Ottimizzazione di circuiti lineari

Michele Corrias

Università degli Studi di Milano
Facoltà di Scienze MM.FF.NN
Dipartimento di Informatica

A.A. 2011/2012

Sommario

1. Introduzione
2. Ottimizzazione sul numero di porte
3. Ottimizzazione sulla profondità

Sommario

1. Introduzione
2. Ottimizzazione sul numero di porte
3. Ottimizzazione sulla profondità

Sommario

1. Introduzione
2. Ottimizzazione sul numero di porte
3. Ottimizzazione sulla profondità

I circuiti combinatori

Circuito

Un circuito combinatorio è un circuito hardware in cui input e output possono assumere solo due stati, corrispondenti ai livelli alto (1) e basso (0): in particolare gli output dipendono dagli input attraverso delle funzioni logiche.

Costo

Il costo di un circuito è essenzialmente determinato da due parametri:

- numero totale di porte logiche (*gate count*)
- profondità (*depth*) = cammino critico (numero di porte logiche sul più lungo percorso input-output)

I circuiti combinatori

Problema

È possibile trovare una tecnica che permetta di costruire un circuito di costo minimo? Col minimo numero di porte o la minima profondità?

NO

Il problema non è risolvibile in tempo polinomiale. Posso solo applicare delle euristiche di ottimo locale, che arrivano ad una soluzione sub-ottimale, sperando che magari sia proprio quella ottima.

I circuiti combinatori

S-Box di AES

I circuiti combinatori sono utilizzati in crittografia, ad esempio, per l'implementazione hardware di cifrari: in AES, nella operazione di SubBytes, vengono utilizzate le S-Box, matrici rappresentabili per mezzo di circuiti: più sono efficienti, più veloci saranno le fasi di cifratura/decifratura.
Le operazioni più utilizzate sono XOR e AND, che equivalgono a somma e prodotto mod 2.

- Circuiti con soli XOR sono detti *lineari*
- circuiti con (anche) AND sono *non-lineari*
    - ottimizzare componenti lineari è molto più facile, e al contempo vengono ottimizzate anche le componenti non lineari.

Ottimizzazione sul numero di porte

Paar[1]

Algoritmo

Data una matrice m x n del tipo $y_k = \bigoplus_{h=0}^{n} b_{k,h}$, con $b_{k,h} = 1 \ \forall \ 0 \leq k \leq m$

1. $\forall$ riga contare tutte le occorrenze di coppie (b_i, b_j), con $0 \leq \mathrm{i} \leq \mathrm{j} \leq$ n, per prendere la coppia più frequente
2. fare lo XOR tra b_i e b_j e sostituire il risultato nel sistema
3. ripetere l'algoritmo fin quando c'è un'unica variabile per ogni uscita

Boyar - Peralta[2]

Algoritmo

Data una matrice m x n del tipo $y_k = \bigoplus_{h=0}^{n} b_{k,h}$, con $b_{k,h} = 1 \ \forall$ $0 \leq k \leq m$

1. ad ogni passo esaminare il vettore di uscita D, che conta gli XOR necessari per ogni output
2. preferire, se c'è, una coppia che elimina un output
3. altrimenti eseguire Paar, utilizzando anche la cancellazione
4. in caso di coppia con frequenza di occorrenze medesime, preferire sempre la coppia che massimizza la norma di D
5. fare lo XOR della coppia considerata, e sostituire il risultato nel sistema
6. ripetere l'algoritmo fin quando D = 0

Boyar - Peralta su matrici opposte

Idea

- data una matrice M posso calcolare la sua opposta, sfruttando la cancellazione (lavoro in GF(2^n)!)
- precomputo t $= w_0 \oplus w_1 \oplus w_2 \oplus w_3 \oplus w_4$ e calcolo una nuova matrice opposta a quella originale
- $$\begin{array}{ccccccccc} z_0 & = & w_0 & \oplus & w_1 & \oplus & w_2 & & \\ z_1 & = & w_1 & \oplus & w_3 & \oplus & w_4 & & \\ z_2 & = & w_0 & \oplus & w_2 & \oplus & w_3 & \oplus & w_4 \\ z_3 & = & w_1 & \oplus & w_2 & \oplus & w_3 & & \\ z_4 & = & w_0 & \oplus & w_1 & \oplus & w_3 & & \\ z_5 & = & w_1 & \oplus & w_2 & \oplus & w_3 & \oplus & w_4 \end{array} = \begin{array}{ccccc} t & \oplus & w_3 & \oplus & w_4 \\ t & \oplus & w_0 & \oplus & w_2 \\ t & \oplus & w_1 & & \\ t & \oplus & w_0 & \oplus & w_4 \\ t & \oplus & w_2 & \oplus & w_4 \\ t & \oplus & w_0 & & \end{array}$$
- è come se considerassi, nella matrice ottenuta, gli XOR delle variabili a 0 anzichè a 1 (aggiungengo anche la variabile precomputata t)
- ho un costo fisso di precomputazione di n XOR, ma risparmio sulla matrice ottenuta (che è di dimensioni più ridotte)

Testing di Boyar - Peralta su matrici opposte

Risultati

Matrici 5×5	**Bias** $\simeq 25\%$	**Bias** $\simeq 50\%$	**Bias** $\simeq 75\%$
Boyar-Peralta	$1 \oplus$	$7 \oplus$	$7 \oplus$
Alg. custom	$14 \oplus$	$10 \oplus$	$10 \oplus$
Matrici 6×5	**Bias** $\simeq 25\%$	**Bias** $\simeq 50\%$	**Bias** $\simeq 75\%$
Boyar-Peralta	$2 \oplus$	$8 \oplus$	$8 \oplus$
Alg. custom	$16 \oplus$	$16 \oplus$	$12 \oplus$
Matrici 6×6	**Bias** $\simeq 25\%$	**Bias** $\simeq 50\%$	**Bias** $\simeq 75\%$
Boyar-Peralta	$3 \oplus$	$10 \oplus$	$11 \oplus$
Alg. custom	$17 \oplus$	$16 \oplus$	$11 \oplus$

- le matrici testate sono random

Risultati di Boyar - Peralta su matrici opposte

- l'algoritmo custom non presenta vantaggi evidenti
  - aumentando le dimensioni delle matrici, la precomputazione della variabile t per l'algoritmo custom diventa più onerosa
  - inoltre, con basse Bias la matrice ha tanti zeri, quindi oltre agli XOR della precomputazione si aggiungono gli XOR delle variabili a zero
- ma è ragionevole pensare che, al crescere delle dimensioni dei circuiti, matrici con "molti 1" (Bias $\simeq 100\%$) che perciò hanno pochi 0, possano essere computate più efficientemente con Boyar-Peralta custom che con Boyar-Peralta originale

Ottimizzazione sulla profondità

LowDepthGreedy [3]

Un'euristica greedy per componenti lineari

L'algoritmo LowDepthGreedy, sviluppato da Boyar e Peralta, tenta di minimizzare la profondità del circuito, tenendo sotto controllo il numero totale di porte [3]

Hamming

- Distanza di Hamming = misura, tra due stringhe di stessa lunghezza, il numero di sostituzioni necessarie per convertire una stringa nell'altra (la *distanza di Hamming* tra 1111 e 1010 è 2)
- Peso di Hamming: misura, per una stringa, la sua *distanza di Hamming* dalla stringa nulla della stessa lunghezza
  - quindi è il numero di elementi diversi da zero di una stringa: per una stringa binaria è semplicemente il numero di 1 (es.: il *peso di Hamming* di 1001 è 2)

LowDepthGreedy

```
Low_Depth_Greedy(M, m, n, k):
{ M is an m × n 0-1 matrix with Hamming weight at most 2^k in any row}
    s := n + 1 { index of the next column }
    i := 0
    ip := n   { columns up to n had depth at most i }
    while there is some row in M with Hamming weight > 2^(k−i−1) do
    { Phase i }
        if some row ℓ in M with weight 2
            had weight 2 at the beginning of the phase
            then let j1 and j2 be the columns in row ℓ with ones
            else find two columns 1 ≤ j1, j2 ≤ ip
                                which maximize |{ℓ | M[ℓ, j1] = M[ℓ, j2] = 1}|
        add an XOR gate with inputs from the variables for columns j1, j2
        the output variable produced will correspond to column s
        for ℓ = 1 to m do
            if M[ℓ, j1] = M[ℓ, j2] = 1
                then M[ℓ, j1] := 0; M[ℓ, j2] := 0; M[ℓ, s] := 1
                else M[ℓ, s] := 0
        s := s + 1
    ip := s − 1   { keep track of which gates had depth at most i }
    i := i + 1
```

Figure 4: Algorithm for creating a minimum depth circuit for linear components

LowDepthGreedy

Parametri

- m righe, n colonne
- $k = \lceil log_2 w \rceil$, con w max Hamming weight tra le righe. k sarà la profondità del circuito dopo l'esecuzione dell'algoritmo
- $s =$ indice della prossima colonna
- $ip =$ colonna con profondità al più i

Euristica:

- LowDepthGreedy mantiene l'approccio greedy di Paar, ma solo fin tanto che questo non incrementa la profondità
- k fasi, partendo da $i \geq 0$. All'inizio di ogni fase viene fatto un check su ogni riga: se ha Hamming weight 2, allora la coppia di variabili a 1 viene XORata; altrimenti viene scelta, tra le coppie candidate, quella che compare più frequentemente tra le righe della matrice (come prevede Paar)
- il risultato dello XOR, in ogni caso, viene sostituito nel sistema

LowDepthGreedy

Teorema

- alla fase *i* solo le variabili input o gli XOR prodotti fino a profondità *i* sono considerati come possibili input di nuove porte logiche da produrre in fase *i*. Quindi le porte prodotte in fase *i* hanno profondità al massimo *i+1*
- l'algoritmo termina in *k-1* fasi, ottenendo un circuito di profondità massima *k*. Il tempo di esecuzione è $O(mt^3)$, con numero finale delle colonne $t \leq m * n + n - m$

Esecuzione di LowDepthGreedy

LowDepthGreedy(M, 6, 5, 2)

$$
\begin{array}{ccccccccccc}
z_0 & = & w_0 & \oplus & w_1 & \oplus & w_2 & & & & \\
z_1 & = & & & w_1 & \oplus & & & w_3 & \oplus & w_4 \\
z_2 & = & w_0 & \oplus & & & w_2 & \oplus & w_3 & \oplus & w_4 \\
z_3 & = & & & w_1 & \oplus & w_2 & \oplus & w_3 & & \\
z_4 & = & w_0 & \oplus & w_1 & \oplus & & & w_3 & & \\
z_5 & = & & & w_1 & \oplus & w_2 & \oplus & w_3 & \oplus & w_4
\end{array}
=
\begin{array}{ccccc}
1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1
\end{array}
$$

- max Hamming weight (H.w.) $= 4$
- $s = 6$; $i = 0$; $ip = 5$

Esecuzione di LowDepthGreedy

Fase 0

- $H.w.4 > 2^{k-i-1=1} = 2$ (condizione soddisfatta)
- $s_0 = w_1 \oplus w_3$ (ultima colonna)
- $s = 7$

$$
\begin{array}{ccccccccc}
z_0 & = & w_0 & \oplus & w_1 & \oplus & w_2 & & \\
z_1 & = & w_4 & \oplus & s_0 & & & & \\
z_2 & = & w_0 & \oplus & w_2 & \oplus & w_3 & \oplus & w_4 \\
z_3 & = & w_2 & \oplus & s_0 & & & & \\
z_4 & = & w_0 & \oplus & s_0 & & & & \\
z_5 & = & w_2 & \oplus & w_4 & \oplus & s_0 & &
\end{array}
=
\begin{array}{ccccc|c}
1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1
\end{array}
$$

Esecuzione di LowDepthGreedy

Fase 0

- $H.w.4 > 2$ (condizione soddisfatta)
- $s_1 = w_0 \oplus w_2$ (ultima colonna)
- $s = 8$

$$
\begin{array}{lcccccc}
z_0 & = & w_1 & \oplus & s_1 & & \\
z_1 & = & w_4 & \oplus & s_0 & & \\
z_2 & = & w_3 & \oplus & w_4 & \oplus & s_1 \\
z_3 & = & w_2 & \oplus & s_0 & & \\
z_4 & = & w_0 & \oplus & s_0 & & \\
z_5 & = & w_2 & \oplus & w_4 & \oplus & s_0
\end{array}
=
\begin{array}{cccccc|c}
0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0
\end{array}
$$

Esecuzione di LowDepthGreedy

Fase 0

- $H.w.3 > 2$ (condizione soddisfatta)
- $s_2 = w_2 \oplus w_4$ (ultima colonna)
- $s = 9$

$$
\begin{array}{lllllll}
z_0 & = & w_1 & \oplus & s_1 & & \\
z_1 & = & w_4 & \oplus & s_0 & & \\
z_2 & = & w_3 & \oplus & w_4 & \oplus & s_1 \\
z_3 & = & w_2 & \oplus & s_0 & & \\
z_4 & = & w_0 & \oplus & s_0 & & \\
z_5 & = & s_0 & \oplus & s_2 & &
\end{array}
=
\begin{array}{ccccccc|c}
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1
\end{array}
$$

Esecuzione di LowDepthGreedy

Fase 0

- $H.w. 3 > 2$ (condizione soddisfatta)
- $s_3 = w_3 \oplus w_4$ (ultima colonna)
- $s = 10$

$$\begin{array}{ccccc} z_0 & = & w_1 & \oplus & s_1 \\ z_1 & = & w_4 & \oplus & s_0 \\ z_2 & = & s_1 & \oplus & s_3 \\ z_3 & = & w_2 & \oplus & s_0 \\ z_4 & = & w_0 & \oplus & s_0 \\ z_5 & = & s_0 & \oplus & s_2 \end{array} = \begin{array}{cccccccc|c} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{array}$$

Esecuzione di LowDepthGreedy

Fase 0

$H.w.2 \not> 2$ (condizione insoddisfatta)

- Aggiorno le variabili e passo ad una nuova fase
- $ip = 9$; $i = 1$; $s = 10$ (non considero più solo gli input per contare le occorrenze, ma anche gli XOR prodotti alla fase precedente)

Fase 1

- $H.w.2 > 2^{k-i-1=0} = 1$ (condizione soddisfatta)
- $s_4 = w_0 \oplus s_0$ (considero la prima riga con H.w. $= 2$)

$$
\begin{array}{lllll}
z_0 & = & w_1 & \oplus & s_1 \\
z_1 & = & w_4 & \oplus & s_0 \\
z_2 & = & s_1 & \oplus & s_3 \\
z_3 & = & w_2 & \oplus & s_0 \\
z_4 & = & s_4 & & \\
z_5 & = & s_0 & \oplus & s_2
\end{array}
=
\begin{array}{ccccccccc|c}
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}
$$

Esecuzione di LowDepthGreedy

Fase 1

- $H.w.2 > 1$ (condizione soddisfatta)
- $s_5 = w_1 \oplus s_1$
- $s_6 = w_2 \oplus s_0$
- $s_7 = w_4 \oplus s_0$
- $s_8 = s_0 \oplus s_2$
- $s_9 = s_1 \oplus s_3$
- $s = 16$

Fase 1

$H.w.1 \not> 1$ (condizione insoddisfatta)

- Aggiorno le variabili e passo ad una nuova fase
- $ip = 15;\ i = 2;\ s = 16$
- $H.w.2^{k-i-1=2-3<0} \Rightarrow$STOP

Esecuzione di LowDepthGreedy

Matrice finale

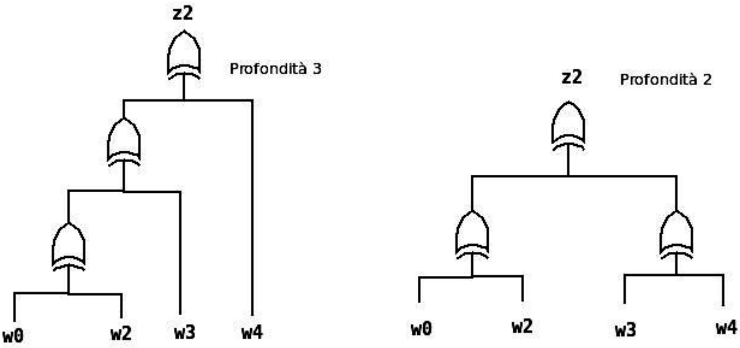
$$
\begin{array}{lllllll}
z_0 & = & s_5 & = & w_1 \oplus s_1 & = & w_1 \oplus (w_0 \oplus w_2) \\
z_1 & = & s_7 & = & w_4 \oplus s_0 & = & w_4 \oplus (w_1 \oplus w_3) \\
z_2 & = & s_9 & = & s_1 \oplus s_3 & = & (w_0 \oplus w_2) \oplus (w_3 \oplus w_4) \\
z_3 & = & s_6 & = & w_2 \oplus s_0 & = & w_2 \oplus (w_1 \oplus w_3) \\
z_4 & = & s_4 & = & w_0 \oplus s_0 & = & w_0 \oplus (w_1 \oplus w_3) \\
z_5 & = & s_8 & = & s_0 \oplus s_2 & = & (w_1 \oplus w_3) \oplus (w_2 \oplus w_4)
\end{array}
$$

$$
= \begin{matrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{matrix}
$$

Risultati di Low_Depth_Greedy

Profondità

- L'algoritmo termina in $k = 2$ fasi, con 10 XOR.
- La profondità del circuito è 2, mentre inizialmente era 3.



Possibili improvements di LowDepthGreedy

LowDepthGreedy Custom

- un'altra possibilità per produrre circuiti lineari di profondità ottimale è considerare come input delle porte logiche da produrre inizialmente solo le coppie di input (depth 0), successivamente considerare come input delle porte che avranno depth 2 solo coppie di valori a depth 1 (e al limite i restanti input non accoppiati alla fase precedente)...
- alla fase $i + 1$ considerare come input delle porte logiche da produrre solo quei valori prodotti alla fase i (più eventuali valori non accoppiati nelle fasi precedenti)

Possibili improvements di LowDepthGreedy

- cerco così di utilizzare tutti i valori precedenti: in questo modo si sviluppa un circuito non in profondità, ma in larghezza (come un albero bilanciato)
- tuttavia, il metodo prescelto dal LowDepthGreedy originale, rispetto a questa variante permette maggiore flessibilità nella scelta di porte, garantendo così maggiori possibilità di creare porte logiche utilizzabili più di una volta
  - preservo *gate count*!

Possibili improvements di LowDepthGreedy

Ottimizzare profondità localmente

- attraverso tecniche di LOCAL REPLACEMENT nei cammini critici del circuito, è possibile abbassare la profondità totale
- qualsiasi porta logica prodotta è il risultato dello XOR tra diversi precedenti valori: questi possono essere XORati in qualsiasi ordine per ottenere lo stesso output

1. $w_3 = w_0 \oplus w_1$ con $w_0 \rightarrow$ profondità d_0
2. $z = w_2 \oplus w_3$ con $w_2 \rightarrow d_1, w_3 \rightarrow d_2$

- $d_0, d_1 \leq d_2 - 2$

La profondità di z è $d_2 + 1$. Il risultato è equivalente a:

1. $x = w_0 \oplus w_2$
2. $z = w_1 \oplus x$

Possibili improvements di LowDepthGreedy

- evito però di calcolare w_3 che era il collo di bottiglia. Risparmio così sulla profondità totale di 1, infatti $z \to d_1 + 1$
- le tecniche di *local replacement* risultano efficaci nel caso in cui il vecchio output w_3 non sia utilizzato in altre parti del circuito

Cancellazione

- CANCELLAZIONE di variabili non utilizzata sin da subito, dal momento che spesso è necessario che qualcosa con un grande Hamming weight sia già stato calcolato per poterla applicare efficacemente
- si potrebbe introdurre la cancellazione ad una certa fase i, dopo aver precomputato un certo numero di variabili
- potrebbe ridurre il *gate count*

Possibili improvements di LowDepthGreedy

Ottimizzare numero porte

- dopo l'esecuzione di LowDepthGreedy ottengo un circuito di profondità ottimale
- si può mirare a decrementare il *gate count* su questo circuito
- ottimizzando sulla profondità si aumentano inevitabilmente i costi sull'uso di porte logiche
  - LowDepthGreedy impiega 10 XOR sull'esempio fatto sopra e 128 porte su AES S-box
  - l'algoritmo di Boyar-Peralta per la minimizzazione del *gate count* arriva a 8 XOR sullo stesso esempio, e 115 porte su AES[2]
  - c'è quindi ancora un margine di miglioramento sul numero totale di porte
- si può "rilassare" ogni percorso di profondità $i < max(\text{depth})$, cioè percorsi più corti del cammino critico, portandoli ad una profondità maggiore $j > i$ per risparmiare XOR

Possibili improvements di LowDepthGreedy

- ad esempio, XOR di profondità 3 che non sono determinanti per la profondità totale, potrebbero essere la somma di una coppia di qualsiasi altro output di profondità maggiore
- LowDepthGreedy potrebbe essere modificato utilizzando un array di appoggio che tenga conto, per ogni riga, quanto ogni coppia può essere "calcolata più tardi", se non è determinante per la profondità
- ciò permette la possibilità di cercare input ad una profondità maggiore per eventuali cancellazioni

Conclusioni

Gate Count Optimization

- Boyar-Peralta Custom, per matrici opposte, non ha dato vantaggi su input di ridotte dimensioni
- dal testing si evince che in circuiti di dimensioni reali e con alta frequenza di 1 può apportare vantaggi

Depth Optimization

- LowDepthGreedy per ottenere profondità ottimale con un discreto numero di XOR
- produce AES S-box di *depth* 16 con *gate count* 128 (prima *Nogami* riusciva ad ottenere S-box di *depth* 22 con *gate count* 148 [4])
  - LowDepthGreedy Custom
  - *local replacement*
  - cancellazione
  - *gate count optimization* sul circuito di minima profondità

Bibliografia

- C. Paar. Optimized arithmetic for Reed-Solomon encoders. In *IEEE International Symposium on Information Theory*, page 250, 1997.
- J. Boyar and R. Peralta. A new combinational logic minimization technique with applications to cryptology. In P. Festa, editor, *SEA*, volume 6049 of *Lecture Notes in Computer Science*, pages 178-189. Springer, 2010.
- J. Boyar and R. Peralta. A depth-16 circuit for the AES S-box. In *Cryptology ePrint Archive, Report 2011/332*, 2011.
- Y. Nogami, K. Nekado, T. Toyota, N. Hongo, and Y. Morikawa. Mixed bases for efficient inversion in $f(((2^2)^2)^2)$ and conversion matrices of subbytes of AES. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 234-247. Springer, 2010.