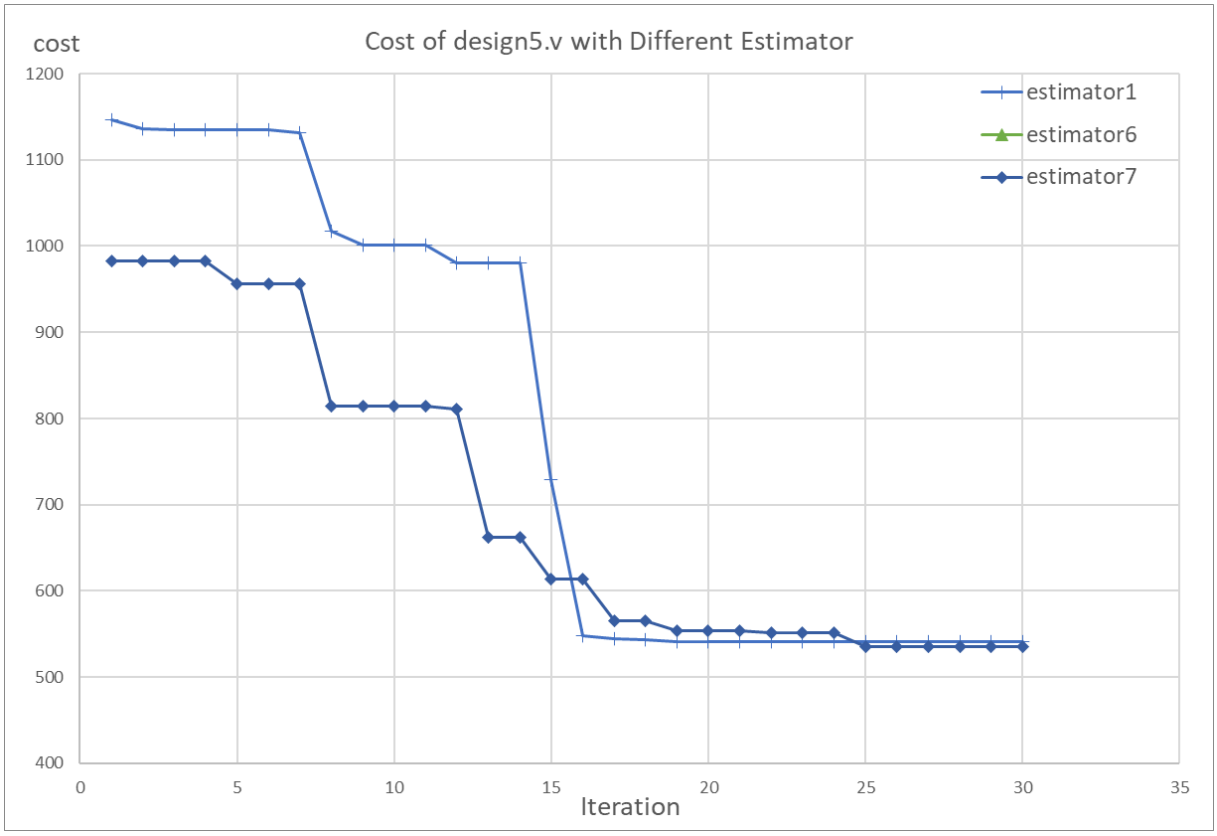


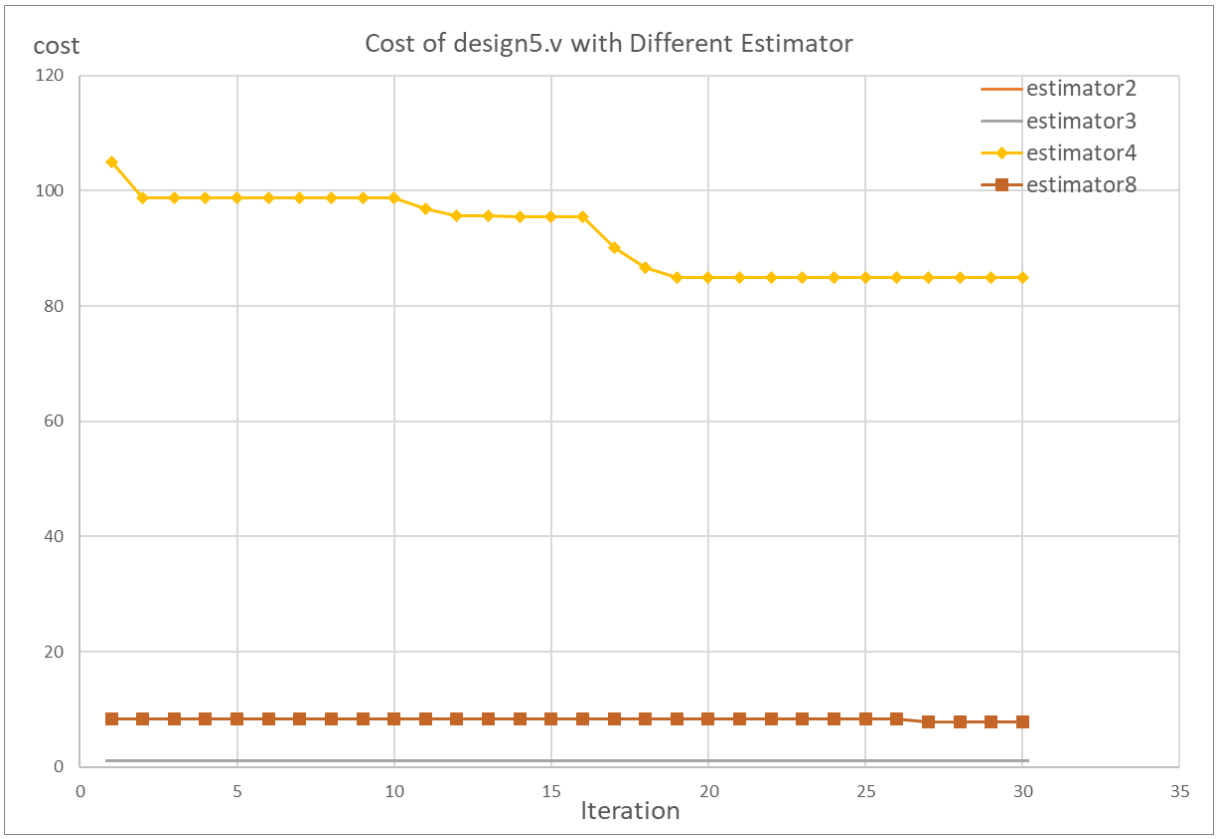
<design4.v> (estimator 2 及 estimator 3 所產生的值皆為 1.0)





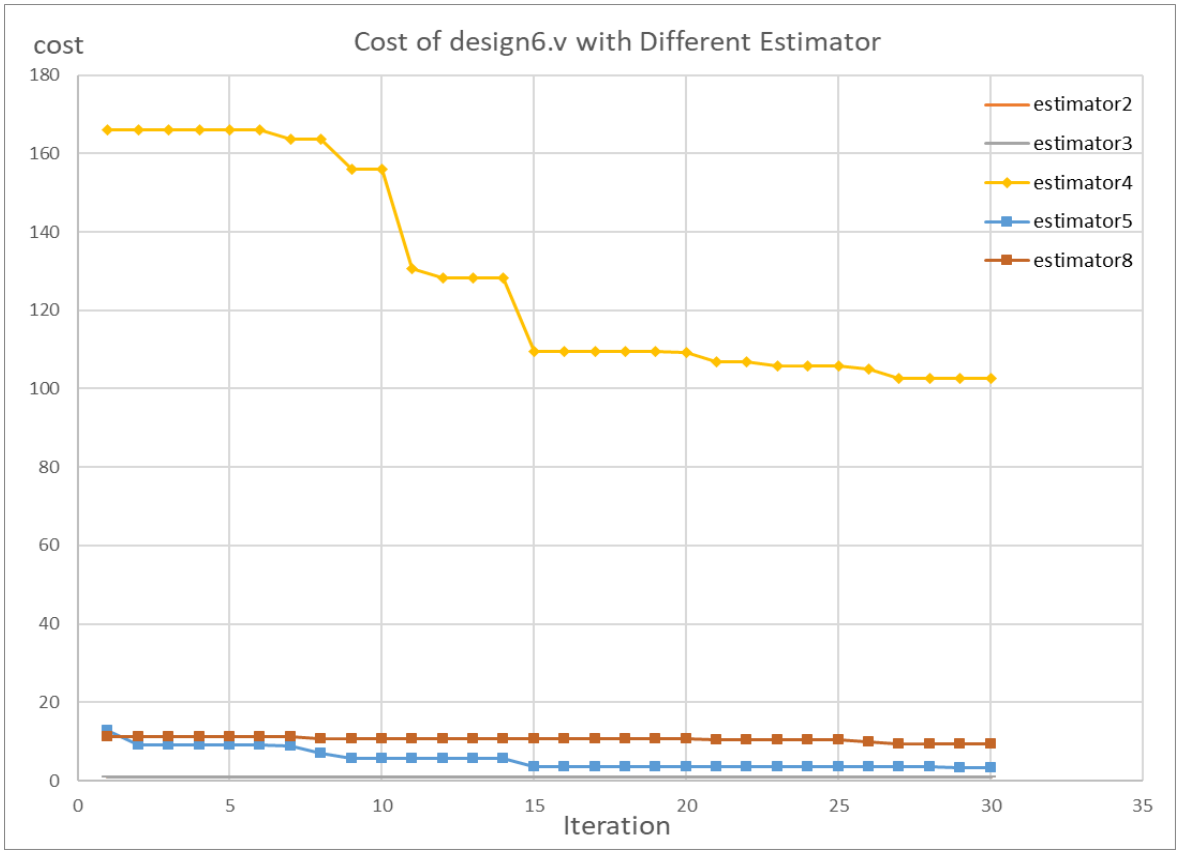
<design5.v> (estimator 2 及 estimator 3 所產生的值皆為 1.0)

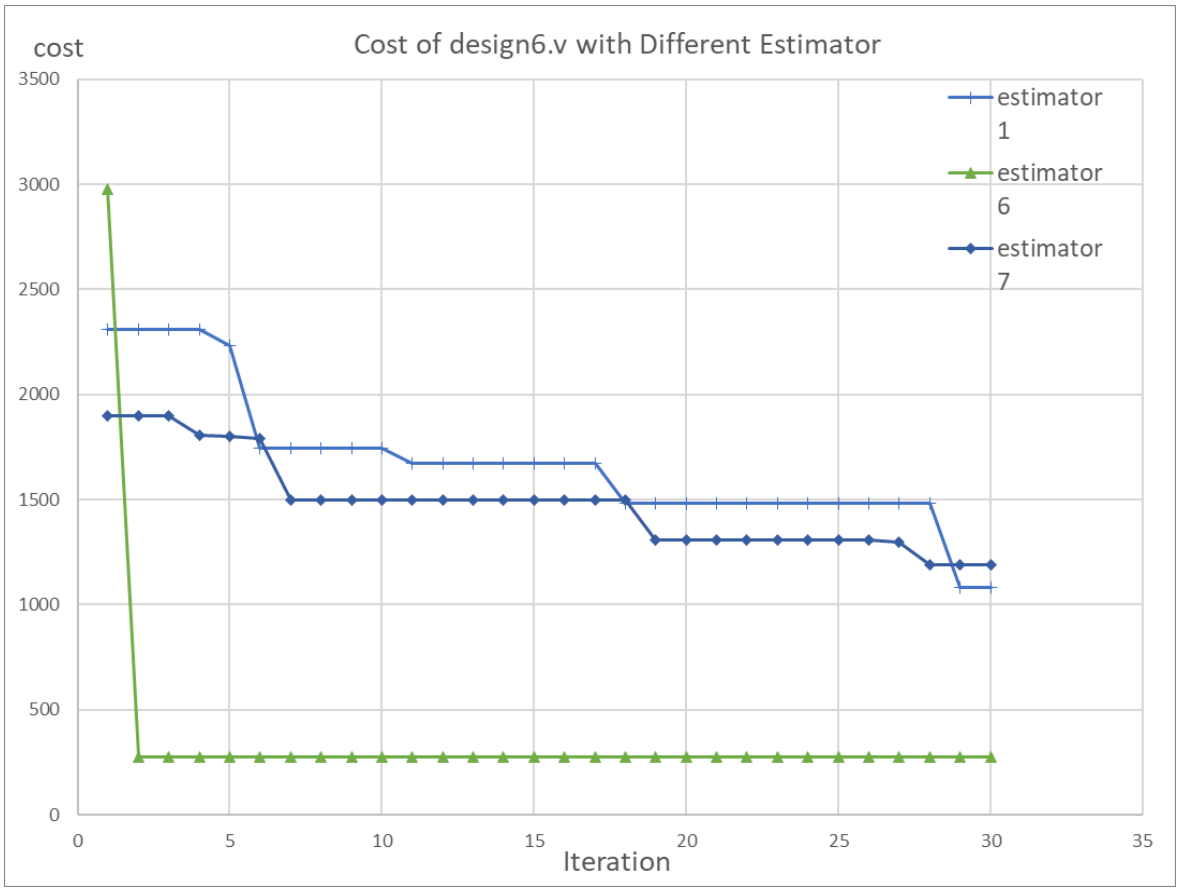




*design5.v 在 cost_estimator6 會出現 cost 為 0 的情形，並不合比賽方的敘述(non-zero positive real number)，因此沒有列入

<design6.v> (estimator 2 及 estimator 3 所產生的值皆為 1.0)





實驗結果分析

從實驗結果來看，能注意到的是，對於每個 design 以及 cost estimator，我們的 cost 都是遞減的，因為我們會保留每個 Iteration 中最好的結果；我們使用的架構會有 Exploration 和 Exploitation: Exploration 是在初期時，模型會利用隨機的方式盡量往外探索，尋找更多可能解，訓練度隨著 Iteration 增加的同時，會讓模型有越來越高的機會去進行 Exploitation，也就是根據經驗，做出會使 cost 降低的決策，因此跟 Simulated Annealing 的方法類似，不會容易在 local minimum 中卡住，但與之相比，我們的決策不全然是用隨機的方式產生。

以下是由實驗結果比較沒有使用 yosys 化簡以及 RL 方式進行優化的原始電路*之 cost 比較 (estimator6 會有不符合預期的 cost 出現，因此分析暫先忽略)，其中 improvement 之計算方式如下:

$$improvement = \frac{initial\ design's\ cost - \ improved\ design's\ cost}{initial\ design's\ cost}$$

estimator / design1	1	2	3	4	5	6	7	8
initial cost	7.45	28.38	1.67	6.67	1.04	0.00	20020006.00	2.59
after 30 iteration	2.63	1.00	1.00	2.81	1.01	0.00	20004005.00	1.61
difference	4.82	27.38	0.67	3.86	0.03	0.00	16001.00	0.98
improvement	64.74%	96.48%	40.16%	57.85%	2.61%	ERROR	0.08%	37.93%

estimator / design2	1	2	3	4	5	6	7	8
initial cost	153.46	44.84	1.46	48.52	1.76	45.00	20456129.00	5.79
after 30 iteration	46.92	3.42	1.03	26.32	1.11	6.00	75.58	4.97
difference	106.53	41.42	0.43	22.20	0.64	39.00	20456053.42	0.82
improvement	69.42%	92.37%	29.34%	45.75%	36.62%	86.67%	100.00%	14.18%

*此處的原始電路是指將原本的電路的所有 logic gate 直接套用 library 的一號 variant (也就是採用 and_1, or_1, xor_1 等等)

estimator / design3	1	2	3	4	5	6	7	8
initial cost	176.76	9.87	1.23	52.54	1.90	27.00	20590149.00	6.07
after 30 iteration	62.48	1.00	1.00	15.71	1.14	99.00	97.25	3.96
difference	114.29	8.87	0.23	36.83	0.76	-72.00	20590051.75	2.11
improvement	64.66%	89.87%	18.96%	70.10%	39.92%	-266.67%	100.00%	34.71%

estimator / design4	1	2	3	4	5	6	7	8
initial cost	1279.37	5058.62	1.43	196.35	5.59	0.00	23439129.00	11.65
after 30 iteration	543.47	1.00	1.00	82.59	1.80	768.00	329.38	7.08
difference	735.90	5057.62	0.43	113.77	3.78	-768.00	23438799.62	4.57
improvement	57.52%	99.98%	29.96%	57.94%	67.76%	ERROR	100.00%	39.20%

estimator / design5	1	2	3	4	5	6	7	8
initial cost	1419.19	717.17	1.76	214.48	8.07	384.00	22421199.38	13.51
after 30 iteration	541.28	1.00	1.00	85.02	2.06	0.00	535.00	7.77
difference	877.91	716.17	0.76	129.46	6.01	384.00	22420664.38	5.73
improvement	61.86%	99.86%	43.32%	60.36%	74.43%	ERROR	100.00%	42.45%

estimator / design6	1	2	3	4	5	6	7	8
initial cost	3007.05	479.00	1.54	347.55	18.06	1143.00	24922486.99	17.70
after 30 iteration	1082.00	1.00	1.00	102.73	3.41	276.00	1186.70	9.42
difference	1925.05	478.00	0.54	244.82	14.64	867.00	24921300.29	8.28
improvement	64.02%	99.79%	35.21%	70.44%	81.09%	75.85%	100.00%	46.79%

能看到對於大部分的情形 (各種 design 搭配 estimator)，我們的方法能有效的降低電路的 cost。對於之後的 beta test，我們會優化模型中每個 Iteration 的行為，並且優化在電路化簡時需要不停讀檔寫檔的行為，爭取時間以執行更多輪 Iteration，也考慮將 cell library 中沒有明確意義的 attribute 納入 RL 的 input layer，透過分析整個電路的各個 attribute 來更加優化我們的模型。

參考資料：

- https://yosyshq.net/yosys/files/yosys_manual.pdf?fbclid=IwZXh0bgNhZW0CMTAAAR3YIaxE_d2li-u54TWQyjJpcSO2_KUvhCIkxr2plL6B1pUP9ZqRTfBWCCY_aem_AbiWrtpov2hKUuf6_8NtN_TKLPBasFmsSEQkqkiJlE8wTfgSVSuTX3T-Obj8cqZ9maVGXUo2LtAXVFyZWTp9p4YS
- https://www.youtube.com/watch?v=V_xro1bcAuA