

# **Population-based Optimization:**

## **Particle Swarm Optimization (PSO)-II**

Lecture 16 – Tuesday July 8, 2014

# Outline

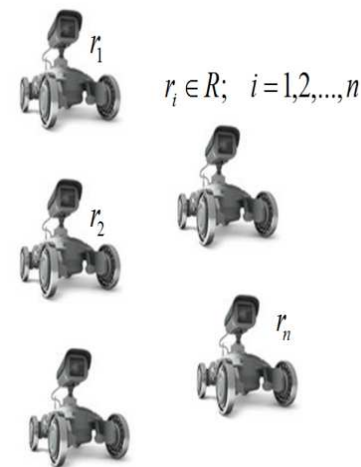
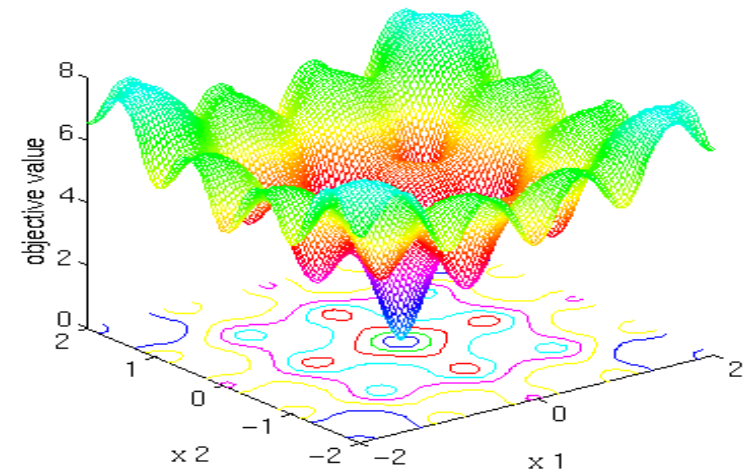
- Discrete PSO
- Binary PSO
- Permutation PSO
- FPGA Placement

# Outline

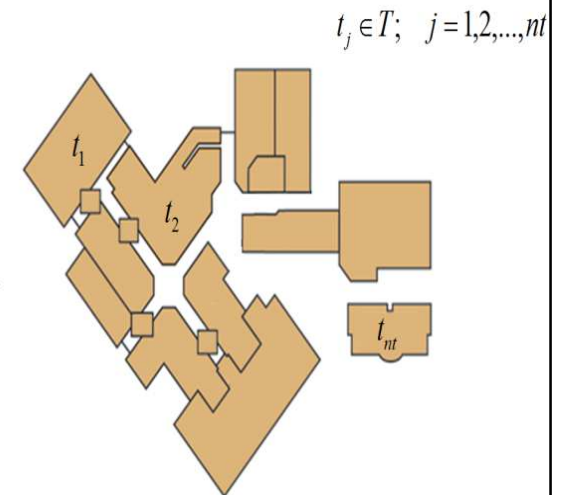
- **Discrete PSO**
- Binary PSO
- Permutation PSO
- FPGA Placement

# Discrete PSO

- PSO was originally developed for **continuous-valued spaces**.
- Many problems are defined for discrete valued spaces
- **Examples:** Feature selection, TSP, Assignment problems, Scheduling,...



A set of  $n$  mobile Robots:  $R$



A set of  $nt$  surveillance tasks:  $T$

# Discrete PSO

- **Regular/Real-valued PSO**

- ◇ **Particle's velocity Update:**

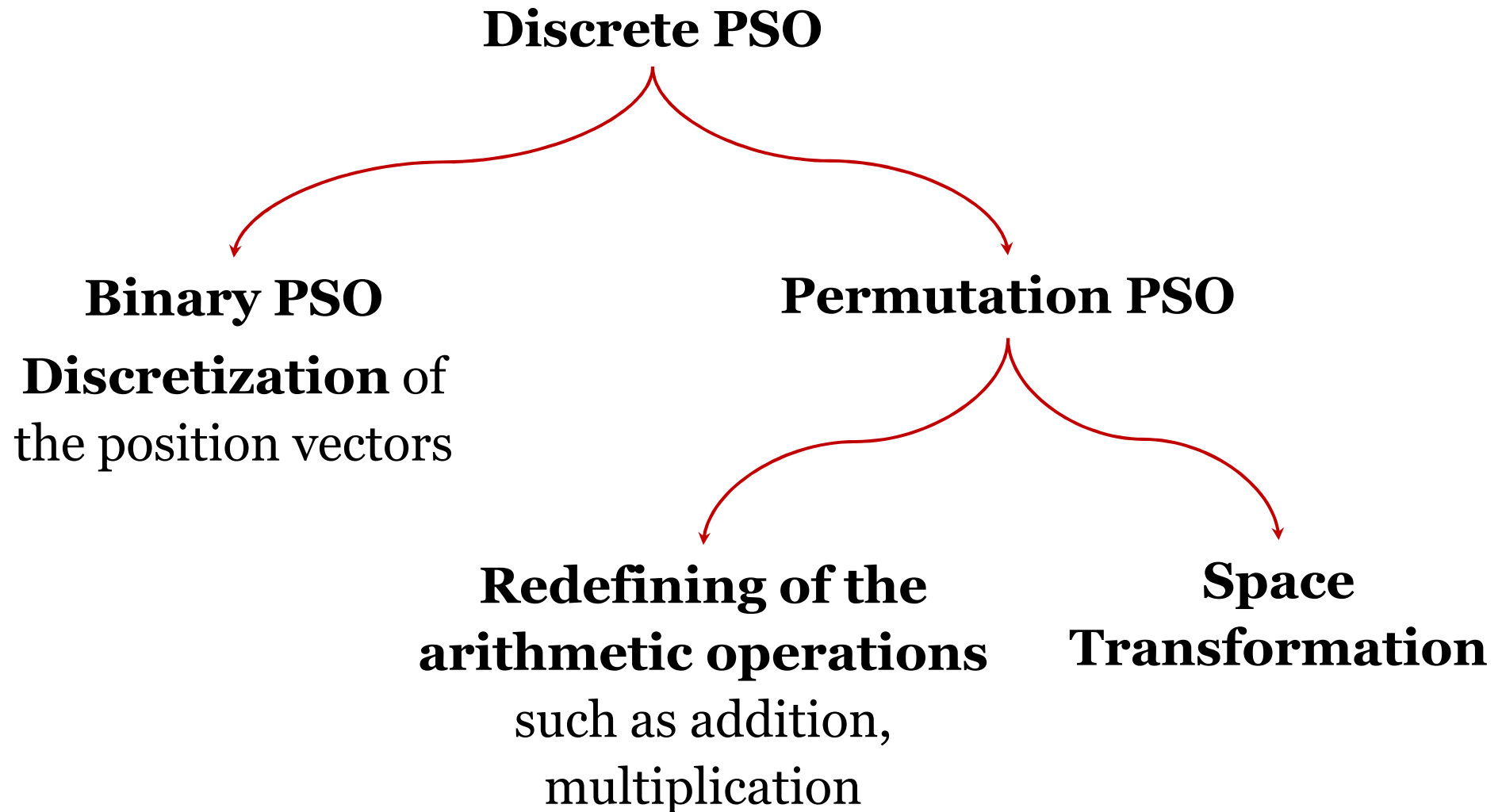
$$v_{t+1}^{id} = w * v_t^{id} + c_1 r_1^{id} (pbest_t^{id} - x_t^{id}) + c_2 r_2^{id} (Nbest_t^{id} - x_t^{id})$$

- ◇ **Particle's Position Update:**

$$x_{t+1}^{id} = x_t^{id} + v_{t+1}^{id}$$

- To be able to handle discrete problems, changes to PSO should be made.

# Discrete PSO



# Outline

- Discrete PSO
- **Binary PSO**
- Permutation PSO
- FPGA Placement

# Binary PSO

- A binary PSO (BPSO) version was proposed in [2].
- In BPSO, each particle represents a position in the binary space.
- Each element can take the value of **0 or 1**.
- Binary strings are **updated bit-by-bit** based on its current value, the value of that bit in the best (fitness) of that particle to date, and the best value of that bit to date of its neighbours

Individual  $i$

Position	1	2	...	d	...	n
State	0	1	*	$x_t^{id}$	*	1



# Binary PSO

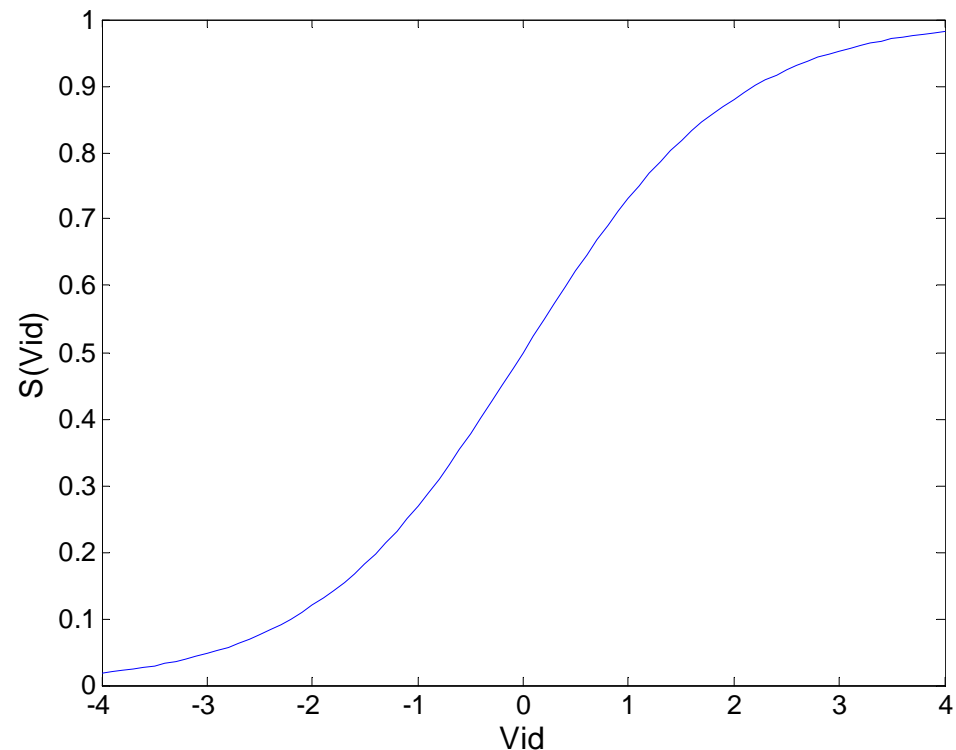
- In regular (real valued) PSO, everything is in terms of a velocity.
- **In BPSO, how does one define a velocity for a single bit?**
- Generally the velocity is defined in terms of a probability of the bit changing.
- If  $V_t^{id} = 0.3$



# Binary PSO

- Since velocities represent **probabilities**, the values of the velocity elements need to be restricted in the range **[0,1]**.
- This is done by using the **sigmoid function**:

$$\text{sig}(V_t^{id}) = \frac{1}{1 + e^{-V_t^{id}}}$$



S-shaped sigmoid

[1]

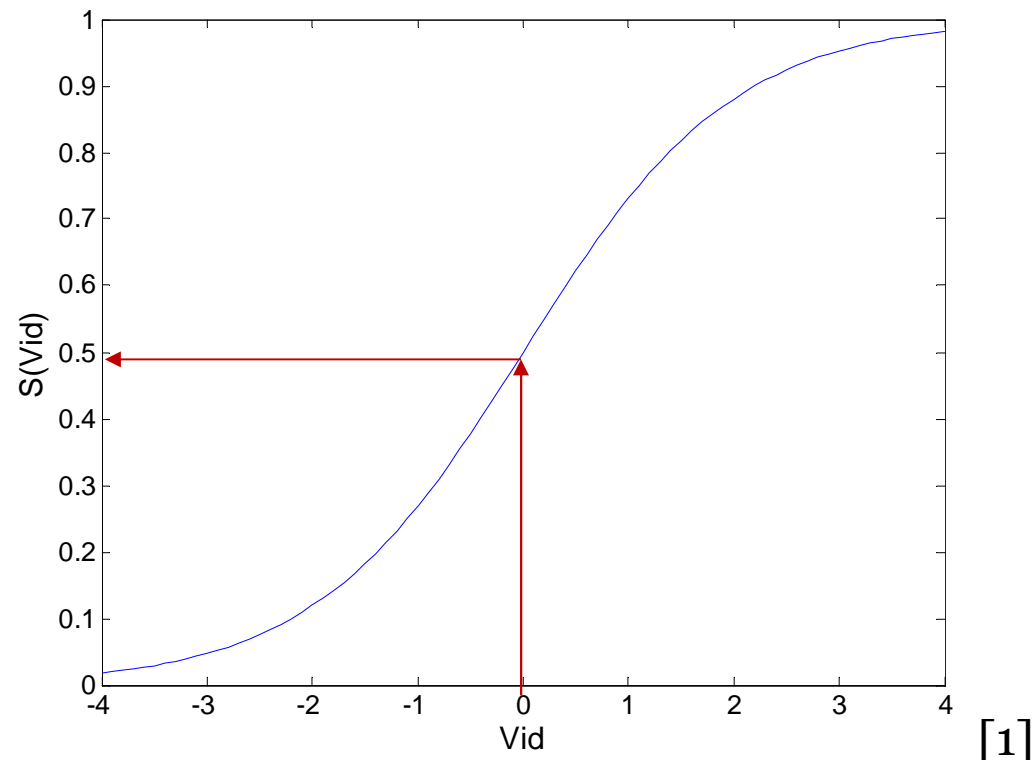
# Binary PSO

- The position update equation then becomes:

$$x_{t+1}^{id} = \begin{cases} 1 & ,if\ r < sig(v_{t+1}^{id}) \\ 0 & ,otherwise \end{cases}$$

where

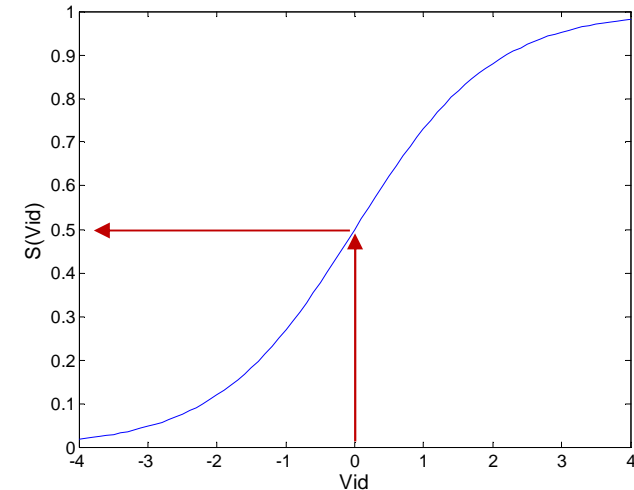
$r$  is a randomly generated number in  $[0, 1]$



# Binary PSO

- This means that:

$$prob(x_{t+1}^{id} = 1) = \begin{cases} 0.5 & , v_{t+1}^{id} = 0 \\ < 0.5 & , v_{t+1}^{id} < 0 \\ > 0.5 & , v_{t+1}^{id} > 0 \end{cases}$$



- Note that the **velocity** components could remain as **real-valued numbers** using the original equation, but fed to the **sigmoid function** before updating the **position vector**.

# Binary PSO

## ◇ Particle's velocity Update:

$$v_{t+1}^{id} = v_t^{id} + \phi_1(p_t^{id} - x_t^{id}) + \phi_2(p_t^{gd} - x_t^{id})$$

$$sig(v_{t+1}^{id}) = \frac{1}{1 + e^{-v_{t+1}^{id}}}$$

## ◇ Particle's Position Update:

$$x_{t+1}^{id} = \begin{cases} 1 & , \text{if } r_{id} < sig(v_{t+1}^{id}) \\ 0 & , \text{otherwise} \end{cases}$$

$\phi_1$  and  $\phi_2$  represent different random numbers drawn from uniform distributions. Sometimes these parameters are chosen from a uniform distribution 0-2, such that the sum of their two limits is 4.0.

# Binary PSO

$v_{t+1}^{id}$  is the **probability** that an individual  $i$  will choose 1 for the bit at the  $d^{\text{th}}$  **site** in the bitstring.

$x_t^{id}$  is the **current state** of string  $i$  at bit  $d$ .

$v_t^{id}$  is a measure of the string's **current probability** to choose 1.

$p_t^{id}$  is the **best state** found so far (to date) for bit  $d$  of individual  $i$ , i.e. a 1 or a 0.

$p_t^{gd}$  is 1 or 0 depending on what the **value of bit  $d$**  is in the best neighbour to date.

Individual  $i$

Position	1	2	...	$d$	...	$n$
State	0	1	*	$x_t^{id}$	*	1

[3]

# Binary PSO

- **Example:** As an example, let's say that we are dealing with a **population of 5 bit binary particles** and a population of **4 particles**

10101

01011

11100

01101

- We are updating **particle 2 (01011), bit 3 (0)**

[3]

# Binary PSO

- Furthermore, we will assume that the **current propensity (velocity)** of this bit to be a **1** is **0.25**.
- Furthermore, assume that the **best value** of this particle (to date) was **00100**.
- And the **best value** of the **whole population** (to date) was **0111**.



# Binary PSO

- **Given**

Particle 2: **01011**

$$v_t^{23} = 0.25 \quad x_t^{23} = 0$$

$$p_t^{23} = 1 \quad p_t^{g3} = 1$$

$$\varphi_1 = 2.5 \quad \varphi_2 = 1.7$$

# Binary PSO

- **Velocity update:**

$$v_{t+1}^{id} = v_t^{id} + \phi_1(p_t^{id} - x_t^{id}) + \phi_2(p_t^{gd} - x_t^{id})$$

$$v_{t+1}^{23} = 0.25 + 2.5(1 - 0) + 1.7(1 - 0) = 4.45$$

- **Sigmoid function:**

$$\text{sig}(v_{t+1}^{23}) = \frac{1}{1 + e^{-v_{t+1}^{id}}} = \frac{1}{1 + e^{-4.45}} = 0.988$$

- **Position update:**

$$x_{t+1}^{23} = \begin{cases} 1 & , \text{if } r_{23} < \text{sig}(v_{t+1}^{23}) \\ 0 & , \text{otherwise} \end{cases}$$

where  $r_{23}$  is a randomly generated number [0 1]

# Binary PSO

- In [2], a comparison was made between the **binary PSO** and **three version of GAs**:
  - ◇ Crossover only GA (GA\_c),
  - ◇ Mutation only GA (GA\_m),
  - ◇ A regular GA.
- The objective was to optimize **different functions** that were generated using a random function generator.

# Binary PSO

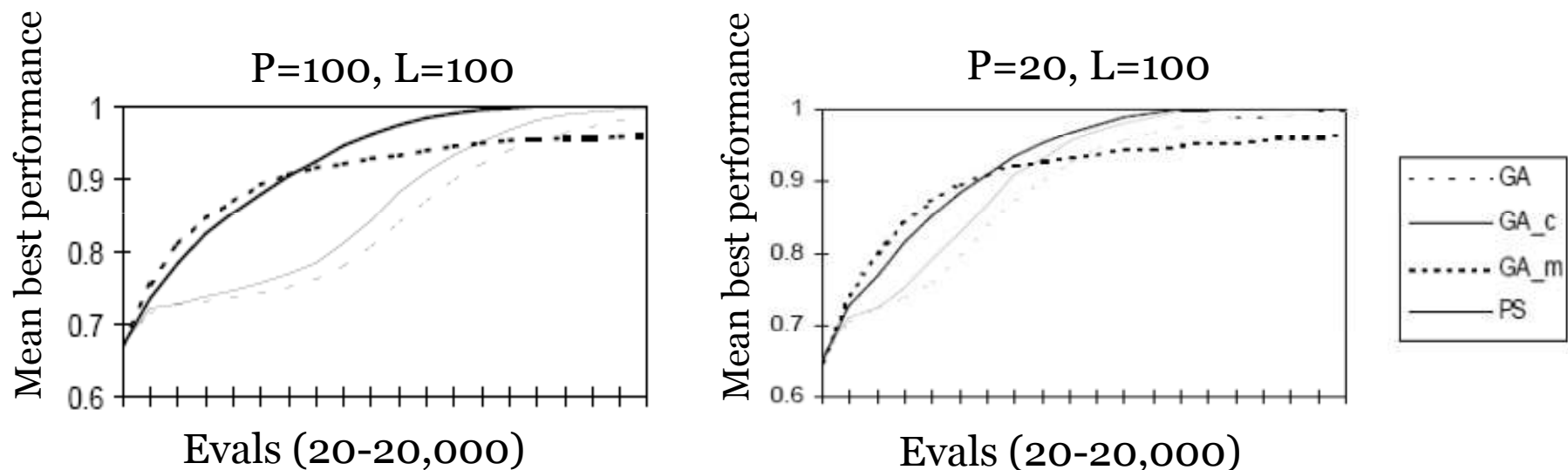
- This allows the creation of random binary problems with some specified characteristics:
  - ◇ Number of local optima, bit strings
  - ◇ Dimension, no of components (bits)
  - ◇ **Fitness function**

$$f(c) = \frac{1}{L} \max_{i=1}^P \{L - \text{Haming}(c, \text{Peak}_i)\}$$

- In that study, the **binary PSO** was the only algorithm that found the **global optimum** on every single trial, regardless of problem features.

# Binary PSO

- The experiments show that the **binary PSO progressed faster** than the other algorithms.



P is the number of peaks and L is the binary vector length

# Outline

- Discrete PSO
- Binary PSO
- **Permutation PSO**
- FPGA Placement

# Permutation PSO

- Several attempts have been made to apply **PSO** to **permutation problems**.
- The difficulty of extending PSO to such problems is that the **notions of velocity and direction** have **no natural extensions** for these problems.

## Permutation PSO

**Redefining of the arithmetic operations** such as addition, multiplication

**Space Transformation**

# Permutation PSO

- **Redefining Arithmetic Operations**
  - ◇ In [4], PSO was applied to solve the **TSP**.
  - ◇ The position of a particle was the solution to a problem (**permutation of cities**).
  - ◇ The **velocity** of a particle was defined as the **set of swaps to be performed on a particle**.



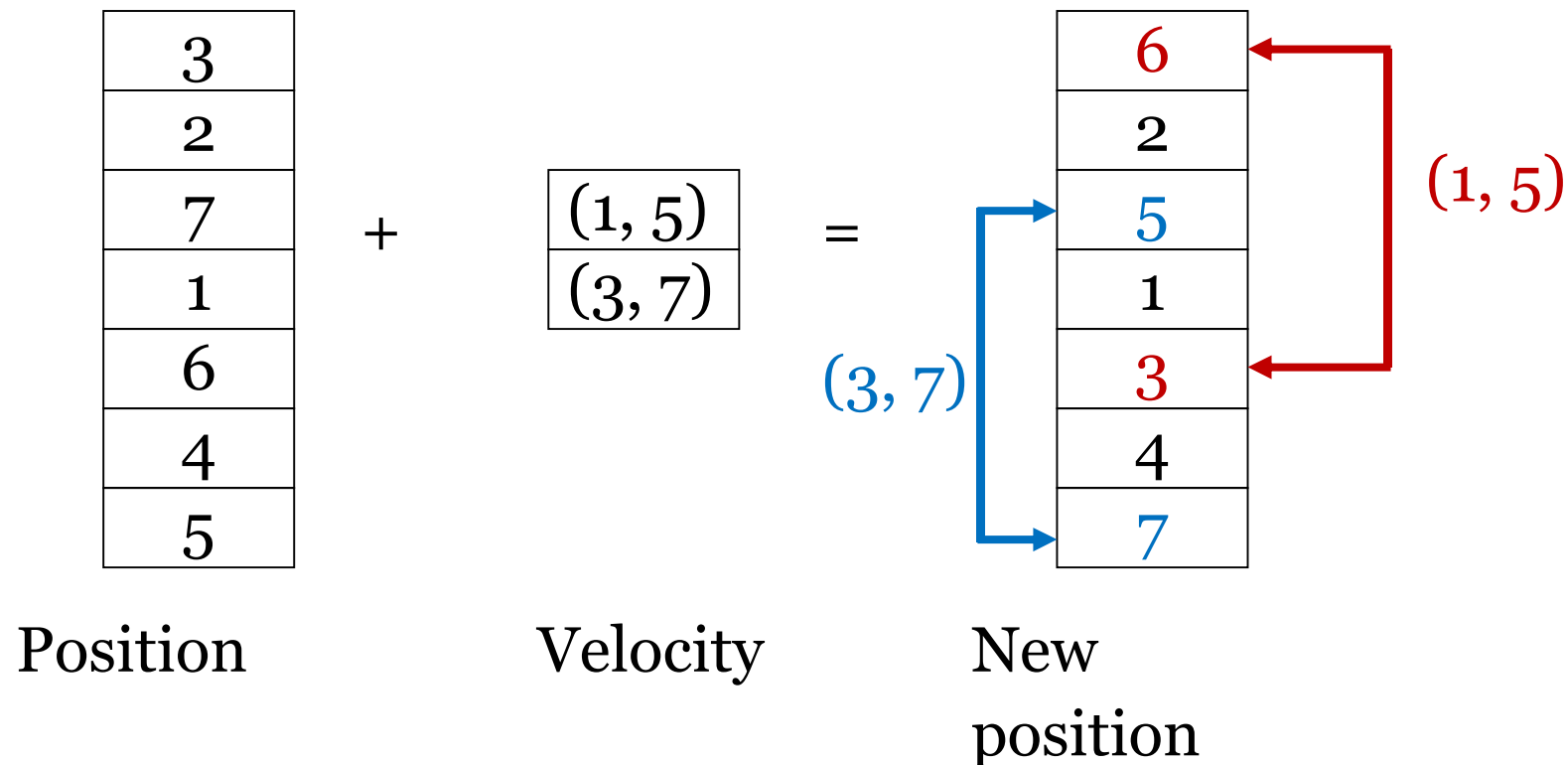
# Permutation PSO

- **Redefining Arithmetic Operations**
  - ◇ Three operations were **re-defined** for the new search space:
    - **Adding** a velocity to a position.
    - **Subtracting** two positions.
    - **Multiplying** a velocity by a constant.

# Permutation PSO

- **Adding a velocity to a position:**

the operation is performed by applying the sequence of swaps defined by the velocity to the position vector.



[1]

# Permutation PSO

- **Subtracting two positions:**
  - ◇ Subtracting two positions should produce a velocity,
  - ◇ This operation produces the sequence of swaps that could transform one position to the other.

3
2
7
1
6
4
5

-

6
2
5
1
3
4
7

=

(3, 6)
(7, 5)

Position

New position

Velocity

[1]

# Permutation PSO

- **Multiplying a velocity by a constant:**

This operation is performed by **changing** the length of the velocity vector (**number of swaps**) according to the **constant c**:

- ◇ If  $c = 0$ , the length is set to **zero**.
- ◇ If  $c < 1$ , the velocity is **truncated**.
- ◇ If  $c > 1$ , the velocity is **augmented**.

# Permutation PSO

- **Multiplying a velocity by a constant:**

- ◇ If  $c < 1$ , the velocity is **truncated**.

$$\begin{array}{|c|} \hline (1, 5) \\ \hline (3, 7) \\ \hline \end{array} \times 0.5 = \begin{array}{|c|} \hline (1, 5) \\ \hline \end{array}$$


Velocity                  Constant                  New Velocity

- ◇ If  $c > 1$ , the velocity is **augmented**.

$$\begin{array}{|c|} \hline (1, 5) \\ \hline (3, 7) \\ \hline \end{array} \times 1.5 = \begin{array}{|c|} \hline (1, 5) \\ \hline (3, 7) \\ \hline (1, 5) \\ \hline \end{array}$$

Velocity                  Constant                  New Velocity

Newly added swaps  
are taken from the  
beginning of the  
velocity vector



# Permutation PSO

- The permutation PSO was successfully applied to TSP.
- Using **16 particles** and a neighbourhood of **4 particles**, it finds the optimal solution for problem with size of **17 (br17.tsp)** using **7990 tour evaluations**.
- Using a **social neighbourhood** (close particles in the **swarm matrix**) is less expensive than using physical neighbourhood (close in the **search space**).

Swarm size	Neighbourhood size	Neighbourhood type	Logical/arithmetic operations
16	4	Social	4.8M
16	4	Physical	6.3M

[4]

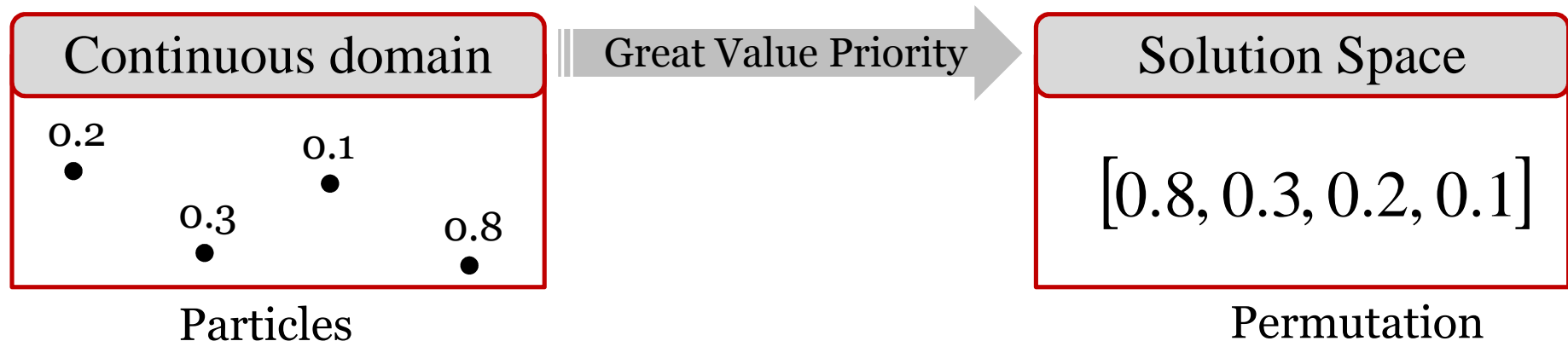
# Permutation PSO

- However, applying it to **larger problems** might be **computationally expensive** (due to the handling of **multiple solutions**).

# Permutation PSO

- **Space Transformation**

- ◇ A different PSO approach was applied to the TSP in [5].
- ◇ A **continuous PSO** version was used.
- ◇ In order to evaluate the **particle fitness**, they performed a **space transformation** from a particle in the **continuous domain** to a **permutation** in the **solution space**.





# Permutation PSO

- **Space Transformation**

- ◇ The used rule known as **Great Value Priority (GVP)**
  - First, all the **elements in the position vector** (along with their indices) were **sorted** to get a sorted list **in descending order**.
  - The **sorted indices** were taken as the **permutation**.

# Permutation PSO

- **Space Transformation**

- ◇ The idea was that if  $x_i$  had the **highest value** in the position vector, it means that **city number  $i$  comes first** in the permutation vector.
- ◇ This transformation was carried before calculating the particles fitness.

# Permutation PSO

- **Space Transformation: *Example***

if  $x = [0.2, 0.3, 0.1, 0.8]$



4    2    1    3

$$x_{sorted} = [0.8, 0.3, 0.2, 0.1]$$

and the permutation would be:

$$P = [4, 2, 1, 3] \leftarrow \text{Tour to be evaluated and its cost is the particles' x fitness}$$

# Permutation PSO

- **Space Transformation**

- ◇ The work also experimented with applying **local search to the permutation** with a certain probability to get a better one,
- ◇ The **local search** was applied by **selecting two cities to swap**,
- ◇ If a **better tour** is found, in order to go back to the continuous domain, the **same swap** is applied to the **original particle x**.

# Permutation PSO

- **Space Transformation: *Example***

if  $P = [4, 2, 1, 3]$  and  $x = [0.2, 0.3, 0.1, 0.8]$

and the **local search** produced the better tour:

$$P = [2, 4, 1, 3]$$

then the new position would be:

$$x = [0.2, 0.8, 0.1, 0.3]$$

  
by swapping the same  
elements

# Permutation PSO

- The **PSO** and **PSO\_LS** both were applied to **4 TSP** problems using:
  - 50 particles and 2000 iterations,
  - The position constrained in  $[-1, 1]$ .
  - The velocity constrained in  $[-0.1, 0.1]$ .
  - $w=1, c_1=c_2=2$ .
  - A local search probability of 1%.
  - The results are the averages taken over 10 runs.

# Permutation PSO

Instance	Optimal Solution	PSO	PSO_LS
burma14	30.8785	33.6948	<b>30.8785</b>
eil51	426	582.501	<b>459.273</b>
berlin52	7542	8100.105	<b>7938.041</b>
eil76	538	588.712	<b>564.523</b>

[1]

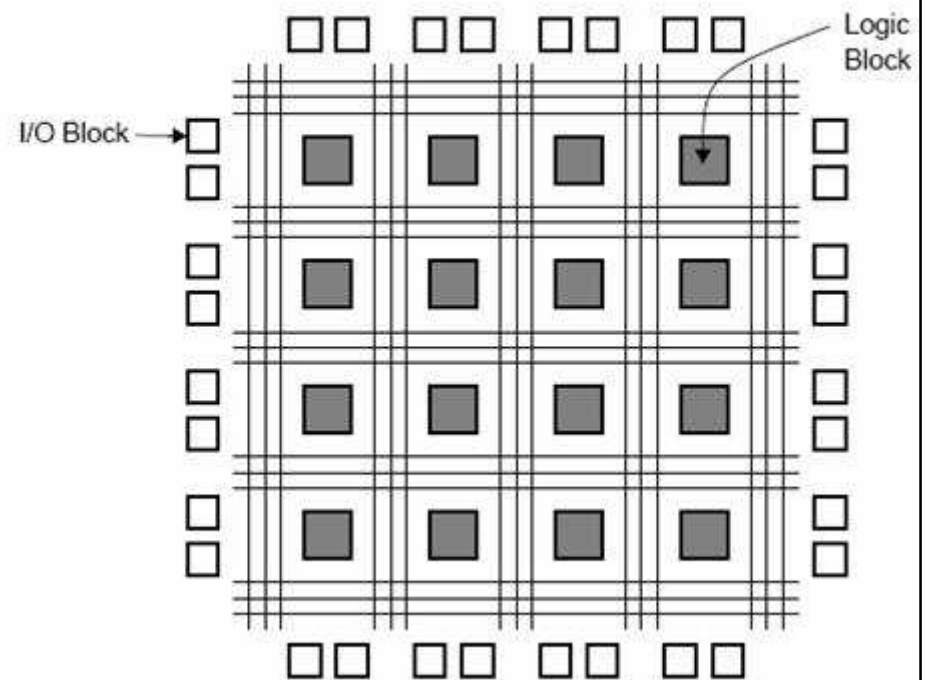
# Outline

- Discrete PSO
- Binary PSO
- Permutation PSO
- **FPGA Placement**



# FPGA Placement

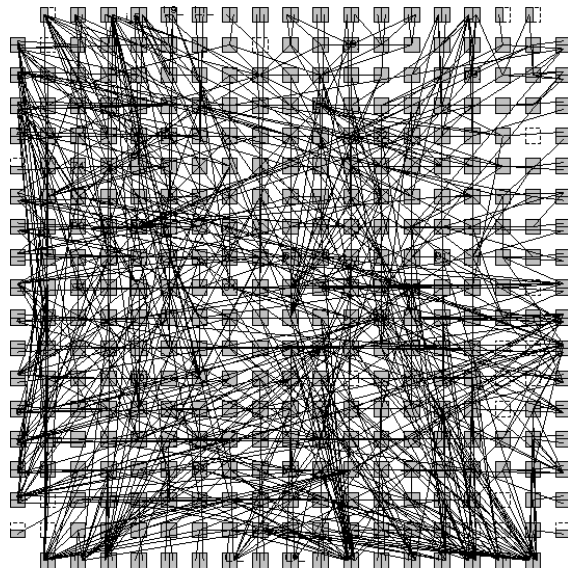
- **Field Programmable Gate Array (FPGA)**, is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence “field-programmable”.
- **Placement in FPGA** decides the physical **locations and inter connections** of each **logic block** in the circuit design, which now becomes the bottleneck of the circuit performance.



[6]

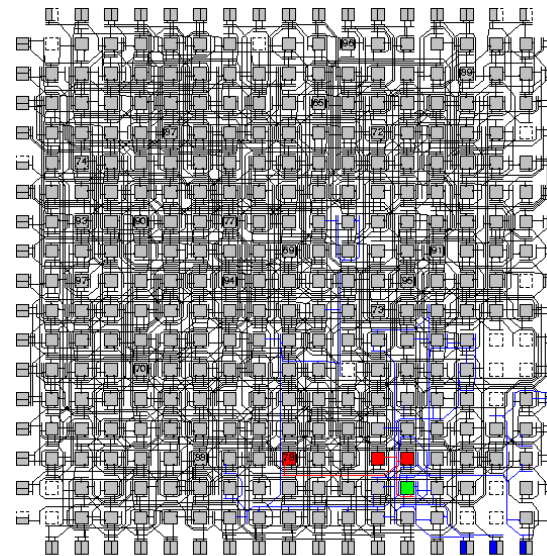
# FPGA Placement

- PSO was applied to the FPGA placement problem and compared to **Versatile Place and Route (VPR)**.  
<http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html>
- VPR is an academic tool adopting a simulated annealing approach.
- The objective was to minimize the total wire length.



Final Placement. Cost: 28.5384. Channel Factor: 100

Final  
Placement

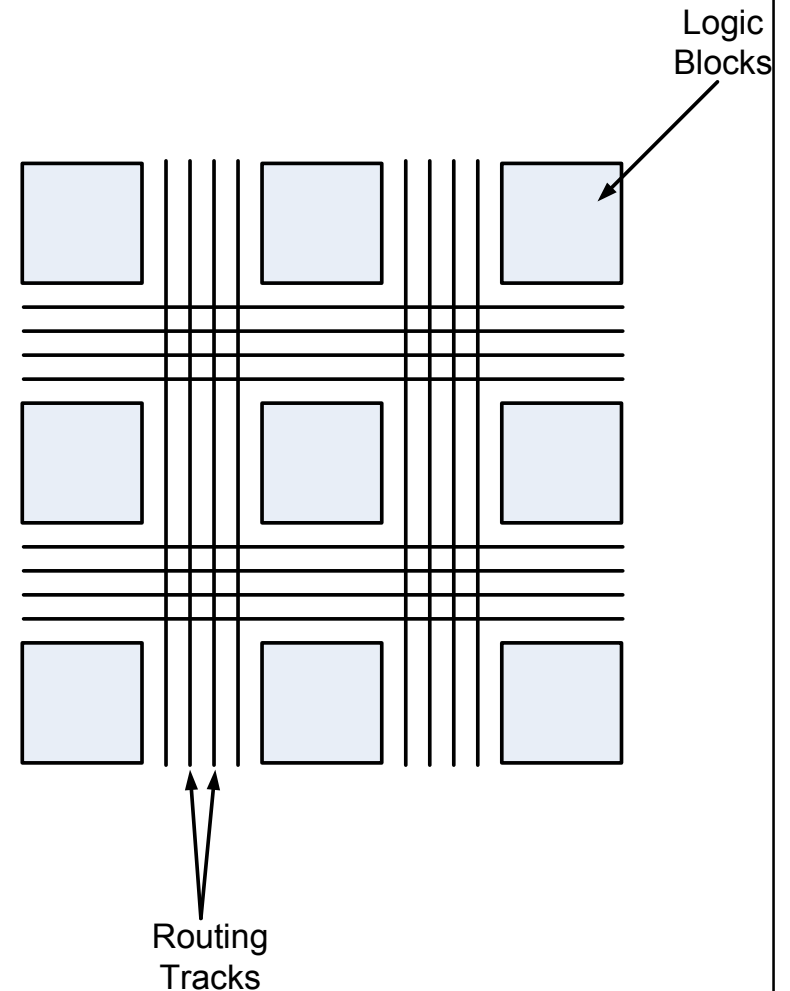


Routing succeeded with a channel width factor of 7.

Completely  
(Detailed)  
Routed Circuit

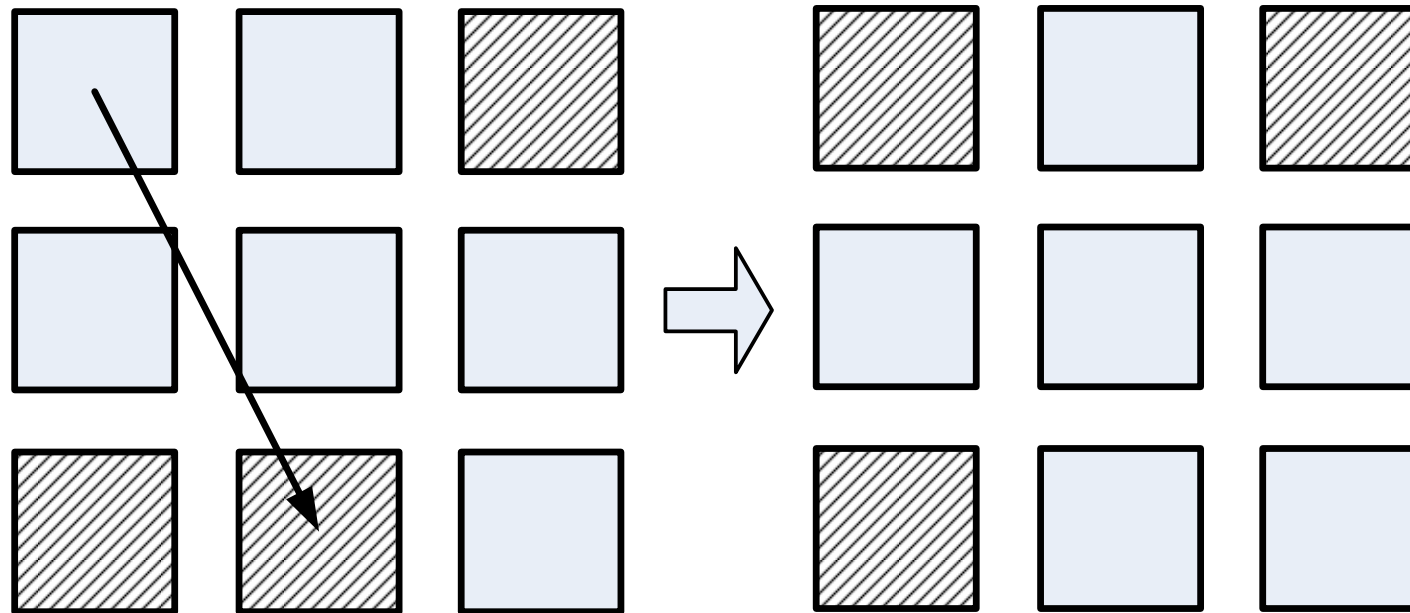
# FPGA Placement

- FPGA placement is performed in **discrete locations**.
- All logic blocks have the **same height and width**.



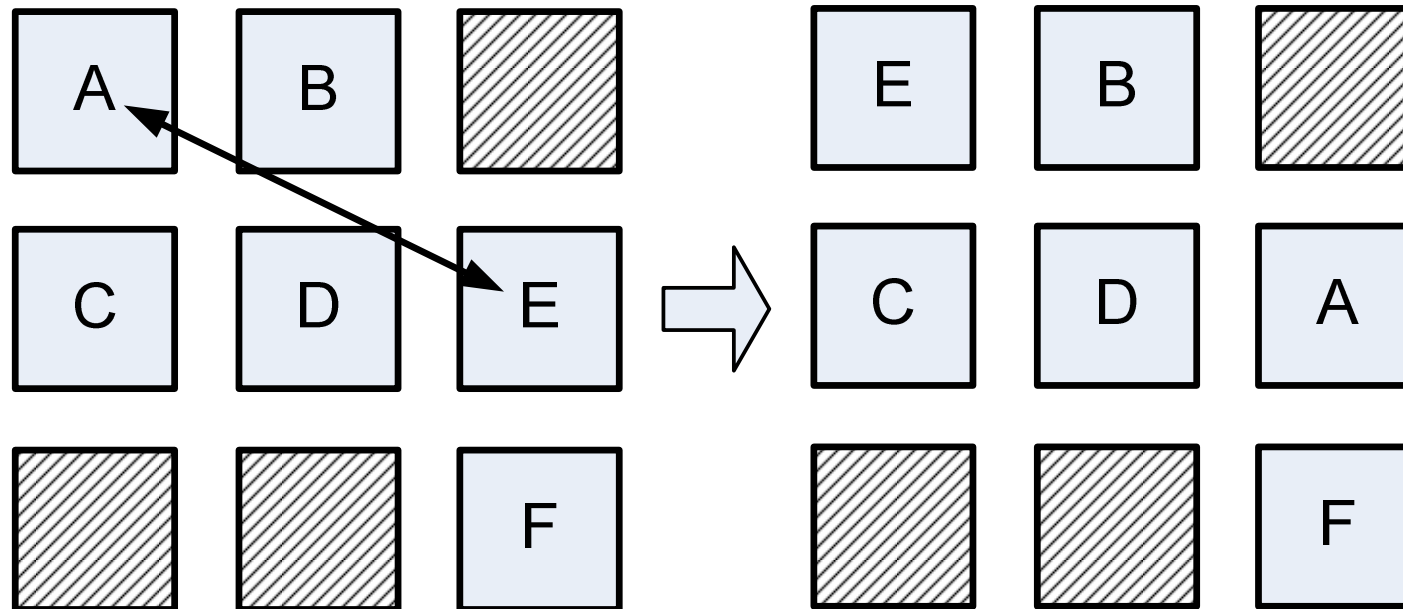
# FPGA Placement

- Move to an empty location



# FPGA Placement

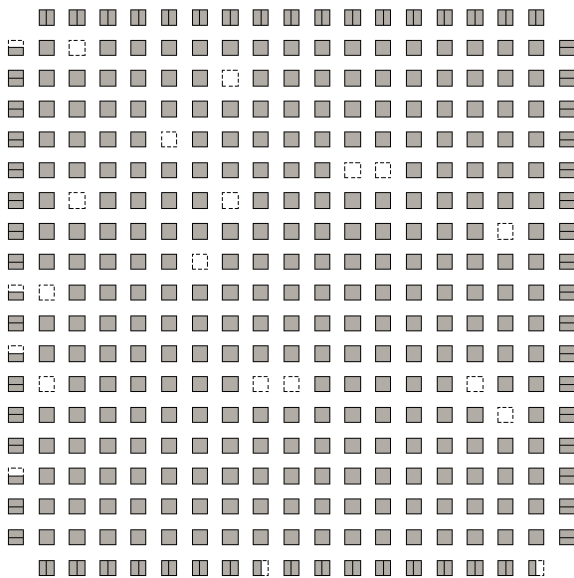
- Swap with another block



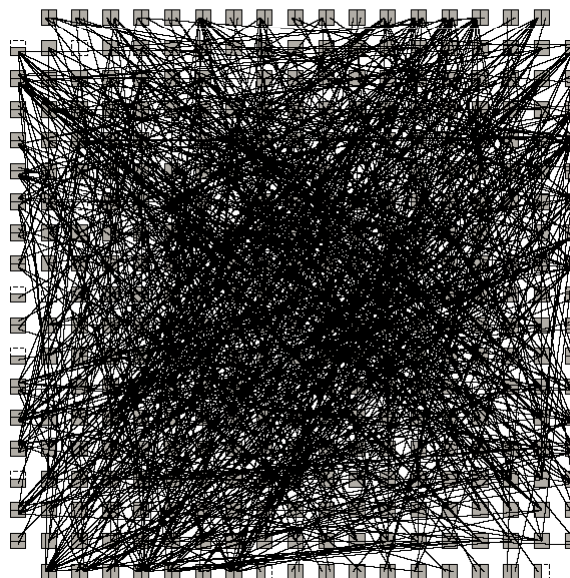
# FPGA Placement

- Placement by Versatile Place and Route (VPR) tool:
  - ◇ VPR uses SA,
  - ◇ VPR is the core of Altera's placement tools

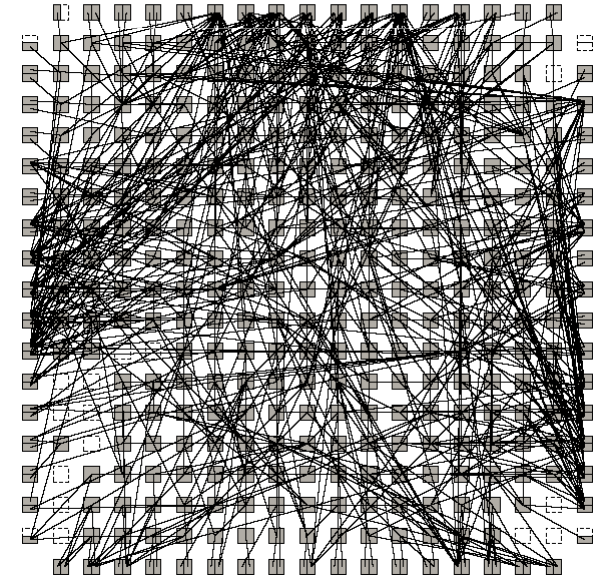
**Initial placement: wire-length=74.69**



**Initial placement: wire-length=74.69**

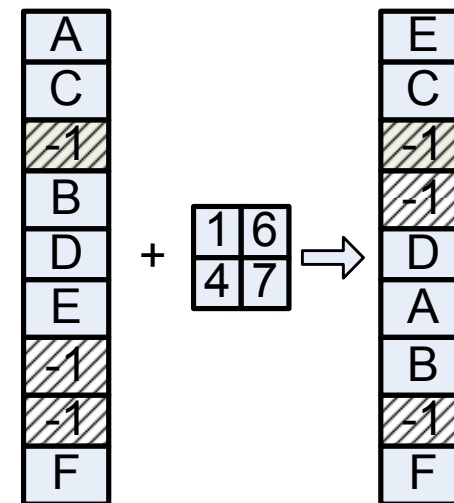
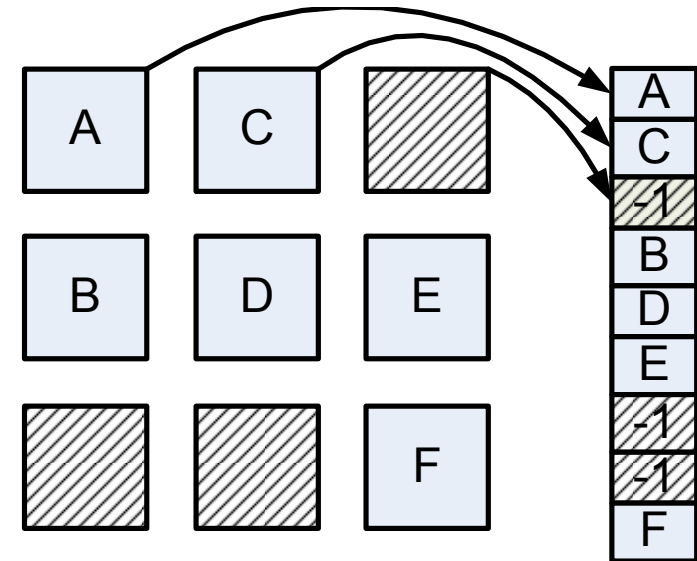


**Final placement: wire-length=30.56**



# FPGA Placement

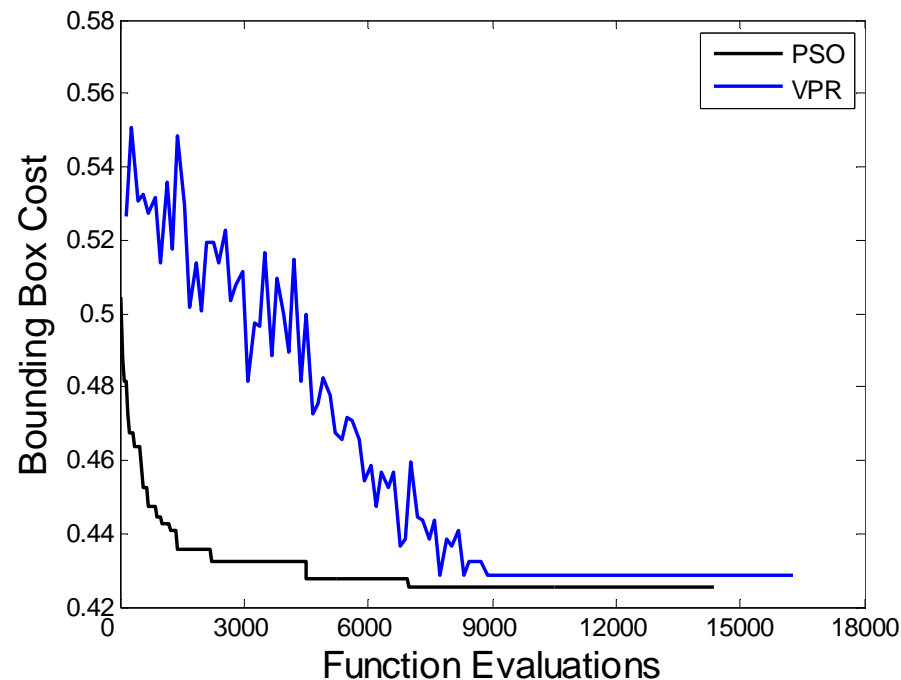
- Each particle contains information of all the available locations.
- Velocities are swap pairs



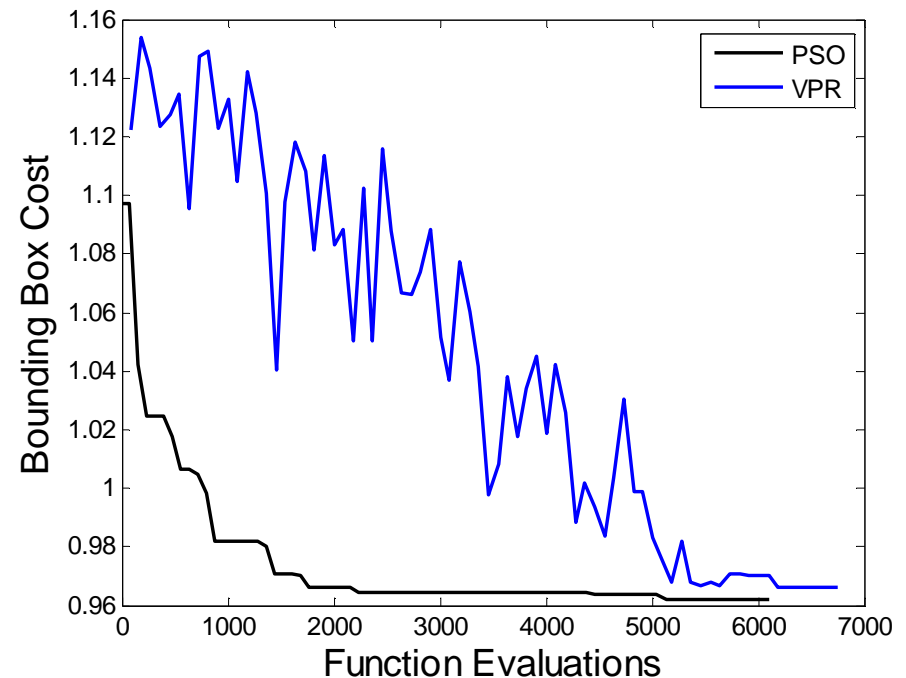
[7]

# FPGA Placement

- PSO has faster convergence than SA



**cm42a: problem size=36**

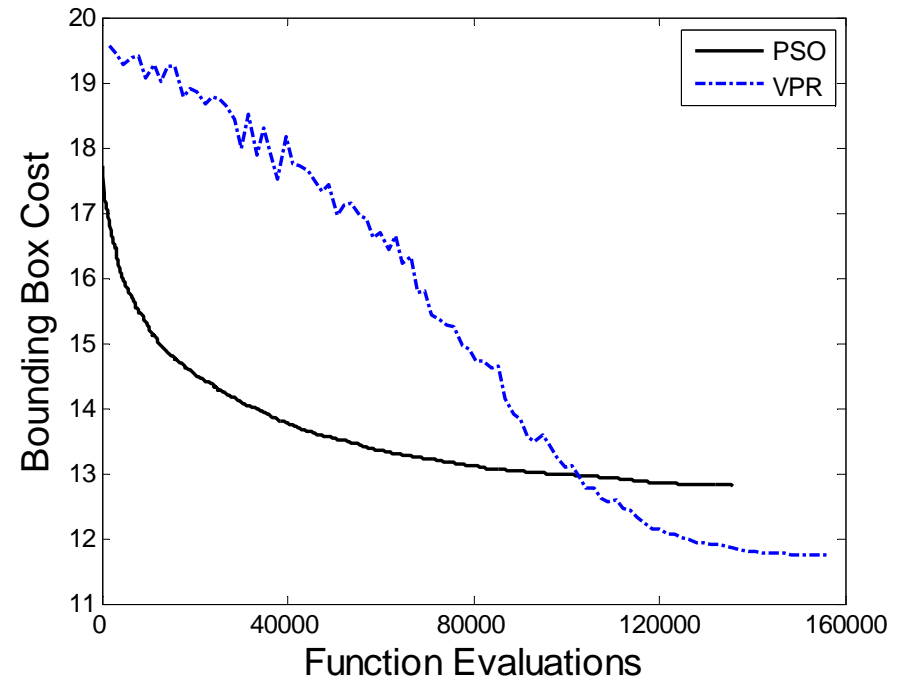


**dk27: problem size=80**



# FPGA Placement

- For problems with larger dimensionality, **PSO gets stuck in local minima**,
- PSO still has **faster** convergence rate than SA.



**s832: problem size=225**

# FPGA Placement

- PSO always starts from a better solution due to the initialization of a number of particles,
- However, it has a **higher computational cost than SA** due to the handling of a complete swarm.

# References

1. M. Kamel. ECE457A: Cooperative and Adaptive Algorithms. University of Waterloo, 2010-2011.
2. J.Kennedy and R. C. Eberhart. “A Discrete Binary Version of The Particle Swarm Algorithm”. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, pp. 4101-4109, 1997.
3. M.A. Khanesar, K. N. Toosi, M. Teshnehlab, M.A. Shoorehdeli, A novel binary particle swarm optimization", editerranean Conference on Control & Automation, 2007.
4. M. Clerc, “Discrete Particle Swarm Optimization,” in New Optimization Techniques in Engineering. Springer-Verlag, 2004.
5. W. Pang, K. Wang, C. Zhou, L. Dong, M. Liu, H. Zhang and J. Wang. “Modified Particle Swarm Optimization Based On Space Transformation for Solving Travelling Salesman Problem”. Proceedings of the third international conference on Machine Learning and Cybernetics, pp. 26-29, 2004.
6. El-Abd, M., Hassan, H., Anis, M., Kamel, M. and El-Masry, M. I., “Discrete Cooperative Particle Swarm Optimization for FPGA Placement”, Applied Soft Computing, Vol. 10, No. 1, pp. 284-295, January 2010.
7. Hassan, H, VLSI Placement, ECE493T8 Presentation, University of Waterloo, 2009.