

What is Local Search?

Informed Search Strategies such as A* search and the Greedy Best Search algorithm are particularly useful for finding optimal paths to the goal state using heuristic.

On the other hand, some problems focus solely on **generating a goal state or a good state**, regardless of the specific actions taken. Local search addresses **optimization tasks (maximization or minimization)**, focusing on improving a measure rather than finding a path. It is used to evaluate large search spaces for a good solution but it is not guaranteed to find the absolute best solution; **it attempts to find the global best solution while avoiding local best solution.**

Local search begins with an initial setup and makes incremental adjustments to the solution, exploring nearby possibilities, known as the "neighborhood," for better solutions.

What is a Genetic Algorithm?

Genetic algorithms are a class of optimization algorithms inspired by the principles of natural evolution.

It copies how nature works:

- Survival of the fittest (organisms best adjusted to their environment are the most successful in surviving and reproducing)
- **Selection** (choose the fitter individuals to reproduce)
- **Crossover** (reproduction of the selected individuals to create offsprings with good traits)
- **Mutation** (alter genes in the offsprings to create diversity within the population)

Algorithm:

Algorithm 5 Genetic Algorithm

```
1: Input: Population size, fitness function, mutation rate, crossover rate, maximum generations
2: Output: Best solution found
3: begin
4:   Initialize population with random candidates
5:   Evaluate the fitness of each candidate
6:   while termination condition not met do
7:     Select parents from the current population
8:     Perform crossover on parents to create new offspring
9:     Perform mutation on offspring
10:    Evaluate the fitness of new offspring
11:    Select individuals for the next generation
12:    best solution in new generation > best solution so far
13:    Update best solution found
14:  end while
15:  return best solution found
16: end
```

Lab Task:

1. Problem Overview

You are simulating a simplified version of VLSI floorplanning within a chip circuit

Objective: Determine the **most optimal placement of 6 common chip** components on a 2D chip grid, **minimizing both wire length** (that interconnects certain components) and the **overall required chip area**, while **avoiding overlapping chip components**

2. Grid and Components

Suppose the chip grid is of dimension **25 x 25** unit square

Say, the chip contains the following 6 functional blocks, each with a fixed size listed below:

Block Name	Width (unit)	Height (unit)
ALU	5	5
Cache	7	4
Control Unit	4	4
Register File	6	6
Decoder	5	3
Floating Unit	5	5

3. Required Interconnections

One of the three goals of your layout design is to **minimize** the wiring distance between the following 6 connected component pairs:

Register File → ALU

Control Unit → ALU

ALU → Cache

Register File → Floating Unit

Cache → Decoder

Decoder → Floating Unit

You can assume all wiring connections are bidirectional, incur no additional space, and wires run **center-to-center** between components. Generally, the more compact the chip component placements are, the shorter the total wiring length becomes, and vice versa.

4. Fitness Objectives

Wiring Distance Between Two Chip Components

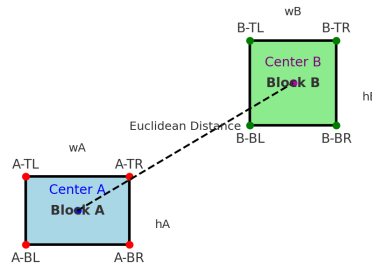


Figure 1

1. **Minimizing the total wiring distance:** Figure 1 above explains how the center-to-center wiring length can be calculated when both the blocks' height, width (w_A , h_A , w_B , h_B), and any single pair of corner points are given [suppose, bottom left of A (A-BL) and bottom left of B (B-BL) are given]. This is essentially the Euclidean Distance between two geometric points (two centers) in a 2D space. The **less total** wiring distance needed, the **more desirable** the layout is.

If the bottom-left coordinates, X and Y , are given for each block A and B,

<p>For Block A:</p> $X_1 = X_A + W_A/2$ $Y_1 = Y_A + H_A/2$	<p>For Block B:</p> $X_2 = X_B + W_B/2$ $Y_2 = Y_B + H_B/2$
<p>Euclidean Distance:</p> $D = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$	

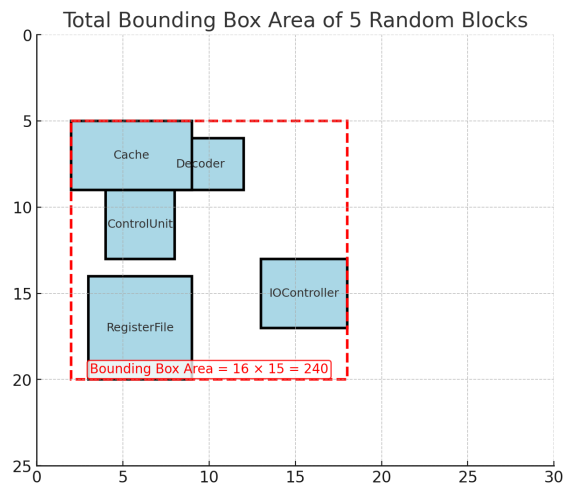


Figure 2

2. **Minimizing the total bounding area:** Figure 2 above hints at how the total bounding area can be calculated for a random placement of 5 component blocks. The red dashed rectangle is the bounding box- aka the smallest rectangle that contains all of the blocks inside. Given the bottom left coordinates (x,y) for all block components, you can easily find out the minimum and maximum of all x and y values, which will help you to calculate the area of such a bounding box $[(x_{\max} - x_{\min}) * (y_{\max} - y_{\min})]$. The **smaller** the bounding box area is, the **better** the layout is.

If the bottom-left coordinates, X and Y, are given for each block,

<p>For X_{\max} and X_{\min}:</p> $X_{\text{bottom-right}} = X + W$ $X_{\max} = \max \text{ of all } X_{\text{bottom-right}}$ $X_{\min} = \min \text{ of all } X$	<p>For Y_{\max} and Y_{\min}:</p> $Y_{\text{top-left}} = Y + H$ $Y_{\max} = \max \text{ of all } Y_{\text{top-left}}$ $Y_{\min} = \min \text{ of all } Y$
<p>Area of Bounding Box:</p> $A = (X_{\max} - X_{\min}) * (Y_{\max} - Y_{\min})$	

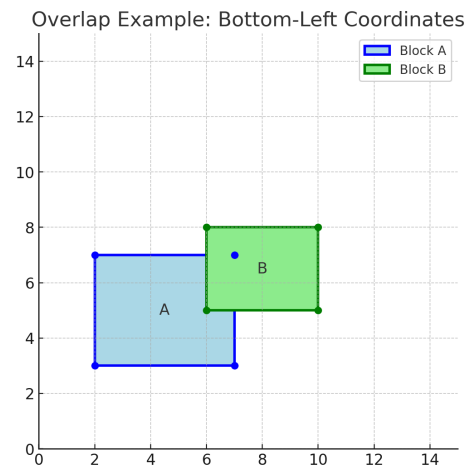


Figure 3

3. **Minimizing total component overlaps:** In Figure 2, there exists an overlap between the two components (cache and decoder). A single block can, in fact, be overlapped with multiple other blocks in a random placement. **Overlaps are not desirable.** Hence, a heavy penalty will be applied for each overlapping block pair. For any two block pairs A and B (Figure 3), one possible way to figure out the overlap is by checking the following condition:

$$\text{overlap} = \text{not} (\\
\begin{aligned}
&A_{\text{right}} \leq B_{\text{left}} \text{ or} \\
&A_{\text{left}} \geq B_{\text{right}} \text{ or} \\
&A_{\text{bottom}} \geq B_{\text{top}} \text{ or} \\
&A_{\text{top}} \leq B_{\text{bottom}}
\end{aligned}
)$$

where right, left, and top, bottom can be derived from the bottom-left x,y coordinate values of both blocks and the width and height of the corresponding blocks, respectively. The idea is that no overlap exists if one rectangle is completely to the left, right, above, **or** below the other.

If the bottom-left coordinates, X and Y, are given for each block A and B,

<p>For Block A:</p> $ \begin{aligned} A_{\text{left}} &= X_A \\ A_{\text{right}} &= X_A + W \\ A_{\text{top}} &= Y_A + H \\ A_{\text{bottom}} &= Y_A \end{aligned} $	<p>For Block B:</p> $ \begin{aligned} B_{\text{left}} &= X_B \\ B_{\text{right}} &= X_B + W \\ B_{\text{top}} &= Y_B + H \\ B_{\text{bottom}} &= Y_B \end{aligned} $
<p>Overlap if:</p> $(A_{\text{left}} < B_{\text{right}}) \& (A_{\text{right}} > B_{\text{left}}) \& (A_{\text{bottom}} < B_{\text{top}}) \& (A_{\text{top}} > B_{\text{bottom}})$	

Task 1 Instructions

Start with a random initial population of 6 members, where each chromosome represents a candidate layout. Suppose the layout is specified by all the **bottom-left coordinates** of the 6 components (outlined in section 2 in that exact order). Check section 7 for a sample input analysis.

1. Chromosome Representation / Encoding:

- Devise a **suitable encoding scheme** to represent all the block components specified in the question. Note that the placement coordinate values will be in the range of [0, 25]. Be as concise as possible in your encoding; that is, try not to keep redundant information in your encoded version.
- Then generate an initial population of 6 chromosomes to start the GA loop, where each chromosome represents a placement strategy.

eg.

P1 → (9,3), (12, 15), (13, 16), (1,13), (4,15), (9, 6)

P2 → (8, 0), (7,12), (4,11), (1,13), (14,10), (9,11)

P3 → (6, 5), (12, 9), (9, 7), (8, 6), (2, 7), (3, 1)

P4 → (3,11), (11, 12), (14, 11), (6, 10), (3,11), (3,0)

P5 → (10, 12) (8, 16), (10, 4), (13, 6), (6, 0), (3, 7)

P6 → (0, 2), (0, 0), (14, 12), (4, 5), (12, 4), (3, 10)

2. Fitness Function Implementation:

- Based on the 3 fitness objectives outlined above, design a **suitable fitness function** that focuses on penalizing bad placement combinations. Implementation hints are outlined in detail in section 4. Follow a prioritized order of these objectives as mentioned below, and reflect that in your fitness calculation.
 - Overlap counts should be considered the least desirable, hence penalized way more than the other two
 - The total wiring distance and the total bounding area should be considered next
- Calculate the fitness of the chromosome pool using the hints specified above. A **weighted sum** can be a good fitness function to start with. Check the sample input output section for some ideas to start with.

eg.

$$Total\ Fitness\ Value = - (\alpha * P_{overlap} + \beta * P_{wiring} + \gamma * P_{area})$$

Consider, $\alpha = 1000$, $\beta = 2$, $\gamma = 1$

$P_{overlap}$ = number of overlaps among all pairs of layouts/chromosomes in a population

P_{wiring} = the total wiring distance of the population, $D_1 + D_2 + D_3 + D_4 + D_5 + D_6$

P_{area} = the total bounding area of the population, A

Chromosome	Overlap	Wiring	Area	Fitness
P1	3	67.1	306	-3440.23
P2	3	57.7	342	-3457.40
P3	7	37.8	204	-7279.63
P4	5	53.3	240	-5346.51
P5	2	54.5	320	-2429.00
P6	3	52.6	288	-3393.81

3. Parent Selection:

- Choose two parents based on **random selection** at each generation. Tournament selection, Roulette Wheel selection are some of the other popular selection techniques.

4. Cross-over:

- Perform **single-point crossover** to create 2 new offspring from each pair of selected parents.
- To achieve this, pick a random split point from the chromosome representation of the parents and swap the complementary pieces to generate offspring.

eg.

Let the random 2 parents be:

P1 → (9,3), (12, 15), (13, 16), (1,13), (4,15), (9, 6) [Fitness = -3440.23]

P2 → (8, 0), (7,12), (4,11), (1,13), (14,10), (9,11) [Fitness = -3457.40]

For single-point crossover: Select a random split, let $k = 3$

P1 → (9,3), (12, 15), (13, 16), | (1,13), (4,15), (9, 6)

P2 → (8, 0), (7,12), (4,11), | (1,13), (14,10), (9,11)

After crossover:

C1 → (9,3), (12, 15), (13, 16), (1,13), (14,10), (9,11) [Fitness = -2419.88]

C2 → (8, 0), (7,12), (4,11), (1,13), (4,15), (9, 6) [Fitness = -4372.33]

For two-point crossover: Select two random splits, let $k_1 = 2$ and $k_2 = 4$

P1 → (9,3), (12, 15), | (13, 16), (1,13), | (4,15), (9, 6)

P2 → (8, 0), (7,12), | (4,11), (1,13), | (14,10), (9,11)

After crossover:

C1 → (9,3), (12, 15), (4,11), (1,13), (4,15), (9, 6) [Fitness = -3419.68]

C2 → (8, 0), (7,12), (13, 16), (1,13), (14,10), (9,11) [Fitness = -2371.53]

5. Mutation:

- Mutation ensures genetic diversity and prevents the algorithm from getting stuck in local optima.
- Implement the following mutation strategy to introduce random changes: pick any arbitrary component block from the chromosome representation and introduce **random coordinate (i.e., the value of x and y) changes** to that single particular block with a low probability (**5-10% mutation rate**).

eg.

Choose the offsprings and mutate the gene blocks with 5-10% mutation rate.

Let's say,

Choose **C2** → (8, 0), (7,12), (4,11), (1,13), (4,15), (9, 6)

Pick a block/gene from C2, (4,15)

Replace the values with random values within the range, (4,15) → (0,0)

Mutated C2 → (8, 0), (7,12), (4,11), (1,13), (0,0), (9, 6) [Fitness = -2415.51]

6. New Generation Creation:

- a. Remember, the population size should be the same for all generations. Apply **elitism** to select new individuals (allow 1 or 2 elites to be carried forward to the next generation). Select the remaining candidates from the created offspring for the next generation.

eg.

Choose 1 or 2 population from initial population list with highest fitness value

Choose the rest from the new offsprings

7. GA Loop:

- a. Run genetic algorithms on such a population until the **best fitness** (a plateau) is achieved or the maximum number of **15 iterations** is reached. Keep this generation count as a variable input for further experiments.

8. Required Output:

- a. Output the best possible placement strategy found once the algorithm halts. Deliverables include the **best total fitness value**, the corresponding **total wiring length**, **the total bounding box area**, and **the total overlap counts**. Also, decode the best chromosome representation to reflect the optimal placement of bottom-left coordinates.

Algorithm:

Algorithm 1: GeneticAlgorithm_Floorplanning_Task1

Input: popSize = 6;
mutationRate $\in [0.05, 0.10]$;
 $\alpha = 1000$, $\beta = 2$, $\gamma = 1$;
maxGenerations = 15.
Output: Best placement solution found.
Population \leftarrow InitializeRandomPopulation(popSize)
foreach chromosome G in Population **do**
 $G.fitness \leftarrow$ Fitness(G)
generation $\leftarrow 0$
bestSolution \leftarrow BestIndividual(Population)
while generation < maxGenerations **do**
 NewPopulation \leftarrow TopElite(Population, 1)
 while size(NewPopulation) < popSize **do**
 ($parent1, parent2$) \leftarrow RandomSelect(Population)
 ($child1, child2$) \leftarrow SinglePointCrossover($parent1, parent2$)
 $child1 \leftarrow$ Mutate($child1$, mutationRate)
 $child2 \leftarrow$ Mutate($child2$, mutationRate)
 $child1.fitness \leftarrow$ Fitness($child1$)
 $child2.fitness \leftarrow$ Fitness($child2$)
 if size(NewPopulation) < popSize **then**
 add $child1$ to NewPopulation
 if size(NewPopulation) < popSize **then**
 add $child2$ to NewPopulation
 Population \leftarrow NewPopulation
 currentBest \leftarrow BestIndividual(Population)
 if currentBest.fitness > bestSolution.fitness **then**
 bestSolution \leftarrow currentBest
 generation \leftarrow generation + 1
return bestSolution

Task 2

Take a look at **Step 4 of Task 1**. Can you incorporate the **Two-point crossover mechanism** into it?

- To implement this task, randomly select two parents from your initial population. Then perform a two-point crossover to generate two children. The two points have to be chosen randomly, but it has to be ensured that the second point always comes after the first point.

Sample Input and Output

A sample initial population set of 6 chromosomes, where each line below represents the bottom-left position coordinate of 6 chip component blocks:

P1 → (9,3), (12, 15), (13, 16), (1,13), (4,15), (9, 6)

P2 → (8, 0), (7,12), (4,11), (1,13), (14,10), (9,11)

P3 → (6, 5), (12, 9), (9, 7), (8, 6), (2, 7), (3, 1)

P4 → (3,11), (11, 12), (14, 11), (6, 10), (3,11), (3,0)

P5 → (10, 12) (8, 16), (10, 4), (13, 6), (6, 0), (3, 7)

P6 → (0, 2), (0, 0), (14, 12), (4, 5), (12, 4), (3, 10)

For **P1**,

Pairwise block overlap count = 3

Total wiring distance (center-to-center) of the specified connected pairs = 12.91 + 12.98 + 12.18 + 10.61 + 9.01 + 9.43 = 67.1

Total bounding box area = (x_max - x_min) * (y_max - y_min) = (19-1) * (20-3) = 306

Total fitness value = - (alpha * overlap penalty + beta * wiring length penalty + gamma * bounding area penalty)

Considering, alpha = 1000, beta = 2, gamma = 1:

Fitness for generation 1 chromosome 1 = - (1000 * 3) - (67.1 * 2) - 306 = -3440.23