Project 2 Report by Ishan Arya

gShare

We know the the size constraints are 1KB - 8KB or 8192 bits - 65,000,000 bits. Since we have a table of size 2^G with 2 bits in each entry, we are restricted with 12 <= G <= 14.

Experimenting with various G sizes, we get:

`G: 12, accuracy: 0.87`

`G: 13, accuracy: 0.906`

`G: 14, accuracy: 0.9305`

From initial tests while building the simulator, however, we can predict that Yeh-Patt and perceptron will perform better.

Yeh-Patt

With Yeh-Patt, our storage usage is decided by 2^G * P + 2^(P + 1) bits (history table has P bits in each entry and prediction table has 2 bits). Thus P <= 15 and G <= 16 (scaling the complement accordingly).

We start with a static G = 12:

`G: 12, P: 3; accuracy = 0.9208`

`G: 12, P: 8; accuracy = 0.936`

`G: 12, P: 10; accuracy = 0.9389`

`G: 12, P: 12; accuracy = 0.941`

So the general trend is increasing P will increase accuracy. However, increase in accuracy is not significant enough to be worth the storage cost. We try alternating G with max P at each value:

`G: 10, P: 14; accuracy = 0.8868`

`G: 11, P: 14; accuracy = 0.920`

`G: 13, P: 7; accuracy = 0.9472`

`G: 14, P: 3; accuracy = 0.945`

At G: 14, P: 3, the prediction table is too small and frequently mispredicts, leading to lower accuracy.

Perceptron

With similar reasoning to previous methods, the storage usage of perceptron is (G + 1) * 2^P * (log_2(1.93) * G + 14) so P <= 16 and scale G accordingly.

`G: 10, P: 10; accuracy = 0.965`

`G: 11, P: 10; accuracy = 0.9667`

Immediately we see perceptron has better accuracy. Even if we try a small storage size:

`G: 2, P: 10; accuracy = 0.921`

We see we outperform the other algorithms at the same budget.

We conclude perceptron is definitely the way to go.

Since G: 11, P: 10 uses maximal storage and G: 2, P: 10 uses almost minimal storage, we conclude that a middle ground will provide the optimal solution in terms of accuracy for storage usage.

We find an ideal number of:

`G: 5, P: 10; accuracy = 0.950`

using about 3KB of storage.