類比電路佈局合成自動化
Automatic Layout Synthesis for Analog Circuits

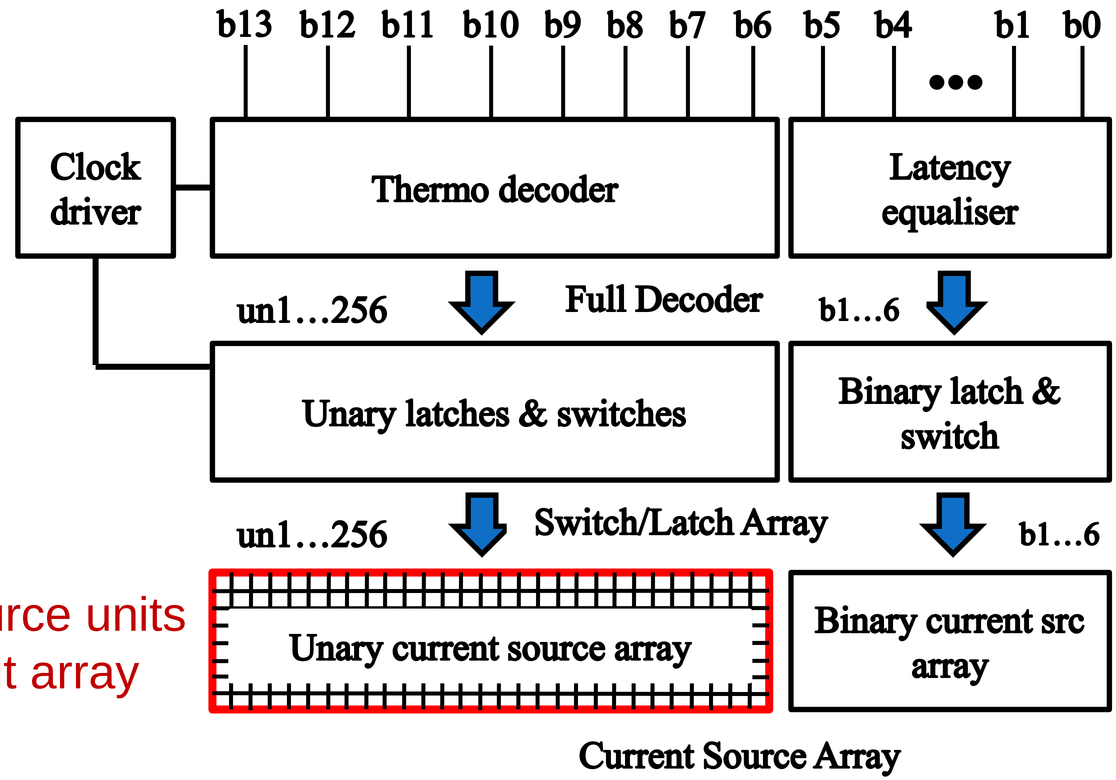
單元三
同時考慮一階與二階系統製程變異的巨型矩陣元件擺置

Lecturer: Shao-Yun Fang
The Electronic Design Automation Laboratory
Department of Electrical Engineering
National Taiwan University of Science and Technology
Taipei 106, Taiwan

THE EDA LAB
TAIWAN TECH

Recall the DAC

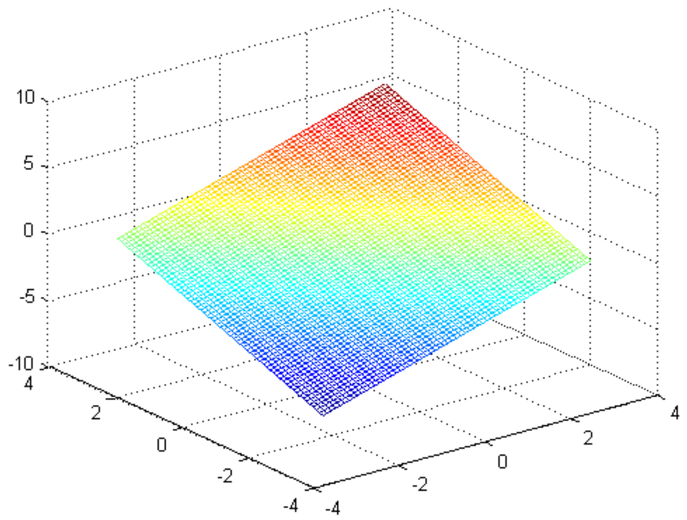
- **14-bit current-steering digital-to-analog converter (DAC)**
  - The 8 MSBs are decoded from a thermometer decoder that steers a unary-weighted current source array
  - The 6 LSBs steer the binary weighted current source array



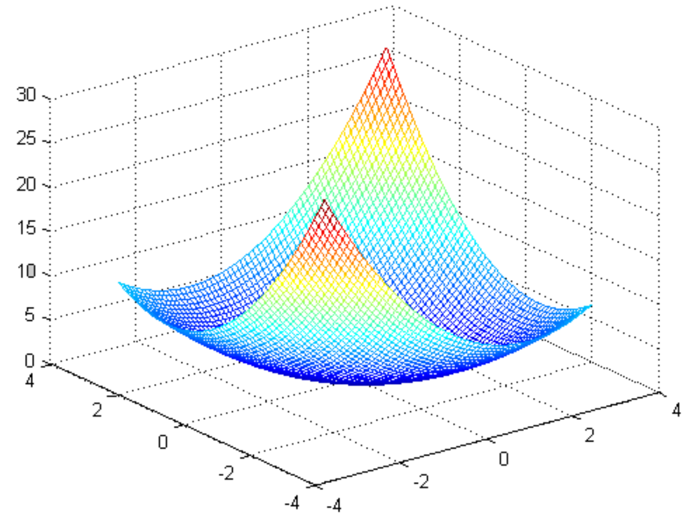
4096 current source units
$\Longrightarrow$ 64 × 64 unit array

Systematic Mismatch Recap

- **Caused by process variation, thermal distribution, uneven mechanical stress from other circuit layers, etc.**
- **Show up as spatial gradients in device parameters**



First-order gradient



Second-order gradient

Gradient Error Computation

- **Thermal-induced or technology-rated gradient error can be approximated with a Taylor series expansion**

$$p(x, y) = \sum_{i=1}^{\infty} G_i(x, y) + C$$

  - (x, y): the location of a unit
  - C: a constant independent of location
  - $G_i(x, y)$: the i-th order gradient error

$$G_i(x, y) = \sum_{j=0}^{i} g_{j,i-j} x^j y^{i-j}$$

    - $g_{j,i-j}$: the coefficient of $x^j y^{i-j}$
  - The lower order terms will have greater impacts on the mismatches among devices

Gradient Error Computation (cont'd)

- **The overall error by considering the first-order (linear) and the second-order (quadratic) gradients:**

$$p(x,y) \approx G_1(x,y) + G_2(x,y)$$
$$= g_{1,0}x + g_{0,1}y + g_{2,0}x^2 + g_{1,1}xy + g_{0,2}y^2$$

- **Placements satisfying the common centroid constraint may have quite different 2nd order gradient errors**

a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2
d_3	c_3	b_3	a_3
d_4	c_4	b_4	a_4

1st order error: 0
Max 2nd order error: 20

b_1	c_1	d_1	b_2
d_2	a_1	a_2	c_3
c_2	a_3	a_4	d_3
b_3	d_4	c_4	b_4

1st order error: 0
Max 2nd order error: 18

a_1	c_2	d_1	b_1
c_1	a_2	b_2	d_2
d_4	b_3	a_3	c_3
b_4	d_3	c_4	a_4

1st order error: 0
Max 2nd order error: 15

Integral Nonlinearity (INL)

- **The maximum difference between the ideal and the actual output levels for a DAC**
  - The actual output level of the *u*-th unit of the *k*-th current source

$$p_{k,u}(x,y) = p_d + p(x,y)$$

    - p_d: the designed (default) output level
  - The total output level of the *k*th current source

$$p_k = \sum_{u=1}^{num} p_{k,u}$$

  - The accumulated output level from the first current source to the *l*-th current source:

$$P(l) = \sum_{k=1}^{l} p_k$$

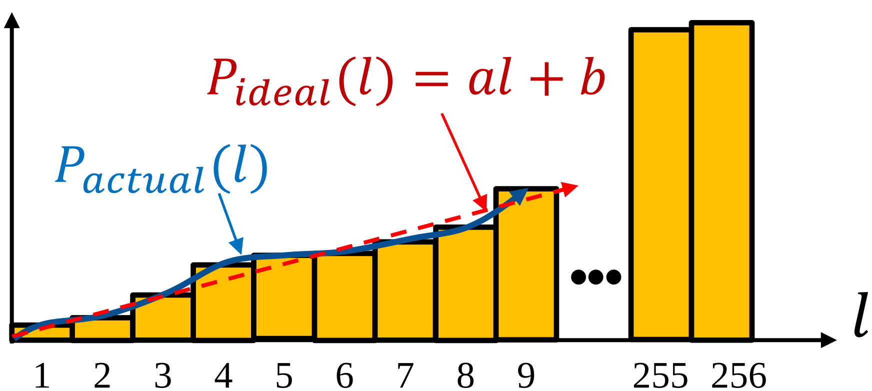
Integral Nonlinearity (cont'd)

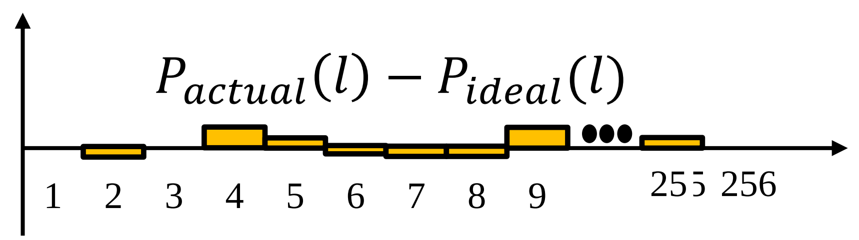
- **The ideal output level can be found by**
  - Find a best-fit line with a linear curve fitting method

$$P_{ideal}(l) = al + b$$

  - INL is the maximum difference between the ideal and the actual output levels

$$INL = \max_{l} |P_{actual}(l) - P_{ideal}(l)|$$

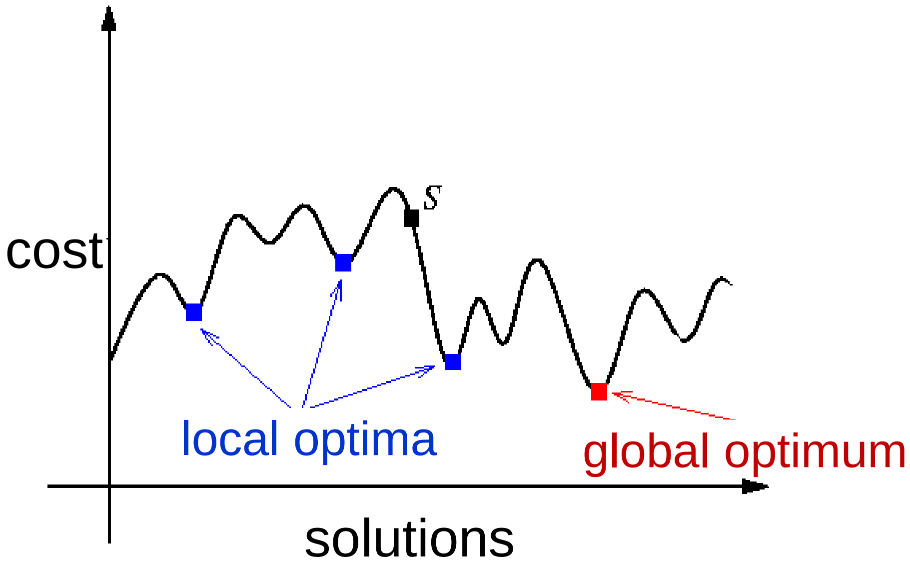




Simulated Annealing

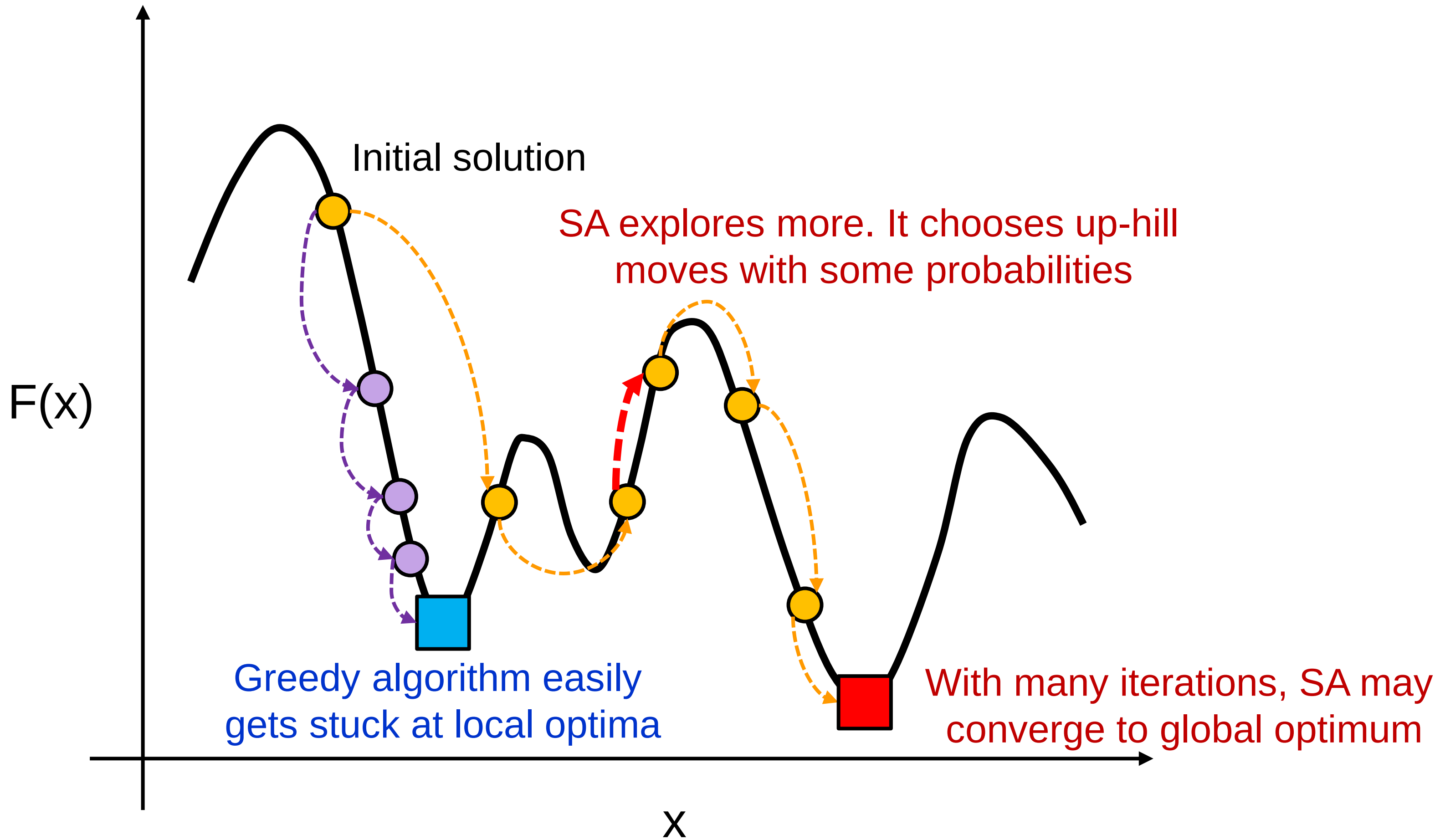
Simulated Annealing (SA)

- **Kirkpatrick, Gelatt, and Vecchi, "Optimization by simulated annealing," *Science*, May 1983.**
- **Simulated annealing is**
  - Motivated by the physical annealing process
  - Used to solve optimization problems
  - A stochastic algorithm
  - Escaping from local optima and finding the global optimum by allowing worsening moves



Simulated Annealing vs. Greedy Strategy

- **Use up-hill moves to escape from local optima**



Simulated Annealing Basics

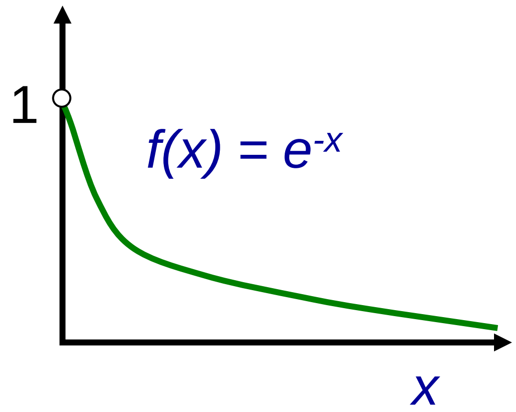
- **Given an initial/a current solution S with its cost $cost(S)$ computed by an objective function**
  - A neighboring solution S' is randomly picked
  - If S' has a lower cost, it is accepted
  - If S' has a higher cost, it still has a probability to be accepted

- $$Prob(S \rightarrow S') = \begin{cases} 1 & , if\ \Delta C \leq 0 \\ e^{-\frac{\Delta C}{T}} & , if\ \Delta C > 0 \end{cases}$$

Down-hill moves

Up-hill moves

  - $\Delta C = cost(S') - cost(S)$
  - T: control parameter (temperature)
  - Probability (T) depends on
    - Magnitude of the “up-hill” movement
    - Total search time



Generic Simulated Annealing Algorithm

At a fixed temperature

```
begin
Get an initial solution S;
Get an initial temperature T > 0;
while not yet "frozen" do
    for 1 ≤ i ≤ P do
        Pick a random neighbor S' of S;
        Δ ← cost(S') - cost(S);
        /* downhill move */
        if  Δ ≤ 0 then S ← S'
        /* uphill move */
        if  Δ > 0 then S ← S' with probability e^(-Δ/T);
    T ← rT;  /* reduce temperature */
return Best solution found
end
```

Basic Ingredients for Simulated Annealing

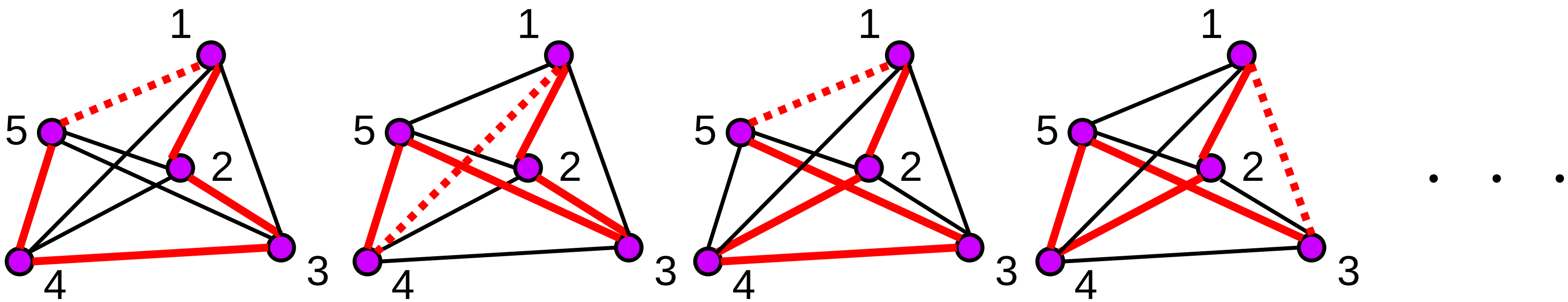
- **Analogy:**

Physical system	**Optimization problem**
System state	Solution
Energy	Cost function
Ground state	optimal solution
Quenching	Local search
Real annealing	Simulated annealing

- **Basic Ingredients for Simulated Annealing:**
  - **Solution space:** What are the feasible solutions?
  - **Neighborhood structure:** How to find a neighboring solution from the current one?
  - **Cost function:** How to evaluate the quality of a solution?
  - **Annealing schedule:** How to conduct the search process to find a desired solution? Starting temperature, stop criteria, cooling function?

An Example: Traveling Salesman Problem

- **The well-known traveling salesman problem (TSP)**
  - **Input:** A set of points P (cities) together with a distance $d(p, q)$ between any pair $p, q \in P$
  - **Output:** The shortest circular route that starts and ends at a given point and visits all the points (cities)



<1,2,3,4,5> <1,2,3,5,4> <1,2,4,3,5> <1,2,4,5,3>

4 of 5! solutions

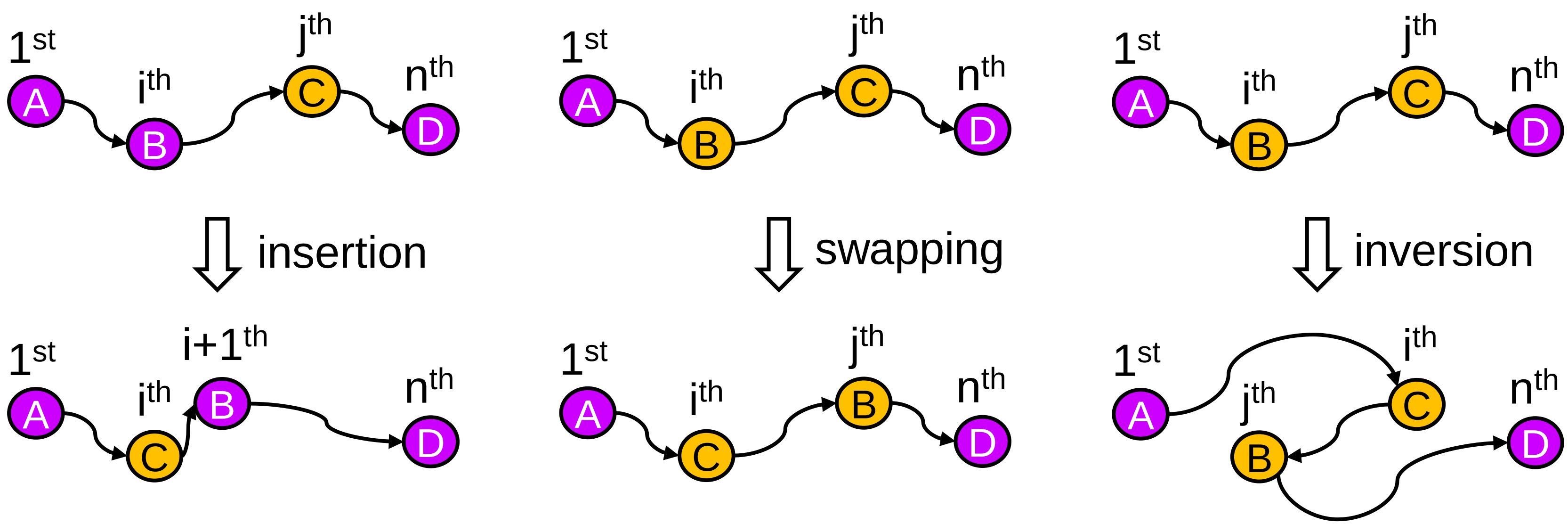
- **Solve with simulated annealing?**

Simulated Annealing for TSP

- **Solution space**
  - All permutations ($n!$) of the n points
- **Cost function**
  - The total distance of the route represented by each permutation
- **Annealing schedule**
  - $T =< T_0, T_1, T_2, \ldots >$, where $T_i = r^i T_0, r < 1$
  - At each temperature, try kn moves ($k = 5 \sim 10$).
  - Terminate the annealing process if, for example
    - # of accepted moves < 5%,
    - Temperature is low enough, or
    - Run out of time.

Simulated Annealing for TSP (cont'd)

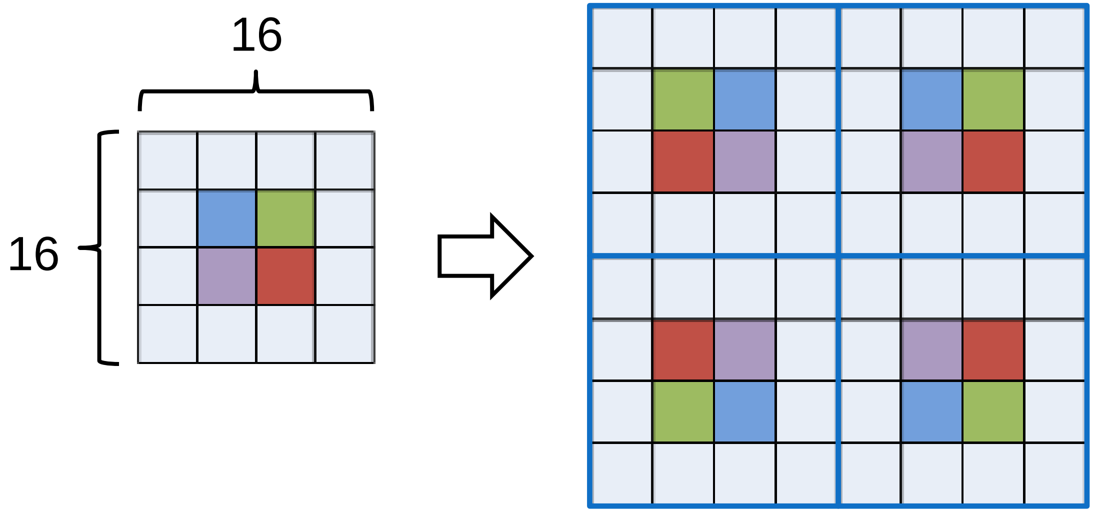
- **Neighborhood structure: given a current route R, a neighbor R' may be generated by many operations:**
  - Insert a point from position j to position i
  - Swap a pair of points at positions i and j
  - Invert the positions of the points between positions i and j



Device Unit Array Placement for Quadratic Gradient Error Mitigation (LAB 3)

The Target Device Unit Array

- **An $n \times n$ device unit array will be automatically placed**
  - Each current source consists of 4 units (a simplified version)
  - Ex: 256 current sources will form a 32×32 current source unit array
- **To follow the common centroid constraint**
  - A quarter of the current source array is first determined $\left(\frac{n}{2} \times \frac{n}{2}\right)$
  - The sub-array is then mirrored by the x- and y-axes



Device Array Placement

- **Fill the sub-array with the indices of the current sources**
  - The current sources will be sequentially switched on according to the indices
  - Given the dimension $\left(\frac{n}{2}\right)$ of the sub-array, the total number of current sources will be $\left(\frac{n}{2}\right)^2$, which are indexed in the range $\left[0, \left(\frac{n}{2}\right)^2 - 1\right]$
- **A placement of a sub-array with 16 current sources**

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Objective

- **Derive a placement with the minimized INL**
  - The INL can be computed for the mirrored array of the original dimensions

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Mirror

15	14	13	12	12	13	14	15
11	10	9	8	8	9	10	11
7	6	5	4	4	5	6	7
3	2	1	0	0	1	2	3
3	2	1	0	0	1	2	3
7	6	5	4	4	5	6	7
11	10	9	8	8	9	10	11
15	14	13	12	12	13	14	15

INL

94.3529

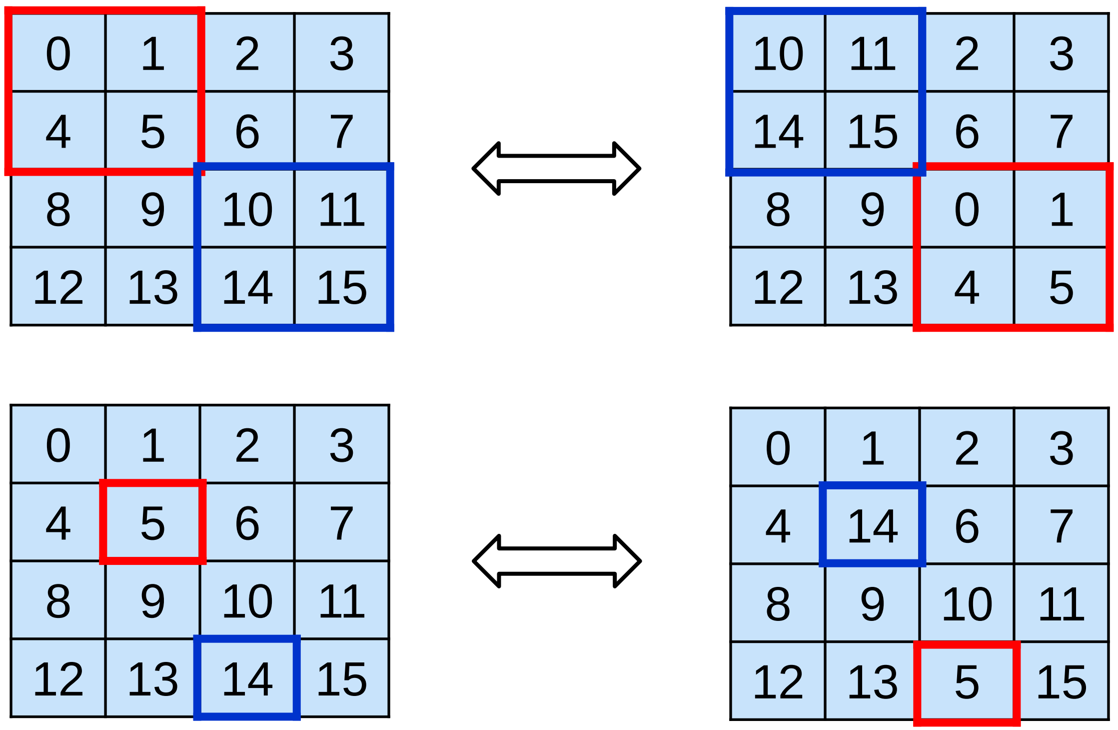
4	10	11	15
14	5	13	1
6	9	3	7
8	2	0	12

. . .

34.7765

SA-Based INL Minimization

- Any operation swapping two sets (of the same size) of current source units may be conducted during SA
- Different swapping operations may be suitable for different stages in the annealing schedule



Lab 3: Device Array Placement

- **Implement an SA-based current source unit array placement to minimize INL**
  - **Input:** the dimension of the sub-array $\frac{n}{2}$, the default output level p_d
  - **Output:** a unit sub-array (2D vector) with current source indices
  - Output format

$$
\begin{array}{c}
[I_{0,0}]\ [I_{0,1}]\ [I_{0,2}]\ \ldots\ [I_{0,\frac{n}{2}}] \\
[I_{1,0}]\ [I_{1,1}]\ [I_{1,2}]\ \ldots\ [I_{1,\frac{n}{2}}] \\
\ldots \\
[I_{\frac{n}{2},0}]\ [I_{\frac{n}{2},1}]\ [I_{\frac{n}{2},2}]\ \ldots\ [I_{\frac{n}{2},\frac{n}{2}}]
\end{array}
$$

Sample output file:

```
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
```

  - $[I_{i,j}]$: the index of the current source placed at (i, j) of the 2D vector

Sample Code for Input and Output

- **The provided code demonstrates command-line parameter, output file, and two placements with INL**

```
int main(int argc, char **argv){
    if(argc != 4){
        cout << "Usage: ./[exe] [dimension of matrix] [p_d]
                                                    [result.out]" << endl;
        exit(1);
    }
    int dimension = atoi(argv[1]);
    int pd = atoi(argv[2]);
    Placer unit_placer(dimension, pd);
    unit_placer.demoBasicPlacement();
    unit_placer.demoRandomPlacement();

    fstream fout;
    fout.open(argv[3],fstream::out);
    if(!fout.is_open()){
        cout << "Error: the output file is not opened!!" << endl;
        exit(1);
    }
    fout << "Hello world!" << endl;
    return 0;
}
```

Sample Code for INL Computation

- **The computation of INL of a given sub-array has also been provided**

```
double Placer::INL(vector< vector<int> > &matrix)
{
    vector<double> x;
    vector<double> y;
    vector<double> x2;
    vector<double> xy;
    const int n = _dimension * _dimension;
    x.resize(n);
    y.resize(n);
    x2.resize(n);
    xy.resize(n);
    for(int i = 0; i < _dimension; ++i)
    {
        for(int j = 0; j < _dimension; ++j)
        {
            int deviceIdx = matrix[i][j];
            y[deviceIdx] = ((i + 0.5) * (i + 0.5) +
                                    (j + 0.5) * (j + 0.5) + _pd) * 4;
        }
    }
```

Sample Code for INL Computation (cont’d)

```
    for(int i = 1; i < n; ++i){
        y[i] += y[i - 1];
    }
    for(int i = 0; i < n; ++i){
        x[i] = i;
        x2[i] = i * i;
        xy[i] = i * y[i];
    }
    double sumX = accumulate(x.begin(), x.end(), 0.0);
    double sumY = accumulate(y.begin(), y.end(), 0.0);
    double sumX2 = accumulate(x2.begin(), x2.end(), 0.0);
    double sumXY = accumulate(xy.begin(), xy.end(), 0.0);

    // y = ax + b
    double a = (n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX);
    double b = (sumY - a * sumX) / n;
    vector<double> diff;
    diff.resize(n);
    for(int i = 0; i < n; ++i){
        diff[i] = abs(y[i] - (a * i + b));
    }
    double inl = *max_element(diff.begin(), diff.end());
    return inl;
}
```