

# Metaheurísticas

## Seminario 2. Problemas de optimización con técnicas basadas en búsqueda local

---

### 1. Problema de la Máxima Diversidad (MDP)

- Definición y Variantes del Problema
- Ejemplo de Aplicación
- Análisis del Problema
- Solución Greedy
- Búsquedas por Trayectorias Simples
- Casos del problema
- Agradecimientos

### 2. Problema del Agrupamiento con Restricciones (PAR)

# Definición del Problema

---

- El Problema de la Máxima Diversidad (*Maximum Diversity Problem*, MDP) es un problema de optimización combinatoria con una formulación sencilla pero una resolución compleja (**es NP-completo**)
- El problema general consiste en seleccionar un subconjunto  $SeI$  de  $m$  elementos ( $|SeI|=m$ ) de un conjunto inicial  $S$  de  $n$  elementos (obviamente,  $n > m$ ) de forma que se **maximice** la diversidad entre los elementos escogidos
- Además de los  $n$  elementos ( $e_i, i=1, \dots, n$ ) y el número de elementos a seleccionar  $m$ , se dispone de una matriz  $D=(d_{ij})$  de dimensión  $n \times n$  que contiene las distancias entre ellos

# Variantes del Problema

---

Existen distintos modelos (variantes) del problema que dependen de la forma en la que se calcula la **diversidad**:

- *MaxSum (MDP)*: La diversidad se calcula como la suma de las distancias entre cada par de elementos seleccionados
- *MaxMin (MMDP)*: La diversidad se calcula como la distancia mínima entre los pares de elementos seleccionados
- *Max Mean Model*: La diversidad se calcula como el promedio de las distancias entre los pares de elementos seleccionados
- *Generalized Max Mean*: Existen pesos asociados a los elementos empleados en el denominador al calcular el promedio de distancias (*hay un orden de importancia de los elementos*)

# Definición del Problema MaxSum (MDP)

---

- Así, la definición matemática del problema *MaxSum Diversity Problem* (MDP), **con el que trabajaremos en prácticas**, es:

$$\text{Maximizar } z_{MS}(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij} x_i x_j$$

$$\text{Sujeto a } \sum_{i=1}^n x_i = m$$

$$x_i = \{0, 1\}, \quad i = 1, \dots, n.$$

donde  $x$  es el vector binario solución al problema

- Las distancias entre pares de elementos se usan para formular el modelo como un problema de optimización binario cuadrático
- Esa formulación es poco eficiente. Se suele resolver como un problema equivalente de programación lineal entera

# Definición del Problema MaxMin (MMDP)

---

- La definición matemática del problema *MaxMin Diversity Problem* es:

$$\begin{array}{ll} \text{Maximizar} & z_{MM} = \min_{i < j} d_{ij} x_i x_j \\ \text{s.a.} & \sum_{i=1}^n x_i = m \\ & x_i \in \{0, 1\} \end{array}$$

# Definición del Problema Generalized Max Mean

---

- La definición matemática del problema *Generalized Max Mean Diversity Problem* es:

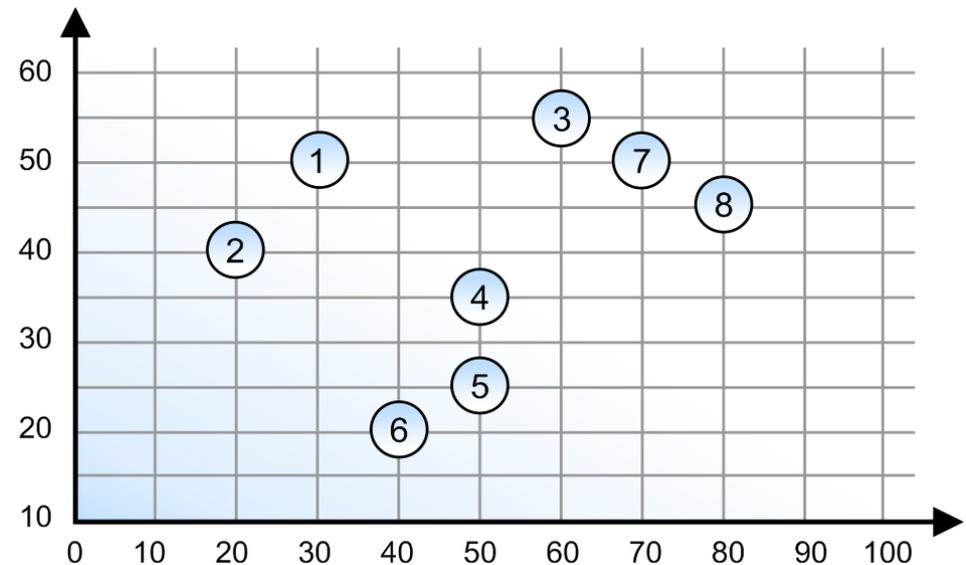
$$\begin{array}{l} \text{Maximizar} \\ \text{s.a.} \end{array} \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij} x_i x_j}{\sum_{i=1}^n w_i x_i}$$
$$\sum_{i=1}^n x_i = m$$
$$x_i \in \{0, 1\}$$

donde  $x$  es el vector binario solución al problema y  $w$  es un vector de pesos que indica la importancia de los elementos ( $w_i \in [0,1]$  y normalmente  $\sum_{i=1}^n w_i = 1$ )

# Ejemplo de Aplicación: Selección de miembros de un Comité

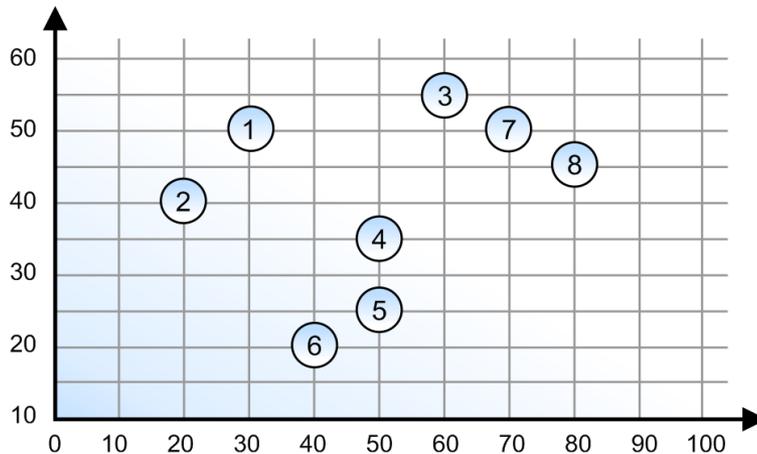
- En un centro médico, tenemos que crear un comité formado por  $m=4$  empleados. Queremos diseñar el comité más diverso de entre los  $n=8$  miembros de plantilla con respecto a su edad e ingresos:

	Edad años	Ingresos miles €/año
1	50	30
2	40	20
3	55	60
4	35	50
5	25	50
6	20	40
7	50	70
8	45	80



# Ejemplo de Aplicación: Selección de miembros de un Comité

- La distancia entre los puntos del gráfico refleja la diferencia entre los empleados representados
- La matriz  $D$  contiene los valores de dichas distancias. En este ejemplo se ha empleado la distancia Euclídea aunque se pueden usar otras métricas



	2	3	4	5	6	7	8
1	14	30	25	32	32	40	50
2		43	30	34	28	51	60
3			22	32	40	11	22
4				10	18	25	32
5					11	32	36
6						42	47
7							11

Matriz de distancias  $D$

# Ejemplo de Aplicación: Selección de miembros de un Comité

## EJEMPLO DEL MODELO MAXSUM (MDP)

- La diversidad entre los elementos escogidos es la **suma** de las distancias existentes entre ellos:

	2	3	4	5	6	7	8
1	14	30	25	32	32	40	50
2		43	30	34	28	51	60
3			22	32	40	11	22
4				10	18	25	32
5					11	32	36
6						42	47
7							11

Empleados seleccionados:

$$x = \{ 3, 4, 6, 8 \}$$

$$\underbrace{22 \ 40 \ 22 \ 18 \ 32 \ 47}$$

suma



$$z_{MS}(x) = 181$$

La solución del modelo **MaxSum** consiste en encontrar el conjunto de 4 empleados con **la mayor suma de distancias** entre ellos

# Ejemplo de Aplicación: Selección de miembros de un Comité

## EJEMPLO DEL MODELO MAXMIN (MMDP) (1/3)

- La diversidad entre los elementos escogidos es el **mínimo** de las distancias existentes entre ellos:

	2	3	4	5	6	7	8
1	14	30	25	32	32	40	50
2		43	30	34	28	51	60
3			22	32	40	11	22
4				10	18	25	32
5					11	32	36
6						42	47
7							11

Sol 1: Empleados seleccionados:

$$x = \{ 3, 4, 6, 8 \}$$

22 40 22 18 32 47

mínimo



$$z_{MS}(x_1) = 18$$

La solución del modelo **MaxMin** consiste en encontrar el conjunto de 4 empleados con **la mayor distancia mínima** entre ellos

# Ejemplo de Aplicación: Selección de miembros de un Comité

## EJEMPLO DEL MODELO MAXMIN (MMDP) (2/3)

- La diversidad entre los elementos escogidos es el **mínimo** de las distancias existentes entre ellos:

	2	3	4	5	6	7	8
1	14	30	25	32	32	40	50
2		43	30	34	28	51	60
3			22	32	40	11	22
4				10	18	25	32
5					11	32	36
6						42	47
7							11

Sol 2: Empleados seleccionados:

$$x = \{ 1, 3, 6, 7 \}$$

30 32 40 40 11 42

mínimo



$$z_{MS}(x_2) = 11$$

La solución del modelo **MaxMin** consiste en encontrar el conjunto de 4 empleados con **la mayor distancia mínima** entre ellos

# Ejemplo de Aplicación: Selección de miembros de un Comité

## EJEMPLO DEL MODELO MAXMIN (MMDP) (3/3)

- La diversidad entre los elementos escogidos es el **mínimo** de las distancias existentes entre ellos:

Sol 1: Empleados seleccionados:

$$x = \{ 3, 4, 6, 8 \}$$

22 40 22 18 32 47

mínimo

$$z_{MS}(x_1) = 18$$

Sol 2: Empleados seleccionados:

$$x = \{ 1, 3, 6, 7 \}$$

30 32 40 40 11 42

mínimo

$$z_{MS}(x_2) = 11$$

La solución del modelo **MaxMin** consiste en encontrar el conjunto de 4 empleados con **la mayor distancia mínima** entre ellos

# Otras Aplicaciones

---

- **Planificación de Mercado:** Maximización del número y la diversidad en las fortalezas del perfil de una marca comercial

Keely, A. (1989). From Experience– Maxi-niching the Way to a Strong Brand: Positioning According to System Dynamics, *Journal of Product Innovation Management* 6:3, 202-206

- Cultivo de plantas, preservación ecológica y gestión de recursos genéticos
- Diseño de productos
- Gestión de equipos de trabajo en empresas (*workforces*)
- Diseño de currícula
- **Otras áreas:** contabilidad y auditoría, experimentación química, diseño experimental, investigación médica, exploración geológica, selección de portafolios, ingeniería estructural, ...

Kuo, C.C., Glover, F., Dhir, K.S. (1993). Analyzing and modeling the maximum diversity problem by zero-one programming, *Decision Sciences* 24:6, 1171-1185

# Análisis del Problema

## CORRELACIÓN ENTRE LOS DISTINTOS MODELOS

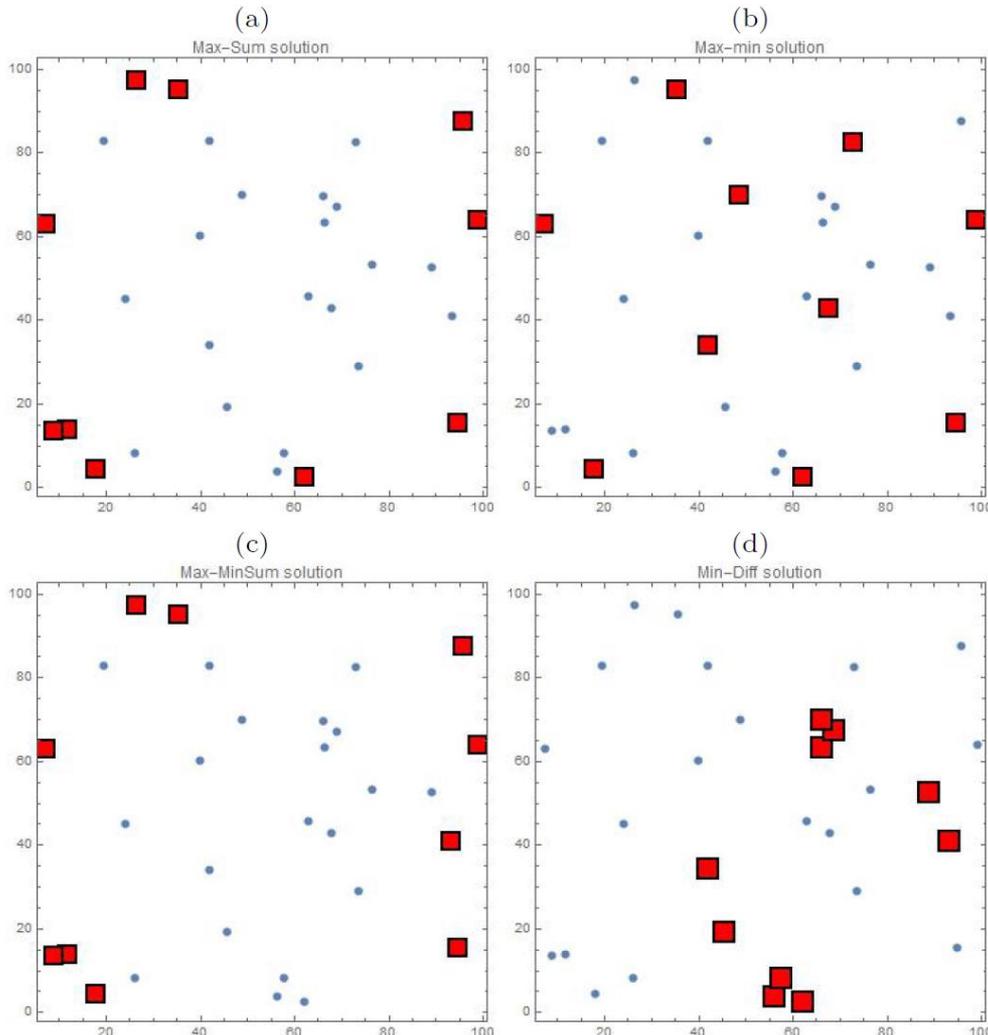
- Aunque los modelos están relacionados, no se puede esperar que un método de resolución diseñado para uno concreto funcione bien en el resto
- El grupo de investigación del Prof. Rafael Martí realizó un experimento con 30 casos aleatorios con tamaños  $n=20$  y  $m=5$ . Calcularon la correlación entre las soluciones óptimas para cuatro modelos:

<b>Opticom</b> Research Group	Max-Sum	Max-Min	Max-MinSum	Min-Diff
Max-Sum	1	0.60*	0.96*	-0.17
Max-Min	0.60*	1	0.73*	-0.63*
Max-MinSum	0.96*	0.73	1	-0.44
Min-Diff	-0.17	-0.63*	-0.44	1

\* diferencias significativas de acuerdo a un test estadístico

# Análisis del Problema

## DIFERENCIAS EN LAS SOLUCIONES ÓPTIMAS



- No se trata únicamente de diferencias en los valores de las funciones objetivo sino en la propia composición de las soluciones

# Solución Greedy

---

Glover, Kuo, Dhir. Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences* 19:1 (1998) 109–132

- La complejidad del problema ha provocado que se hayan aplicado muchos algoritmos aproximados para su resolución
- Podemos determinar que una buena fórmula heurística para resolver el problema es:

*Añadir secuencialmente el elemento no seleccionado que más diversidad aporte con respecto a los ya seleccionados*

# Solución Greedy

---

- El algoritmo Greedy de Glover hace uso del concepto de elemento más central de un conjunto  $X$ :

$$s\_central(X) = \frac{\sum_{s_i \in X} s_i}{|X|}$$

- El primer elemento seleccionado es el de mayor distancia del elemento más central
- Cada vez que se añade un nuevo elemento al conjunto de seleccionados  $Sel$ , se actualiza el elemento más central
- El proceso itera hasta seleccionar los  $m$  elementos deseados

# Solución Greedy

---

## ALGORITMO GREEDY MDP DE GLOVER 1998:

---

1.  $Sel = \emptyset$

2. Compute  $s_c = s\_center(S)$

**Centroide del conjunto  
INICIAL de elementos  $S$**

**while** ( $|Sel| < m$ )

3. Let  $i^* / d(s_{i^*}, s_c) = \max_{s_i \in S} \{d(s_i, s_c)\}$

4.  $Sel = Sel \cup \{s_{i^*}\}$

5.  $S = S - \{s_{i^*}\}$

6.  $s_c = s\_center(Sel)$

**Centroide del conjunto  
ACTUAL de elementos  
seleccionados  $Sel$**

**end while**

---

# Solución Greedy

---

- Como en nuestros casos del problema no dispondremos de los valores concretos de los elementos sino sólo de las distancias entre ellos, no podemos calcular los centroides de los conjuntos
- Por ello, modificaremos levemente el algoritmo:

1.  $Sel = \emptyset$

2. Calcular la distancia acumulada de cada elemento al resto:  $DistAc(s_i) = \sum_{s_j \in S} d(s_i, s_j)$ , incluir el elemento  $s_{i^*}$  que la maximice en  $Sel$ :  $Sel = Sel \cup \{s_{i^*}\}$  y eliminarlo de  $S$ :  $S = S - \{s_{i^*}\}$

**while** ( $|Sel| < m$ )

3. Calcular las distancias de los elementos no seleccionados,  $s_i \in S$ , al conjunto de elementos seleccionados  $Sel$ :  $Dist(s_i, Sel) = \min_{s_j \in Sel} d(s_i, s_j)$

4. Incluir el elemento  $s_{i^*}$  que la maximice en  $Sel$ :  $Sel = Sel \cup \{s_{i^*}\}$

5. Eliminar el elemento  $s_{i^*}$  seleccionado de  $S$ :  $S = S - \{s_{i^*}\}$

**end while**

# Búsquedas por Trayectorias Simples: Búsqueda Local del Mejor

---

- **Representación:** Problema de selección: un conjunto  $Se/\equiv\{s_1, \dots, s_m\}$  que almacena los  $m$  elementos seleccionados de entre los  $n$  elementos del conjunto  $S$

Para ser una solución candidata válida, tiene que **satisfacer las restricciones (ser un conjunto de tamaño  $m$ )**:

- No puede tener elementos repetidos
- Ha de contener exactamente  $m$  elementos
- El orden de los elementos no es relevante

# Búsquedas por Trayectorias Simples: Búsqueda Local del Mejor

---

- **Operador de vecino de intercambio y su entorno:** El entorno de una solución  $Sel$  está formado por las soluciones accesibles desde ella a través de un movimiento de intercambio

Dada una solución (conjunto de elementos seleccionados) se escoge un elemento y se intercambia por otro que no estuviera seleccionado ( $Int(Sel, i, j)$ ):

$$Sel = \{s_1, \dots, i, \dots, s_m\} \Rightarrow Sel' = \{s_1, \dots, j, \dots, s_m\}$$

- $Int(Sel, i, j)$  verifica las restricciones: si la solución original  $Sel$  es factible y el elemento  $j$  se escoge de los no seleccionados en  $Sel$ , es decir, del conjunto  $S - Sel$ , siempre genera una solución vecina  $Sel'$  factible

# Búsquedas por Trayectorias Simples: Búsqueda Local del Mejor

---

- Su aplicación provoca que el tamaño del entorno sea:

$$|E(Sel)| = m \cdot (n - m)$$

- La BL del Mejor del MDP explora todo el vecindario, las soluciones resultantes de los  $m \cdot (n - m)$  intercambios posibles, escoge el mejor vecino y se mueve a él siempre que se produzca mejora
- Si no la hay, detiene la ejecución y devuelve la solución actual
- El método funciona bien pero es muy lento incluso para casos no demasiado grandes ( $n=500$ ) y usando un **cálculo factorizado del coste  $z_{MS}(Sel)$**  para acelerar la ejecución ( $O(n)$ )
- Es recomendable utilizar una estrategia avanzada más eficiente

# Búsqueda Local del Primer Mejor para el MDP

---

Duarte, Martí, Tabu search and GRASP for the maximum diversity problem, European Journal of Operational Research 178 (2007) 71–84

- Algoritmo de **búsqueda local del primer mejor**: en cuanto se genera una solución vecina que mejora a la actual, se aplica el movimiento y se pasa a la siguiente iteración
  - Se detiene la búsqueda cuando se ha explorado el vecindario completo sin obtener mejora (o tras un número fijo de evaluaciones)
- Se **explora el vecindario de forma inteligente**:
  - Se calcula la **contribución** de cada elemento seleccionado al coste de la solución actual (valor de la función objetivo  $z_{MS}(Se/)$ )
  - Se aplican primero los intercambios de elementos que menos contribuyen
- Se considera una **factorización para calcular el coste** de  $Se/$  a partir del de  $Se/$  considerando sólo el cambio realizado en la función objetivo por el movimiento aplicado. Además, se **“factoriza” también el cálculo de la contribución**

# BL-MDP: Exploración Inteligente del Vecindario

---

- Técnica que permite focalizar la BL en una zona del espacio de búsqueda en la que potencialmente puede ocurrir algo
- Reduce significativamente el tiempo de ejecución con una reducción muy pequeña de la eficacia de la BL del Mejor (incluso puede mejorarla en algunos problemas)
- Se basa en definir un orden de aplicación de los intercambios (exploración de los vecinos) en una BL del primer mejor
- En cada iteración, se define un valor  $d_i$  para cada elemento seleccionado  $s_i$  de la solución actual que mide la **contribución del elemento a su coste**  $z_{MS}(Sel)$ :

$$d_i = \sum_{s_j \in Sel} d_{ij} = d(s_i, Sel)$$

# BL-MDP: Exploración Inteligente del Vecindario

---

- En lugar de calcular el valor del movimiento  $Int(Sel, i, j)$  para todos los intercambios posibles, se escoge el elemento  $s_{i^*}$  de  $Sel$  que presenta el menor aporte (es decir, el valor mínimo  $d_{i^*}$ )
- Tras escoger el elemento a extraer, se prueban sucesivamente los intercambios por los elementos no seleccionados:
  - Si se encuentra un movimiento de mejora, se aplica. Si no, se pasa al siguiente elemento con menor aporte y se repite el proceso
  - Si ningún movimiento del vecindario provoca mejora, se finaliza la ejecución y se devuelve la solución actual

# BL-MDP: Factorización del Movimiento de Intercambio

- Sea  $z_{MM}(Sel)$  el coste de la solución original  $Sel$ :

$$z_{MM}(Sel) = \sum_{s_i, s_j \in Sel} d_{ij}$$

- Para generar  $Sel'$ , el operador de vecino  $Int(Sel, i, j)$  escoge un elemento seleccionado  $i$  y lo cambia por uno no seleccionado  $j$ :

$$Sel = \{s_1, \dots, i, \dots, s_m\} \Rightarrow Sel' \leftarrow Sel - \{i\} + \{j\} \Rightarrow Sel' = \{s_1, \dots, j, \dots, s_m\}$$

- **No es necesario recalcular todas las distancias de la función objetivo:**
  - Al añadir un elemento, las distancias entre los que ya estaban en la solución se mantienen y basta con sumar la distancia del nuevo elemento al resto de elementos seleccionados
  - Al eliminar un elemento, las distancias entre elementos que se quedan en la solución se mantienen y basta con restar la distancia del elemento eliminado al resto de elementos en la solución
- Por tanto, quedan afectados  $2 \cdot (m-1)$  sumandos de la función objetivo, las  $m-1$  distancias del elemento que **se elimina** y las  $m-1$  del que **se añade**

# BL-MDP: Factorización del Movimiento de Intercambio

- El coste del movimiento (la **diferencia de costes entre las dos soluciones**)  $\Delta z_{MM}(Sel, i, j) = z_{MM}(Sel') - z_{MM}(Sel)$  se puede factorizar:

$$\Delta z_{MM}(Sel, i, j) = \sum_{s_k \in Sel - \{s_i\}} \boxed{d_{kj}} - \boxed{d_{ki}} \text{ *eliminada* } \\ \text{añadida}$$

- Si  $\Delta z_{MM}(Sel, i, j)$  es positivo ( $\Delta z_{MM}(Sel, i, j) > 0$ ), la solución vecina  $Sel'$  es mejor que la actual  $Sel$  (**el MDP es un problema de maximización**) y se acepta. Si no, se descarta y se genera otro vecino
- Podemos combinar fácilmente la factorización del coste con el cálculo de la contribución de los elementos para mejorar aún más la eficiencia:
  - Las distancias del elemento eliminado equivalen directamente a la contribución de dicho elemento,  $d_i = \sum_{s_k \in Sel - \{s_i\}} d_{ki}$
  - El cálculo de las aportaciones de los elementos actualmente seleccionados también se puede factorizar. No es necesario recalcularlo completamente, basta con restar la distancia del elemento eliminado y sumar la del añadido:  $d_k = d_k + d_{kj} - d_{ki}$

# BL-MDP: Factorización del Movimiento de Intercambio

---

- El coste  $z_{MM}(Sel')$  de la nueva solución vecina es:

$$z_{MM}(Sel') = z_{MM}(Sel) + \Delta z_{MM}(Sel, i, j)$$

- Sólo es necesario calcularlo al final de la ejecución. Durante todo el proceso, basta con trabajar con el coste del movimiento
- El pseudocódigo de la BL del Primer Mejor del Tema 2 de Teoría queda (**partimos de una solución inicial factible aleatoria**):

## Repetir

$Sel' \leftarrow \text{GENERA\_VECINO}(Sel);$

**Hasta** ( $\Delta z_{MM}(Sel, i, j) > 0$ ) **O**  
(se ha generado  $E(Sel)$  al completo)

# Casos del Problema

---

- Existen distintos grupos de casos del problema para los que se conoce la solución óptima que permiten validar el funcionamiento de los algoritmos de resolución
- Para el MDP, disponemos de cuatro grandes grupos de casos:
  - **Casos GKD** (Glover, Kuo and Dhir, 1998): Entre otras, 20 matrices  $n \times n$  con distancias Euclideas calculadas a partir de puntos con  $r$  coordenadas ( $r \in \{2, \dots, 21\}$ ) aleatorias en  $[0,10]$ .  $n=500$  elementos y  $m=50$
  - **Casos SOM** (Silva, Ochi y Martins, 2004): Entre otras, 20 matrices  $n \times n$  con distancias enteras aleatorias en  $\{0,9\}$  con  $n \in \{100, \dots, 500\}$  elementos y  $m \in \{0.1 \cdot n, \dots, 0.4 \cdot n\}$ . P.ej. para  $n=100$  hay 4 casos con  $m=10, 20, 30, 40$
  - **Casos MDG** (Duarte y Martí, 2007):
    - **Tipo a**: 40 matrices  $n \times n$  con distancias enteras aleatorias en  $\{0,10\}$ : 20 con  $n=500$  y  $m=50$ ; y 20 con  $n=2000$  y  $m=200$
    - **Tipo b**: 40 matrices  $n \times n$  con distancias reales aleatorias en  $[0,1000]$ : 20 con  $n=500$  y  $m=50$ ; y 20 con  $n=2000$  y  $m=200$
    - **Tipo c**: 20 matrices  $n \times n$  con distancias enteras aleatorias en  $\{0,1000\}$ .  $n=3000$  y  $m=\{300,400,500,600\}$

# Casos del Problema

---

- Los casos están recopilados en la **biblioteca MDPLib**, accesible en la Web en la dirección siguiente:

<https://grafo.etsii.urjc.es/opticom/mdp/>

- En dicha dirección pueden encontrarse tanto los datos como los valores de las mejores soluciones encontradas para 315 casos del problema
- Además, están disponibles los resultados de un ejemplo de una experimentación comparativa de distintos algoritmos con 10 minutos de tiempo de ejecución por caso

# La Biblioteca MDPLIB

---

- El **formato de los ficheros de datos** es un fichero de texto con la siguiente estructura:

$$n \ m$$
$$D$$

donde  $n$  es el número de elementos,  $m$  es el número de elementos seleccionados y  $D$  es la matriz de distancias entre elementos que está precalculada

- Al ser  $D$  una matriz simétrica, sólo se almacena la diagonal superior. El fichero contendrá  $n \cdot (n-1)/2$  entradas, una por línea, con el siguiente formato:

$$i \ j \ d_{ij}$$

donde  $i, j \in \{0, \dots, n-1\}$  son respectivamente la fila y la columna de la matriz  $D$ , mientras que  $d_{ij}$  es el valor de la distancia existente entre los elementos  $i+1$  y  $j+1$

# La Biblioteca MDPLIB

---

## EJEMPLO: FICHERO DEL CASO GKD-c\_1\_n500\_m50:

```
500 50
0 1 11.17945
0 2 12.18565
0 3 15.82056
0 4 7.17287
0 5 12.63171
0 6 10.45706
0 7 12.37497
0 8 12.13219
0 9 13.07364
0 10 10.54751
0 11 9.96995
0 12 12.55428
0 13 12.86351
0 14 7.08237
...
496 497 14.48240
496 498 11.37189
496 499 13.94453
497 498 15.47191
497 499 17.05433
498 499 10.37931
```

# Agradecimientos

---

- Para la preparación de las transparencias de presentación del problema MDPLIB se han usado materiales de los profesores:
  - Rafael Martí. Universidad de Valencia
  - Abraham Duarte. Universidad Rey Juan Carlos
  - Jesús Sánchez-Oro. Universidad Rey Juan Carlos
- Su grupo de investigación ha realizado muchas publicaciones sobre el problema y mantiene la biblioteca MDPLIB:
  - Tabu Search for the **Maximum Diversity** Problem, Abraham Duarte and Rafael Martí, **European Journal of Operational Research** 178, 71-84 (2007)
  - Hybrid heuristics for the **Maximum Diversity** Problem, M. Gallego, A. Duarte, M. Laguna and R. Martí, **Computational Optimization and App.** 44(3), 411-426 (2009)
  - A Branch and Bound algorithm for the **Maximum Diversity** Problem, Martí, Gallego and Duarte, **European Journal of Operational Research** 200(1), 36-44 (2010)
  - Heuristics and Metaheuristics for the **Maximum Diversity** Problem, Rafael Martí, Micael Gallego and Abraham Duarte, **Journal of Heuristics**, 19, 591-615 (2013)

# Metaheurísticas

## Seminario 2. Problemas de optimización con técnicas basadas en búsqueda local

---

1. Problema de la Máxima Diversidad (MDP)
2. Problema del Agrupamiento con Restricciones (PAR)
  - Agrupamiento Clásico
  - Agrupamiento con Restricciones: Definición y Variantes del Problema
  - Análisis del Problema
  - Solución Greedy
    - Ejemplo de Aplicación
  - Búsquedas por Trayectorias Simples
    - Ejemplo de Aplicación
  - Casos del problema
  - Agradecimientos

# El Problema del Agrupamiento

---

- El **agrupamiento** o **análisis de clusters** o ***clustering*** en inglés, persigue la clasificación de objetos de acuerdo a posibles similitudes entre ellos.
- Así, el agrupamiento es una **técnica de aprendizaje no supervisado** que permite descubrir grupos (inicialmente desconocidos) en un conjunto de datos o agrupar objetos similares entre sí
- Conjuntos grandes de datos se agrupan en clusters de subconjuntos de datos similares
- El hecho de que no exista una clasificación a priori de los datos marca el carácter de las técnicas de agrupamiento como herramientas de exploración de los datos para **extracción de conocimiento**

# El Problema del Agrupamiento

---

- El agrupamiento provoca una reducción de la dimensionalidad del problema. Se busca un espacio de menor dimensión que el original que preserve la estructura de las observaciones
- Ejemplo: Conjunto de datos:   
Después del agrupamiento: 
- Su principal aplicación es la clasificación en entornos en los que las clases son desconocidas a priori
- Existen otras muchas aplicaciones, tales como la clasificación en entornos con clases cambiantes o la reducción de la complejidad de una muestra inicial de ejemplos

# El Problema del Agrupamiento

---

- Son ejemplos típicos de análisis de clusters en Informática e Ingeniería los caracteres escritos a mano, las muestras de diálogo, las huellas dactilares y las imágenes
- En Ciencias de la Vida los objetos de análisis son plantas, animales e insectos. El agrupamiento se puede aplicar desde la creación de taxonomías completas hasta la clasificación de especies en subespecies
- En Biología, Química y Farmacología está muy extendido el uso de agrupamiento para moléculas, proteínas, genes, etc.
- También se aplica en Ciencias de la Información, Empresariales y Políticas: documentos, votos, análisis de mercados, segmentación de clientes, programas de ventas, etc.

# ¿Qué es un Cluster?

---

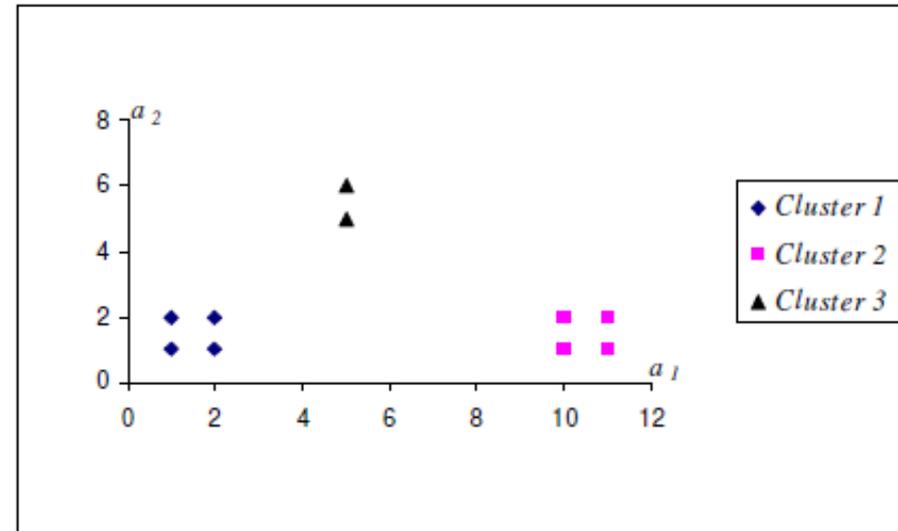
- Cada grupo de objetos similares entre sí obtenidos tras el proceso se denomina **cluster**. Al conjunto de clusters resultante se le denomina **configuración de clusters**
- El termino **similitud** se suele entender en su sentido matemático, es decir, definido mediante una **distancia**, habitualmente la **Euclídea**
- De este modo, cada cluster se caracteriza por un **centroide**, centro geométrico del cluster en el espacio d-dimensional de características de los objetos
- Cada objeto pertenece a un único cluster y su distancia Euclídea al centroide de éste es siempre menor que a los restantes

# Configuración de Clusters

## Ejemplo gráfico de una configuración de clusters

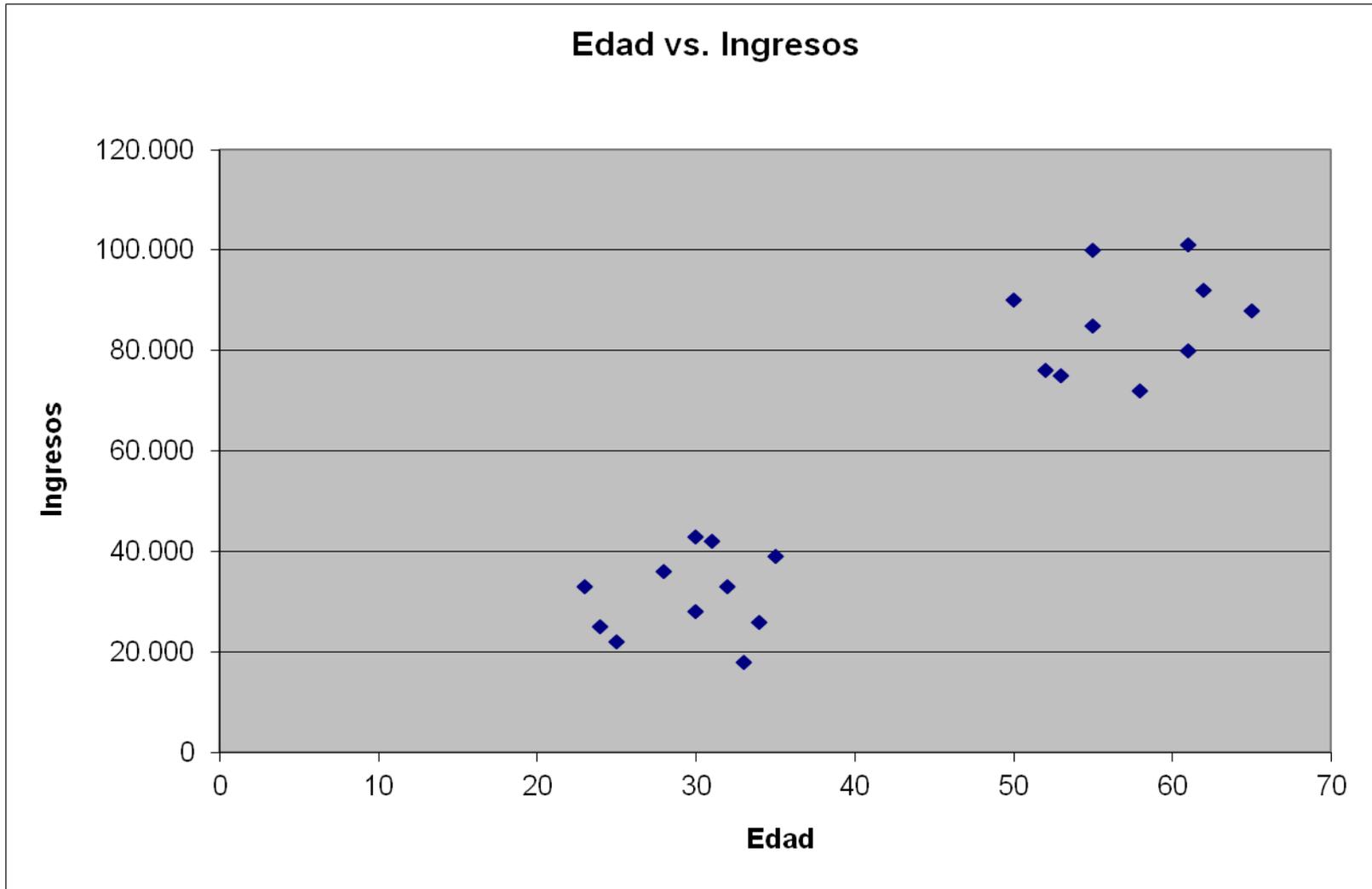
TABLE I. PEDAGOGICAL DATA SET.

Object ( $x_i$ )	$a_1$	$a_2$	Cluster - $C_j$
$x_1$	1	1	Cluster 1 ( $C_1$ )
$x_2$	1	2	Cluster 1 ( $C_1$ )
$x_3$	2	1	Cluster 1 ( $C_1$ )
$x_4$	2	2	Cluster 1 ( $C_1$ )
$x_5$	10	1	Cluster 2 ( $C_2$ )
$x_6$	10	2	Cluster 2 ( $C_2$ )
$x_7$	11	1	Cluster 2 ( $C_2$ )
$x_8$	11	2	Cluster 2 ( $C_2$ )
$x_9$	5	5	Cluster 3 ( $C_3$ )
$x_{10}$	5	6	Cluster 3 ( $C_3$ )



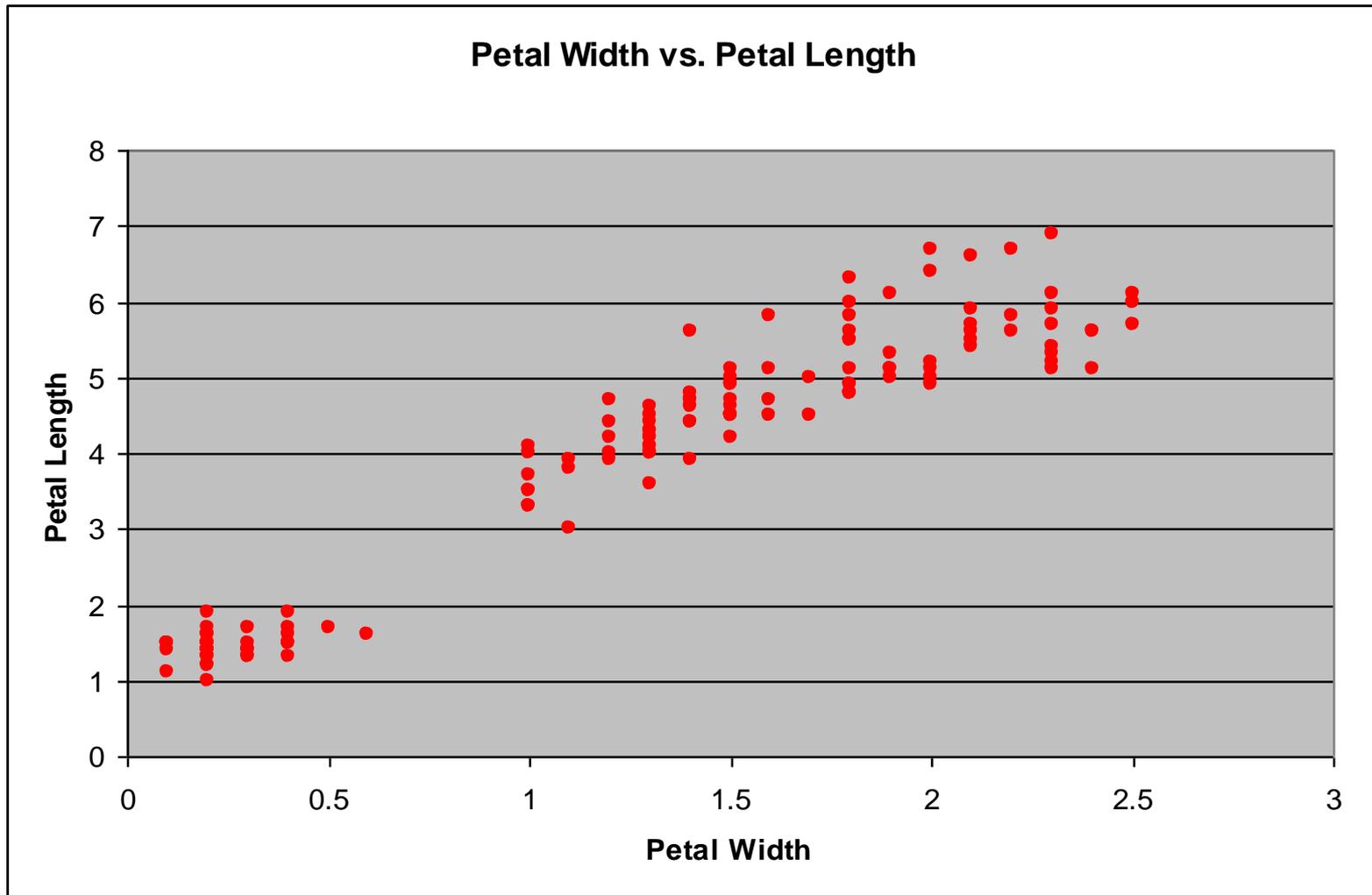
# Configuración de Clusters

## Ejemplos: Segmentación de clientes



# Configuración de Clusters

## Ejemplos: Clasificación (Iris)



# Agrupamiento con Restricciones: Definición del Problema

---

- El Problema del Agrupamiento con Restricciones (PAR) (en inglés *Constrained Clustering, CC*) consiste en una **generalización del problema del agrupamiento clásico**. Permite incorporar al proceso de agrupamiento un **nuevo tipo de información: las restricciones**
- Al incorporar esta información la tarea de aprendizaje deja de ser no supervisada y se convierte en semi-supervisada
- El problema consiste en, dado un conjunto de datos  $X$  con  $n$  instancias, encontrar una partición  $C$  del mismo que minimice la desviación general y cumpla con las restricciones del conjunto de restricciones  $R$

# Agrupamiento con Restricciones: Definición del Problema

---

- Se puede formular en términos de optimización combinatoria. Dicha formulación es sencilla pero de compleja resolución (**es NP-completo**)
- Entre algunos ejemplos de aplicación del PAR encontramos:
  - robótica aplicada,
  - márketing aplicado,
  - detección de comunidades terroristas,
  - diseño de distritos electorales,
  - análisis de la apnea obstructiva del sueño, y
  - detección de carriles de vehículos en aplicaciones GPS, entre otras

# Variantes del Problema

---

Existen variantes del problema que dependen del **tipo de restricciones**:

- *Restricciones de distancia*: Las instancias separadas por una distancia mayor a una dada deben pertenecer a diferentes clusters. Las instancias separadas por una distancia menor que una dada deben pertenecer al mismo cluster
- *Restricciones de instancia*: Dada una pareja de instancias, se establece una restricción de tipo **Must-Link (ML)**, si **deben pertenecer al mismo cluster**, o de tipo **Cannot-Link (CL)**, si **no pueden pertenecer al mismo cluster**

# Variantes del Problema

---

Existen variantes del problema que dependen del **modo en que se interpretan las restricciones**:

- *Restricciones Fuertes (Hard)*: Todas las restricciones deben satisfacerse en la partición  $C$  del conjunto de datos  $X$ . Se dice que una partición es **factible si y solo si cumple con todas las restricciones en  $R$**
- *Restricciones Débiles (Soft)*: La partición  $C$  del conjunto de datos  $X$  debe minimizar el número de restricciones incumplidas pero puede incumplir algunas

# Definición del Problema: Formalización

---

- El conjunto de datos es una matriz  $X$  de  $n \times d$  valores reales. El conjunto de datos esta formado por  $n$  instancias en un espacio de  $d$  dimensiones notadas como:

$$\vec{x}_i = \{x_{[i,1]}, \dots, x_{[i,d]}\} | x_{[i,j]} \in \mathbb{R} \forall j \in \{1, \dots, d\}$$

- Una partición  $\mathcal{C}$  consiste en un conjunto de  $k$  clusters  $\mathcal{C} = \{c_1, \dots, c_k\}$  que asignan una etiqueta a cada instancia de  $X$ . Cada instancia debe pertenecer a un y solo a un cluster
- Un cluster  $c_i$  consiste en un subconjunto de instancias de  $X$ . Notamos el número de elementos de  $c_i$  como  $|c_i|$ . Cada cluster  $c_i$  tiene asociada una etiqueta (nombre de cluster)  $l_i$

# Definición del Problema: Formalización

---

- Para cada cluster  $c_i$  se puede calcular su **centroide** asociado  $\vec{\mu}_i$  como el vector promedio de las instancias de  $X$  que lo componen:

$$\vec{\mu}_i = \frac{1}{|c_i|} \sum_{\vec{x}_j \in c_i} \vec{x}_j$$

- Definimos la **distancia media intra-cluster**  $\bar{c}_i$  como la media de las distancias de las instancias que lo conforman a su centroide:

$$\bar{c}_i = \frac{1}{|c_i|} \sum_{\vec{x}_j \in c_i} \|\vec{x}_j - \vec{\mu}_i\|_2$$

# Definición del Problema: Formalización

- Definimos la **desviación general de la partición**  $C = \{c_1, \dots, c_k\}$  como la media de las desviaciones intra-cluster  $\bar{C}$  :

$$\bar{C} = \frac{1}{k} \sum_{c_i \in C} \bar{c}_i$$

- Para que una partición  $C = \{c_1, \dots, c_k\}$  del conjunto de datos  $X$  sea válida debe cumplir que (**restricciones fuertes**):

- Todos los clusters deben contener al menos una instancia:

$$c_i \neq \emptyset \forall i \in \{1, \dots, k\} \leftrightarrow |c_i| > 0 \forall i \in \{1, \dots, k\}$$

- Cada instancia debe pertenecer a un solo cluster:

$$c_i \cap c_j = \emptyset \forall i, j \in \{1, \dots, k\} | i \neq j$$

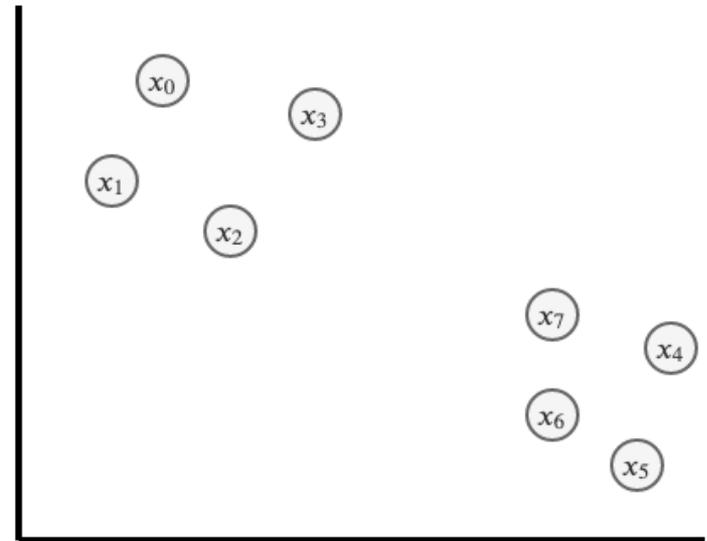
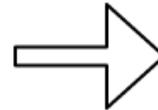
- La unión de los clusters debe ser el conjunto de datos  $X$ :

$$\bigcup_{c_i \in C} c_i = X$$

# Definición del Problema: Ejemplo

Consideramos el conjunto de datos  $X$  formado por 8 instancias en dos dimensiones ( $n = 8, d = 2$ ).

$$X = \begin{pmatrix} x_{0,1} & x_{0,2} \\ x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \\ x_{5,1} & x_{5,2} \\ x_{6,1} & x_{6,2} \\ x_{7,1} & x_{7,2} \end{pmatrix}$$



# Definición del Problema: Ejemplo

---

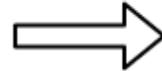
Consideramos la partición  $C$  que produce los clusters  $c_1$  y  $c_2$  asignando etiquetas a las instancias de  $X$ .

$$C = \{\overset{x_0}{1}, \overset{x_1}{1}, \overset{x_2}{1}, \overset{x_3}{1}, \overset{x_4}{2}, \overset{x_5}{2}, \overset{x_6}{2}, \overset{x_7}{2}\}$$
$$c_1 = \{\vec{x}_0, \vec{x}_1, \vec{x}_2, \vec{x}_3\} \qquad c_2 = \{\vec{x}_4, \vec{x}_5, \vec{x}_6, \vec{x}_7\}$$

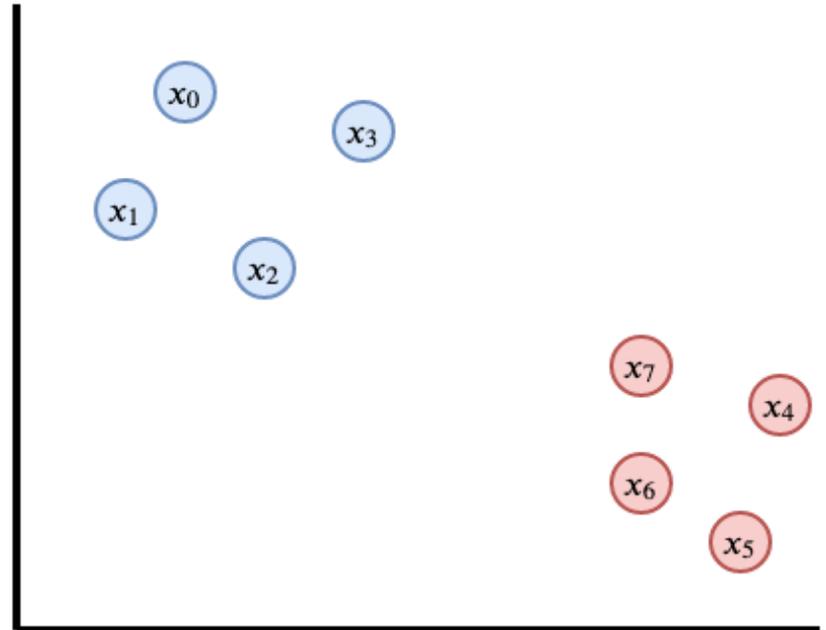
# Definición del Problema: Ejemplo

Consideramos la partición  $\mathcal{C}$  que produce los clusters  $c_1$  y  $c_2$  asignando etiquetas a las instancias de  $X$  (**azul** y **rojo**):

$$c_1 = \{\vec{x}_0, \vec{x}_1, \vec{x}_2, \vec{x}_3\}$$



$$c_2 = \{\vec{x}_4, \vec{x}_5, \vec{x}_6, \vec{x}_7\}$$

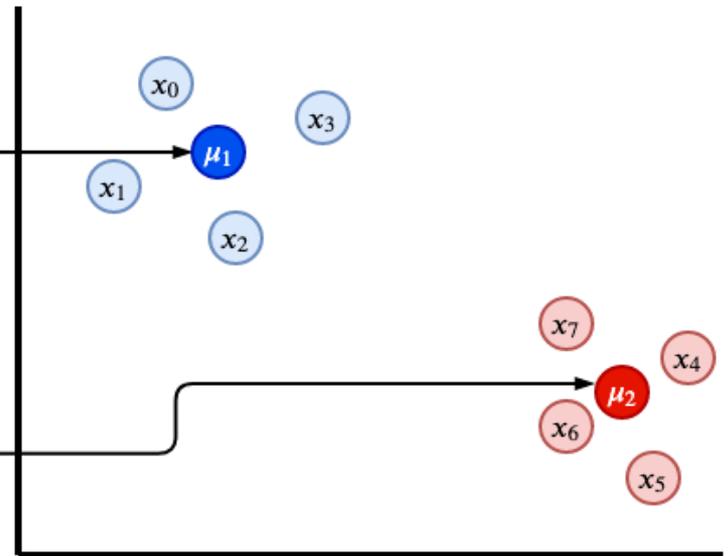


# Definición del Problema: Ejemplo

De esta forma los centroides  $\vec{\mu}_1$  y  $\vec{\mu}_2$  se calculan como:

$$\vec{\mu}_1 = \frac{\vec{x}_0 + \vec{x}_1 + \vec{x}_2 + \vec{x}_3}{4}$$

$$\vec{\mu}_2 = \frac{\vec{x}_4 + \vec{x}_5 + \vec{x}_6 + \vec{x}_7}{4}$$



# Definición del Problema: Ejemplo

---

Las distancias medias intra-cluster  $\bar{c}_1$  y  $\bar{c}_2$  se calculan como:

$$\bar{c}_1 = \frac{\|\vec{x}_0 - \vec{\mu}_1\|_2 + \|\vec{x}_1 - \vec{\mu}_1\|_2 + \|\vec{x}_2 - \vec{\mu}_1\|_2 + \|\vec{x}_3 - \vec{\mu}_1\|_2}{4}$$

$$\bar{c}_2 = \frac{\|\vec{x}_4 - \vec{\mu}_2\|_2 + \|\vec{x}_5 - \vec{\mu}_2\|_2 + \|\vec{x}_6 - \vec{\mu}_2\|_2 + \|\vec{x}_7 - \vec{\mu}_2\|_2}{4}$$

Y la desviación general  $\bar{c}$  se calcula como:

$$\bar{c} = \frac{\bar{c}_1 + \bar{c}_2}{2}$$

# Definición del Problema: Formalización

---

- Durante el desarrollo de la práctica trabajaremos con las restricciones a nivel de instancias Must-Link (ML) y Cannot-Link (CL):

$$R = ML \cup CL$$

- Podemos formalizar las restricciones como sigue:
  - Restricciones Must-Link  $ML(\vec{x}_i, \vec{x}_j)$ : las instancias  $\vec{x}_i$  y  $\vec{x}_j$  deben ser asignadas al mismo cluster
  - Restricciones Cannot-Link  $CL(\vec{x}_i, \vec{x}_j)$ : las instancias  $\vec{x}_i$  y  $\vec{x}_j$  no pueden ser asignadas al mismo cluster

# Definición del Problema: Formalización

---

- Notamos el número de restricciones  $ML$  con  $|ML|$  y el de restricciones  $CL$  con  $|CL|$ , de forma que el número total de restricciones  $|R| = |ML| + |CL|$
- Notamos la  $i$ -ésima restricción  $ML$  como  $ML_i$ . Notamos la  $i$ -ésima restricción  $CL$  como  $CL_i$
- Notamos la primera instancia implicada en  $ML_i$  como  $\overrightarrow{ML_{[i,1]}}$  y la segunda como  $\overrightarrow{ML_{[i,2]}}$ . Notamos la primera instancia implicada en  $CL_i$  como  $\overrightarrow{CL_{[i,1]}}$  y la segunda como  $\overrightarrow{CL_{[i,2]}}$ .
- Definimos  $h_C(\cdot)$  como la función que dada una instancia  $\vec{x}_i$  devuelve la etiqueta  $l_j$  asociada al cluster  $c_j$  al que  $\vec{x}_i$  pertenece según la partición  $\mathcal{C}$

# Definición del Problema: Formalización

---

- Dada una partición  $C$  y los conjuntos de restricciones  $ML$  y  $CL$ , definimos *infeasibility* ("grado de infactibilidad") como el número de restricciones que  $C$  incumple:

$$infeasibility = \sum_{i=0}^{|ML|} \mathbb{1}(h_C(\overrightarrow{ML_{[i,1]}}) \neq h_C(\overrightarrow{ML_{[i,2]}})) + \sum_{i=0}^{|CL|} \mathbb{1}(h_C(\overrightarrow{CL_{[i,1]}}) = h_C(\overrightarrow{CL_{[i,2]}}))$$

donde  $\mathbb{1}$  es la función indicadora, que devuelve 1 en caso de ser cierta la expresión Booleana que toma como argumento y 0 en otro caso

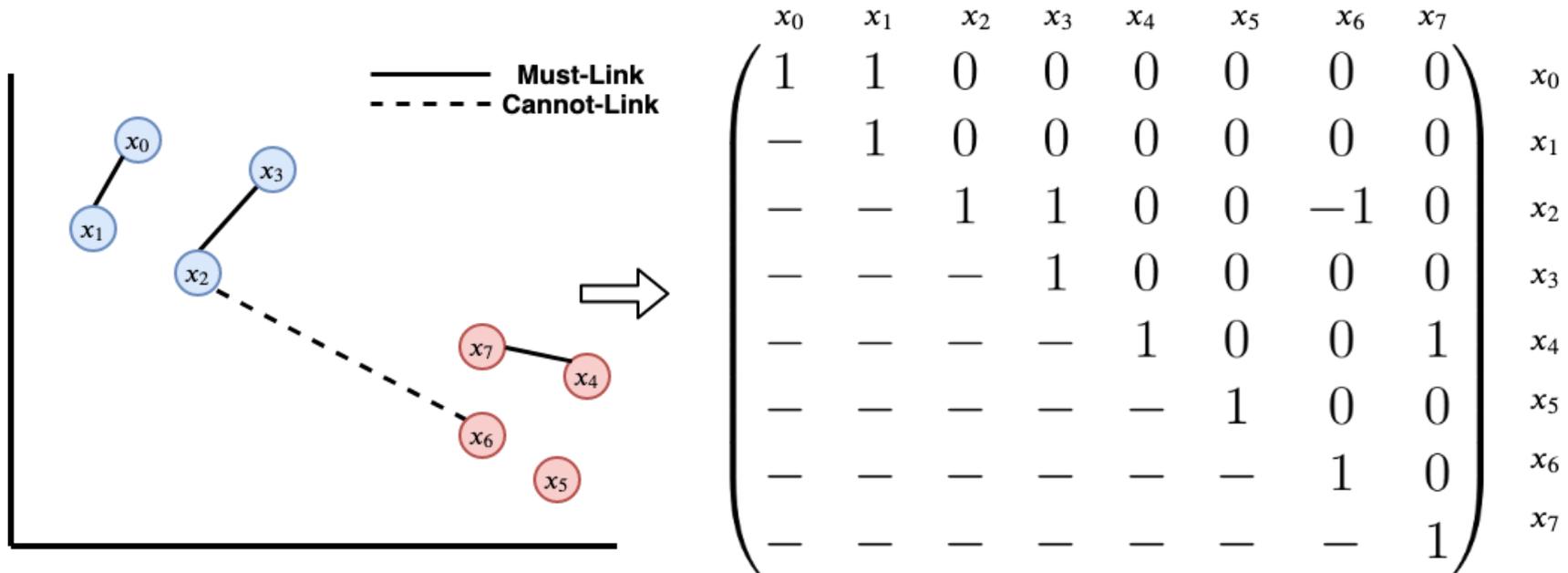
# Definición del Problema: Formalización

---

- Las restricciones pueden venir dadas en forma de lista de  $|R|$  entradas ( $LR$ ) o de matriz de  $n^2$  entradas ( $MR$ )
- En formato lista las restricciones consiste en tripletas en las que las dos primeras posiciones contienen los identificadores de las instancias implicadas y el tercero contiene el tipo de restricción
- En formato matriz las restricciones consisten en valores dentro de la matriz que indican el tipo de restricción que existe entre las instancias asociadas a los índices que identifican el valor

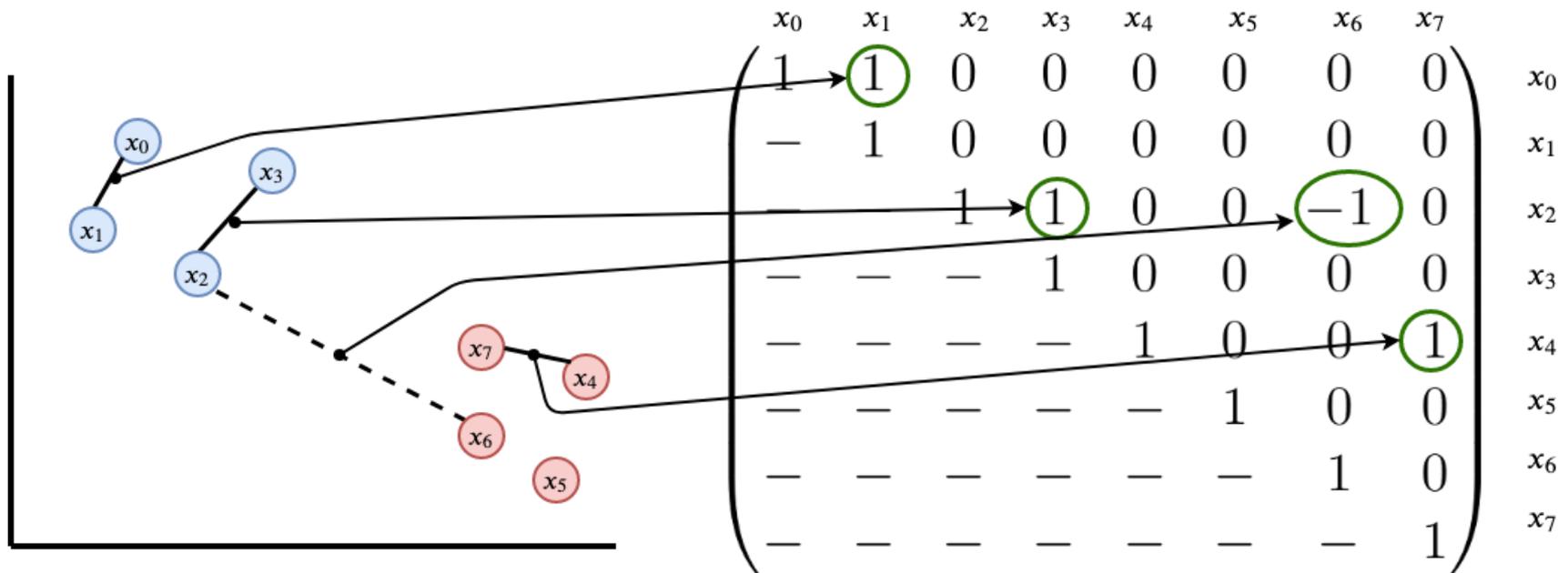
# Definición del Problema: Ejemplo

Las restricciones vienen dadas en forma de una matriz  $MR$  de  $n^2$  entradas. Utilizamos un 1 para indicar restricción de tipo  $ML$ , un -1 para indicar  $CL$  y 0 para indicar que no hay restricción. La matriz es siempre **simétrica** y su diagonal contiene 1s:



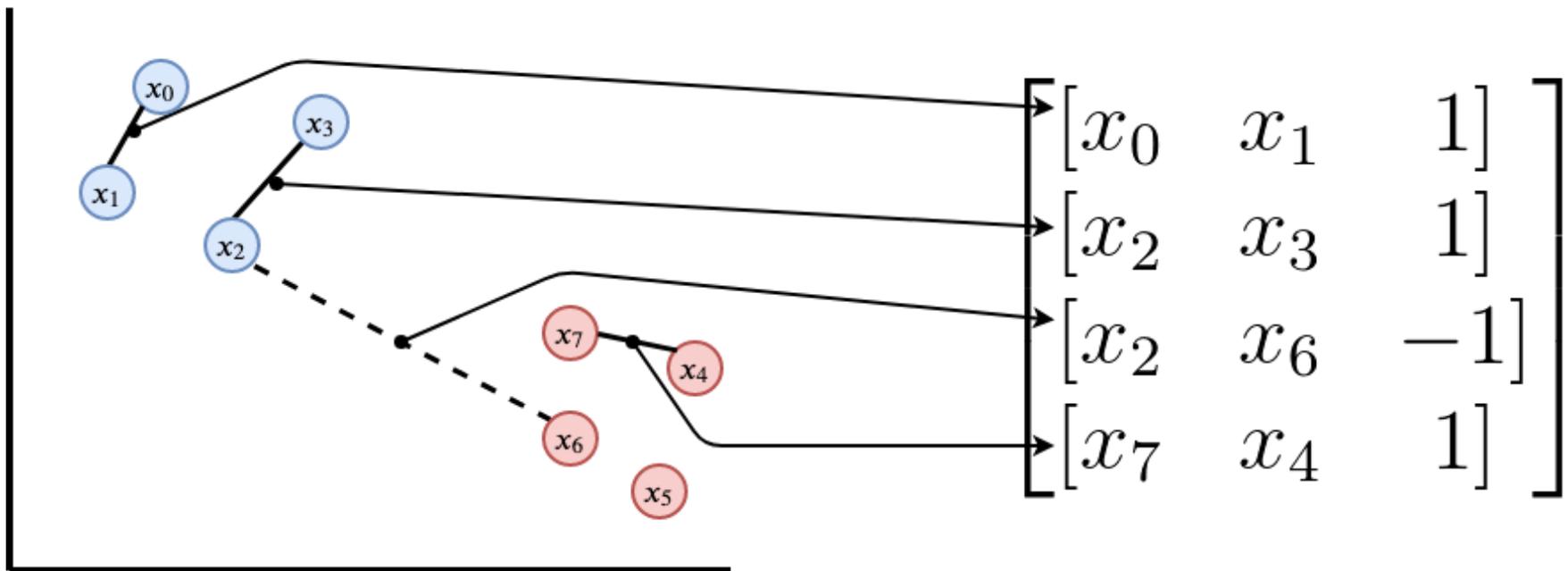
# Definición del Problema: Matriz de restricciones

Las restricciones vienen dadas en forma de una matriz  $MR$  de  $n^2$  entradas. Utilizamos un 1 para indicar restricción de tipo  $ML$ , un -1 para indicar  $CL$  y 0 para indicar que no hay restricción. La matriz es siempre **simétrica** y su diagonal contiene 1s:



# Definición del Problema: Lista de restricciones

En formato lista,  $LR$ , las restricciones consisten en tripletas en las que las dos primeras posiciones contienen los identificadores de las instancias implicadas y el tercero contiene el tipo de restricción. Utilizamos un 1 para indicar una restricción  $ML$  y un -1 para indicar una  $CL$ :



# Definición del Problema: CC

---

- **Bajo Restricciones Fuertes:** Minimizar  $\bar{C}$  sujeto a que *infeasibility* = 0
- **Bajo Restricciones Débiles:** Minimizar una agregación de  $\bar{C}$  e *infeasibility*, permitiendo *infeasibility* > 0

**Para el desarrollo de la práctica trabajaremos siempre bajo restricciones débiles**

# Definición del Problema: Formalización

---

- Dado que disponemos de estructuras para almacenar las restricciones, podemos calcular el valor para *infeasibility* de manera sencilla y eficiente
- Definimos  $V(\vec{x}_i, \vec{x}_j)$  como la función que, dada una pareja de instancias, devuelve 1 si la asignación de dichas instancias viola una restricción y 0 en otro caso
- Podemos calcular *infeasibility* como el número de restricciones incumplidas por una partición  $\mathcal{C}$  del conjunto de datos  $X$  dada una matriz de restricciones  $MR$  o una lista de restricciones  $LR$ :

$$infeasibility = \sum_{i=0}^n \sum_{j=i+1}^n V(\vec{x}_i, \vec{x}_j)$$

# Análisis del Problema

---

- El problema del agrupamiento simple es NP-Duro. Al añadirle restricciones de tipo *ML* y *CL* pasa a ser NP-Completo
- Resolverlo de forma óptima es intratable, debemos buscar soluciones aproximadas
- La representación de las restricciones en forma de lista es más eficiente cuando es necesario recorrer las restricciones para realizar operaciones sobre ellas, ya que solo es necesario recorrer una lista de longitud  $|R|$  en lugar de recorrer una matriz de  $n^2$  posiciones
- La representación en forma de matriz es más eficiente cuando se sabe que restricción se quiere comprobar, ya que las matrices implementan acceso aleatorio

# Análisis del Problema

---

## ■ **COMENTARIOS:**

- ✓  $k$  debe ser fijo y conocido a priori. El aprendizaje del número óptimo de clusters para un conjunto de datos concreto es un problema complejo
- ✓ La función minimizar  $\{\bar{C}\}$  puede presentar muchos mínimos locales
- Para diseñar un método de resolución del PAR es necesario disponer de dos componentes:
  - a) Una medida de la bondad con la que la configuración de clusters actual representa la muestra original de instancias (habitualmente el valor  $\bar{C}$  combinado con *infeasibility*)
  - b) Un algoritmo que busque la configuración de clusters que optimice dichas medidas

# La Solución Greedy

---

La solución Greedy para el PAR consiste en **modificar el algoritmo k-medias para considerar dichas restricciones**

Para la solución greedy haremos una interpretación débil de las restricciones

De esta forma, **para cada instancia se llevará a cabo la asignación que menos restricciones viole y que más disminuya la desviación general  $\bar{c}$**

# La Solución Greedy: El Algoritmo

---

**Algorithm 1:** K-medias Restringido Débil

---

**Input:** Conjunto de datos  $X$ , conjunto de restricciones  $R$ , número de clusters  $k$ , centroides iniciales  $\{\mu_1, \dots, \mu_k\}$ .

/\* Barajamos los índices para recorrer  $X$  de forma aleatoria  
sin repetición \*/

```
[1]  $RSI \leftarrow \text{RandomShuffle}(\{1, \dots, n\})$ 
[2] do
[3]   for  $i \in RSI$  do
[4]     - Calcular el incremento en infeasibility que produce la
        asignación de  $x_i$  a cada cluster  $c_j$ .
[5]     - De entre las asignaciones que producen menos incremento en
        infeasibility, seleccionar la asociada con el centroide  $\mu_j$  más
        cercano a  $x_i$ .
[6]   end
[7]   for  $i \in \{1, \dots, k\}$  do
[8]     Actualizar el centroide  $\mu_i$  promediando las instancias de su
        cluster asociado  $c_i$ .
[9]   end
[10] while Se produzca cambio en  $C$ ;
[11] return  $C = \{c_1, \dots, c_k\}$ 
```

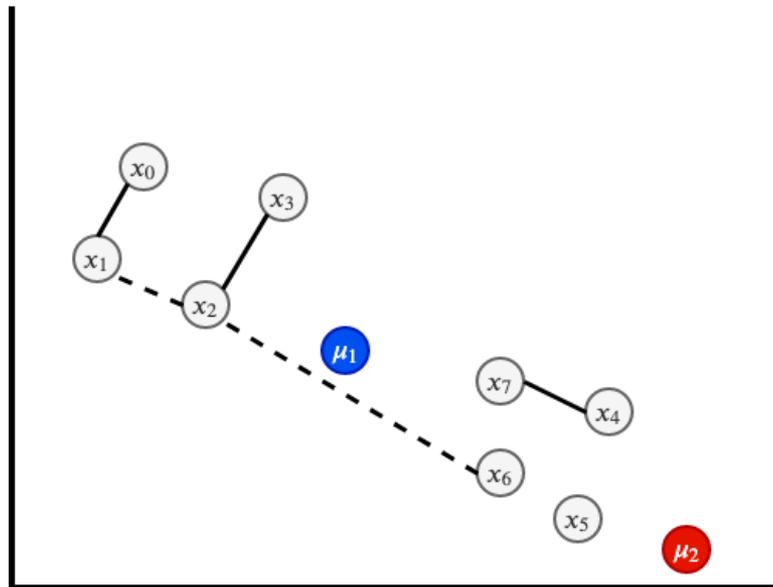
---

# La Solución Greedy: Ejemplo

Supongamos que las instancias de  $X$  se exploran en el siguiente orden:

$$\vec{x}_0 \rightarrow \vec{x}_2 \rightarrow \vec{x}_1 \rightarrow \vec{x}_3 \rightarrow \vec{x}_7 \rightarrow \vec{x}_4 \rightarrow \vec{x}_6 \rightarrow \vec{x}_5$$

Y supongamos  $k = 2$  y la siguiente especificación de restricciones e inicialización de centroides:

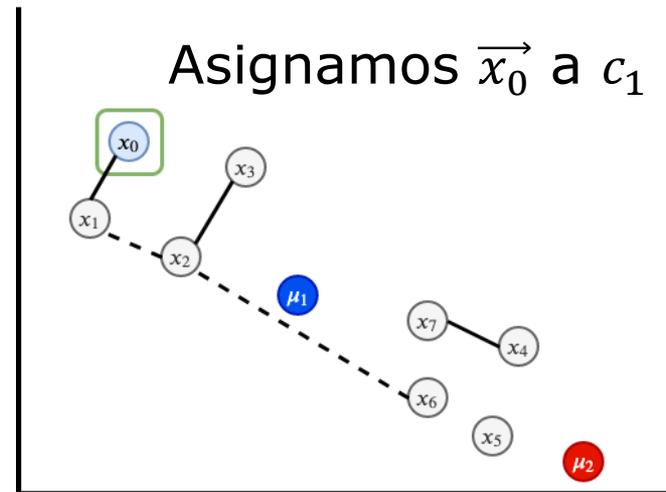
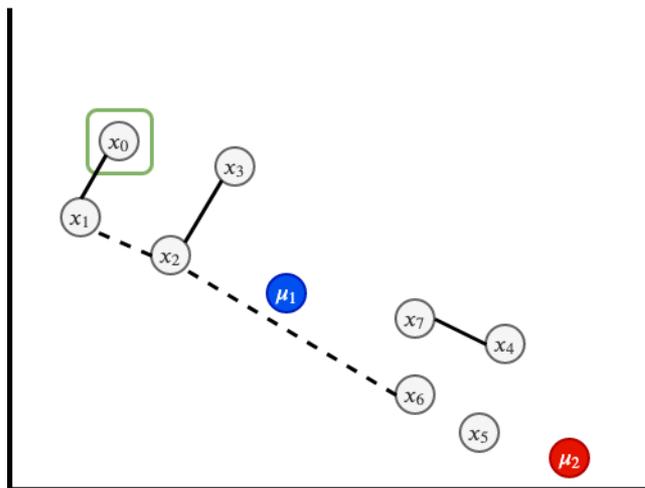


# La Solución Greedy: Ejemplo

$$\vec{x}_0 \rightarrow \vec{x}_2 \rightarrow \vec{x}_1 \rightarrow \vec{x}_3 \rightarrow \vec{x}_7 \rightarrow \vec{x}_4 \rightarrow \vec{x}_6 \rightarrow \vec{x}_5$$

Asignar  $\vec{x}_0$  a  $c_1$   $\longrightarrow \Delta \textit{infeasibility} = 0$

Asignar  $\vec{x}_0$  a  $c_2$   $\longrightarrow \Delta \textit{infeasibility} = 0$



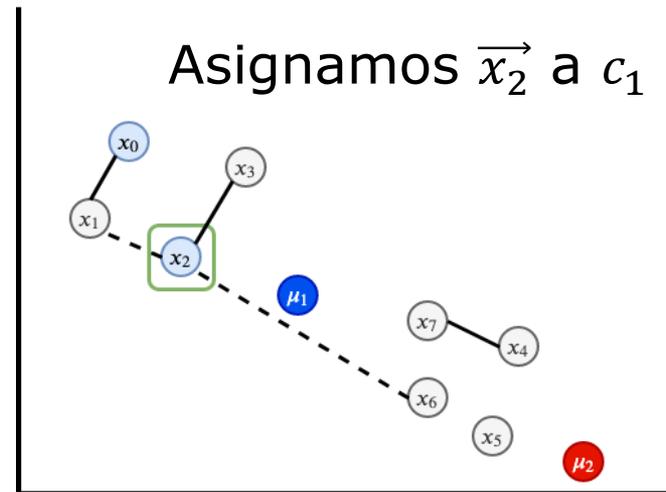
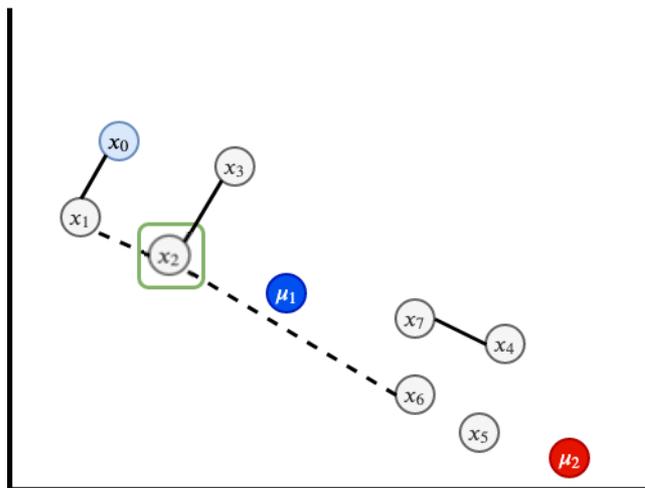
Ambos producen incremento de 0 en *infeasibility* así que asignamos  $\vec{x}_0$  al cluster asociado con el centroide más cercano,  $c_1$

# La Solución Greedy: Ejemplo

$$\vec{x}_0 \rightarrow \vec{x}_2 \rightarrow \vec{x}_1 \rightarrow \vec{x}_3 \rightarrow \vec{x}_7 \rightarrow \vec{x}_4 \rightarrow \vec{x}_6 \rightarrow \vec{x}_5$$

Asignar  $\vec{x}_2$  a  $c_1$   $\longrightarrow$   $\Delta \textit{infeasibility} = 0$

Asignar  $\vec{x}_2$  a  $c_2$   $\longrightarrow$   $\Delta \textit{infeasibility} = 0$



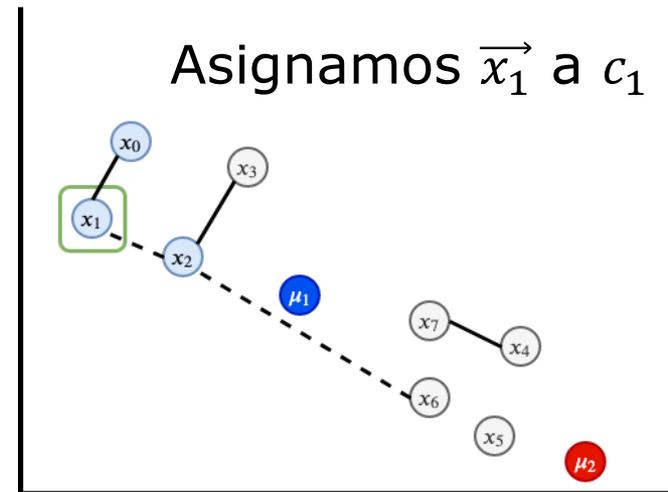
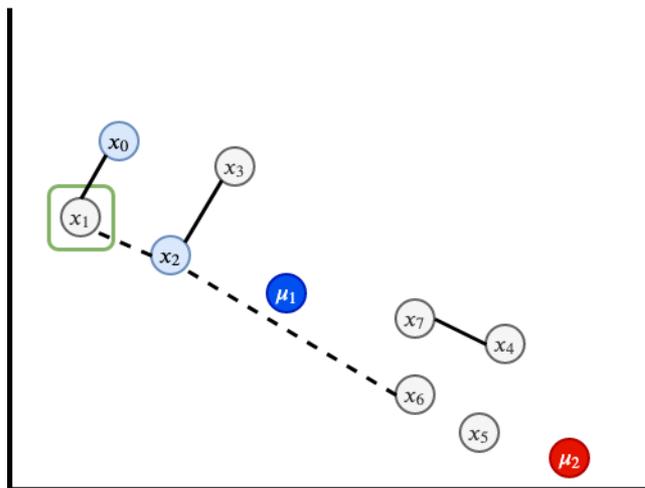
Ambos producen incremento de 0 en *infeasibility* así que asignamos  $\vec{x}_2$  al cluster asociado con el centroide más cercano,  $c_1$

# La Solución Greedy: Ejemplo

$$\vec{x}_0 \rightarrow \vec{x}_2 \rightarrow \vec{x}_1 \rightarrow \vec{x}_3 \rightarrow \vec{x}_7 \rightarrow \vec{x}_4 \rightarrow \vec{x}_6 \rightarrow \vec{x}_5$$

Asignar  $\vec{x}_1$  a  $c_1$   $\longrightarrow$   $\Delta \textit{infeasibility} = 1$  (Viola  $CL(\vec{x}_1, \vec{x}_2)$ )

Asignar  $\vec{x}_1$  a  $c_2$   $\longrightarrow$   $\Delta \textit{infeasibility} = 1$  (Viola  $ML(\vec{x}_1, \vec{x}_0)$ )



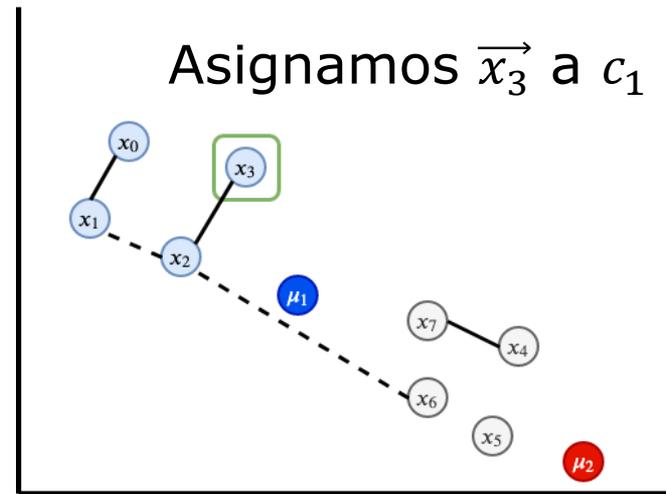
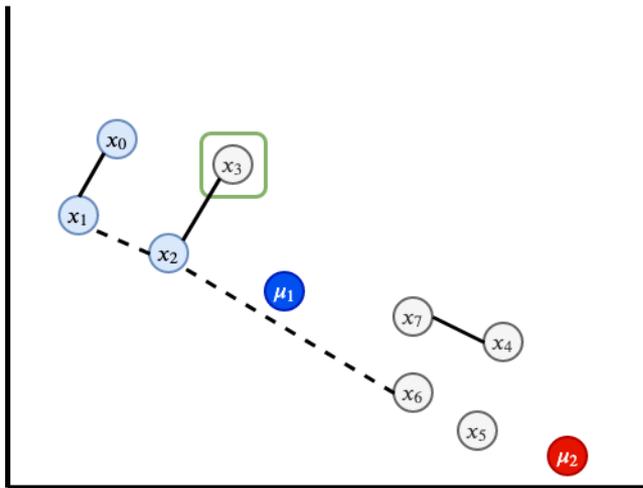
Ambos producen incremento de 1 en *infeasibility* así que asignamos  $\vec{x}_1$  al cluster asociado con el centroide más cercano,  $c_1$

# La Solución Greedy: Ejemplo

$$\vec{x}_0 \rightarrow \vec{x}_2 \rightarrow \vec{x}_1 \rightarrow \vec{x}_3 \rightarrow \vec{x}_7 \rightarrow \vec{x}_4 \rightarrow \vec{x}_6 \rightarrow \vec{x}_5$$

Asignar  $\vec{x}_3$  a  $c_1$   $\longrightarrow$   $\Delta \textit{infeasibility} = 0$

Asignar  $\vec{x}_3$  a  $c_2$   $\longrightarrow$   $\Delta \textit{infeasibility} = 1$  (Viola  $ML(\vec{x}_2, \vec{x}_3)$ )



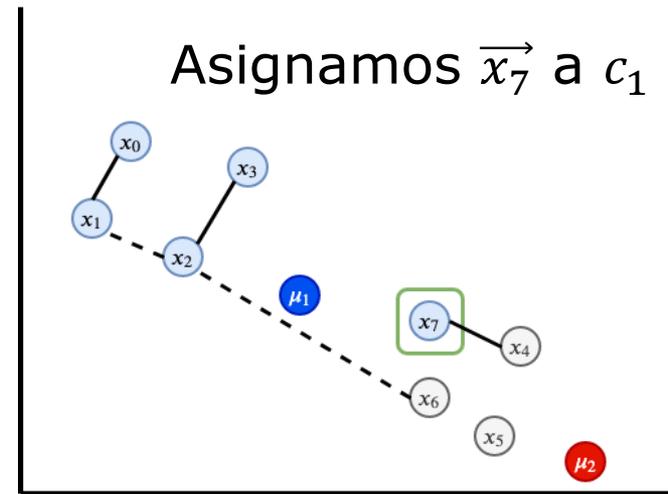
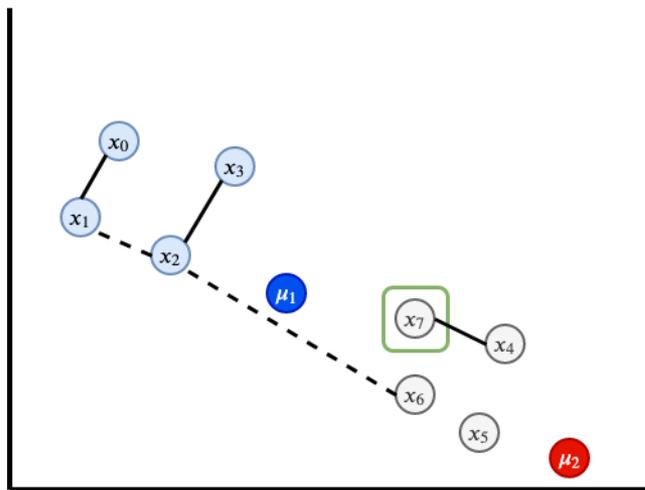
En este caso una de las asignaciones produce peor *infeasibility* que otra. **Escogeremos siempre la que menos incrementa *infeasibility***, que resulta ser la del cluster  $c_1$ , que no incumple restricciones

# La Solución Greedy: Ejemplo

$$\vec{x}_0 \rightarrow \vec{x}_2 \rightarrow \vec{x}_1 \rightarrow \vec{x}_3 \rightarrow \vec{x}_7 \rightarrow \vec{x}_4 \rightarrow \vec{x}_6 \rightarrow \vec{x}_5$$

Asignar  $\vec{x}_7$  a  $c_1$   $\longrightarrow$   $\Delta \textit{infeasibility} = 0$

Asignar  $\vec{x}_7$  a  $c_2$   $\longrightarrow$   $\Delta \textit{infeasibility} = 0$



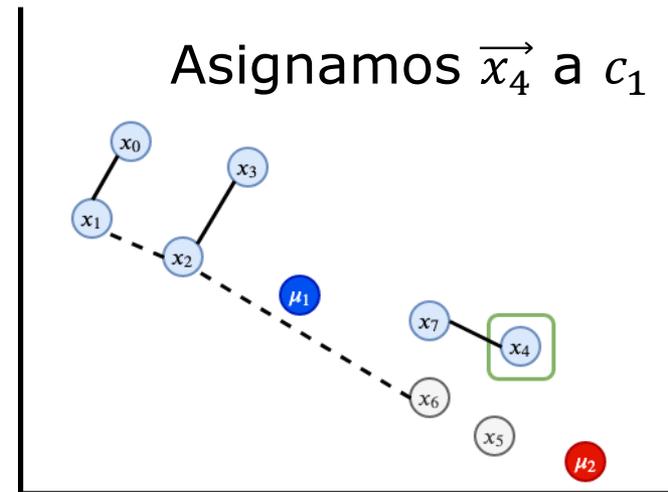
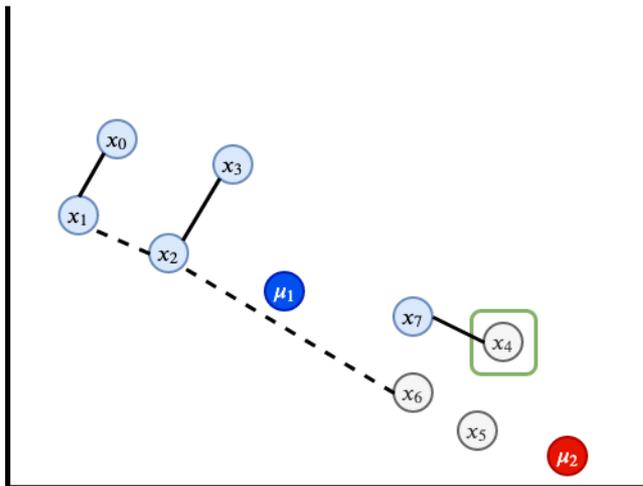
Ambos producen incremento de 0 en *infeasibility* así que asignamos  $\vec{x}_7$  al cluster asociado con el centroide más cercano,  $c_1$

# La Solución Greedy: Ejemplo

$$\vec{x}_0 \rightarrow \vec{x}_2 \rightarrow \vec{x}_1 \rightarrow \vec{x}_3 \rightarrow \vec{x}_7 \rightarrow \vec{x}_4 \rightarrow \vec{x}_6 \rightarrow \vec{x}_5$$

Asignar  $\vec{x}_4$  a  $c_1$   $\longrightarrow$   $\Delta \textit{infeasibility} = 0$

Asignar  $\vec{x}_4$  a  $c_2$   $\longrightarrow$   $\Delta \textit{infeasibility} = 1$  (Viola  $ML(\vec{x}_4, \vec{x}_7)$ )



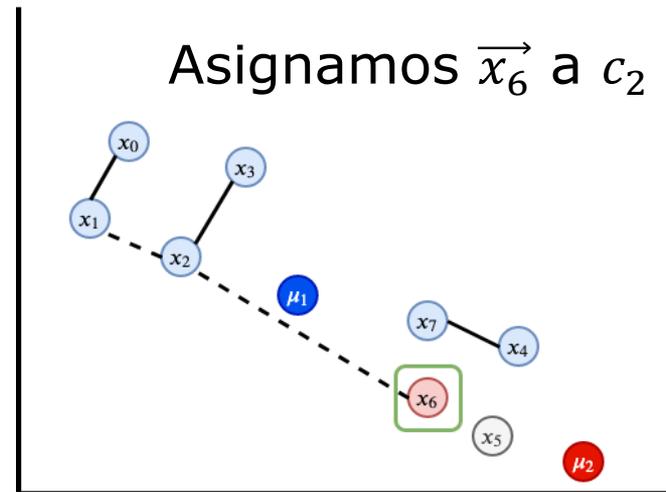
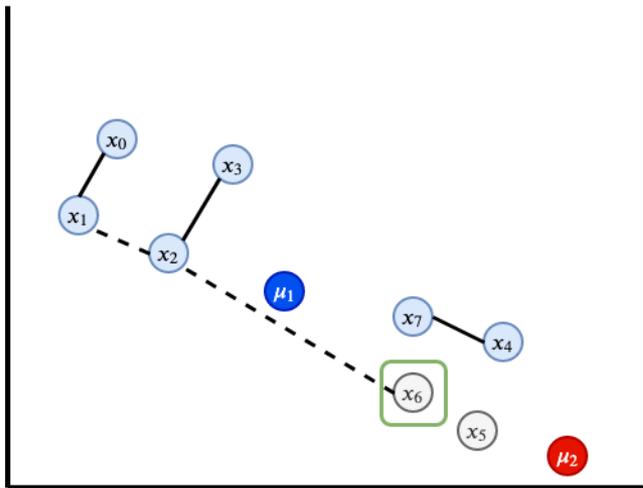
En este caso una de las asignaciones produce peor *infeasibility* que otra. **Escogeremos siempre la que menos incrementa *infeasibility***, que en resulta ser la del cluster con el **segundo centroide más cercano**,  $c_1$

# La Solución Greedy: Ejemplo

$$\vec{x}_0 \rightarrow \vec{x}_2 \rightarrow \vec{x}_1 \rightarrow \vec{x}_3 \rightarrow \vec{x}_7 \rightarrow \vec{x}_4 \rightarrow \vec{x}_6 \rightarrow \vec{x}_5$$

Asignar  $\vec{x}_6$  a  $c_1$   $\longrightarrow$   $\Delta \textit{infeasibility} = 1$  (Viola  $CL(\vec{x}_2, \vec{x}_6)$ )

Asignar  $\vec{x}_6$  a  $c_2$   $\longrightarrow$   $\Delta \textit{infeasibility} = 0$



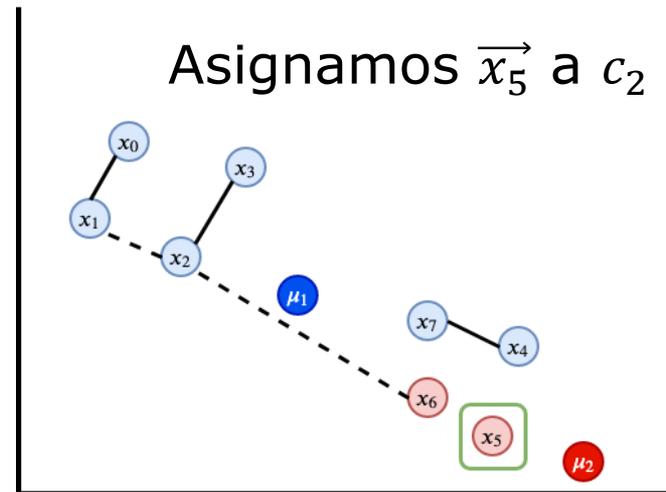
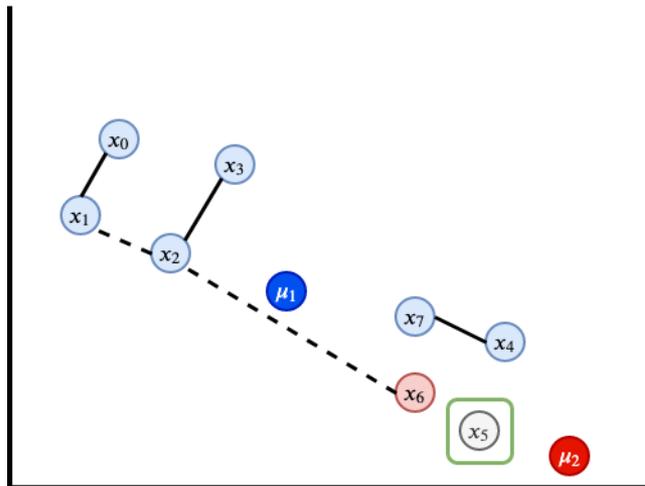
En este caso una de las asignaciones produce peor *infeasibility* que otra. **Escogeremos siempre la que menos incrementa *infeasibility***, que resulta ser la del cluster con el centroide más cercano,  $c_2$

# La Solución Greedy: Ejemplo

$$\vec{x}_0 \rightarrow \vec{x}_2 \rightarrow \vec{x}_1 \rightarrow \vec{x}_3 \rightarrow \vec{x}_7 \rightarrow \vec{x}_4 \rightarrow \vec{x}_6 \rightarrow \vec{x}_5$$

Asignar  $\vec{x}_5$  a  $c_1$   $\longrightarrow \Delta \textit{infeasibility} = 0$

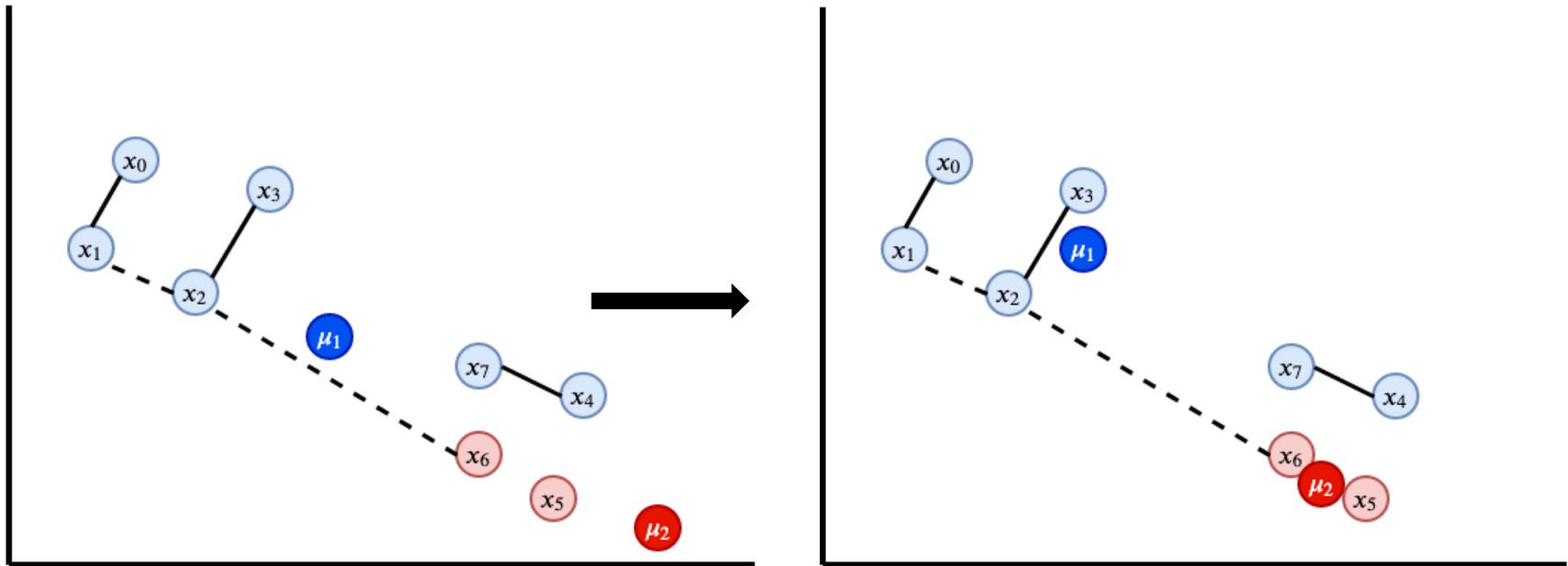
Asignar  $\vec{x}_5$  a  $c_2$   $\longrightarrow \Delta \textit{infeasibility} = 0$



Ambos producen incremento de 0 en *infeasibility* así que asignamos  $\vec{x}_5$  al cluster asociado con el centroide más cercano,  $c_2$

# La Solución Greedy: Ejemplo

Se han explorado todas las instancias. Se pasa a actualizar los centroides:



Esta solución candidata tendría una *infeasibility* = 1, solo se incumple la restricción  $CL(\vec{x}_1, \vec{x}_2)$

**Como se ha producido cambio en la partición el proceso de agrupamiento continúa**

# La Solución Greedy: Resumen

---

Para cada instancia  $\vec{x}_i$ , calculamos el cambio que se produce en *infeasibility* cuando la asignamos a cada cluster  $c_i$ . De entre las asignaciones que menos incremento producen elegimos la asociada al cluster  $c_j$  con el centroide más cercano  $\vec{\mu}_j$

Este método es altamente dependiente del orden en el que se exploran las instancias de  $X$ . **Ordenes de exploración distintos producen particiones con distintos valores para *infeasibility* y para  $\bar{C}$**

Por tanto, **este algoritmo greedy no es determinista**, puede dar distintas soluciones ante distintos órdenes de exploración del conjunto de datos y ante distintos centroides iniciales

# Búsqueda por Trayectorias Simples

---

El método de búsqueda por trayectorias simples que usaremos para aplicar al PAR es la **Búsqueda Local**

De nuevo haremos una **interpretación débil de las restricciones**

Para construir un esquema de optimización por búsqueda local tenemos que definir la **función objetivo y el operador de vecino**, así como el esquema de representación del problema

# Búsqueda por Trayectorias Simples: Función Objetivo

---

Al utilizar restricciones débiles tenemos que plantear una función objetivo que optimice a la vez la desviación general y el número de restricciones incumplidas (minimización):

$$f = \overline{C} + (infeasibility * \lambda)$$

Desviación General

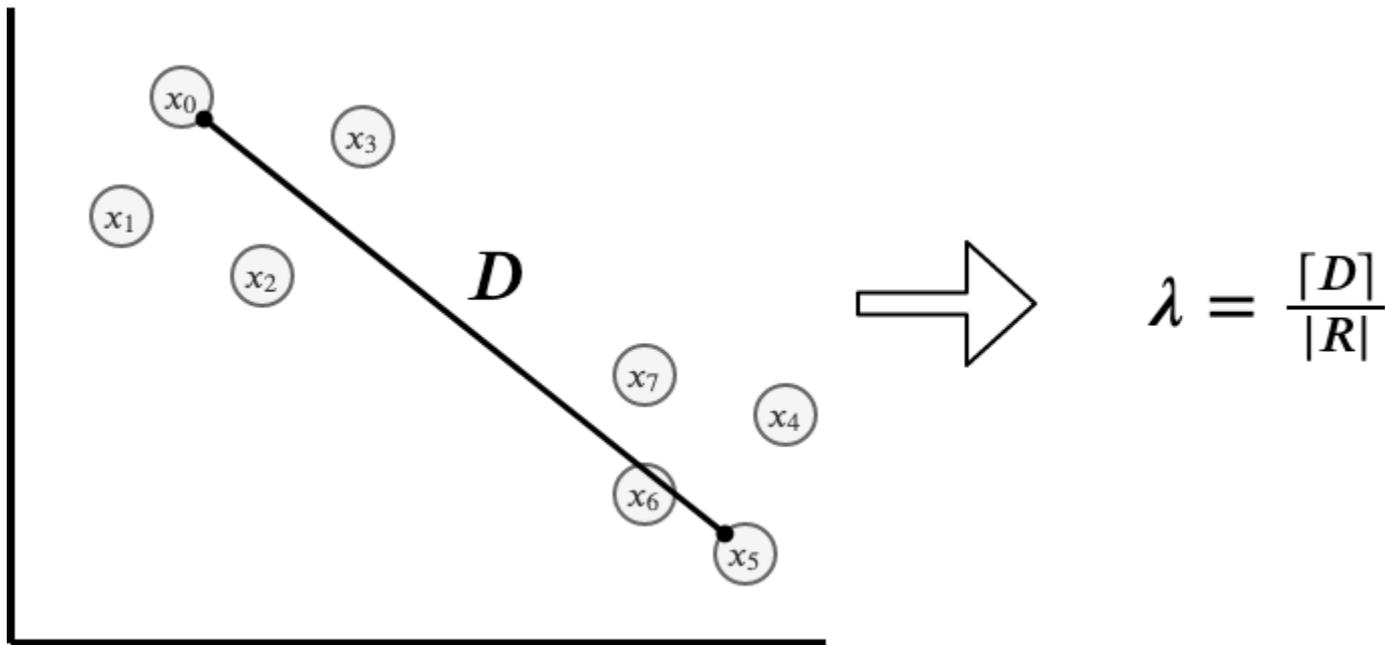
Número De Restricciones Incumplidas

La función propuesta utiliza el parámetro  $\lambda$  como parámetro de escalado para dar relevancia al término *infeasibility*. Si se establece correctamente, la búsqueda local optimiza primero el número de restricciones incumplidas y luego la desviación general

# Búsqueda por Trayectorias Simples: Función Objetivo

---

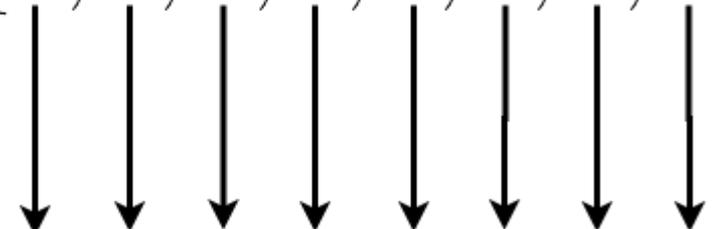
Para asegurar que el factor *infeasibility* tiene la suficiente relevancia establecemos  $\lambda$  siempre como **el cociente entre un valor mayor a la distancia máxima existente en el conjunto de datos y el número de restricciones del problema:**



# Búsqueda por Trayectorias Simples: Representación

---

Para representar la solución al problema emplearemos un vector de  $n$  posiciones en correspondencia con las  $n$  instancias del conjunto de datos  $X$ . En cada posición se almacena el cluster al que se asigna la instancia de dicha posición:

$$C = \{ \overset{x_0}{1}, \overset{x_1}{1}, \overset{x_2}{1}, \overset{x_3}{1}, \overset{x_4}{2}, \overset{x_5}{2}, \overset{x_6}{2}, \overset{x_7}{2} \}$$

$$S = [ \overset{s_0}{1}, \overset{s_1}{1}, \overset{s_2}{1}, \overset{s_3}{1}, \overset{s_4}{2}, \overset{s_5}{2}, \overset{s_6}{2}, \overset{s_7}{2} ]$$

# Búsqueda por Trayectorias Simples: Operador de Vecino

---

El operador de vecino considerado es el **cambio de asignación de cluster de una instancia** ( $Cambio\_Cluster(S, i, l)$ ). Dada una solución (configuración de clusters)  $S$  se escoge un índice  $i$  (instancia  $\vec{x}_i$ ) y se cambia su asignación a otro cluster distinto  $l$ :

$$S = \{s_0, \dots, \mathbf{s}_i, \dots, s_n\} \Rightarrow S' = \{s_1, \dots, \mathbf{l}, \dots, s_n\}$$

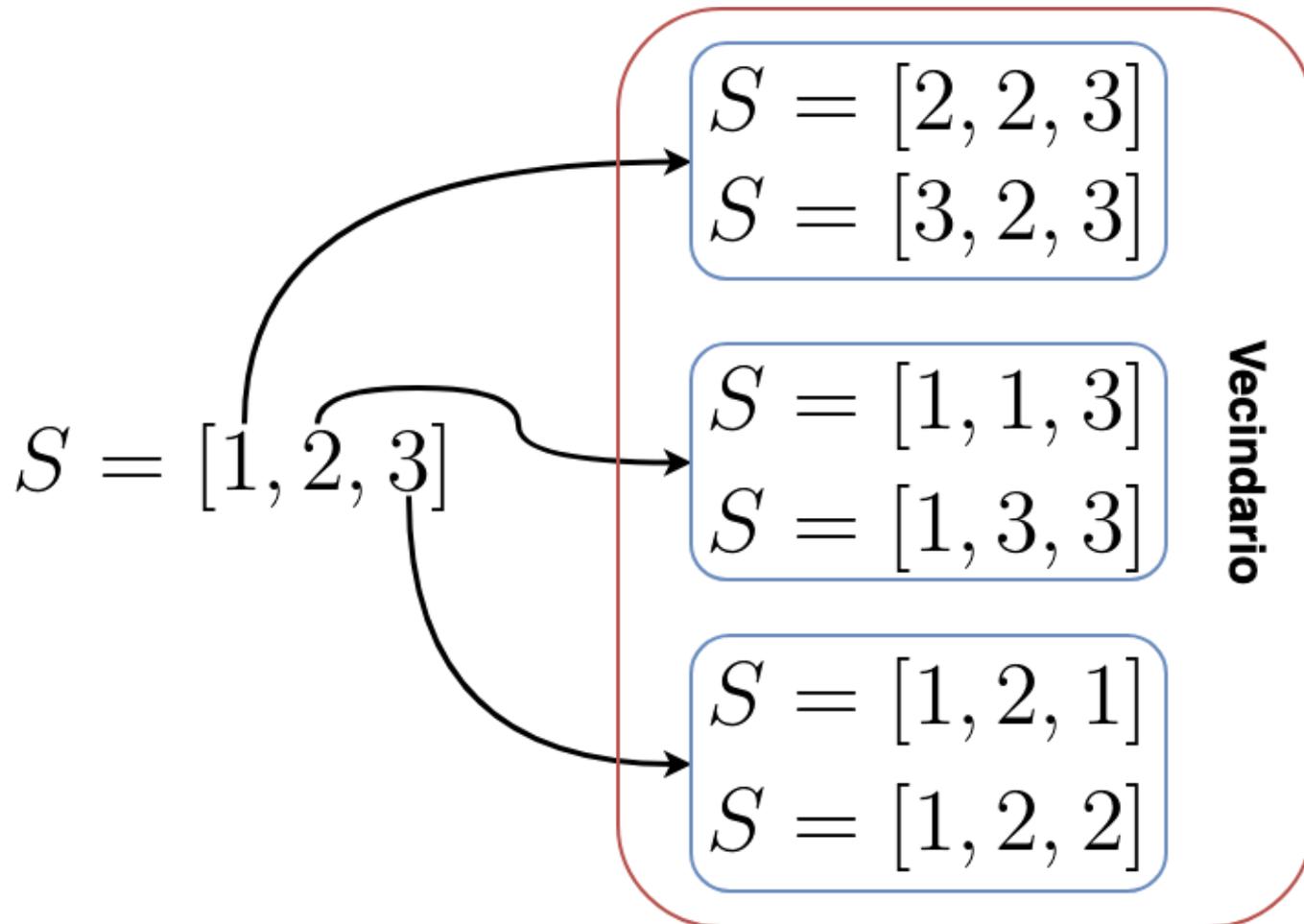
Es necesario chequear que el cambio no deja ningún cluster vacío, ya que no verificaría las restricciones del problema del clustering

El entorno de una solución  $S$  está formado por las soluciones accesibles desde ella a través de la aplicación de dicho operador sobre todos los índices de  $S$ ,  $i \in \{1, \dots, n\}$ , estudiando todos los posibles valores alternativos de  $s_i$  de  $X$  de un cluster a otro

Por tanto, **el tamaño del entorno de una solución es  $n \cdot (k - 1)$**

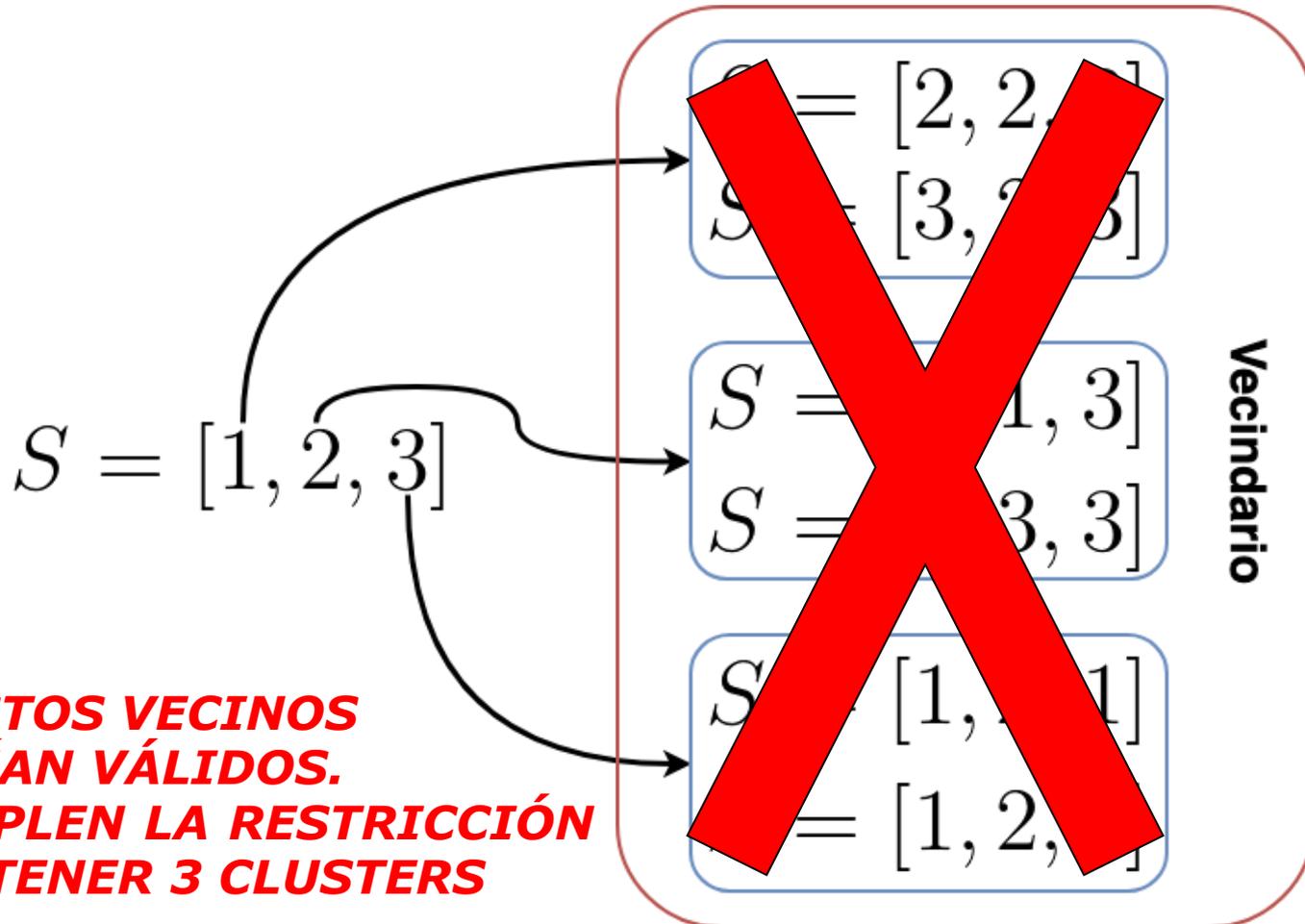
# Búsqueda por Trayectorias Simples: Operador de Vecino

Por ejemplo, para  $n = 3$  y  $k = 3$  (tres clusters:  $l_i \in \{1,2,3\} \forall i \in \{1, \dots, n\}$ ), y dada una solución inicial  $S$ , el operador de vecino genera:



# Búsqueda por Trayectorias Simples: Operador de Vecino

Por ejemplo, para  $n = 3$  y  $k = 3$  (tres clusters:  $l_i \in \{1,2,3\} \forall i \in \{1, \dots, n\}$ ), y dada una solución inicial  $S$ , el operador de vecino genera:



# Búsqueda por Trayectorias Simples: El Algoritmo

---

El algoritmo propuesto utiliza una estrategia tipo “**el primer mejor**” para explorar el entorno de una solución. Esto es, se acepta el primer vecino que mejore a la solución actual como nueva solución

La exploración del entorno no se hace en un orden determinístico sino en un orden aleatorio para dar diversidad

Para evaluar la calidad de las soluciones utilizamos la función objetivo  $f$ , que hay que minimizar. De esta forma se acepta como nueva solución el primer vecino  $S'$  que cumpla  $f(S') < f(S)$

El proceso termina cuando se alcanza el número máximo de evaluaciones de la función objetivo o cuando no se produce mejoría al explorar un vecindario concreto ( $\nexists S' t. q. f(S') < f(S)$ )

La solución inicial se genera de manera aleatoria, comprobando que cada cluster tiene al menos una instancia

# Búsqueda por Trayectorias Simples: El Algoritmo

---

No es necesario generar el entorno completo para explorarlo. En su lugar generamos un vecindario virtual que consiste en parejas índice-valor

Esto es, generamos los cambios posibles sobre la solución actual, no las nuevas soluciones en sí. Al momento de evaluar un vecino de la solución actual se aplica el cambio correspondiente y se evalúa. Gracias a este mecanismo la memoria ocupada por el vecindario es mucho menor

Otra ventaja de este mecanismo es que el proceso de aleatorización es más simple. En lugar de barajar el vecindario real para determinar el orden de exploración, barajamos el vecindario virtual (parejas de tamaño invariable)

¡Cuidado! **Barajar el entorno no es lo mismo que generarlo de forma aleatoria.** Si se barajan los individuos no se repiten, si se selecciona aleatoriamente sí se podrían repetir

# Búsqueda por Trayectorias Simples: Ejemplo vecindarios

Un ejemplo de vecindario virtual para  $n = 3$  y  $k = 3$  (tres clusters:  $l_i \in \{1,2,3\} \forall i \in \{1, \dots, n\}$ ), y dada una solución inicial  $S = [1,2,3]$ , el operador de vecindario genera:

## Vecindario Real

$$\begin{array}{lll} S = [2, 2, 3] & S = [1, 1, 3] & S = [1, 2, 1] \\ S = [3, 2, 3] & S = [1, 3, 3] & S = [1, 2, 2] \end{array}$$

## Vecindario Virtual

$$\begin{array}{lll} (1, 2) & (2, 1) & (3, 1) \\ (1, 3) & (2, 3) & (3, 2) \end{array}$$

El formato de los individuos del vecindario virtual es:

$$(indice, nuevo\ valor) = (i, l_i')$$

# Búsqueda por Trayectorias Simples: Ejemplo vecindarios

Un ejemplo de vecindario virtual para  $n = 3$  y  $k = 3$  (tres clusters:  $l_i \in \{1,2,3\} \forall i \in \{1, \dots, n\}$ ), y dada una solución inicial  $S = [1,2,3]$ , el operador de vecindario genera:

**Vecindario Real**

$S = [1, 2, 3]$   $S = [1, 1, 3]$   $S = [1, 2, 1]$   
 $S = [3, 2, 1]$   $S = [1, 3, 2]$   $S = [1, 2, 2]$

**ESTOS VECINOS  
NO SON VÁLIDOS.  
SÓLO GUARDAMOS  
LOS FACTIBLES**

$(1, 1)$   $(2, 1)$   $(3, 1)$   
 $(1, 2)$   $(2, 2)$   $(3, 2)$   
 $(1, 3)$   $(2, 3)$   $(3, 2)$

El formato de los individuos del vecindario virtual es:

$$(indice, nuevo valor) = (i, l_i')$$

# Casos del problema: Conjuntos de Datos

---

- **Zoo:** Contiene información sobre distintas características (13 atributos) de distintos tipos de animales, y hay que clasificarlos. Tiene 7 clases ( $k=7$ ) y 101 instancias
- **Glass:** Agrupar distintos tipos de vidrio en función de 9 atributos sobre sus componentes químicos. Tiene 7 clases ( $k=7$ ) y 214 instancias
- **Bupa:** Hay que agrupar personas en función de sus hábitos de consumo de alcohol, definido con 5 atributos. Tiene 16 clases ( $k=16$ ) y 345 instancias

# Casos del problema: Conjuntos de Restricciones

---

Para cada conjunto de datos se generan aleatoriamente 2 conjuntos de restricciones, correspondientes al 10% y 20% del total de restricciones posibles

Los llamaremos  $CS_{10}$  y  $CS_{20}$ , respectivamente

Con esto tenemos que, si para cada conjunto de datos generamos dos conjuntos de restricciones, el total de casos de estudio es 6

	Instancias	Atributos	Clases	$CS_{10}$		$CS_{20}$		Distancia Óptima
				ML	CL	ML	CL	
<b>Zoo</b>	101	16	7	107	379	220	692	0,9048
<b>Glass</b>	214	9	7	573	1594	1078	3018	0,3643
<b>Bupa</b>	345	5	16	872	4760	1664	9160	0,2292

## Casos del problema: Formato

---

Los ficheros `.dat` corresponden a los conjuntos de datos. Los ficheros `.const` corresponden a los conjuntos de restricciones. Por ejemplo: `zoo_set.dat` es el fichero de datos de Zoo y los conjuntos de restricciones son `zoo_const_set_10.const` e `zoo_set_const_20.const`

Los ficheros de datos contienen en cada fila una instancia del conjunto en cuestión con sus características separadas por comas (sin espacios)

Los ficheros de restricciones almacenan una matriz de restricciones con el formato descrito anteriormente, de nuevo con sus valores separados por comas y sin espacios

## Algunos consejos

---

- **Las restricciones se dan en forma de matriz pero es recomendable construir la lista de restricciones y tener ambas disponibles para usar la más eficiente en cada caso**
- Estamos trabajando con números reales así que hay que tener cuidado con las comparaciones de igualdad
- Una manera de saber si la implementación es correcta es comprobar que la calidad de la solución escala con el tamaño del conjunto de restricciones y con la infactibilidad conseguida (en el caso medio). Si no es así es posible que haya algún fallo

# Agradecimientos

---

- Para la preparación de las transparencias de presentación del PAR se han usado materiales del estudiante de doctorado Germán González Almagro
- Agradecemos también a Germán la preparación de las instancias del problemas, ficheros y las pruebas previas realizadas