Universidad de Oviedo

Intelligent Scheduling and Optimization

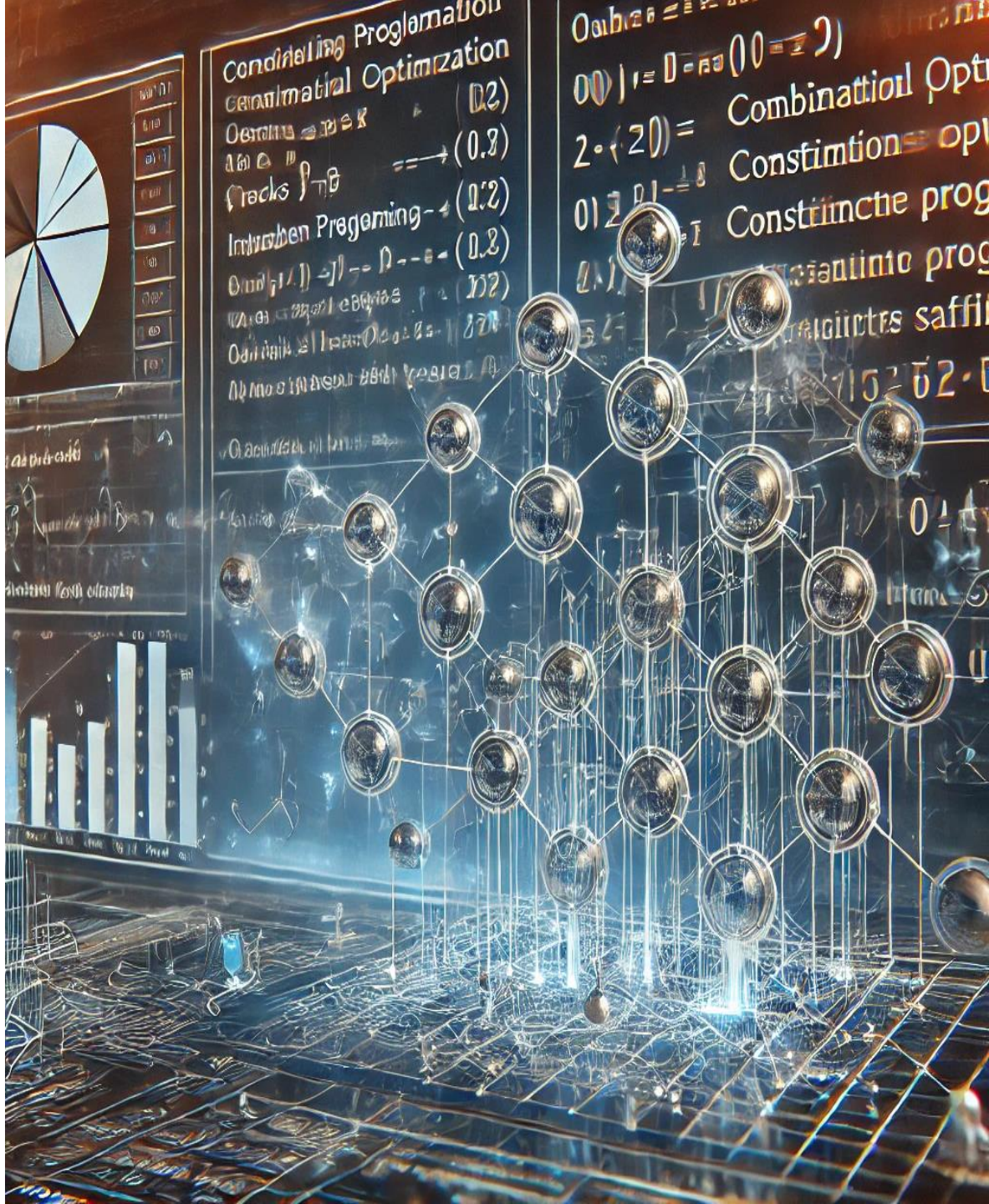
Técnicas de Inteligencia Artificial para la Optimización y Programación de Recursos

Algoritmos Genéticos y Diseño Orientado a Objeto

Jorge Puente Peinador
{puente}@uniovi.es

Ciencia de la Computación e Inteligencia Artificial
Departamento de Informática

ceisIA Centro de Estudios sobre el Impacto Social de la Inteligencia Artificial | Universidad de Oviedo



Objetivo JSSP + “Framework GA”

Algoritmo Genético Simple (SGA)

Algoritmo Genético

Parámetros de Entrada (Pc, Pm, NumGen, PopSize, …);

t ← 0;

1 Init(P(t)); // Población Inicial

2 3 Evaluate(P(t)); // Función Fitness (Decodificación. y Evaluación)

while (t < NroGen) **{**

t ← t+1;

4 P'(t) = Select(P(t-1)); // Selección

5 6 P''(t) = Alter(P'(t)); // Cruce y Mutación

Evaluate(P''(t)); // Función fitness (Decodificación. y Evaluación)

7 P(t) = Replace(P'(t), P''(t)); // Remplazamiento

}

end.

Composición de Operadores

Dependencias de los operadores

	Problema:		**Viajante de Comercio (TSP)**
	Representación Soluciones:		Permutación de símbolos (ciudades)
	DEPENDENCIAS		
OPERADOR	**Representación**	**Problema/Instancia**	
Selección	-	-	Torneo 8:1
Reemplazamiento	-	-	Generacional
Cruce	dependiente	-	Cruce Ordered Crossover (OX)
Mutación	dependiente	-	Mutación SWAP
Generador Inicial (Instancia->Genotipo)	dependiente	dependiente	Permutación Aleatoria / Heurístico: k vecinos más cercanos
Decodificador (Genotipo->Fenotipo)	dependiente	dependiente	Permutación ID-Ciudades -> Secuencia de coordenadas (x, y) a visitar
Evaluador (Fenotipo -> Fitness)	-	dependiente	Secuencia de (x, y) -> Distancia

Dependencias de los operadores

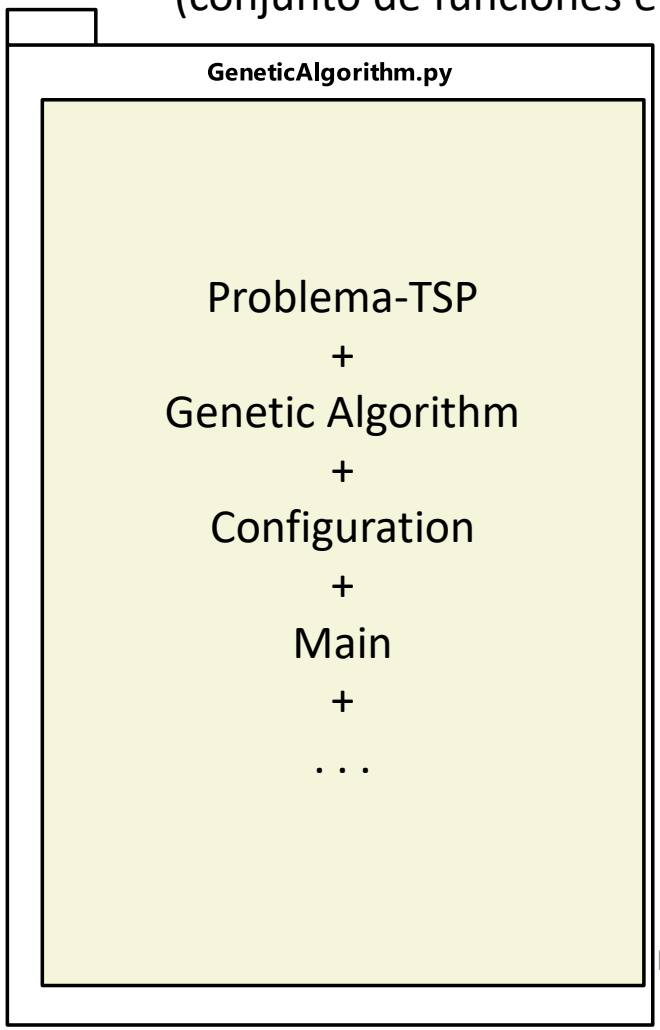
	Problema:		**Viajante de Comercio (TSP)**	**JSSP**
	Representación Soluciones:		Permutación de símbolos (ciudades)	Permutación de símbolos (tareas)
	DEPENDENCIAS			
OPERADOR	**Representación**	**Problema/Instancia**		
Selección	-	-	Torneo 8:1	Torneo 8:1
Reemplazamiento	-	-	Generacional	Generacional
Cruce	dependiente	-	Cruce Ordered Crossover (OX)	Cruce Order CrossOver (GOX) (*)
Mutación	dependiente	-	Mutación SWAP	Mutación SWAP (*)
Generador Inicial (Instancia->Genotipo)	dependiente	dependiente	Permutación Aleatoria / Heurístico: k vecinos más cercanos	Permutación Aleatoria de tareas
Decodificador (Genotipo->Fenotipo)	dependiente	dependiente	Permutación ID-Ciudades -> Secuencia de coordenadas (x, y) a visitar	El Fenotipo: Schedule y startingTimes
Evaluador (Fenotipo -> Fitness)	-	dependiente	Secuencia de (x, y) -> Distancia	El fitness es el Makespan (C_{max})

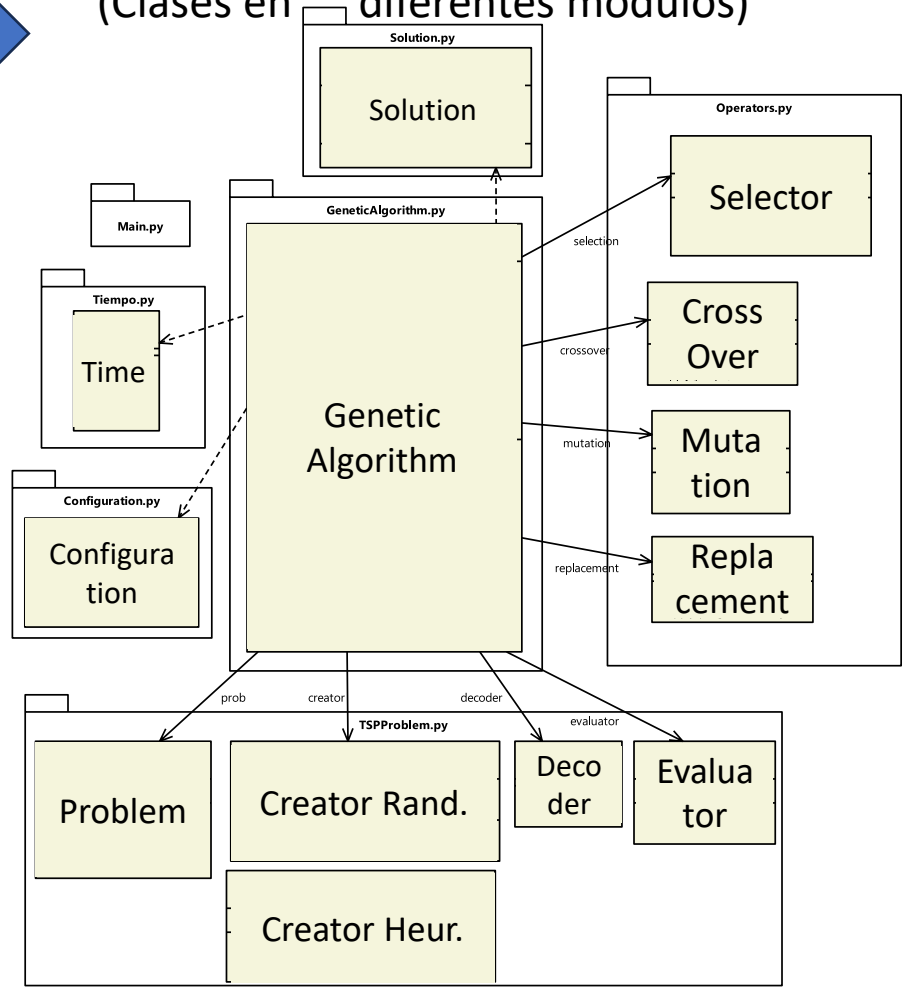
Diseño e Implementación GA-TSP

Monolítico
(conjunto de funciones en 1 módulo)

Rediseño utilizando P.O.O.

Orientado a Objetos
(Clases en diferentes módulos)





Diseño e Implementación GA-TSP

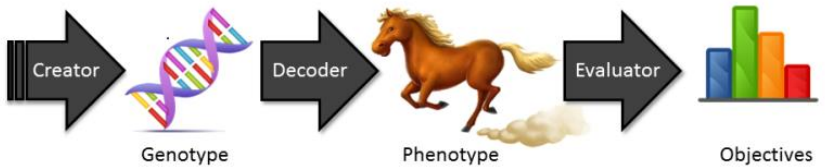
Monolítico
(conjunto de funciones en 1 módulo)

GeneticAlgorithm.py

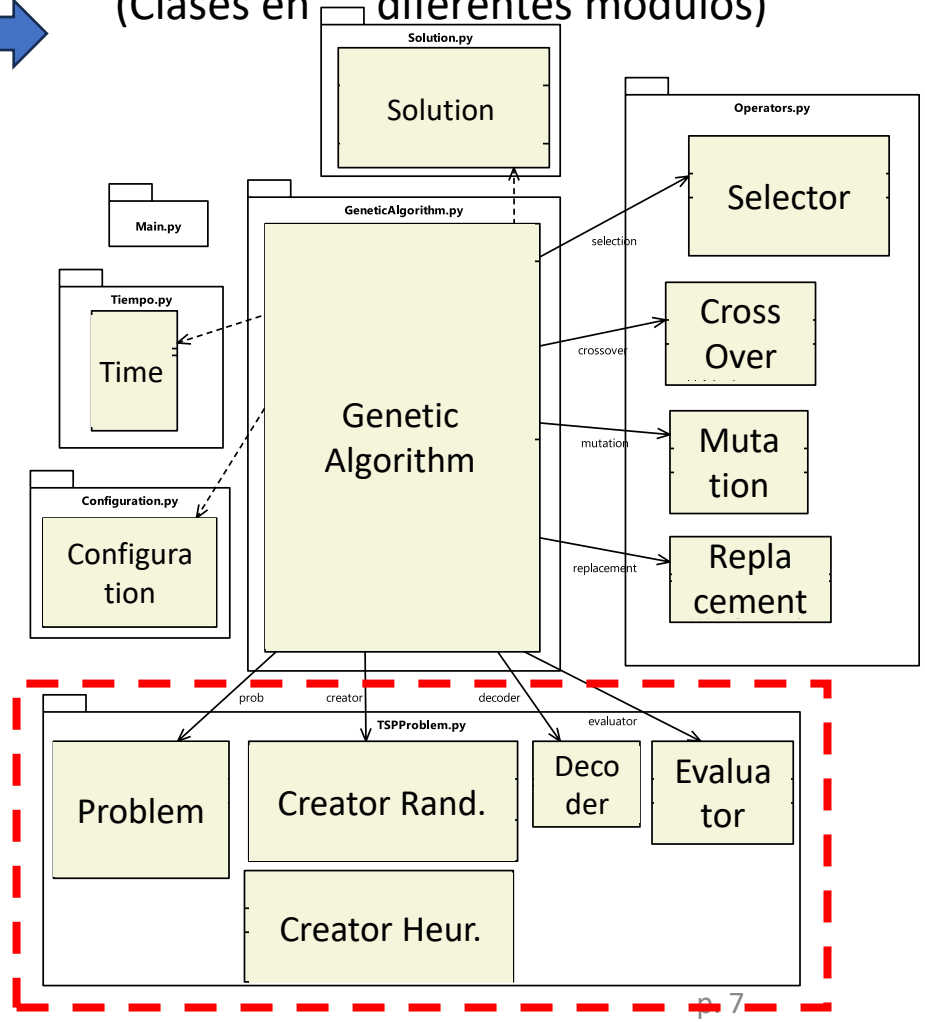
Problema-TSP
+
Genetic Algorithm
+
Configuration
+
Main
+
. . .

Rediseño utilizando P.O.O.

¿Módulos afectados al cambiar el problema?
(TSP -> JSSP)

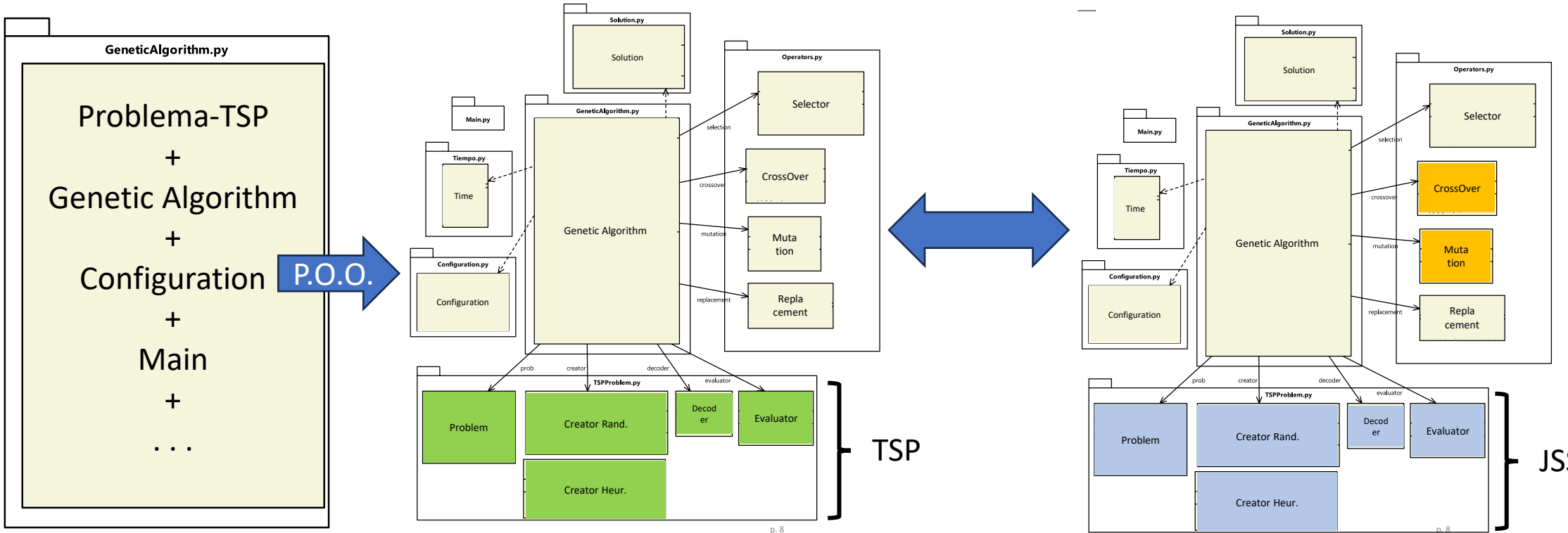


Orientado a Objetos
(Clases en diferentes módulos)



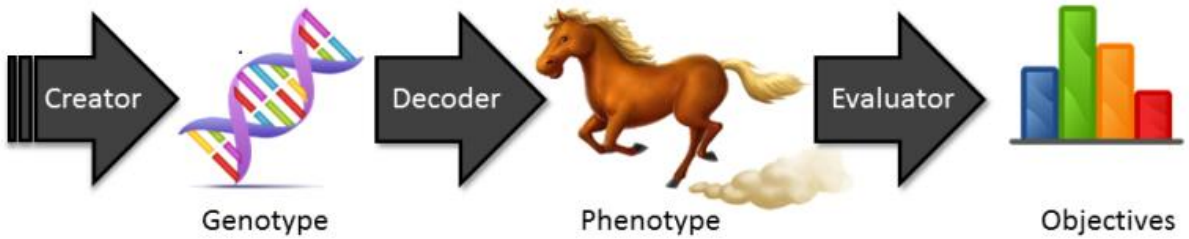
Objetivos

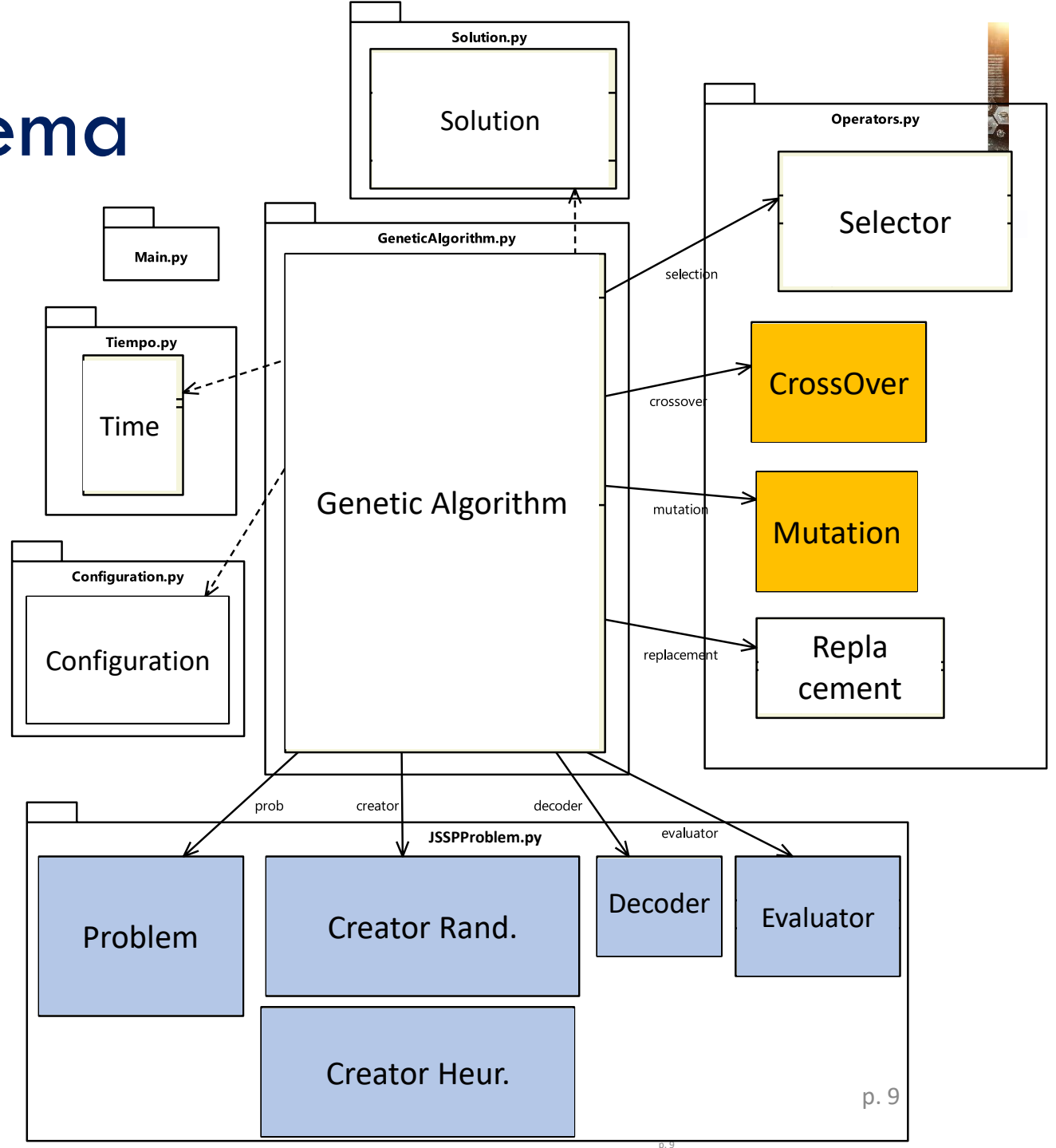
- Optimización problema JSSP mediante Algoritmos Genéticos
  - ¿Implementación "from Scratch"? No
    - Framework de Metaheurísticas
    - Nuestro propio Framework (basado en el GA-TSP de B. I. M. Refactorizado)



Cómo Definir el Problema

- Operadores específicos de JSSP
  - Problema
  - Codificación Cromosomas
    - Generación
    - ¿Cruce?
    - ¿Mutación?
  - Decodificador & Evaluador



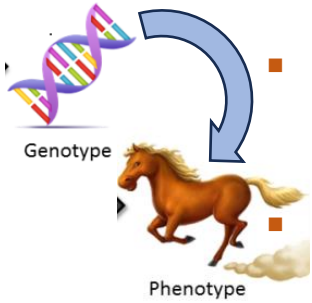


Intelligent Scheduling and Optimization

Algoritmos Genéticos. Población

- La **población** que emplea para evolucionar un Algoritmo Genético es representada mediante cromosomas.

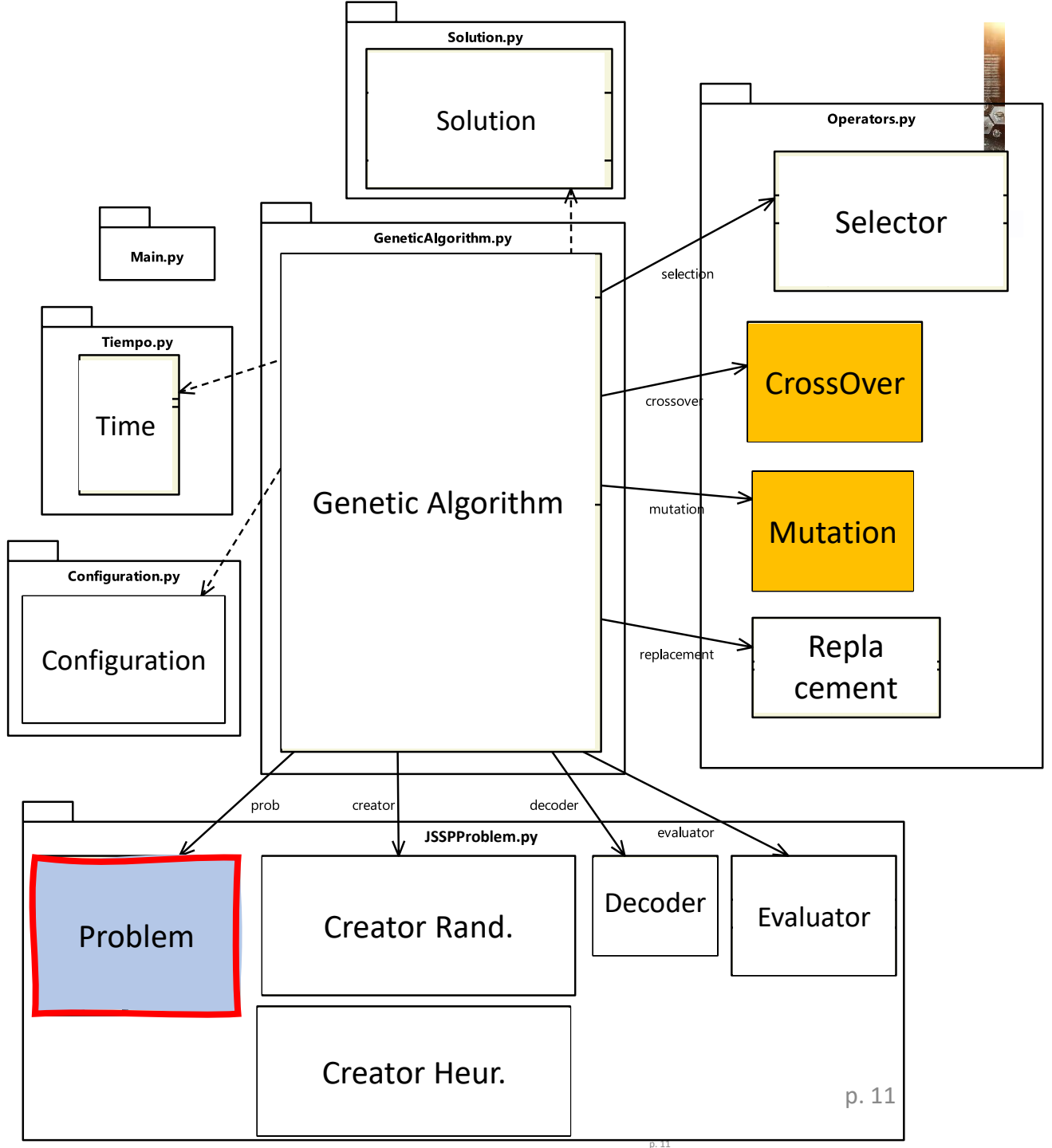
- Un **cromosoma** representa una solución de un problema, y es un individuo de la población. En el caso del problema $J||C_{max}$ éste se puede representar con una permutación de tareas (con o sin repetición).
  - Es deseable que el cromosoma represente una solución factible, que respete las restricciones del problema.
  - Lo bueno o malo que es un cromosoma, nos lo da su valor de **fitness**.
- **Decodificación de un cromosoma**.



  - Para calcular el valor de fitness de un individuo de la población o cromosoma tendremos que decodificarlo. Para ello se ha de construir la solución representada por el cromosoma, es decir obtener a partir de su genotipo, su fenotipo y calcula la función objetivo, lo que nos dará el fitness del cromosoma.
  - Para decodificar un cromosoma podemos realizar una decodificación directa o emplear un Algoritmo Generador de Schedules (SGS), en ambos casos se empleará el orden de tareas expresado por el cromosoma para guiar la construcción de la solución.

Definiendo JSSP

- Operadores específicos de JSSP
  - **Problema**
  - Codificación Cromosomas
    - Generación
    - ¿Cruce?
    - ¿Mutación?
  - Decodificador & Evaluador

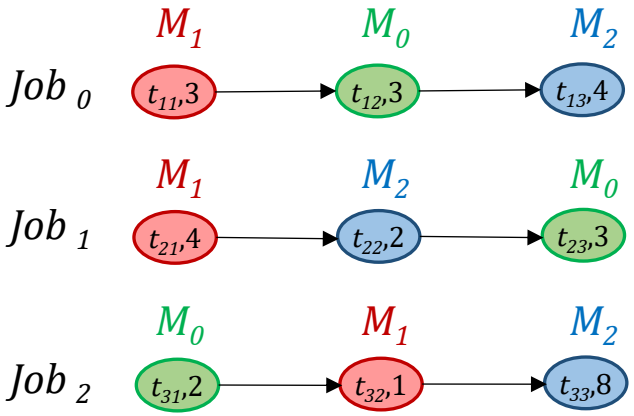


Intelligent Scheduling and Optimization

Recordatorio. JSSP

Datos

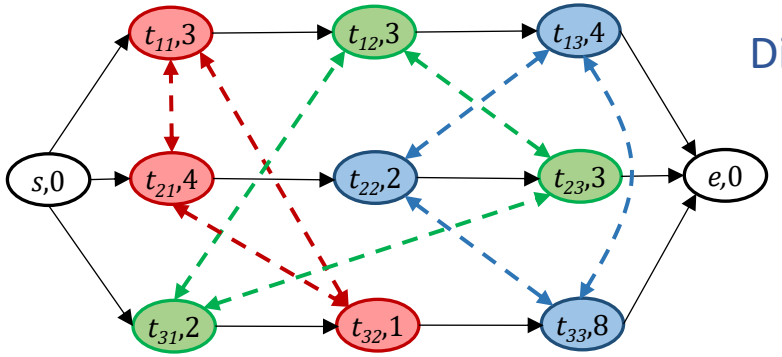
- Número de trabajos y de máquinas
- Secuencia de máquinas que usan las tareas de cada trabajo
- Tiempos de procesamiento (duración) de las tareas, p_{ij}




```
3 trabajos
3 máquinas
Secuencias de máquinas
1 0 2
1 2 0
0 1 2
Duraciones
3 3 4
4 2 3
2 1 8
```

Restricciones

- Secuenciales (de trabajo). Una tarea no puede empezar antes de que termine la anterior en su trabajo
- Capacidad (de máquina). No se pueden solapar la ejecución de dos tareas en una misma máquina
- No-interrupción. No puede interrumpirse la ejecución de las tareas



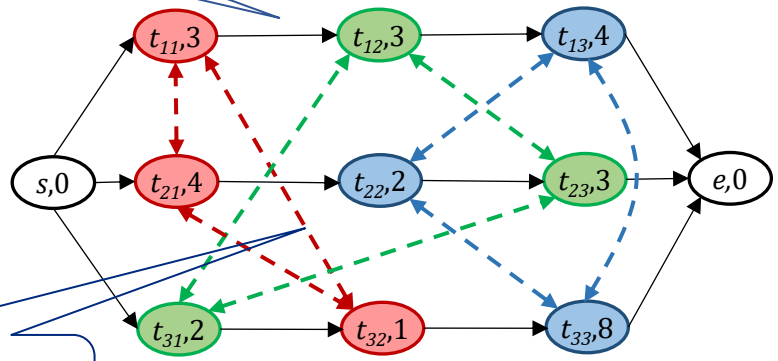
Grafo Disyuntivo

Recordatorio. JSSP

Grafo disyuntivo

- Grafo de restricciones:
  - Vértices: tareas con sus costes. Dos vértices adicionales $\{s, e\}$
  - Arcos: restricciones

R. Secuenciales: *Arcos dirigidos (conjuntivos)*



R. Capacidad: *Arcos no dirigidos o bidireccionales (disyuntivos)*

Solución

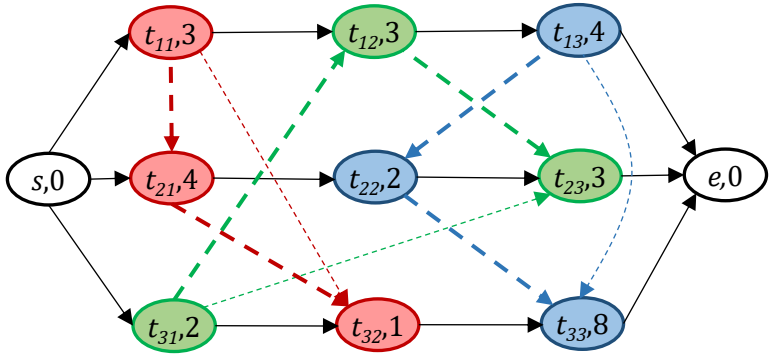
- Orden de procesamiento para las tareas
  - Ordenación topológica (respeta las restricciones)

$$\pi = (s, t_{31}, t_{11}, t_{21}, t_{12}, t_{32}, t_{13}, t_{22}, t_{23}, t_{33}, e)$$

  - (Tiempos de inicio de las tareas)

Resolver un problema

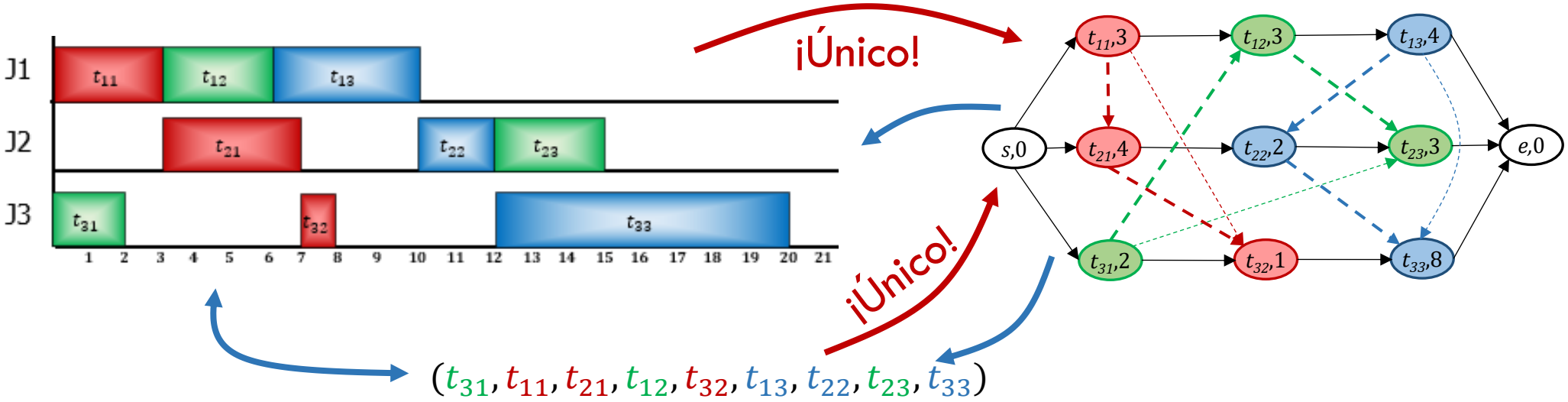
- Elegir un sentido para los arcos disyuntivos (subgrafo)
  - Solución factible = Grafo Dirigido **Acíclico**
- Determinar los tiempos de inicio de las tareas



Recordatorio. JSSP

Diagrama de Gannt

- Representa un Schedule mediante los tiempos de inicio y las duraciones de las tareas



Recordatorio. JSSP

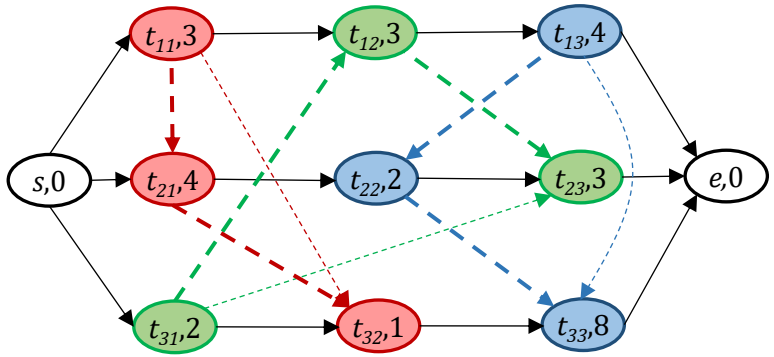
Funciones Objetivo

- $(J|\ \ |C_{max})$ Minimizar el makespan $C_{max} = \max C_{ij}$
- $(J|\ \ |\Sigma C_i)$ Minimizar el flujo total $C_{sum} = \Sigma C_i$
- $(J|\ \ |\Sigma T_i)$ Minimizar el retraso respecto a fechas de entrega de los trabajos (Tardiness),
- $(J|\ \ |L_{max})$ Minimizar el máximo tiempo de retraso (Lateness) …

Coste de una solución

- Calcular la función objetivo de una solución

$$\pi = (s, t_{31}, t_{11}, t_{21}, t_{12}, t_{32}, t_{13}, t_{22}, t_{23}, t_{33}, e)$$



Makespan:

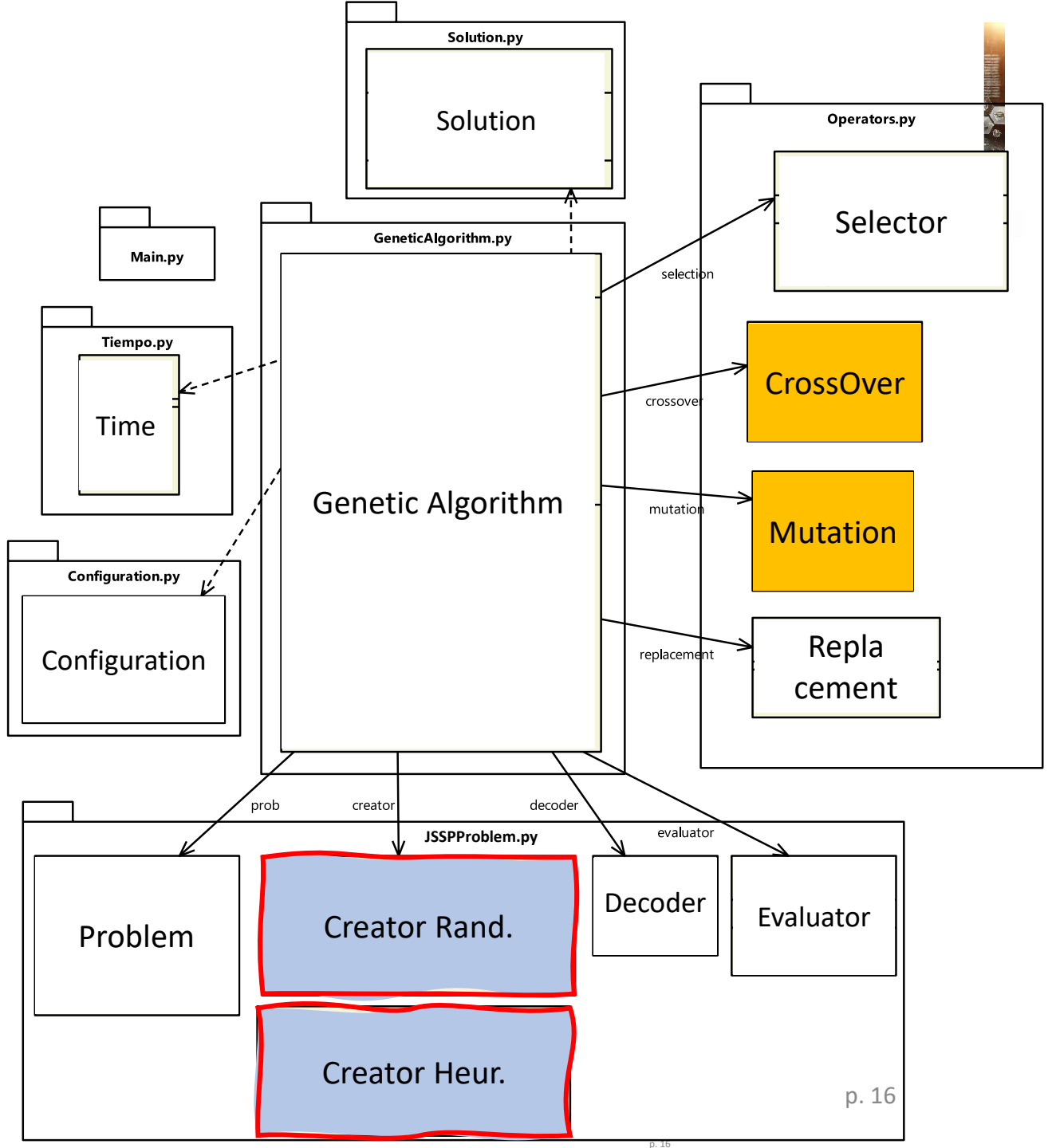
$$C_{max} = max\{10, 15, 20\} = 20$$

Total Completion Time:

$$\sum C_i = \sum_{i=1}^{n} C_i = 10 + 15 + 20 = 45$$

Definiendo JSSP

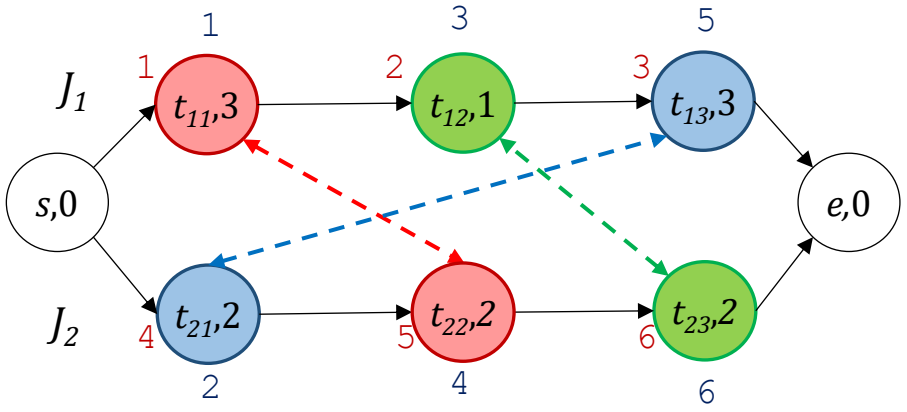
- Operadores específicos de JSSP
  - Problema
  - **Codificación Cromosomas**
    - **Generación**
    - ¿Cruce?
    - ¿Mutación?
  - Decodificador & Evaluador



Genotype

Codificación: Permutaciones sin repetición

Problema $J||C_{max}$



Cromosoma: Orden de tareas.

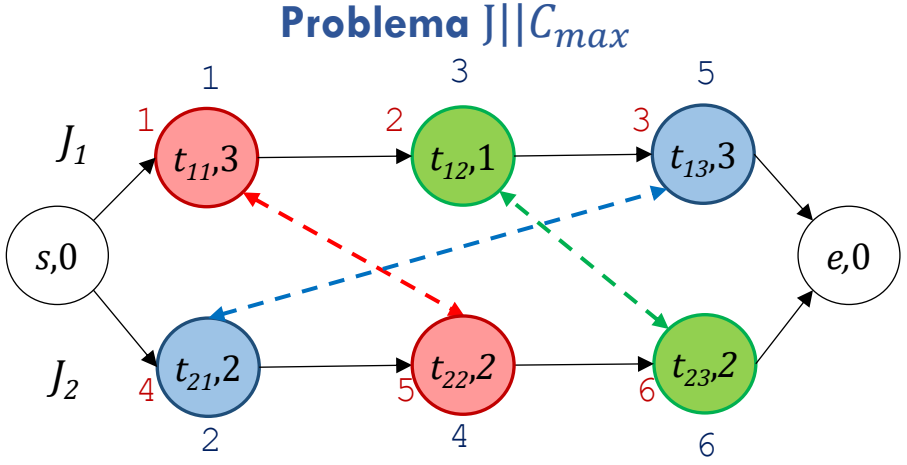
- Permutaciones sin repetición
  - $(t_{21}, t_{22}, t_{11}, t_{12}, t_{13}, t_{23}), (2, 4, 1, 3, 5, 6), (4, 5, 1, 4, 5, 6)$
- **Ventaja:** operadores de cruce y mutación estándar
- **Problema:** podemos tener cromosomas que no representen soluciones factibles y tendrían que ser "reparadas"
  - No respeten las restricciones del problema

Ejemplos:

- $(t_{23}, t_{11}, t_{12}, t_{21}, t_{22}, t_{13}, t_{23})$ La tarea t_{23} no puede ir antes que la tarea t_{21} pues se incumple la restricción de secuencialidad en J_2
- $(2, 3, 4, 1, 5, 6)$ La tarea 3 (t_{12}) no puede ir antes que la tarea 1 (t_{11}) pues se incumple la restricción de secuencialidad en J_1
- $(1, 2, 4, 6, 3, 5)$ La tarea 6 (t_{23}) no puede ir antes que la tarea 5 (t_{22}) pues se incumple la restricción de secuencialidad en J_2

Si queremos emplear esta codificación necesitamos un mecanismo que permita convertir las soluciones en factibles (para evaluarlas) o incluir en el AG un factor de penalización de estos individuos

Codificación: Permutaciones con repetición



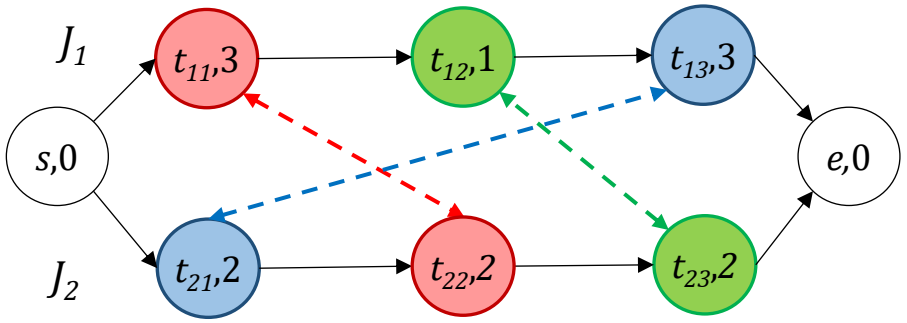
Cromosoma: Orden de tareas.

- Permutaciones con repetición
  - $(2, 2, 1, 1, 1, 2)$
- **Problemas:** Operadores de cruce y mutación no estándar, han de conservar la factibilidad de las soluciones de los cromosomas que generan
- **Ventaja:** la población sólo contiene individuos factibles y la decodificación es más sencilla (una de las operaciones más costosas del AG)

- Esta codificación emplea identificadores de trabajos. Al decodificar…
  - ¿Cómo saber a qué tarea del trabajo nos estamos refiriendo?
    - El primer identificador de un trabajo es la primera tarea del mismo, el segundo la segunda, etc…
    - En el proceso de decodificación debemos llevar la cuenta de las veces que ya ha salido el identificador del mismo trabajo

Codificación: Permutaciones con repetición

Problema $J||C_{max}$



Decodificación directa: planificamos las tareas en el orden expresado por el cromosoma

Cromosoma

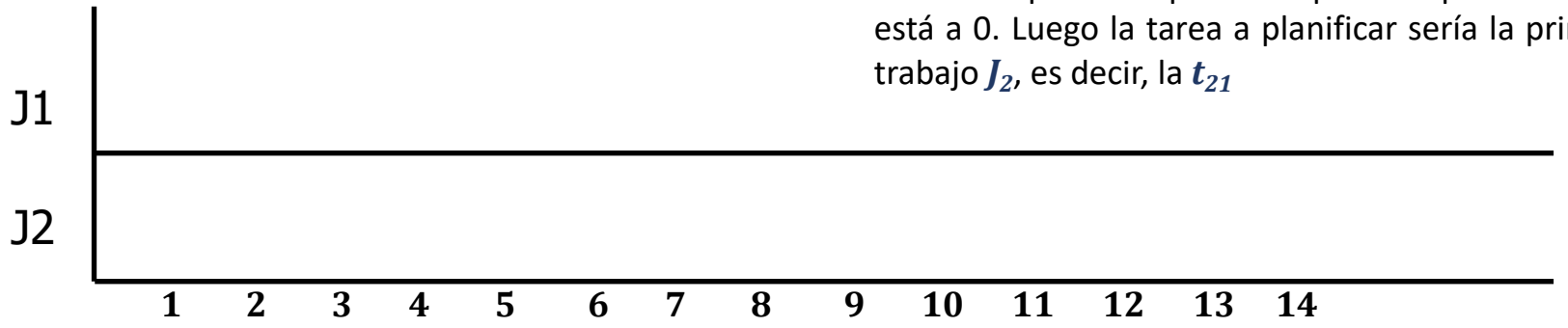
$(2, 2, 1, 1, 1, 2)$

El primer **gen** del cromosoma es el **2**, lo que indica que la tarea a planificar pertenece al trabajo J_2

Primera aparición del 2 (identificador del trabajo J_2). Sabemos que es su primera aparición pues su contador está a 0. Luego la tarea a planificar sería la primera del trabajo J_2, es decir, la t_{21}

Contador

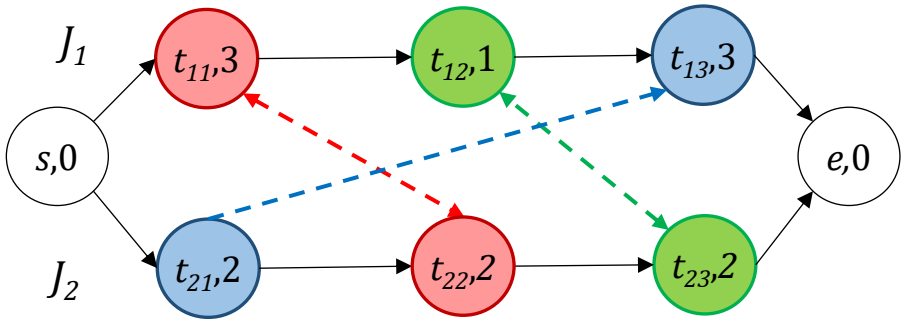
J_1	0
J_2	0



Genotype

Codificación: Permutaciones con repetición

Problema $J||C_{max}$



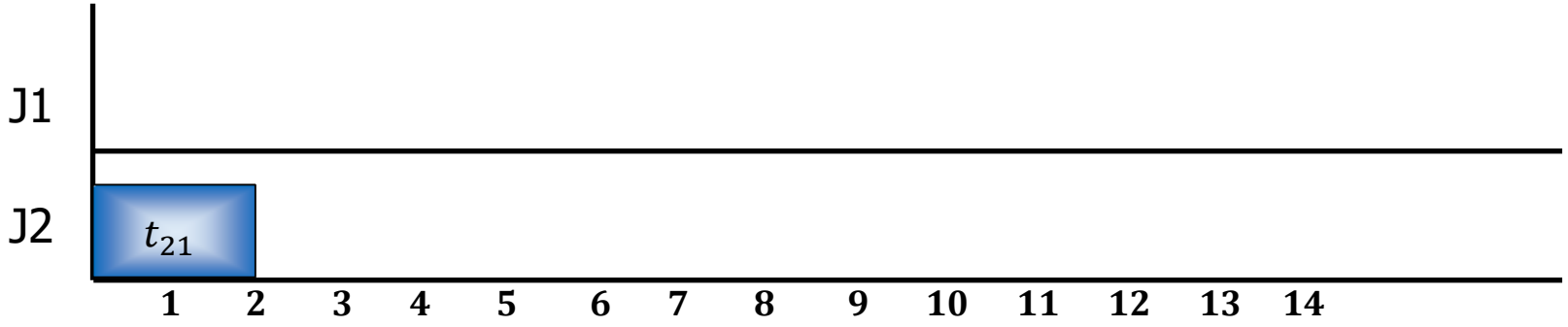
Cromosoma

$(\mathbf{2}, 2, 1, 1, 1, 2)$

Planificamos la primera tarea del trabajo J_2, tarea t_{21} y ponemos el contador para el trabajo J_2 a **1**

Contador

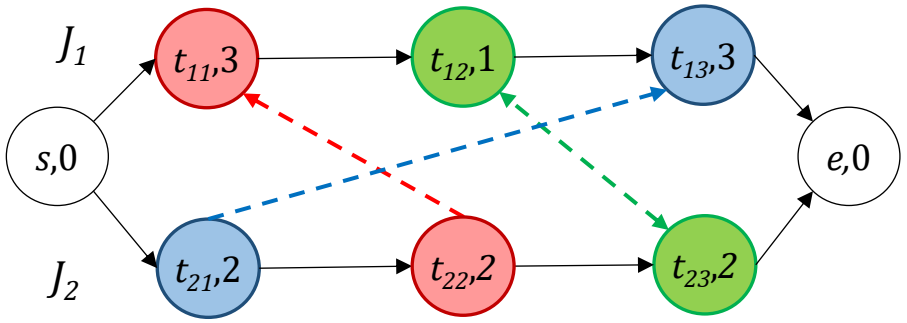
J_1	0
J_2	**1**



Genotype

Codificación: Permutaciones con repetición

Problema $J||C_{max}$



Cromosoma

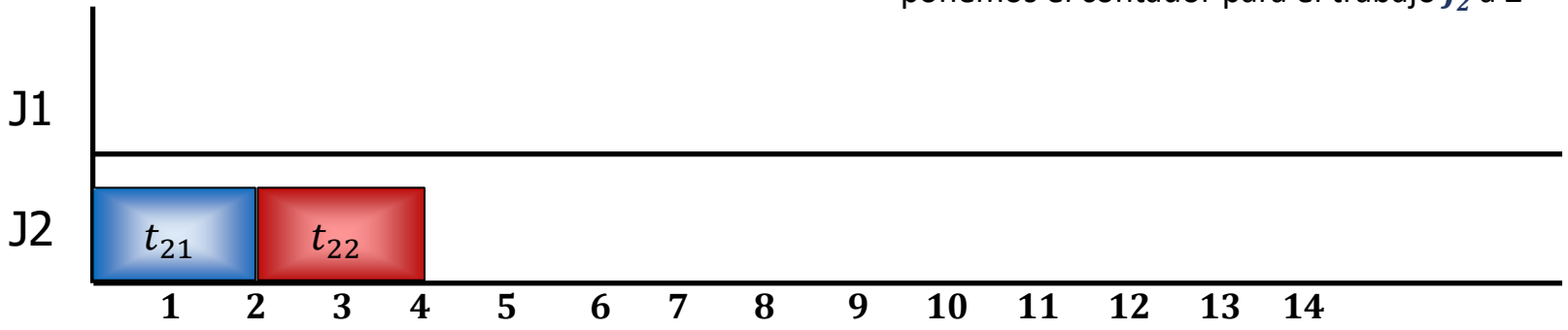
$(2, \mathbf{2}, 1, 1, 1, 2)$

Segunda aparición del 2 (identificador del trabajo J_2).

Planificamos la segunda tarea del trabajo J_2, pues su contador está a 1, luego se planifica la tarea t_{21} y ponemos el contador para el trabajo J_2 a **2**

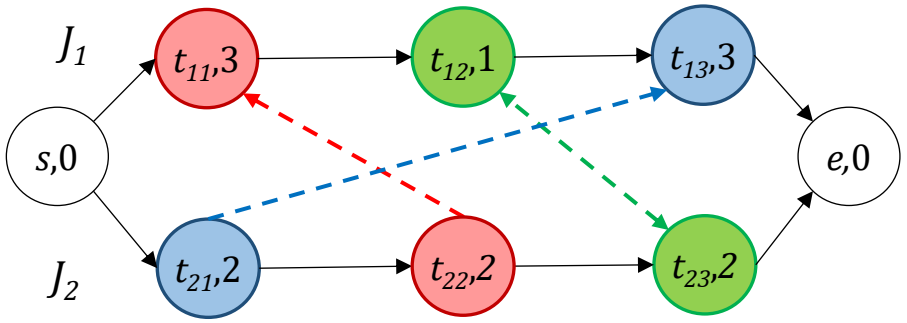
Contador

J_1	0
J_2	**2**



Codificación: Permutaciones con repetición

Problema $J||C_{max}$



Contador

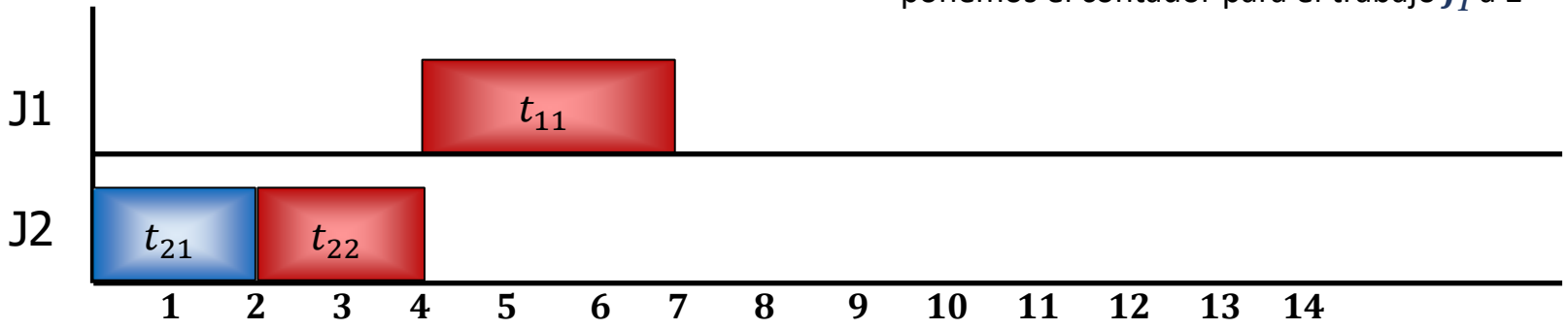
J_1	**1**
J_2	2

Cromosoma

$(2, 2, \mathbf{1}, 1, 1, 2)$

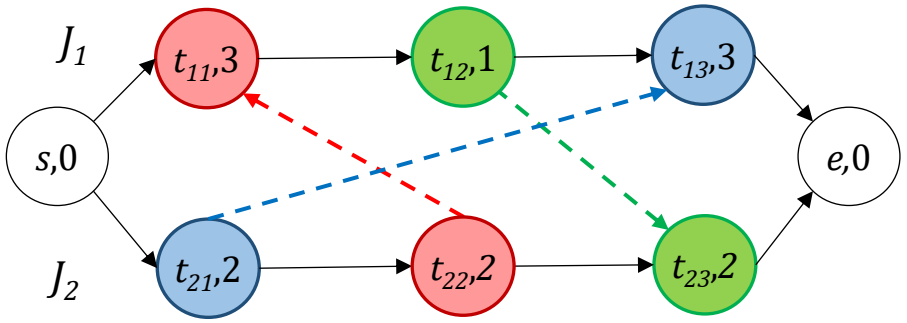
Primera aparición del 1 (identificador del trabajo J_1)

Planificamos la primera tarea del trabajo J_1, pues su contador está a 0, luego se planifica la tarea t_{11} y ponemos el contador para el trabajo J_1 a **1**



Codificación: Permutaciones con repetición

Problema $J||C_{max}$



Cromosoma

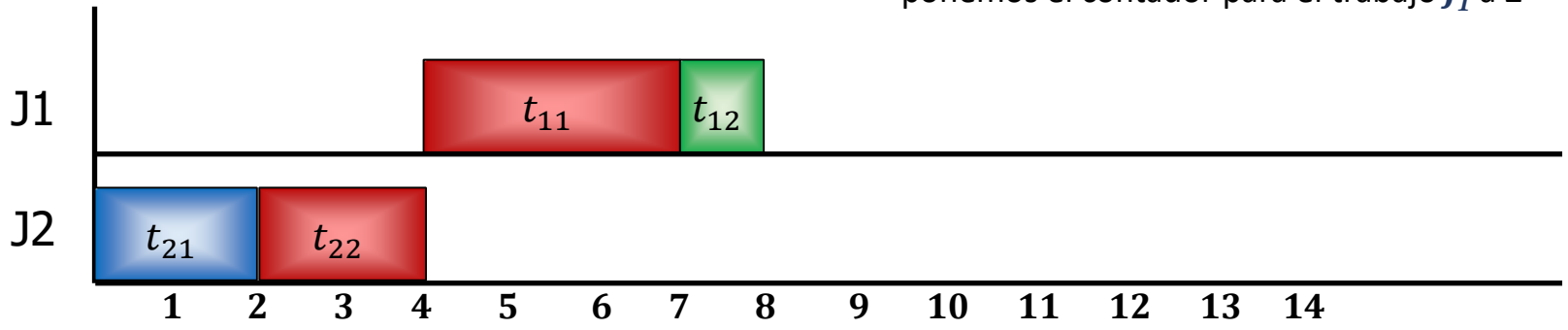
$(2, 2, 1, \mathbf{1}, 1, 2)$

Segunda aparición del 1 (identificador del trabajo J_1)

Planificamos la segunda tarea del trabajo J_1, pues su contador está a 1, luego se planifica la tarea t_{12} y ponemos el contador para el trabajo J_1 a **2**

Contador

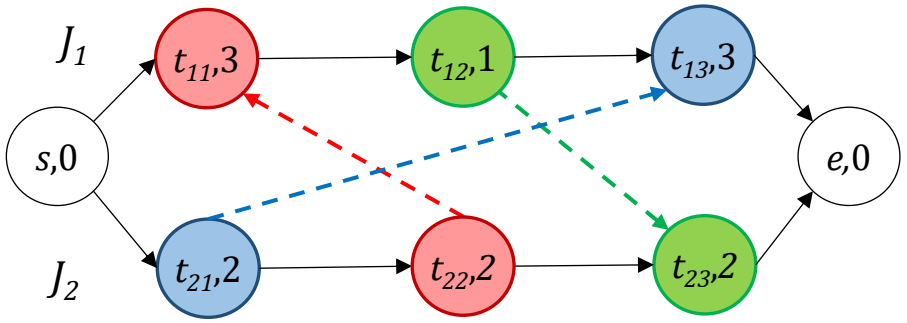
J_1	**2**
J_2	2



Genotype

Codificación: Permutaciones con repetición

Problema $J||C_{max}$



Contador

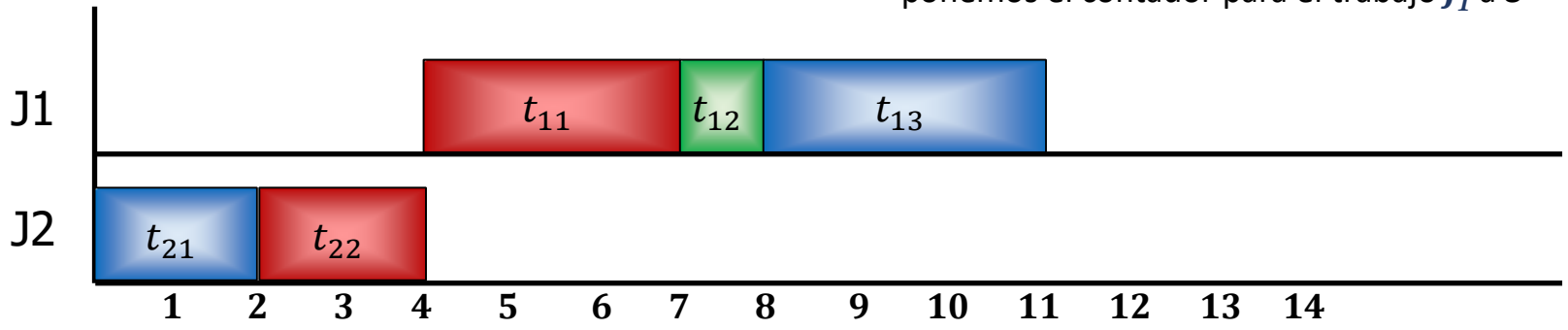
J_1	**3**
J_2	2

Cromosoma

$(2, 2, 1, 1, \mathbf{1}, 2)$

Tercera aparición del 1 (identificador del trabajo J_1)

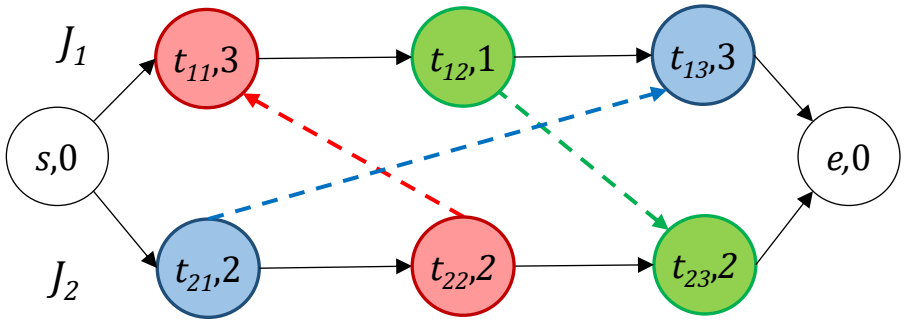
Planificamos la tercera tarea del trabajo J_1, pues su contador está a 2, luego se planifica la tarea t_{13} y ponemos el contador para el trabajo J_1 a **3**



Genotype

Codificación: Permutaciones con repetición

Problema $J||C_{max}$



Contador

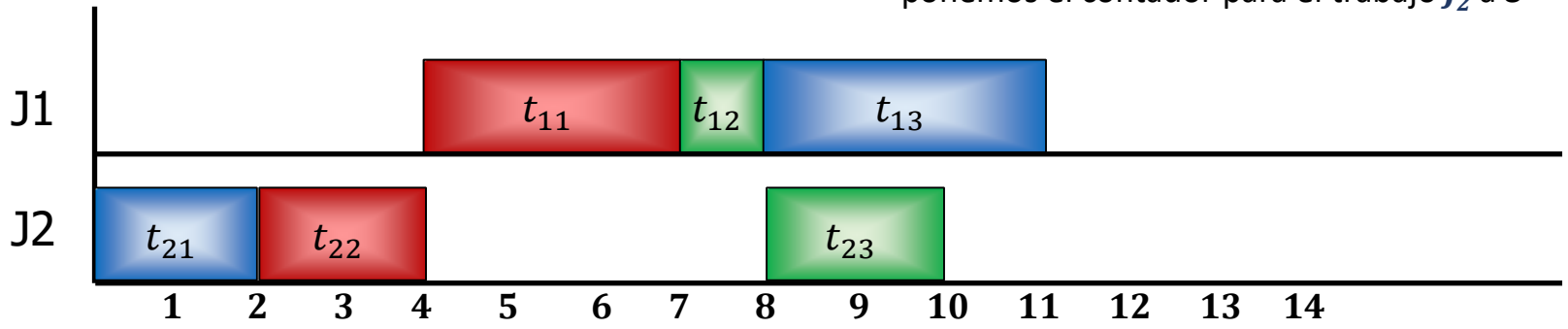
J_1	3
J_2	**3**

Cromosoma

$(2, 2, 1, 1, 1, \mathbf{2})$

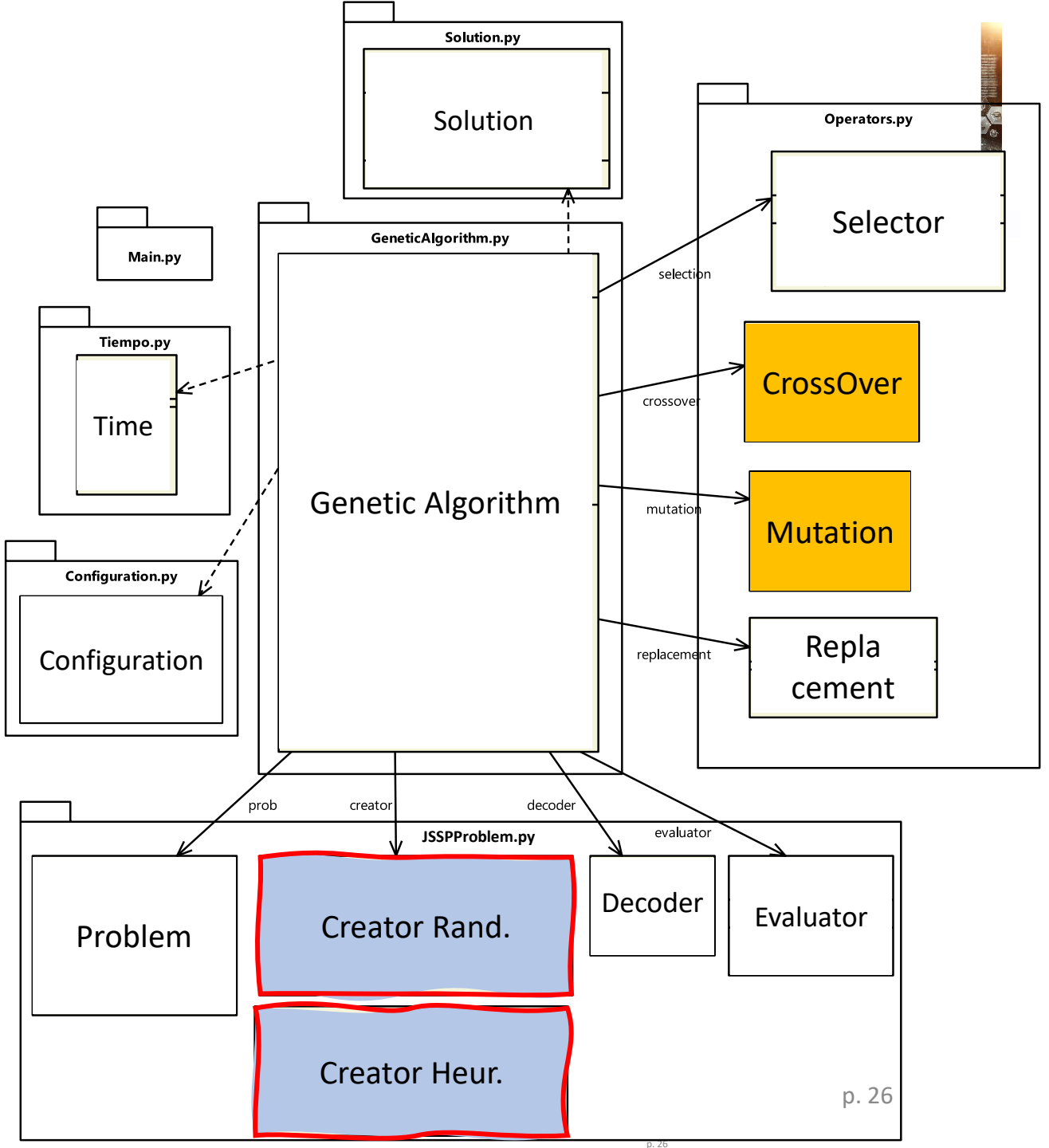
Tercera aparición del 2 (identificador del trabajo J_2)

Planificamos la tercera tarea del trabajo J_2, pues su contador está a 2, luego se planifica la tarea t_{23} y ponemos el contador para el trabajo J_2 a **3**



Definiendo JSSP

- Operadores específicos de JSSP
  - Problema
  - **Codificación Cromosomas**
    - **Generación**
    - ¿Cruce?
    - ¿Mutación?
  - Decodificador & Evaluador

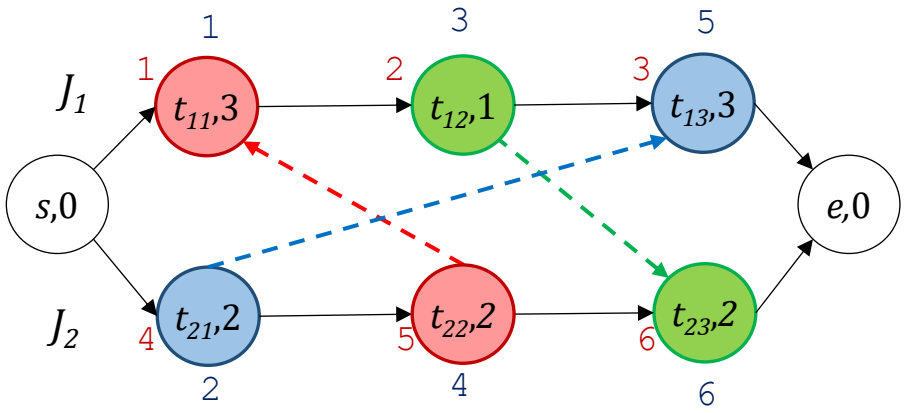


Intelligent Scheduling and Optimization

Genotype

Generación Aleatoria de Soluciones

Problema $J||C_{max}$



Cromosoma:

Secuencia Inicial: Secuencia de identificadores de trabajo, tantas veces como tareas tengan

$$(1, 1, 1, 2, 2, 2)$$

Secuencia final: Permutación aleatoria de la secuencia inicial

Ejemplos: $(1, 2, 2, 1, 2, 1)$ $(2, 2, 2, 1, 1, 1)$ $(1, 2, 1, 2, 1, 2)$ …

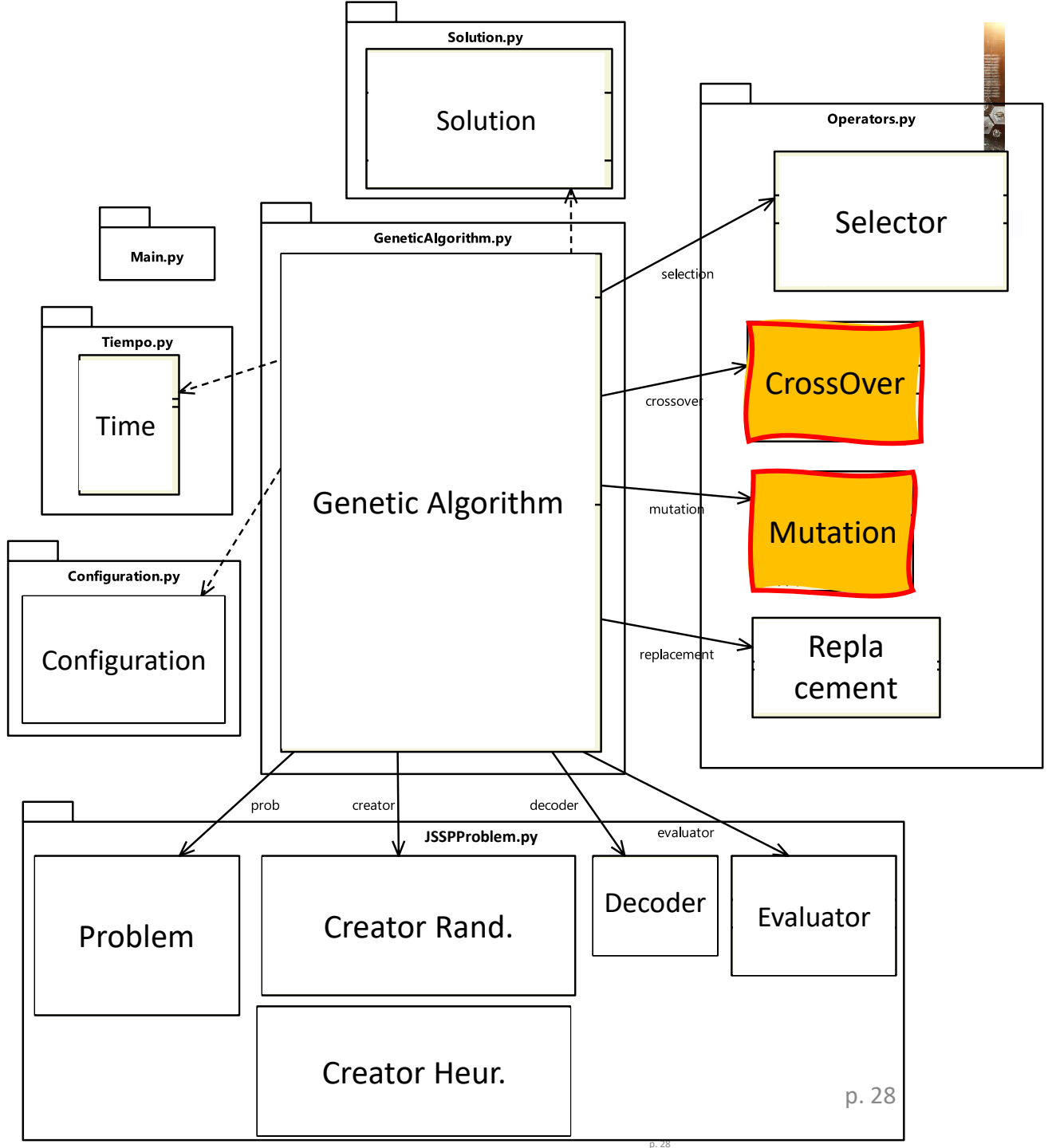
$(\mathrm{t}_{12}, t_{21}, t_{22}, t_{12}, t_{23}, t_{13})$ $(\mathrm{t}_{21}, t_{22}, t_{23}, t_{11}, t_{12}, t_{13})$ $(\mathrm{t}_{11}, t_{21}, t_{12}, t_{22}, t_{13}, t_{23})$

Ventaja (con respecto a Permutaciones Sin Repetición): Todas las permutaciones codifican soluciones factibles

Intelligent Scheduling and Optimization

Definiendo JSSP

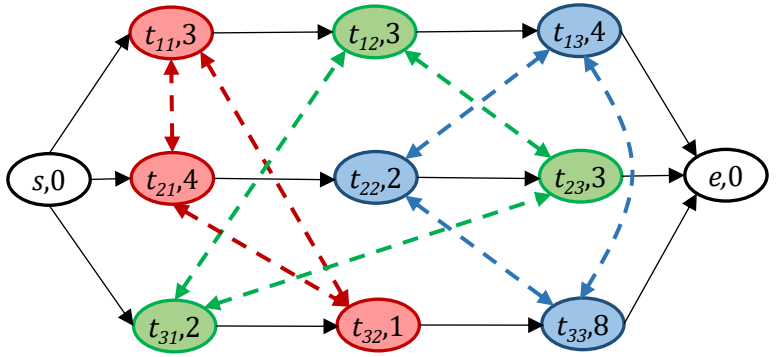
- Operadores específicos de JSSP
  - Problema
  - **Representación Cromosomas**
    - Generación
    - **¿Cruce?**
    - **¿Mutación?**
  - Decodificador & Evaluador



Cruce y Mutación de Cromosomas

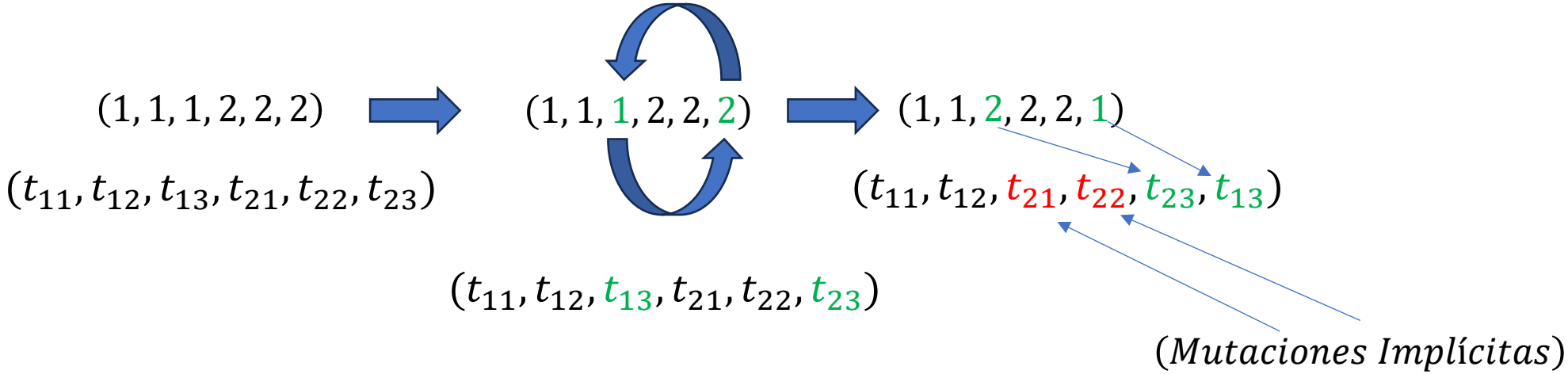
- Cruce
  - Generalized Order Crossover (GOX) [Bierwirth 1996]
  - Generalized Partial Mapped Crossover (GPMX) [Bierwirth 1996]

parent 1	3	2	2	2	3	1	1	1	3
parent 2	1	1	3	2	2	1	2	3	3
GOX offspring	1	3	2	2	2	3	1	1	3
GPMX offspring	1	3	2	2	3	1	2	1	3



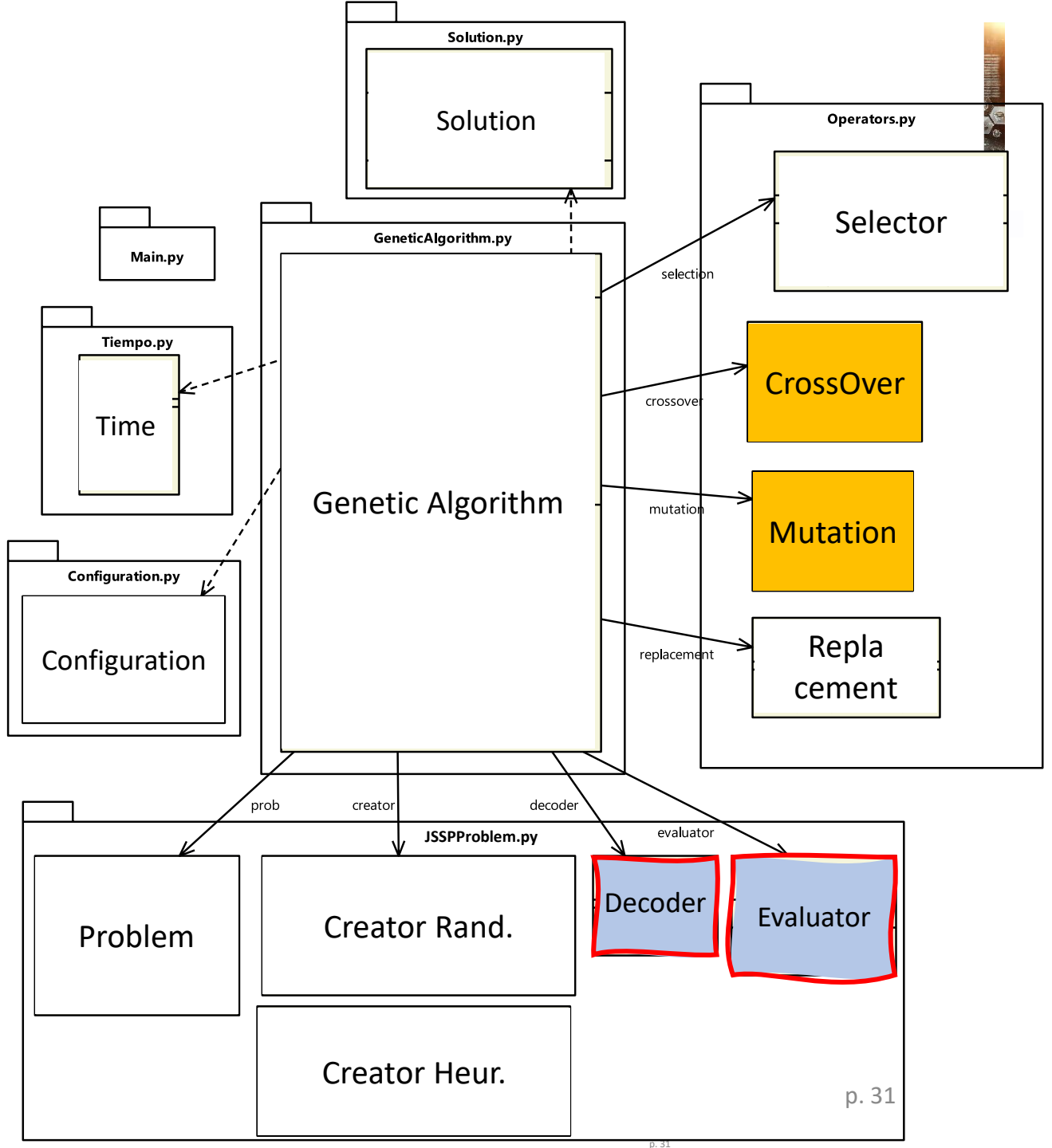
Cruce y Mutación de Cromosomas

- Mutation SWAP
  - TSP: ok
  - JSSP: ok, pero más disruptiva: Mutaciones adicionales implícitas)



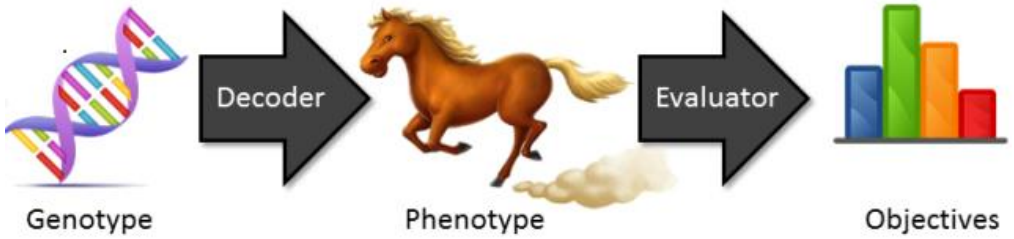
Definiendo JSSP

- Operadores específicos de JSSP
  - Problema
  - Representación Cromosomas
    - Generación
    - ¿Cruce?
    - ¿Mutación?
  - **Decodificador & Evaluador**



Algoritmos Genéticos. Población

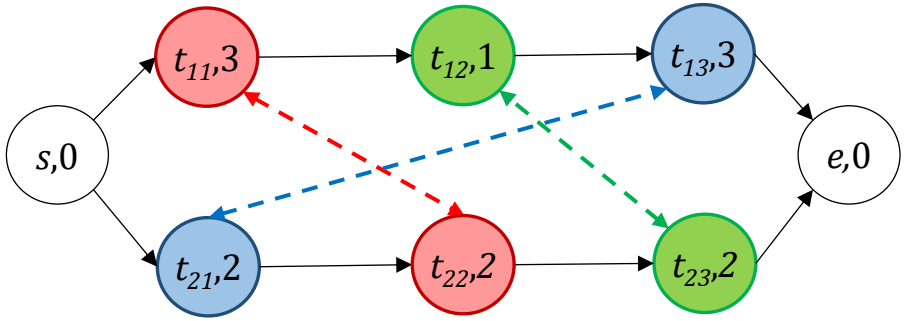
- **Decodificación y Evaluación de un cromosoma.**
  - Para calcular el valor de fitness de un individuo de la población o cromosoma tendremos que decodificarlo. Para ello se ha de construir la solución representada por el cromosoma, es decir obtener a partir de su genotipo, su fenotipo y calcula la función objetivo, lo que nos dará el fitness del cromosoma.
  - Para decodificar un cromosoma podemos realizar una decodificación directa o emplear un Algoritmo Generador de Schedules (SGS), en ambos casos se empleará el orden de tareas expresado por el cromosoma para guiar la construcción de la solución.



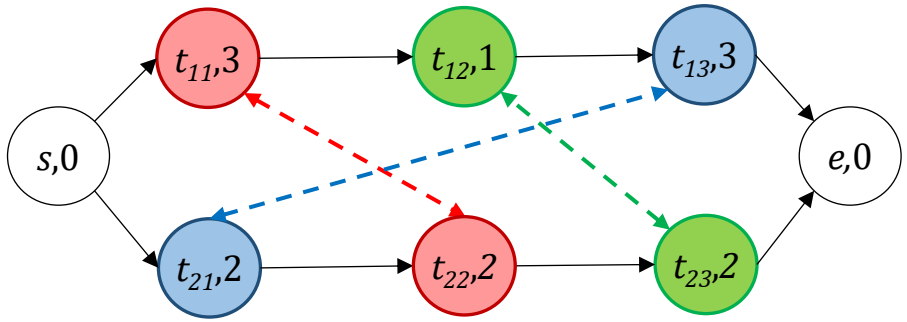
Decodificación directa

- Dado siguiente problema $J||C_{max}$, y el cromosoma que representa el orden topológico π, decodifica la solución representada por el cromosoma mediante una decodificación **directa** y calcula su fitness empleando la función objetivo Makespan ($C_{max} = max\{C_i | \ i = 1, \dots, n\}$). En este problema todos los trabajos llegan en el instante cero.
- Orden topológico $\pi = (s, t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23}, e)$
- **Cromosoma**: $(t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23})$

En la decodificación directa se planifican las tareas en el orden expresado por el cromosoma



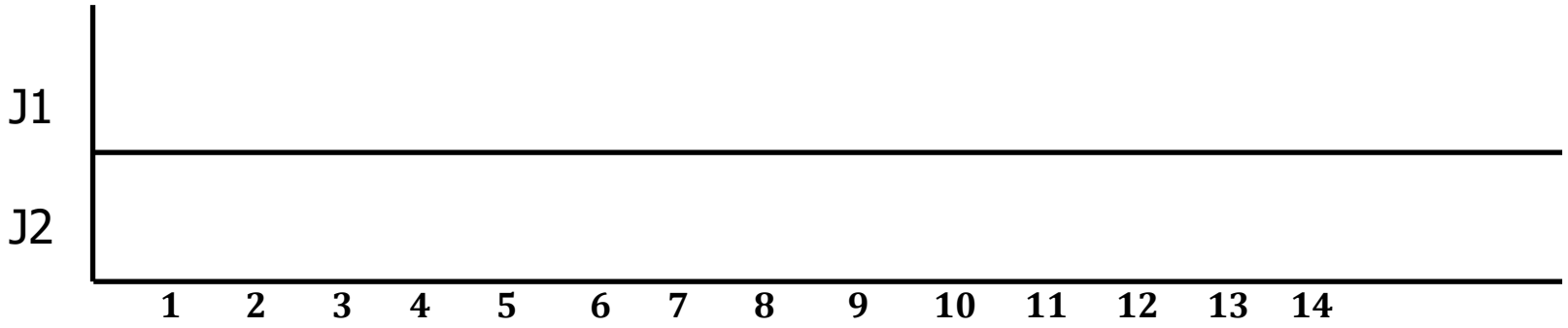
Decodificación directa



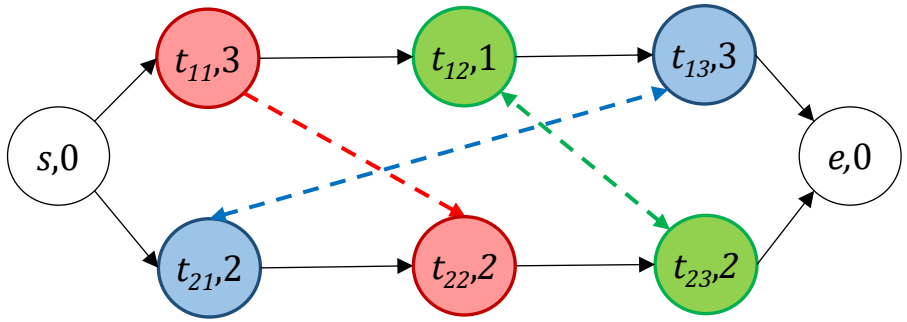
Vamos planificando las tareas en el orden expresado por el cromosoma

Cromosoma

$$(t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23})$$

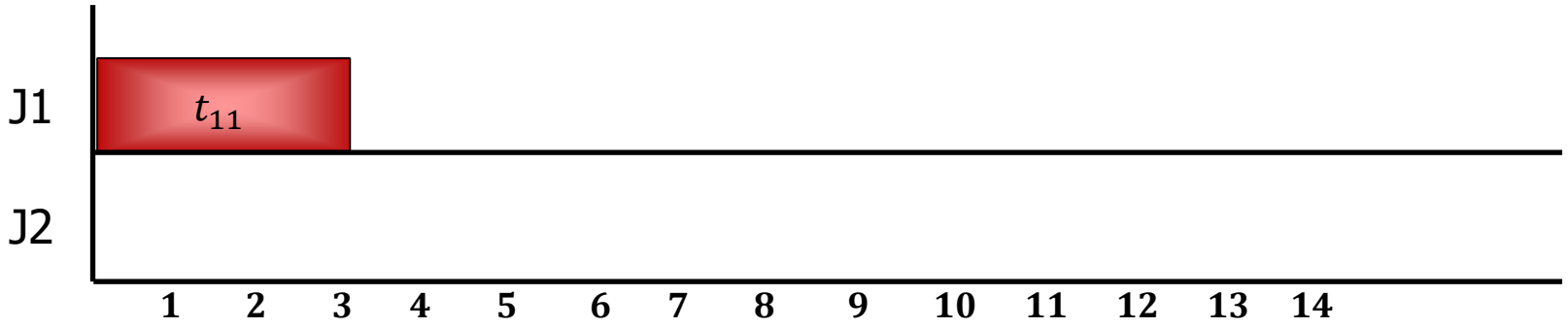


Decodificación directa

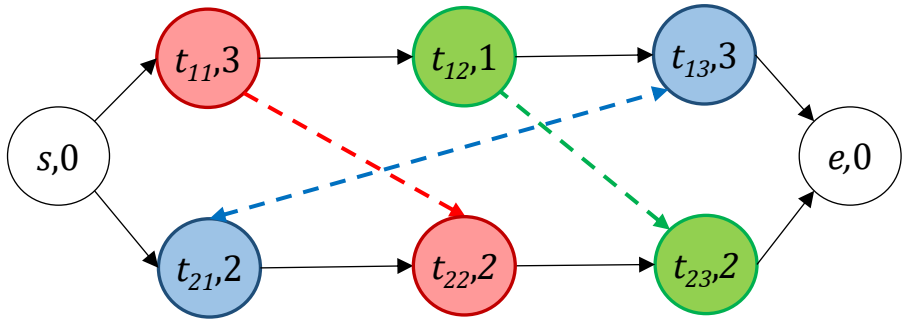


Cromosoma

$$(t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23})$$

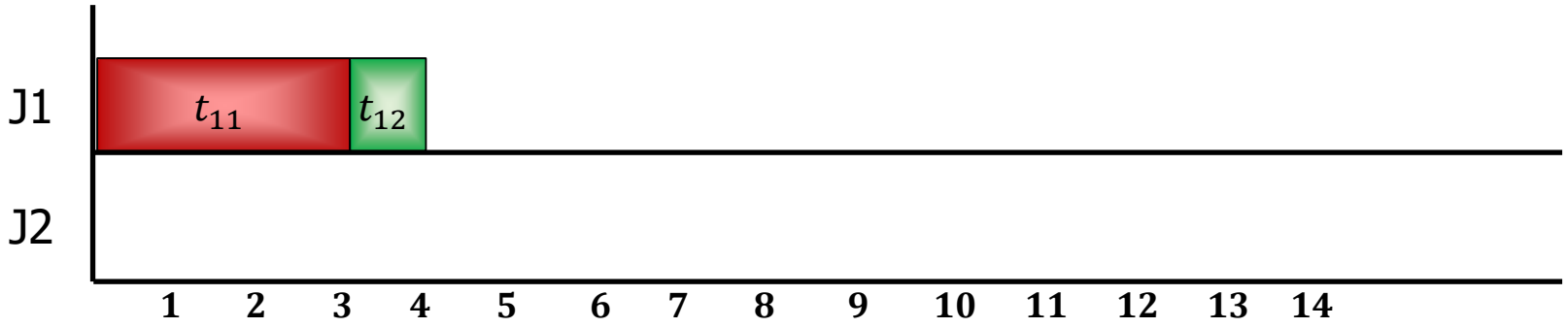


Decodificación directa

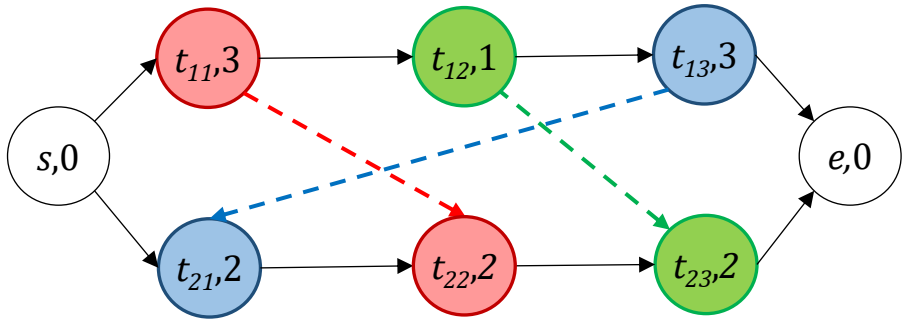


Cromosoma

$$(t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23})$$

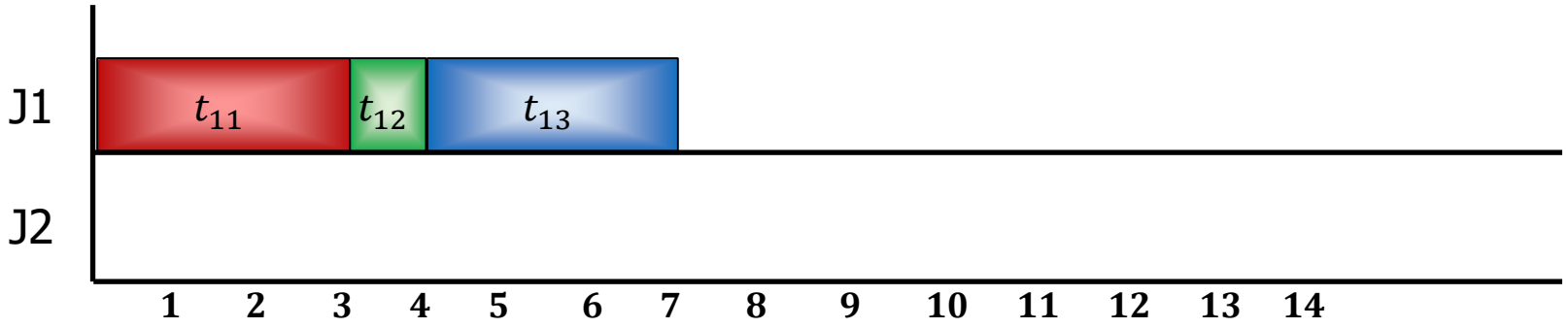


Decodificación directa

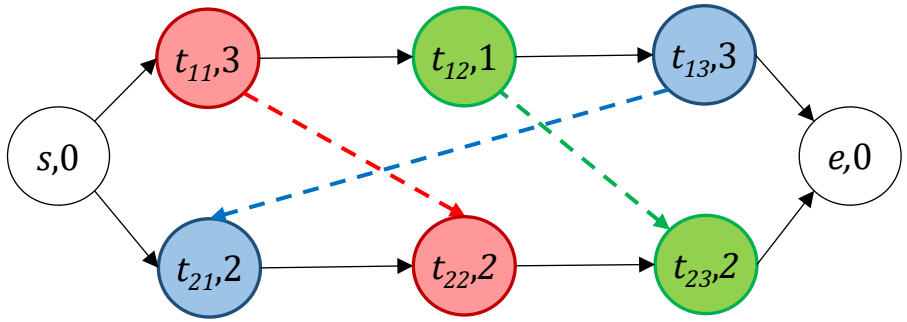


Cromosoma

$$(t_{11}, t_{12}, \boldsymbol{t_{13}}, t_{21}, t_{22}, t_{23})$$

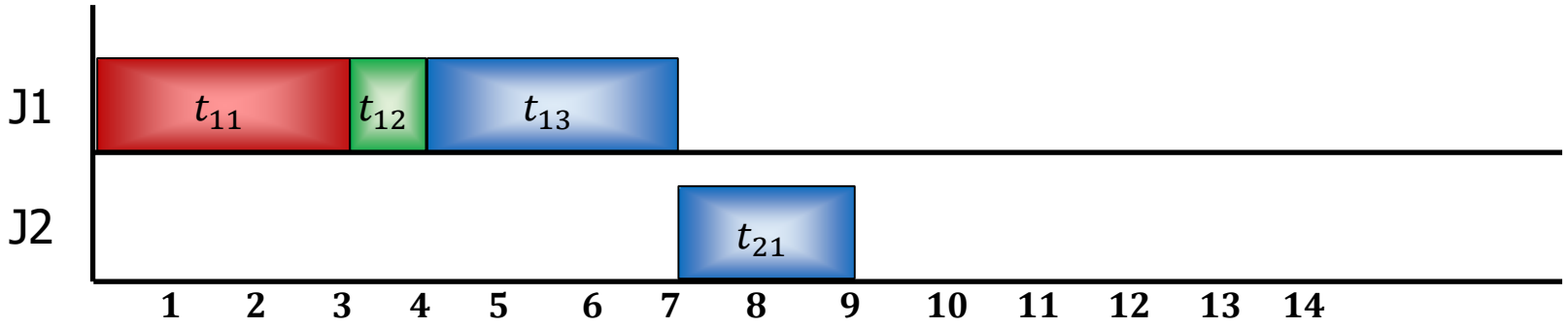


Decodificación directa

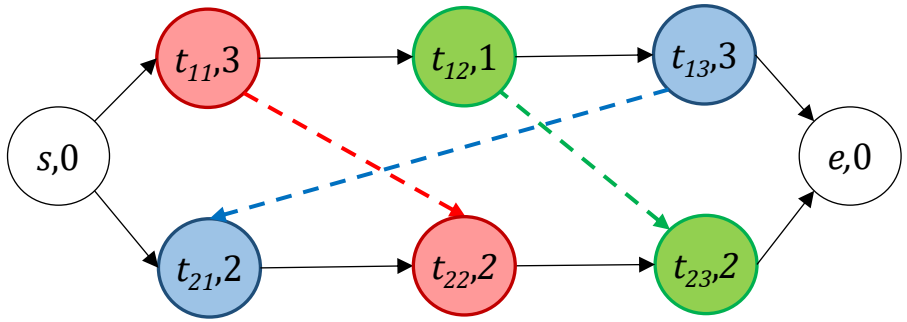


Cromosoma

$$(t_{11}, t_{12}, t_{13}, \boldsymbol{t_{21}}, t_{22}, t_{23})$$

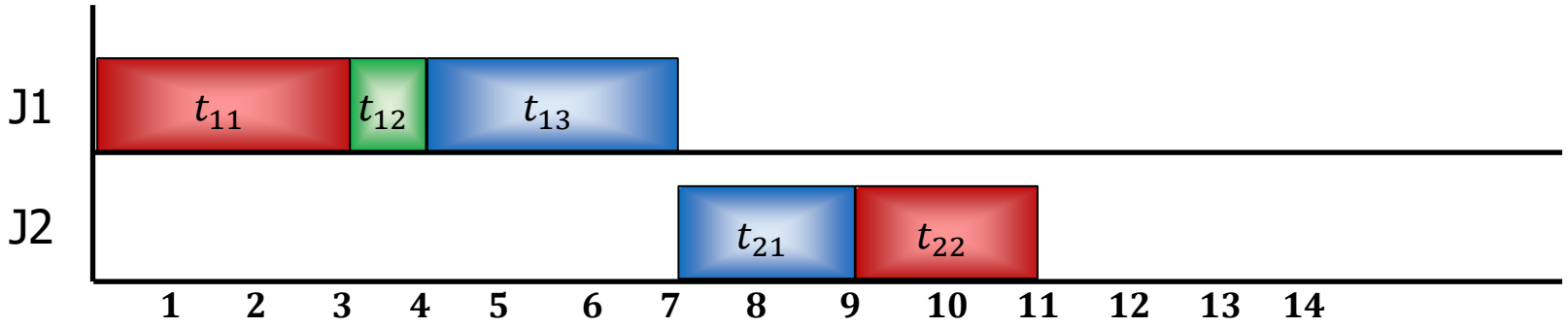


Decodificación directa

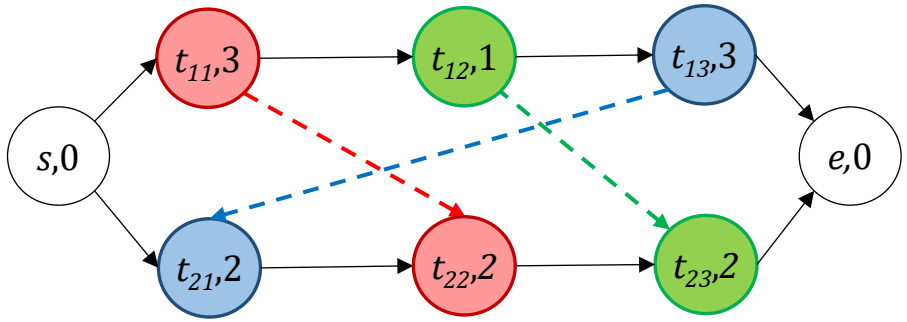


Cromosoma

$$(t_{11}, t_{12}, t_{13}, t_{21}, \boldsymbol{t_{22}}, t_{23})$$

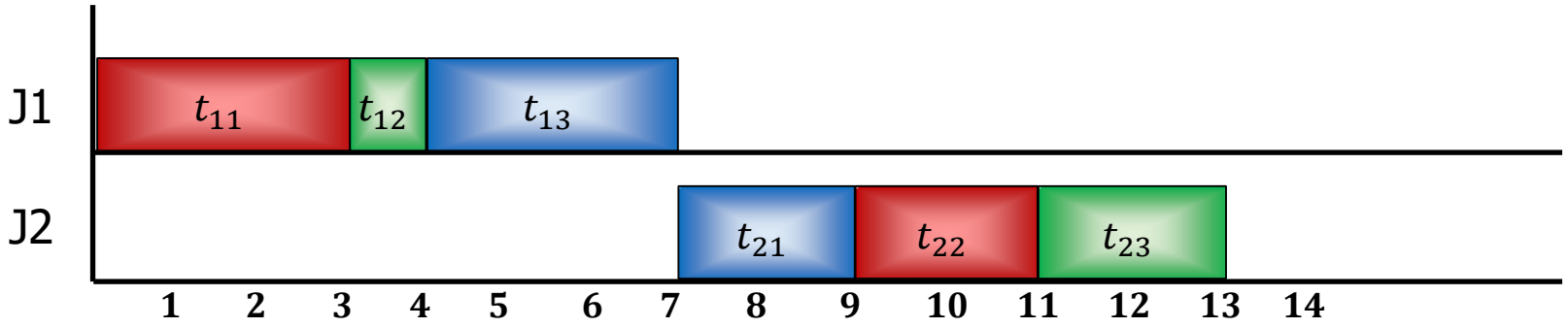


Decodificación directa

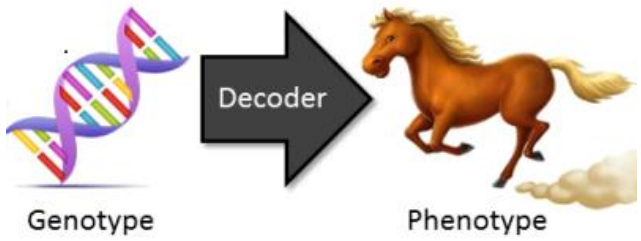


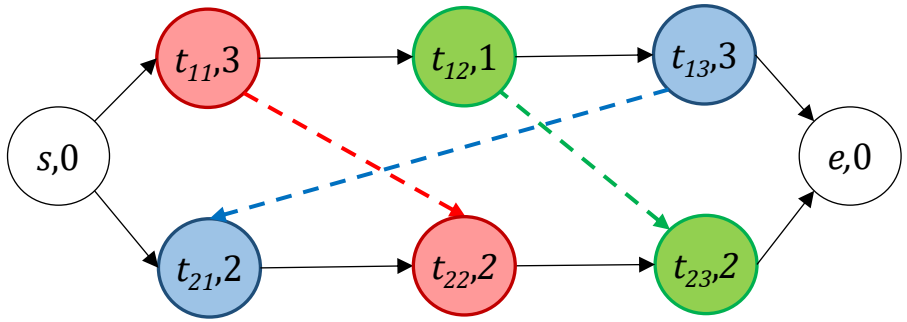
Cromosoma

$$(t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23})$$



Decodificación directa





Cromosoma

$$(t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23})$$

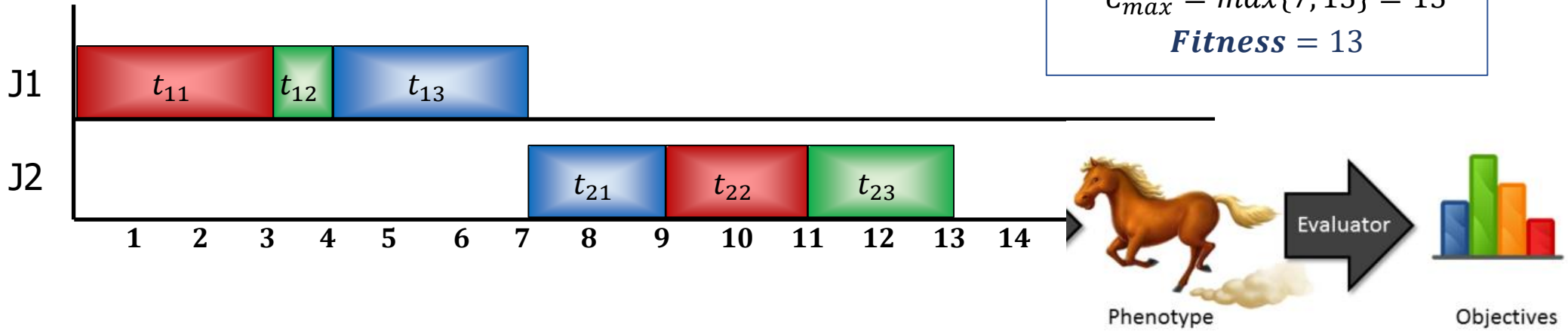
Fenotipo: Solución representada por el cromosoma

Makespan:

$$C_{max} = max\{7, 13\} = 13$$

$$\boldsymbol{Fitness} = 13$$

Planificación semi-activa



Referencias

- [Bierwirth 1996] Bierwirth, C., Mattfeld, D.C., Kopfer, H. (1996). On permutation representations for scheduling problems. In: Voigt, HM., Ebeling, W., Rechenberg, I., Schwefel, HP. (eds) Parallel Problem Solving from Nature — PPSN IV. PPSN 1996. Lecture Notes in Computer Science, vol 1141. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-61723-X_995