Resolviendo el Strip Packing Problem con Algoritmos Genéticos

Sebastián Luis Riccardo

21 de noviembre de 2021

Resumen

El Strip packing problem (Empaquetado en tiras) es un problema de minimización geométrica bidimensional, donde rectángulos alineados se van acomodando uno al lado del otro dentro de un *strip* (tira) siendo el ancho del strip (tira) la única restricción. En el siguiente reporte técnico se detallará la arquitectura del *algoritmo genético* utilizada para la búsqueda de la solución. También se presentarán los resultados obtenidos de la aplicación de dicho algoritmo en sus dos variantes.

1. Introducción

La optimización de problemas está presente en muchos aspectos de la vida cotidiana, que pueden ir desde la optimización en procesos de producción, optimización de redes de comunicaciones, predecir la estructura 3D de una proteína para optimizar su energía potencial y el empaquetado en tiras, entre otros. Estos problemas son naturalmente complejos y por ende difíciles de resolver, es por esto mismo que se utilizan métodos de aproximación para encontrar una solución en un tiempo razonable.

Segun el autor *El-Ghazali Talbi*. "*Las metaheurísticas resuelven instancias de problemas que son considerados por lo general difíciles, explorando el espacio de búsqueda de soluciones generalmente grandes de estas instancias*"[1].

La metaheurística aplicada al problema abordado es un algoritmo evolutivo que pertenece a una clase de metaheurísticas llamadas *Metaheurísticas Basadas en Población* o *Population-Based Metaheuristics* [1]. En estas metaheurísticas se inicia el proceso de búsqueda con una población inicial y de forma iterativa se va generando una nueva población que remplazará a la anterior. El proceso termina una vez alcanzada una condición de frenado.

En el reporte encontrará las siguientes secciones:

- **Sección 1**. Introducción
- **Sección 2**. Descripción del Problema

- **Sección 3**. Propuesta Algorítmica
- **Sección 4**. Resultados
- **Sección 5**. Discusión
- **Sección 6**. Conclusiones
- Referencias
- Repositorio
- Apéndice

2. Descripción del Problema

El problema del Strip packing problem (SPP) es un problema de minimización geométrica bidimensional. Dado un conjunto de rectángulos con ancho $w_i \in \mathbf{R}$ y altura $h_i \in \mathbf{R}$, se van acomodando de izquierda a derecha en un strip de ancho $W \in \mathbf{R}$, donde $w_i \leq W$ para todo w_i. Se debe determinar el empaquetado que **minimice** la altura total, sin superponer las superficies de los rectángulos. La altura del *strip* para esta version del problema es considerada infinita.

En una de las variantes del problema los rectángulos pueden rotar 90° grados. Respecto a como se empaquetan los rectángulos, estos se van colocando uno al lado del otro y una vez que el próximo rectángulo no tiene espacio suficiente se corta el strip *tipo guillotina* a la altura del rectángulo más alto y desde allí se comienza a colocar los rectángulos restantes.

3. Propuesta Algorítmica

Las metaheurísticas basadas en población (Population-Based Metaheuristics) presentan dentro de ellas una clase de algoritmos llamados *Algoritmos evolutivos o EA* los cuales tienen sus bases y se afirman en el concepto de la evolución presentado por C.Darwin en su libro *On the Origin of Species* [1].

Los Algoritmos evolutivos son P-metaheurísticas estocásticas que de forma iterativa simulan la evolución de las especies basándose en la competencia. A grandes rasgos un *algoritmo evolutivo* comienza con una población de inicial que va evolucionando. Esta población generalmente es creada de forma aleatoria [1]. Cada individuo de la población es una codificación de una posible solución al problema, estos individuos tienen asociados una función objetivo y esta es la función que se busca optimizar aplicando el algoritmo. En este problema se utilizó una clase particular de algoritmos evolutivos llamados *Algoritmos Genéticos*, los cuales son una de las clases más populares de algoritmos evolutivos.

Los Algoritmos Genéticos (*Genetic Algorithms o GAs*), poseen la misma línea de funcionamiento que los algoritmos evolutivos, pero en el caso de los

GA se aplica una función de *crossover* entre dos individuos seleccionados de la población por medio de alguna función de *selección*, también se aplica una *mutación* que modifica los *genes* de algún individuo de forma aleatoria para introducir diversidad en la población. *Los GA usan selección probabilística que es originalmente la selección proporcional* [1]. A medida que van pasando las generaciones se van remplazando los *"padres"* por sus *"descendientes"* y tanto las funciones de crossover como de mutación usan una probabilidad fija p_c y p_m respectivamente para simular la frecuencia con la cual son aplicadas. En las siguientes subsecciones se van a detallar las dos modalidades (con posibilidad rotación de 90° en los rectángulos y sin rotación) de funcionamiento del algoritmo genético junto a una explicación de su arquitectura.

- Implementación de individuos
- Selección
- Crossover
- Mutación
- Función de fitness

3.1. Implementación de individuos

A continuación se van a detallar las dos formas en las que se implementaron los individuos.

3.1.1. GAnr

Para la primera modalidad del algoritmo los rectángulos no tienen la posibilidad de rotar, por lo tanto, para representar una posible solución S de tamaño n simplemente se utiliza un arreglo de n posiciones por ejemplo, si $n = 9, S = [4, 7, 2, 1, 3, 0, 5, 6, 8]$.

Cada elemento del arreglo es un *gen* del individuo y representan los subindices de los rectángulos R_i, en ambas modalidades los rectángulos se enumeran desde el 0 al $n-1$ para mayor facilidad en la implementación. El arreglo completo es el *cromosoma* del individuo y define la disposición de los rectángulos en el strip. Gráficamente se ve representado de la siguiente manera:

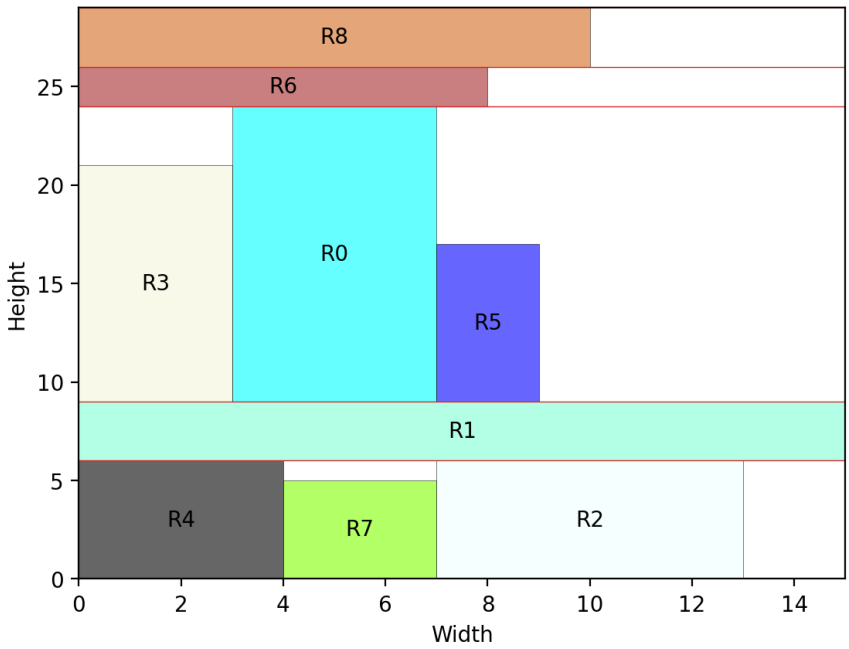


Figura 1: Representación gráfica del individuo con genes $S = [4, 7, 2, 1, 3, 0, 5, 6, 8]$

En el **apéndice A** se encuentra un fragmento de código de la implementación de un individuo.

3.1.2. GAr

La otra modalidad de funcionamiento del algoritmo permite la rotación de 90° de cada uno de los rectángulos. Para representar la solución S de tamaño n se utiliza nuevamente un arreglo de n posiciones, por ejemplo: $n = 15, S = [4, 8, 9, 3, 5, 1, 10, 11, 7, 6, 0, 12, 13, 2, 14]$ y para representar las rotaciones se utiliza otro arreglo $R = [0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0]$. El número 1 en la posición $i - esima$ indica que el rectángulo R_i rota 90° grados, por lo tanto utilizamos su altura como ancho y viceversa.

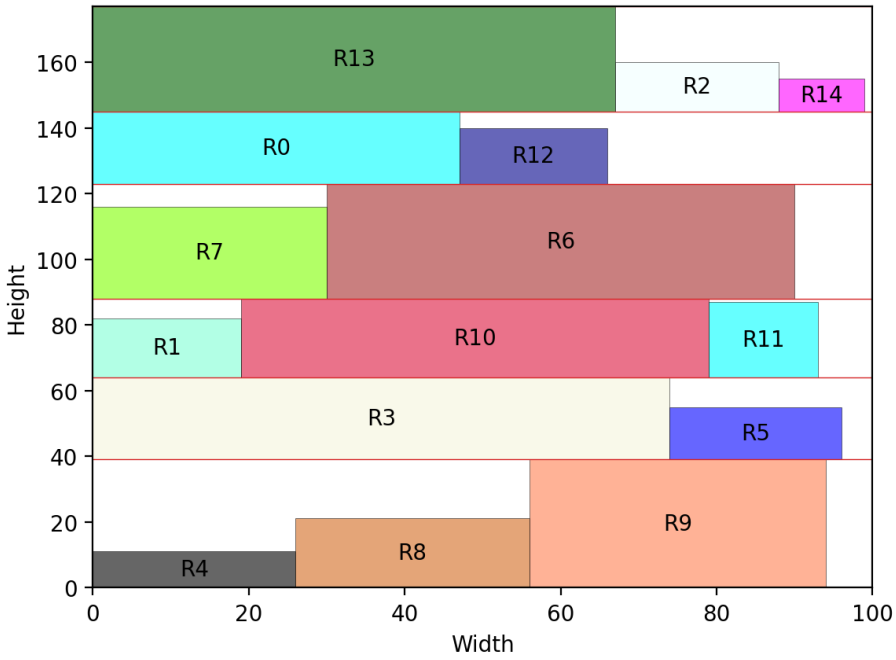


Figura 2: Representación gráfica del individuo con genes $S = [4, 8, 9, 3, 5, 1, 10, 11, 7, 6, 0, 12, 13, 2, 14]$ y lista de rotación $R = [0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0]$

En la figura 2 los rectángulos que rotan 90° serían: $R1, R3, R6, R10, R13$.

3.2. Selección

"La selección es la elección de los individuos que van a participar en la creación de la descendencia que conformara la próxima población".[2] En esta implementación del GA se utilizó la selección por torneo (*Tournament selection*). Se realizan n torneos, donde $n = P, n \in \mathbf{N}$, siendo P el tamaño de la población. En cada torneo se seleccionan m individuos, donde $m \in \mathbf{N}$. Una vez seleccionados los m individuos se los hace "competir"para ver quien tiene el mejor *fitness*. El fitness, es el valor que devuelve la función objetivo que buscamos optimizar y nos da una idea de que tan apto es el individuo. Cuando se terminan los n torneos, queda conformado un nuevo conjunto de tamaño P donde pueden existir individuos repetidos.

3.3. Crossover

El crossover o cruzamiento es el proceso en el cual dos individuos seleccionados generan descendencia a partir de la combinación de sus *genes. El propósito principal del crossover es el intercambio de experiencia y este proceso acelera en*

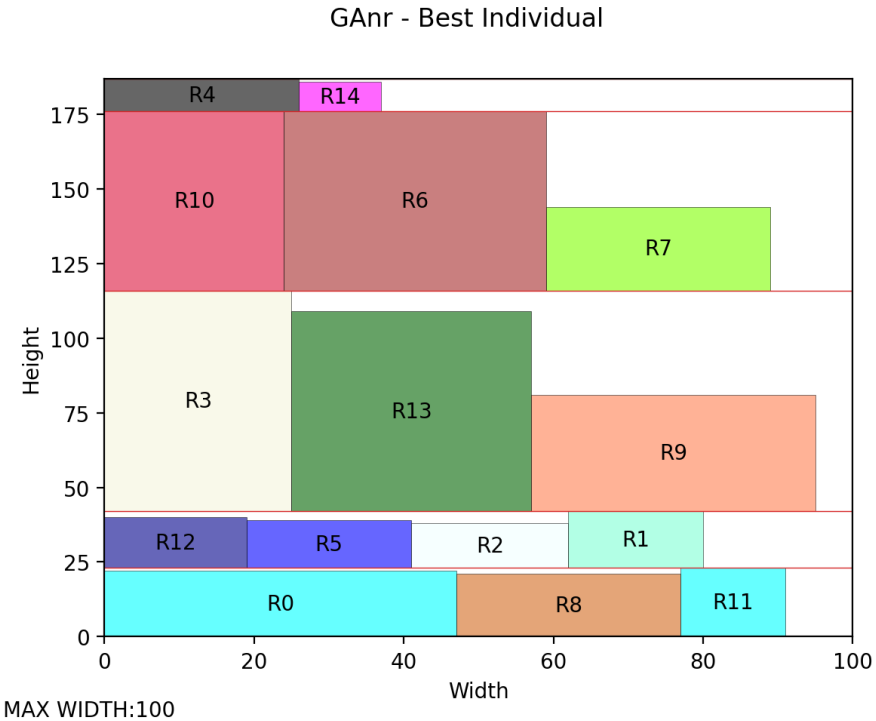
gran medida la búsqueda de una solución aceptable[2]. En esta arquitectura del algoritmo genético se utilizó una función de crossover llamada *order crossover*. *El principio fundamental de este tipo de función de crosover es preservar el orden de los genes de ambos padres*[2].
En el algoritmo implementado, el crossover se realiza con dos individuos extraídos del conjunto que se generó después del realizar los n torneos. Luego, según una probabilidad p_c los individuos se van a cruzar y generan una descendencia, en caso contrario, simplemente se agregarán los individuos seleccionados en el conjunto de la nueva población.
Cuando el algoritmo permite rotación de los rectángulos el crossover también se aplica a la lista de rotación que contiene cada individuo. El order crossover no puede ser aplicado en la lista de rotación porque solo funciona con arreglos con valores distintos entre si y ya que esta lista contiene solo 0s y 1s, un *one point crossover* fue utilizado en este caso. Mas acerca de como se implementa la rotación para un individuo fue explicado en la **sección 3.1**

3.4. Mutación

Para la función de mutación simplemente se intercambian dos bits en dos posiciones aleatorias del cromosoma. Esta mutación ocurre con una probabilidad p_m.

3.5. Función de fitness

La función de fitness da una idea de cuan bien se adapta un individuo a su ambiente[2]. Así, dependiendo del tipo de problema, el mejor individuo es aquel en el cual su función objetivo es un mínimo o un máximo global. En el caso de nuestro algoritmo genético, la función objetivo es la suma total de los rectángulos en cada "nivel" del strip. Esto es, minimizar la suma de las alturas de los rectángulos mas altos en cada uno de los niveles.



MAX WIDTH:100

Fitness:187

Chromosome:[0, 8, 11, 12, 5, 2, 1, 3, 13, 9, 10, 6, 7, 4, 14]

Solution:[[0, 8, 11], [12, 5, 2, 1], [3, 13, 9], [10, 6, 7], [4, 14]]

R0: W:47 H:22	R4: W:26 H:11	R8: W:30 H:21	R12: W:19 H:17
R1: W:18 H:19	R5: W:22 H:16	R9: W:38 H:39	R13: W:32 H:67
R2: W:21 H:15	R6: W:35 H:60	R10: W:24 H:60	R14: W:11 H:10
R3: W:25 H:74	R7: W:30 H:28	R11: W:14 H:23	

Figura 3: Representación de la mejor solución con 15 rectángulos y 1000 generaciones (sin rotación)

En la **Figura 3** tenemos un individuo con un valor de fitness 187 que es el resultado de sumar las alturas de: $R11, R1, R3, R10, R4$ en el lugar de $R10$ se pudo haber elegido $R6$ ya que poseen la misma altura.

4. Resultados

Los resultados obtenidos fueron generados a partir de la ejecución de 6 instancias, donde las dimensiones de los rectángulos son leídas del archivo junto con el ancho máximo del strip. Para cada instancia se hicieron 20 ejecuciones con dis-

tintas semillas y con una cantidad de 500 generaciones por ejecución. El tamaño de la población es de 50 individuos, la probabilidad de crossover es $p_c = 0{,}65$, $p_m = 0{,}1$ para la probabilidad de mutación y el tamaño del torneo es 2.

4.1. Instancias

Las dimensiones de los rectángulos se lista en la siguientes tabla.

4.1.1. Instancia spp9a

Rectángulos

i	1	2	3	4	5	6	7	8	9
w_i	4	15	6	3	4	2	8	3	10
h_i	15	3	6	12	6	8	2	5	3

$$W = 15$$

Resultados Algoritmo con rotación

- **Mejor fitness:** 23
- **Peor fitness:** 26
- **Media aritmética:** 23.95
- **Mediana:** 23.5
- **Desviación:** 1.023
- **Tiempo de ejecución(seg):** 25.13
- **Solución:** [6, 0, 1, 8, 5, 3, 2, 7, 4]

Resultados Algoritmo sin rotación

- **Mejor fitness:** 29
- **Peor fitness:** 29
- **Media aritmética:** 29
- **Mediana:** 29
- **Desviación:** 0.0
- **Tiempo de ejecución(seg):** 23.78
- **Solución:** [6, 8, 2, 4, 7, 5, 0, 3, 1]

4.1.2. Instancia spp9b

Rectángulos

i	1	2	3	4	5	6	7	8	9
w_i	4	15	6	3	3	2	8	3	10
h_i	15	3	6	12	8	8	2	5	3

$$W = 15$$

Resultados Algoritmo con rotación

- **Mejor fitness:** 23
- **Peor fitness:** 25
- **Media aritmética:** 23.95
- **Mediana:** 23.5
- **Desviación:** 0.973
- **Tiempo de ejecución(seg):** 25.55
- **Solución:** [6, 8, 1, 3, 0, 5, 2, 4, 7]

Resultados Algoritmo sin rotación

- **Mejor fitness:** 29
- **Peor fitness:** 29
- **Media aritmética:** 29
- **Mediana:** 29
- **Desviación:** 0.0
- **Tiempo de ejecución(seg):** 24.32
- **Solución:** [6, 2, 7, 8, 1, 0, 5, 3, 4]

4.1.3. Instancia spp10

Rectángulos

i	1	2	3	4	5	6	7	8	9	10
w_i	5	9	2	3	4	6	4	9	3	2
h_i	11	8	8	6	9	8	7	8	11	11

$$W = 20$$

Resultados Algoritmo con rotación

- **Mejor fitness:** 24
- **Peor fitness:** 27
- **Media aritmética:** 26.55
- **Mediana:** 27
- **Desviación:** 0.73
- **Tiempo de ejecución(seg):** 27.02
- **Solución:** [4, 8, 9, 2, 1, 0, 6, 7, 5, 3]

Resultados Algoritmo sin rotación

- **Mejor fitness:** 27
- **Peor fitness:** 27
- **Media aritmética:** 27
- **Mediana:** 27
- **Desviación:** 0.0
- **Tiempo de ejecución(seg):** 26.87
- **Solución:** [7, 2, 5, 3, 4, 0, 9, 8, 6, 1]

4.1.4. Instancia spp11

Rectángulos

i	1	2	3	4	5	6	7	8	9	10	11
w_i	4	7	10	2	6	3	1	4	4	6	4
h_i	13	8	5	8	7	8	12	7	4	12	8

$$W = 20$$

Resultados Algoritmo con rotación

- **Mejor fitness:** 25
- **Peor fitness:** 28
- **Media aritmética:** 26.15
- **Mediana:** 26.0
- **Desviación:** 0.57
- **Tiempo de ejecución(seg):** 29.00
- **Solución:** [5, 9, 6, 0, 2, 4, 3, 1, 10, 7, 8]

Resultados Algoritmo sin rotación

- **Mejor fitness:** 26
- **Peor fitness:** 26
- **Media aritmética:** 26
- **Mediana:** 26
- **Desviación:** 0.0
- **Tiempo de ejecución(seg):** 28.05
- **Solución:** [2, 8, 9, 6, 3, 0, 4, 7, 10, 5, 1]

4.1.5. Instancia spp12

Rectángulos

i	1	2	3	4	5	6	7	8	9	10	11	12
w_i	3	5	4	10	7	6	8	4	18	2	3	9
h_i	6	8	5	8	4	3	8	6	3	3	10	2

$$W = 20$$

Resultados Algoritmo con rotación

- **Mejor fitness:** 24
- **Peor fitness:** 26
- **Media aritmética:** 25.4
- **Mediana:** 25.5
- **Desviación:** 0.66
- **Tiempo de ejecución(seg):** 31.27
- **Solución:** [9, 2, 1, 7, 0, 4, 5, 10, 11, 8, 3, 6]

Resultados Algoritmo sin rotación

- **Mejor fitness:** 27
- **Peor fitness:** 27
- **Media aritmética:** 27
- **Mediana:** 27
- **Desviación:** 0.0
- **Tiempo de ejecución(seg):** 29.80
- **Solución:** [2, 7, 1, 0, 10, 8, 3, 6, 5, 4, 9, 11]

4.1.6. Instancia spp13

Rectángulos

i	1	2	3	4	5	6	7	8	9	10	11	12	13
w_i	7	1	5	9	2	9	5	5	4	3	2	9	2
h_i	5	8	9	3	16	7	9	3	9	7	7	3	16

$$W = 20$$

Resultados Algoritmo con rotación

- **Mejor fitness:** 23
- **Peor fitness:** 31
- **Media aritmética:** 26.4
- **Mediana:** 26
- **Desviación:** 2.28
- **Tiempo de ejecución(seg):** 32.65
- **Solución:** [4, 12, 5, 0, 10, 9, 3, 11, 7, 6, 8, 2, 1]

Resultados Algoritmo sin rotación

- **Mejor fitness:** 31
- **Peor fitness:** 31
- **Media aritmética:** 31
- **Mediana:** 31
- **Desviación:** 0.0
- **Tiempo de ejecución(seg):** 29.03
- **Solución:** [6, 2, 1, 8, 12, 4, 11, 0, 5, 10, 9, 3, 7]

En el **apéndice B** se encuentran las imágenes de la mejor solución de cada instancia.

5. Discusión

A primera vista podemos observar que la *desviación* de los resultados obtenidos con el algoritmo genético sin rotación es 0 (cero) para cada una de las instancias. Esto demuestra la mayor estabilidad del algoritmo sobre todo en los resultados de la instancia **spp13** donde la desviación es la más grande de todas con un valor de 2.28. Sin embargo, las mejores soluciones obtenidas por el algoritmo **con rotación** son mucho menores que las proporcionadas por el **GA sin rotación**. Esto se debe principalmente a que al permitir la rotación de algunos de los rectángulos le estamos dando un grado mayor de libertad al algoritmo para explorar nuevas soluciones.
Una de las desventajas de permitir la rotación se ve reflejada en el tiempo de ejecución del algoritmo, que en promedio es 1.47 segundos mayor para la variante **con rotación** del algoritmo y siendo la mayor diferencia de 3.62 segundos para la instancia **spp13**.

6. Conclusiones

Después de analizar los resultados podemos concluir que los algoritmos genéticos generan soluciones muy buenas en tiempos de ejecución razonables en comparación a lo que sería una búsqueda exhaustiva, considerando que el espacio de búsqueda para la instancia **spp13** con $n = 13$ es de $6,227,020,800$ posibles combinaciones de rectángulos.

Referencias

[1] Metaheuristics - From Design to Implementation. Wiley 2009, ISBN 978-0-470-27858-1, pp. I-XXIX, 1-593

[2] Learning Genetic Algorithms with Python. Ivan Gridin 2021, ISBN 978-81-94837-756.

Repositorio código fuente

`https://github.com/SebaRiccardo/Strip_Packing_GA`

A. Fragmentos de Código

A.1. Main GA flow

```
def GA(number_of_rectangles, values, W, genes, it_rotates,seed,run_number):

    random.seed(seed)
```

```
# Generate reference rectangle list
set_of_rectangles = generate_N_ractangles(number_of_rectangles,values)

# Start inicial population
population = create_starting_population(POPULATION_SIZE, W,
set_of_rectangles, genes, calculate_fitness, it_rotates,seed)

# initial stats
best_ind = get_best_individual(population)
#average_fitness = [get_average_fitness(population)]
best_fitness = [best_ind.fitness]
best_individuals = [best_ind]
best_fitness_ever = [best_ind.fitness]

for generation_number in range(MAX_GENERATIONS):
    new_seed = (seed+generation_number)

    # SELECTION
    selected = select_tournament(population, TOURNAMENT_SIZE, new_seed)

    # CROSSOVER
    crossed_offspring = []
    for ind1, ind2 in zip(selected[::2], selected[1::2]):
        if random.random() < CROSS_OVER_PROBABILITY:
            children = crossover(ind1, ind2, W, set_of_rectangles,
            calculate_fitness, new_seed, it_rotates)
            crossed_offspring.append(children[0])
            crossed_offspring.append(children[1])
        else:
            crossed_offspring.append(ind1)
            crossed_offspring.append(ind2)

    # MUTATION
    mutated = []
    for ind in crossed_offspring:
        if random.random() < MUTATION_PROBABILITY:
            mutated.append(mutate(ind.gene_list, ind.rotation, W,
            set_of_rectangles,
            number_of_rectangles, calculate_fitness, new_seed, it_rotates))
        else:
            mutated.append(ind)

    population = mutated.copy()

    # all values for the best individual in each generation
```

```
        best_ind, best_of_generation, best_fitness, best_fitness_ever = \
            stats(population, set_of_rectangles, W, best_ind, best_fitness,
            best_fitness_ever)

    return best_ind
```

A.2. Clase Individuo

```
class Individual:

    def __init__(self, gene_list, rotation, max_width, rectangles,
        fitness_function, it_rotates):
        self.gene_list = gene_list
        self.rotation = rotation
        self.fitness = fitness_function(self.gene_list, self.rotation,
        rectangles, max_width, it_rotates)

    def __str__(self):
        return "Chromosome: " + str(self.gene_list) + " Fitness: " +
         str(self.fitness) + " Rotation: " + str(self.rotation)

    def get_gene_list(self):
        return self.gene_list

    def get_rotation(self):
        return self.rotation

    def get_fitness(self):
        return self.fitness
```

A.3. Crossover

```
def order_crossover(p1, p2, seed):

    random.seed(seed)
    zero_shift = min(p1)
    length = len(p1)
    start, end = sorted([random.randrange(length) for _ in range(2)])
    c1, c2 = [nan] * length, [nan] * length
    t1, t2 = [x - zero_shift for x in p1], [x - zero_shift for x in p2]

    spaces1, spaces2 = [True] * length, [True] * length
    for i in range(length):
        if i < start or i > end:
```

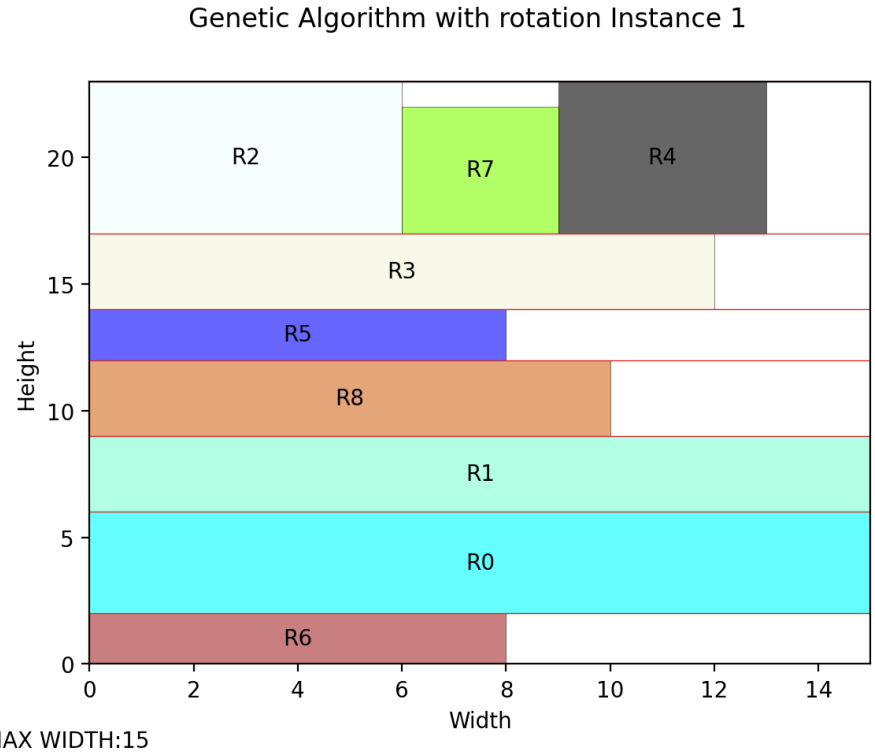
```
            spaces1[t2[i]] = False
            spaces2[t1[i]] = False

    j1, j2 = end + 1, end + 1
    for i in range(length):
        if not spaces1[t1[(end + i + 1) % length]]:
            c1[j1 % length] = t1[(end + i + 1) % length]
            j1 += 1

        if not spaces2[t2[(i + end + 1) % length]]:
            c2[j2 % length] = t2[(i + end + 1) % length]
            j2 += 1

    for i in range(start, end + 1):
        c1[i], c2[i] = t2[i], t1[i]

    return [[x + zero_shift for x in c1], [x + zero_shift for x in c2]]
```

B. Imágenes mejores soluciones

B.1. GAr

Mejores soluciones de las 20 ejecuciones del algoritmo con rotación

B.1.1. spp9a



MAX WIDTH:15

Fitness:23

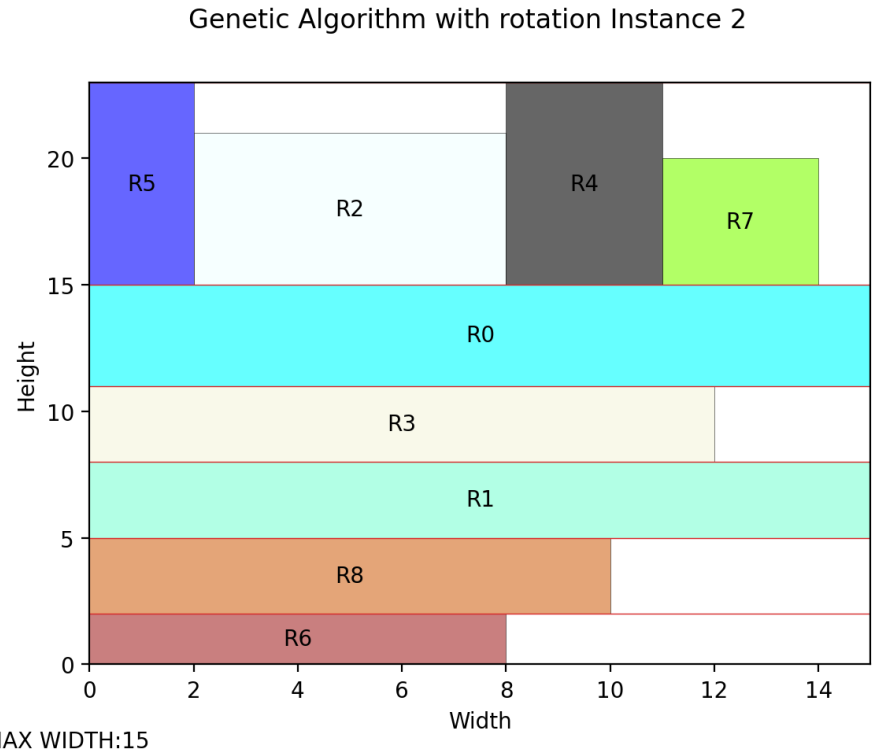
Chromosome:[6, 0, 1, 8, 5, 3, 2, 7, 4]

Solution:[[6], [0], [1], [8], [5], [3], [2, 7, 4]]

Rotation: [1, 0, 0, 1, 0, 1, 0, 0, 0]

R0: W:4 H:15
R1: W:15 H:3
R2: W:6 H:6
R3: W:3 H:12
R4: W:4 H:6
R5: W:2 H:8
R6: W:8 H:2
R7: W:3 H:5
R8: W:10 H:3

B.1.2. spp9b



MAX WIDTH:15

Fitness:23

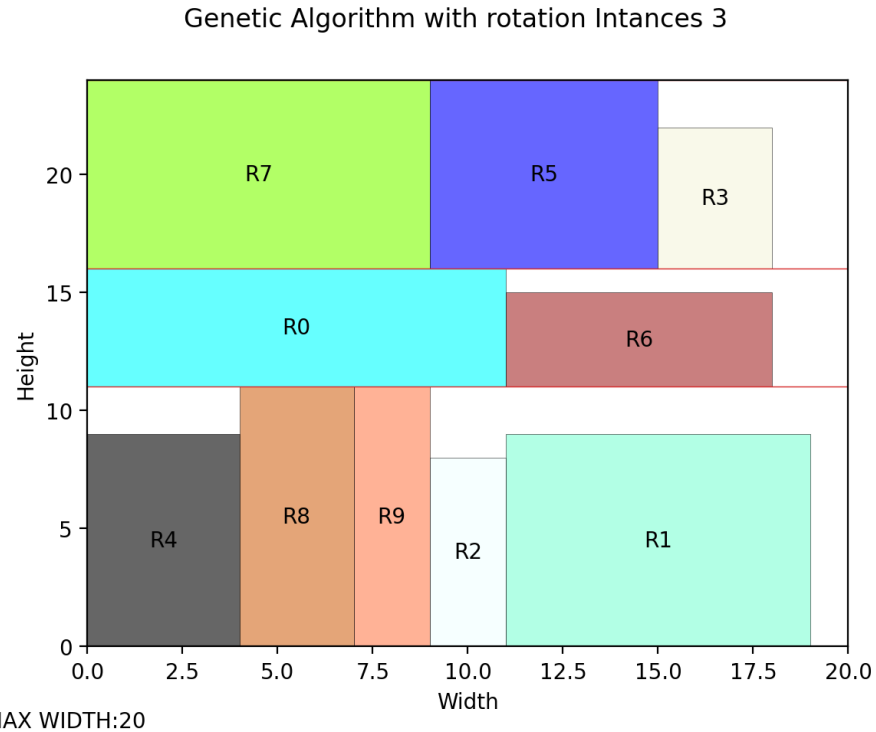
Chromosome:[6, 8, 1, 3, 0, 5, 2, 4, 7]

Solution:[[6], [8], [1], [3], [0], [5, 2, 4, 7]]

Rotation: [1, 0, 0, 1, 0, 0, 0, 0, 0]

R0: W:4 H:15	R4: W:3 H:8	R8: W:10 H:3
R1: W:15 H:3	R5: W:2 H:8	
R2: W:6 H:6	R6: W:8 H:2	
R3: W:3 H:12	R7: W:3 H:5	

B.1.3. spp10



MAX WIDTH:20

Fitness:24

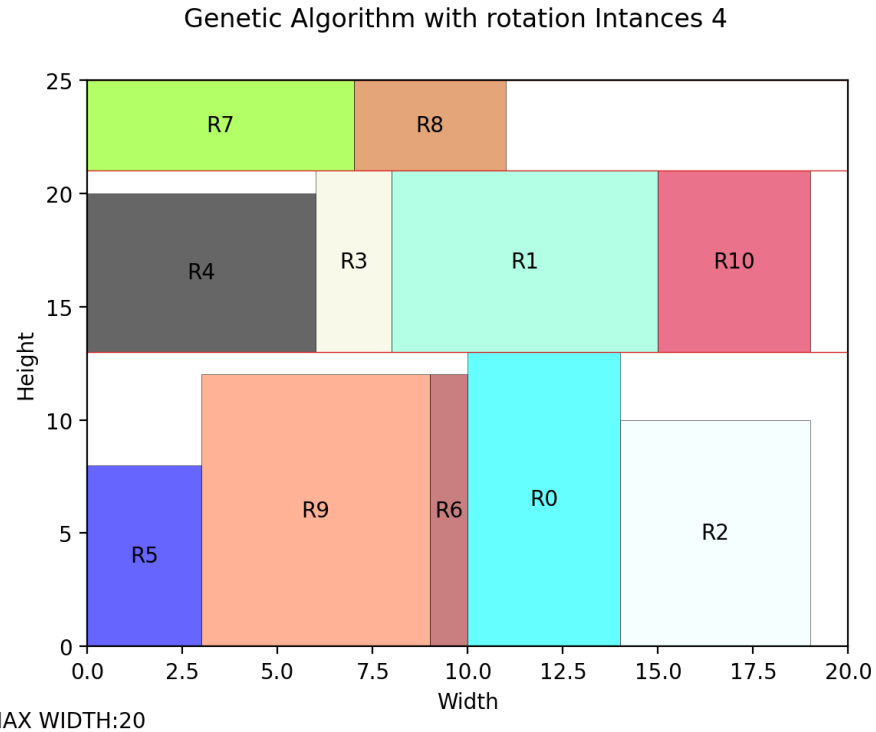
Chromosome:[4, 8, 9, 2, 1, 0, 6, 7, 5, 3]

Solution:[[4, 8, 9, 2, 1], [0, 6], [7, 5, 3]]

Rotation: [1, 1, 0, 0, 0, 0, 1, 0, 0, 0]

R0: W:5 H:11	R4: W:4 H:9	R8: W:3 H:11
R1: W:9 H:8	R5: W:6 H:8	R9: W:2 H:11
R2: W:2 H:8	R6: W:4 H:7	
R3: W:3 H:6	R7: W:9 H:8	

B.1.4. spp11



MAX WIDTH:20

Fitness:25

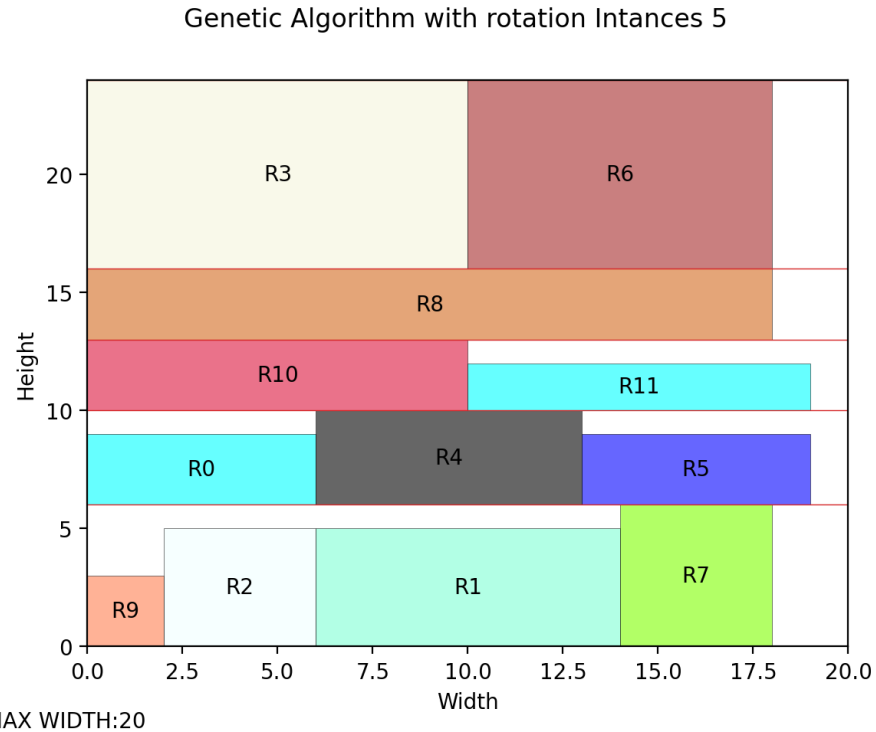
Chromosome:[5, 9, 6, 0, 2, 4, 3, 1, 10, 7, 8]

Solution:[[5, 9, 6, 0, 2], [4, 3, 1, 10], [7, 8]]

Rotation: [0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0]

R0: W:4 H:13	R4: W:6 H:7	R8: W:4 H:4
R1: W:7 H:8	R5: W:3 H:8	R9: W:6 H:12
R2: W:10 H:5	R6: W:1 H:12	R10: W:4 H:8
R3: W:2 H:8	R7: W:4 H:7	

B.1.5. spp12



MAX WIDTH:20

Fitness:24

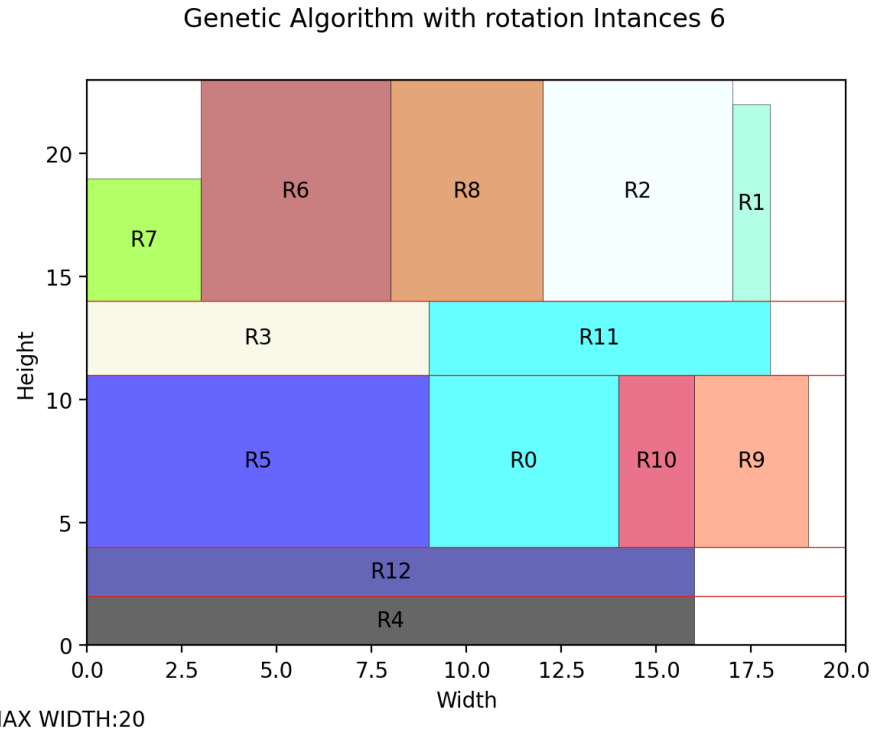
Chromosome:[9, 2, 1, 7, 0, 4, 5, 10, 11, 8, 3, 6]

Solution:[[9, 2, 1, 7], [0, 4, 5], [10, 11], [8], [3, 6]]

Rotation: [1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0]

R0: W:3 H:6	R4: W:7 H:4	R8: W:18 H:3
R1: W:5 H:8	R5: W:6 H:3	R9: W:2 H:3
R2: W:4 H:5	R6: W:8 H:8	R10: W:3 H:10
R3: W:10 H:8	R7: W:4 H:6	R11: W:9 H:2

B.1.6. spp13



MAX WIDTH:20

Fitness:23

Chromosome:[4, 12, 5, 0, 10, 9, 3, 11, 7, 6, 8, 2, 1]

Solution:[[4], [12], [5, 0, 10, 9], [3, 11], [7, 6, 8, 2, 1]]

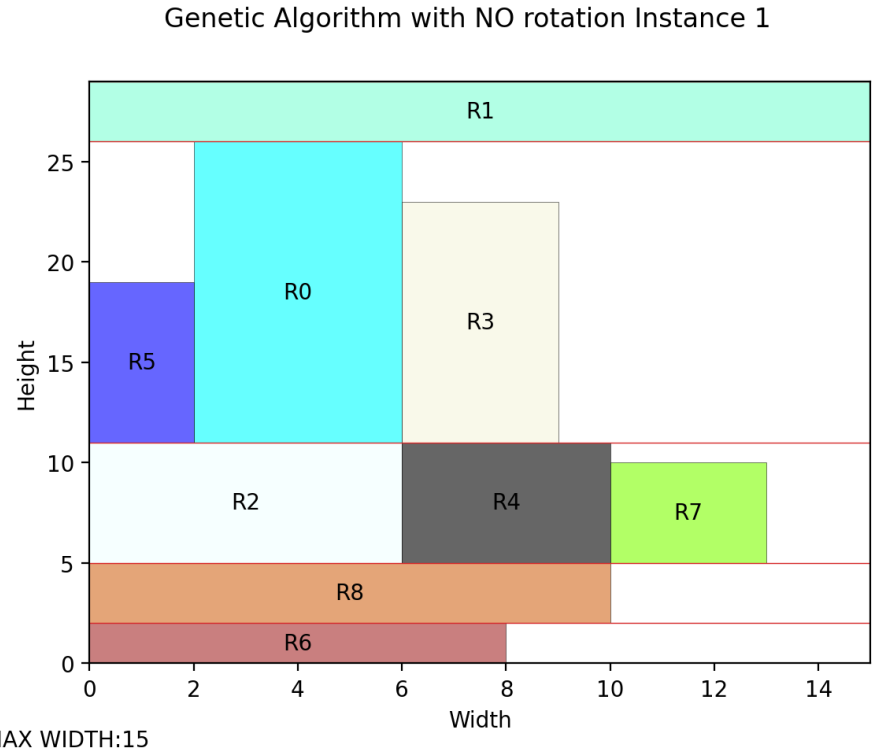
Rotation: [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1]

R0: W:7 H:5	R4: W:2 H:16	R8: W:4 H:9	R12: W:2 H:16
R1: W:1 H:8	R5: W:9 H:7	R9: W:3 H:7	
R2: W:5 H:9	R6: W:5 H:9	R10: W:2 H:7	
R3: W:9 H:3	R7: W:5 H:3	R11: W:9 H:3	

B.2. GAnr

Mejores soluciones de las 20 ejecuciones del algoritmo sin rotación

B.2.1. spp9a



MAX WIDTH:15

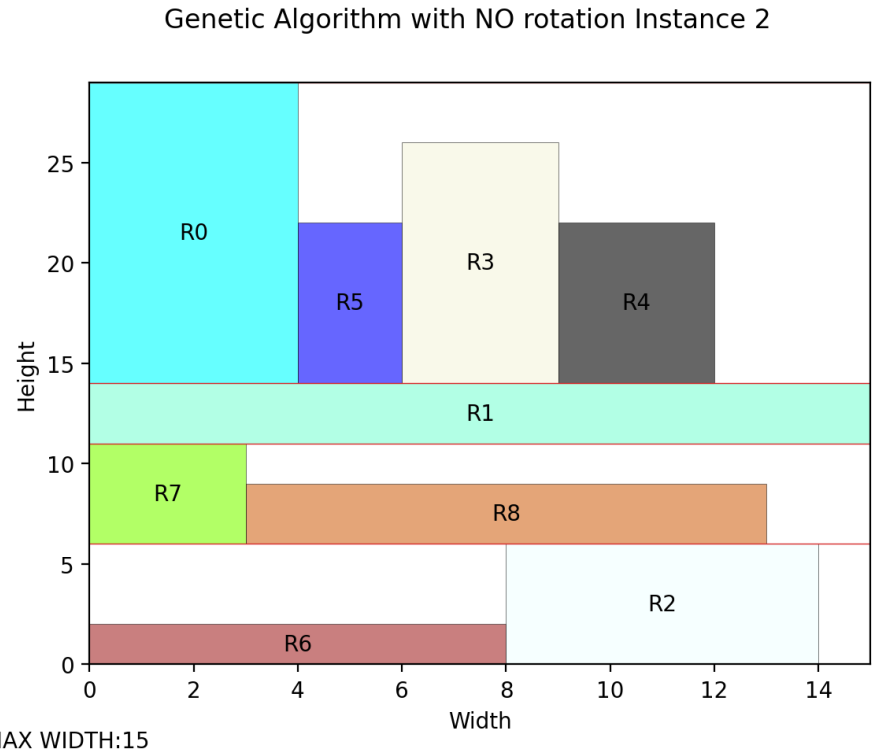
Fitness:29

Chromosome:[6, 8, 2, 4, 7, 5, 0, 3, 1]

Solution:[[6], [8], [2, 4, 7], [5, 0, 3], [1]]

R0: W:4 H:15
R1: W:15 H:3
R2: W:6 H:6
R3: W:3 H:12
R4: W:4 H:6
R5: W:2 H:8
R6: W:8 H:2
R7: W:3 H:5
R8: W:10 H:3

B.2.2. spp9b



MAX WIDTH:15

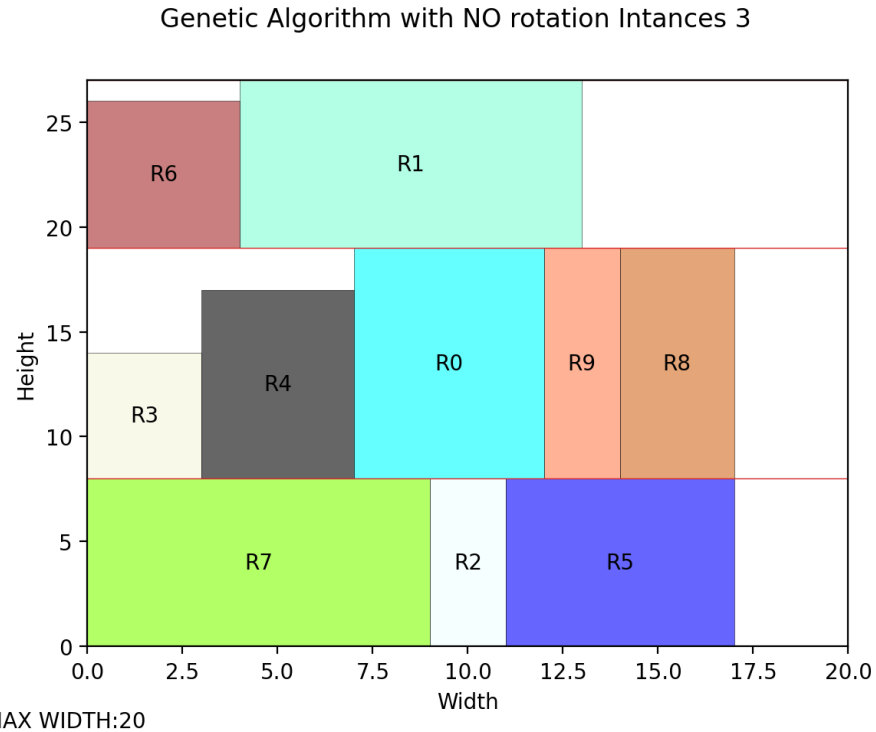
Fitness:29

Chromosome:[6, 2, 7, 8, 1, 0, 5, 3, 4]

Solution:[[6, 2], [7, 8], [1], [0, 5, 3, 4]]

R0: W:4 H:15
R1: W:15 H:3
R2: W:6 H:6
R3: W:3 H:12
R4: W:3 H:8
R5: W:2 H:8
R6: W:8 H:2
R7: W:3 H:5
R8: W:10 H:3

B.2.3. spp10



MAX WIDTH:20

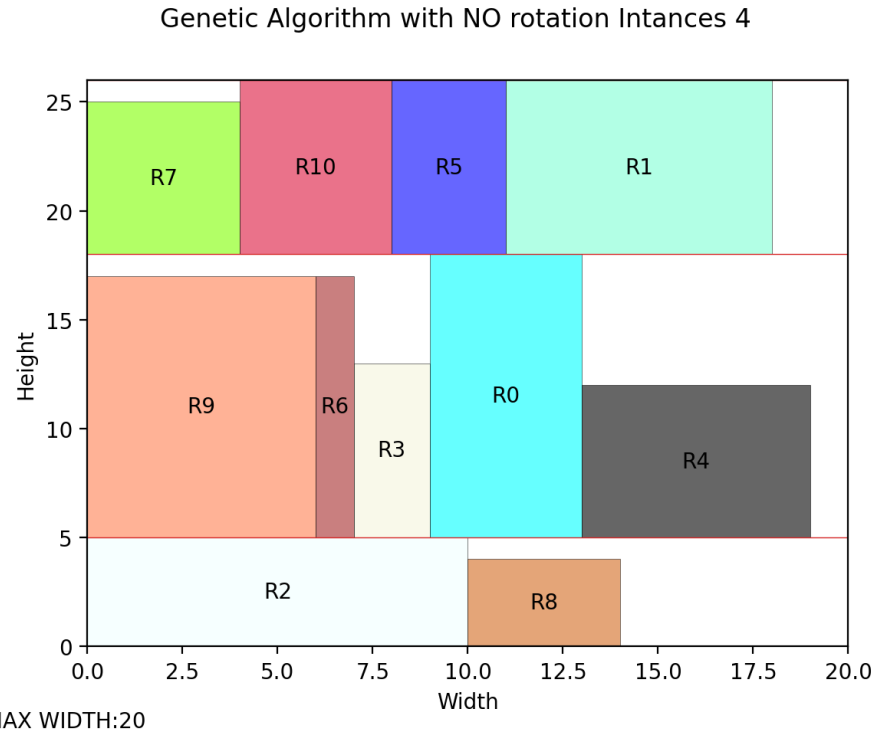
Fitness:27

Chromosome:[7, 2, 5, 3, 4, 0, 9, 8, 6, 1]

Solution:[[7, 2, 5], [3, 4, 0, 9, 8], [6, 1]]

R0: W:5 H:11	R4: W:4 H:9	R8: W:3 H:11
R1: W:9 H:8	R5: W:6 H:8	R9: W:2 H:11
R2: W:2 H:8	R6: W:4 H:7	
R3: W:3 H:6	R7: W:9 H:8	

B.2.4. spp11



MAX WIDTH:20

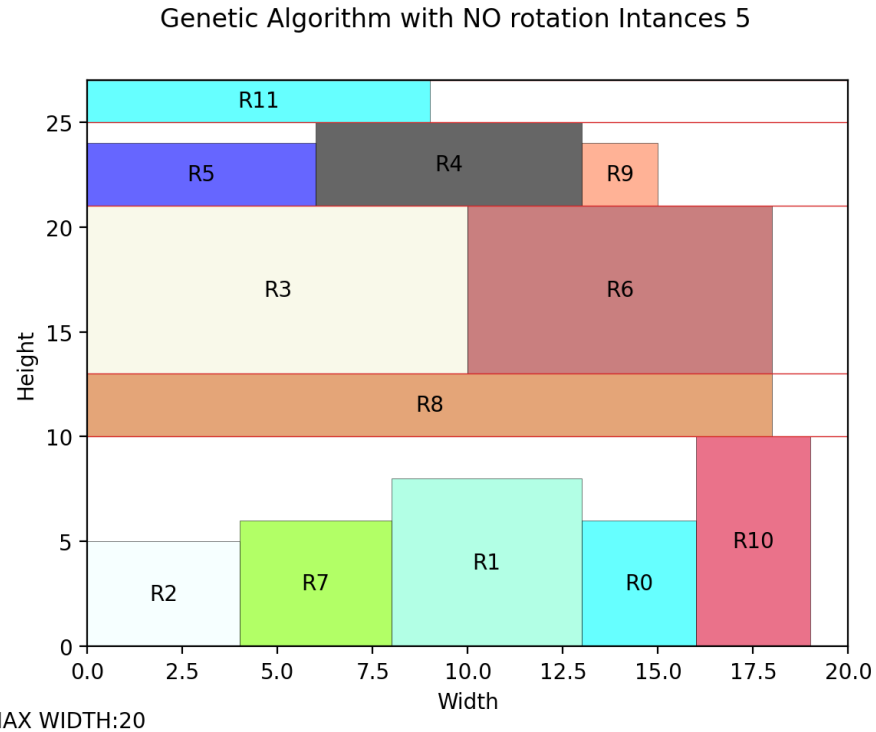
Fitness:26

Chromosome:[2, 8, 9, 6, 3, 0, 4, 7, 10, 5, 1]

Solution:[[2, 8], [9, 6, 3, 0, 4], [7, 10, 5, 1]]

R0: W:4 H:13	R4: W:6 H:7	R8: W:4 H:4
R1: W:7 H:8	R5: W:3 H:8	R9: W:6 H:12
R2: W:10 H:5	R6: W:1 H:12	R10: W:4 H:8
R3: W:2 H:8	R7: W:4 H:7	

B.2.5. spp12



MAX WIDTH:20

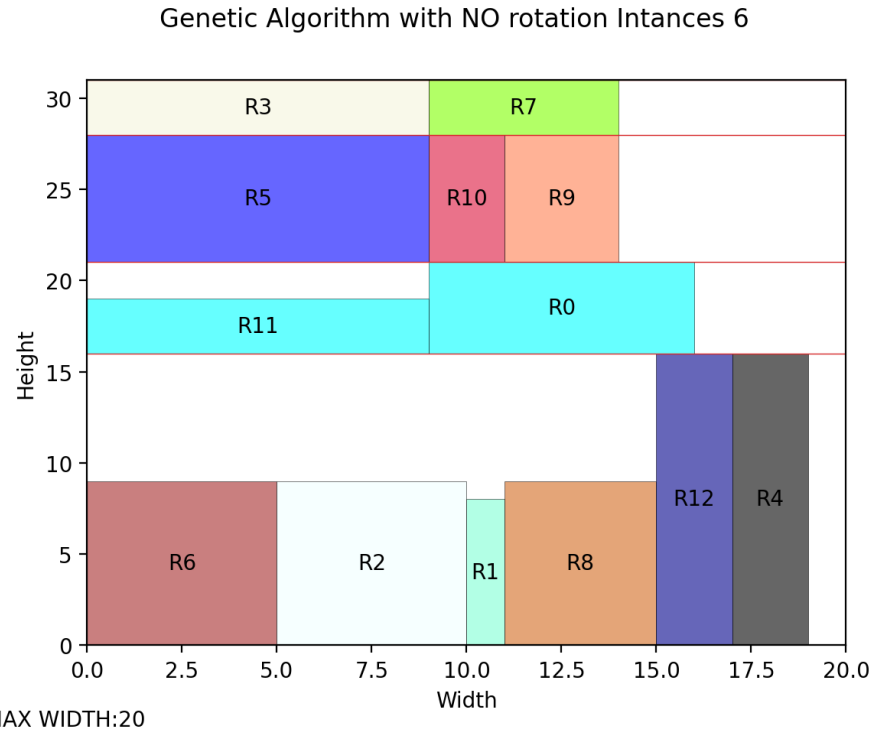
Fitness:27

Chromosome:[2, 7, 1, 0, 10, 8, 3, 6, 5, 4, 9, 11]

Solution:[[2, 7, 1, 0, 10], [8], [3, 6], [5, 4, 9], [11]]

R0: W:3 H:6	R4: W:7 H:4	R8: W:18 H:3
R1: W:5 H:8	R5: W:6 H:3	R9: W:2 H:3
R2: W:4 H:5	R6: W:8 H:8	R10: W:3 H:10
R3: W:10 H:8	R7: W:4 H:6	R11: W:9 H:2

B.2.6. spp13



MAX WIDTH:20

Fitness:31

Chromosome:[6, 2, 1, 8, 12, 4, 11, 0, 5, 10, 9, 3, 7]

Solution:[[6, 2, 1, 8, 12, 4], [11, 0], [5, 10, 9], [3, 7]]

R0: W:7 H:5	R4: W:2 H:16	R8: W:4 H:9	R12: W:2 H:16
R1: W:1 H:8	R5: W:9 H:7	R9: W:3 H:7	
R2: W:5 H:9	R6: W:5 H:9	R10: W:2 H:7	
R3: W:9 H:3	R7: W:5 H:3	R11: W:9 H:3	