ALGORITMOS VORACES

UNIVERSIDAD
COMPLUTENSE
MADRID

ALBERTO VERDEJO

Problema del cambio de monedas

- Pagar de forma exacta cierta cantidad, utilizando el menor número de monedas.

- ¿Cómo pagarías 1,47 €?

Problema del cambio de monedas

- El método voraz construye una solución a través de una secuencia de pasos, hasta que se alcanza una solución completa al problema.
- En cada paso, la elección debe ser:
  - *factible*, es decir, tiene que satisfacer las restricciones del problema
  - *óptima localmente*, es decir, la mejor opción local entre todas las disponibles en ese paso
  - *irrevocable*, es decir, no se puede cambiar en los pasos posteriores del algoritmo

Problema del cambio de monedas, no siempre funciona

- La estrategia voraz no funciona para cualquier sistema monetario.

- Si queremos pagar 30, la solución voraz sería esta, con 6 monedas:

Problema del cambio de monedas, no siempre funciona

- La estrategia voraz no funciona para cualquier sistema monetario.

- Mientras que la solución óptima utiliza solamente 3 monedas:

Método voraz

Las características generales de los algoritmos voraces son:

- Para construir la solución se dispone de un *conjunto de candidatos*. Se van formando dos conjuntos: los candidatos *seleccionados* (formarán parte de la solución), y los *rechazados* definitivamente.
- Existe una *función de selección* que indica cuál es el candidato más prometedor de entre los aún no considerados.
- Existe un *test de factibilidad* que comprueba si un candidato es compatible con la solución parcial construida hasta el momento.

Método voraz

Las características generales de los algoritmos voraces son:

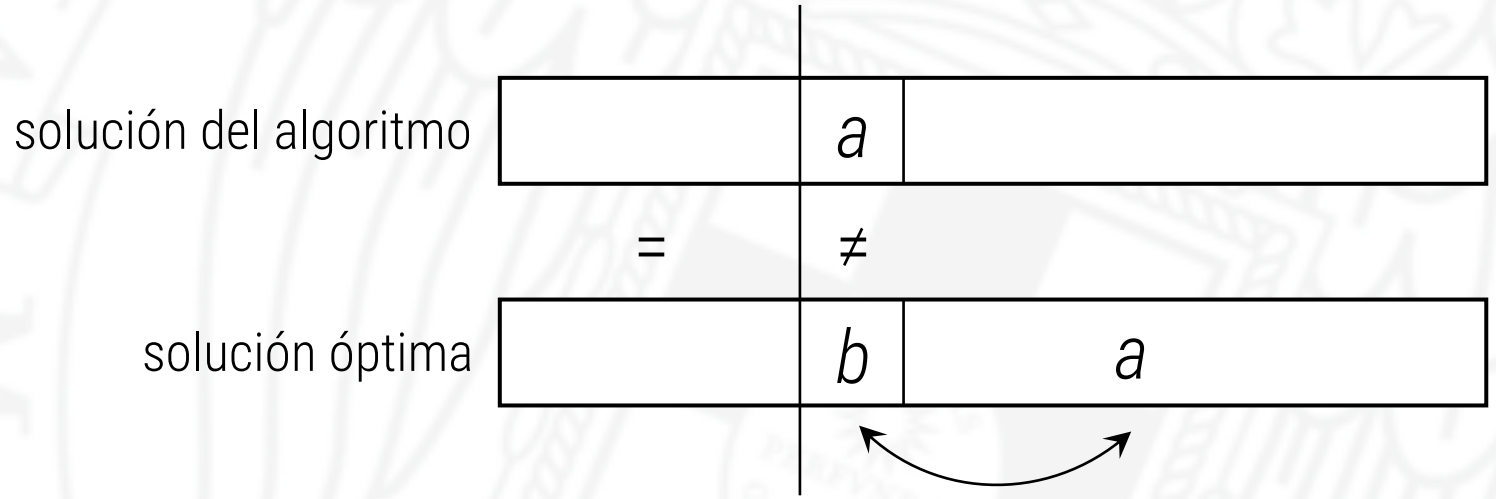
- Existe un *test de solución* que determina si una solución parcial forma una solución "completa".
- Se tiene que obtener una solución óptima según una *función objetivo* que asocia un valor a cada solución.

Método voraz

```
fun voraz(datos : conjunto) dev S : conjunto
var candidatos : conjunto
    S := ∅      { en S se va construyendo la solución }
    candidatos := datos
    mientras candidatos ≠ ∅ ∧ ¬es-solución?(S) hacer
        x := seleccionar(candidatos)
        candidatos := candidatos – {x}
        si es-factible?(S ∪ {x}) entonces S := S ∪ {x} fsi
    fmientras
ffun
```

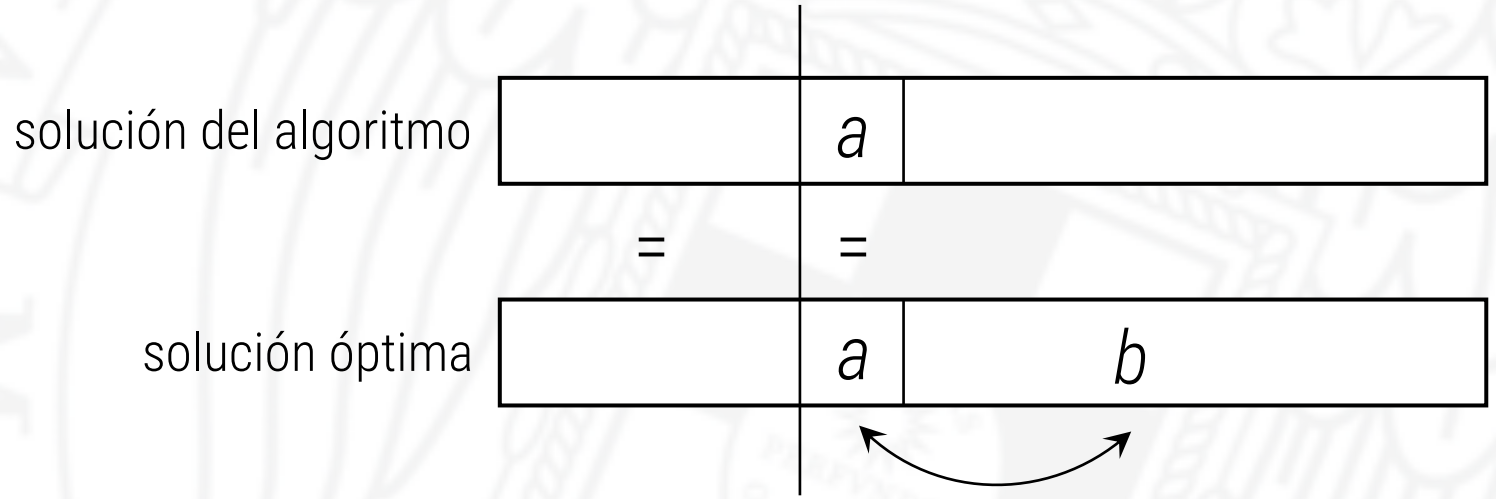
Demostración de corrección

- Método de *reducción de diferencias*: comparar una solución óptima con la solución obtenida por el algoritmo voraz. Si ambas soluciones no son iguales, se va transformando la solución optima de partida de forma que continúe siendo óptima, pero siendo más parecida a la del algoritmo voraz.



Demostración de corrección

- Método de *reducción de diferencias*: comparar una solución óptima con la solución obtenida por el algoritmo voraz. Si ambas soluciones no son iguales, se va transformando la solución optima de partida de forma que continúe siendo óptima, pero siendo más parecida a la del algoritmo voraz.



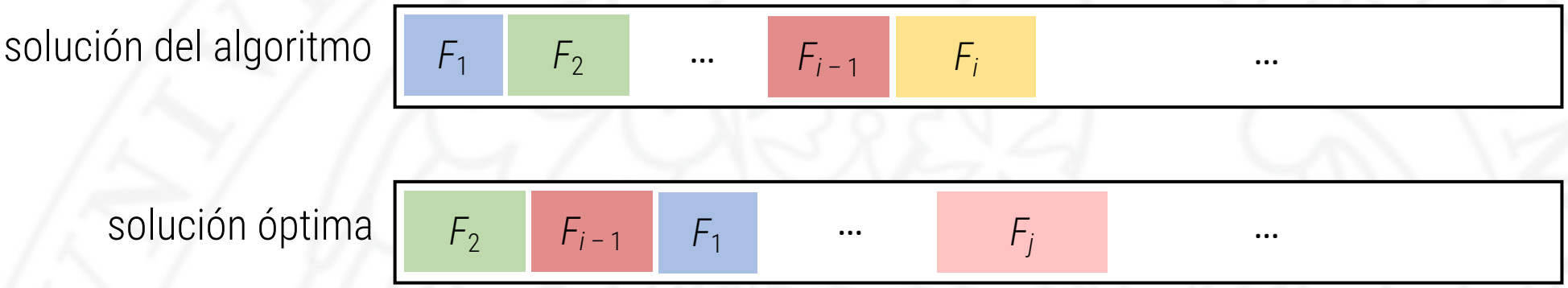
Maximizar número de ficheros en disco

- Tenemos n ficheros $F_1, F_2, \ldots, F_n$ que queremos almacenar en un disco.
- El fichero F_i requiere s_i megabytes de espacio de disco y la capacidad del disco es D megabytes.
- Queremos *maximizar el número de ficheros* almacenados en el disco.
- Supondremos que $D < \sum_{i=1}^{n} s_i$.
- Consideramos los programas *de menor a mayor tamaño*.

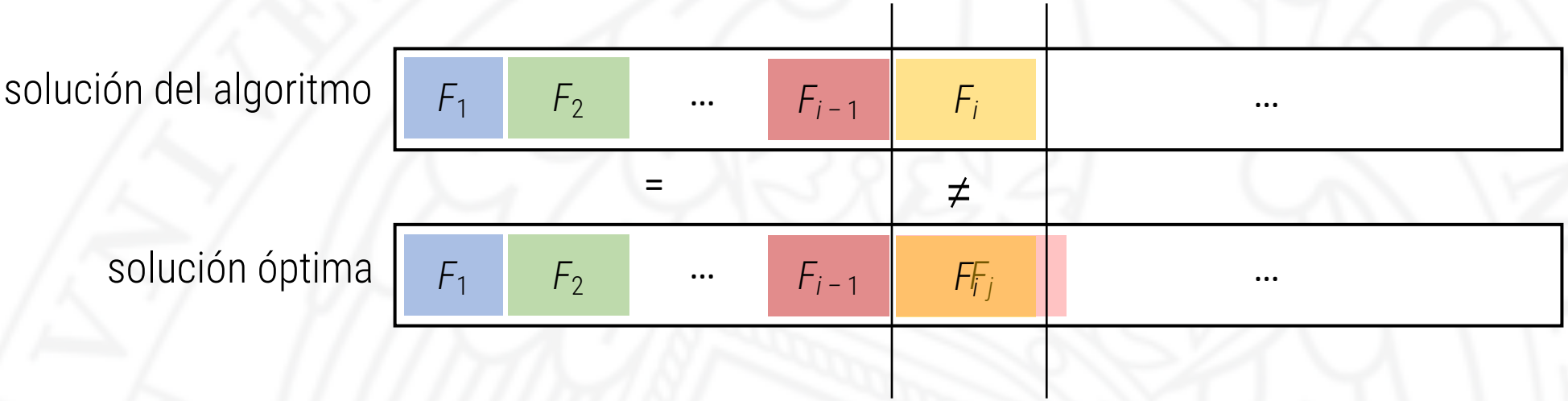
Maximizar número de ficheros en disco

```
// S[0] ≤ S[1] ≤ ... ≤ S[n-1] ∧ D < ∑_{i=0}^{n-1} S[i]
vector<bool> ficheros_en_disco(vector<int> const& S, int D) {
    vector<bool> solucion(S.size(), false);
    int acumula = 0;
    for (int i = 0; acumula + S[i] <= D; ++i) {
        solucion[i] = true;
        acumula += S[i];
    }
    return solucion;
}
```

Maximizar número de ficheros en disco, corrección

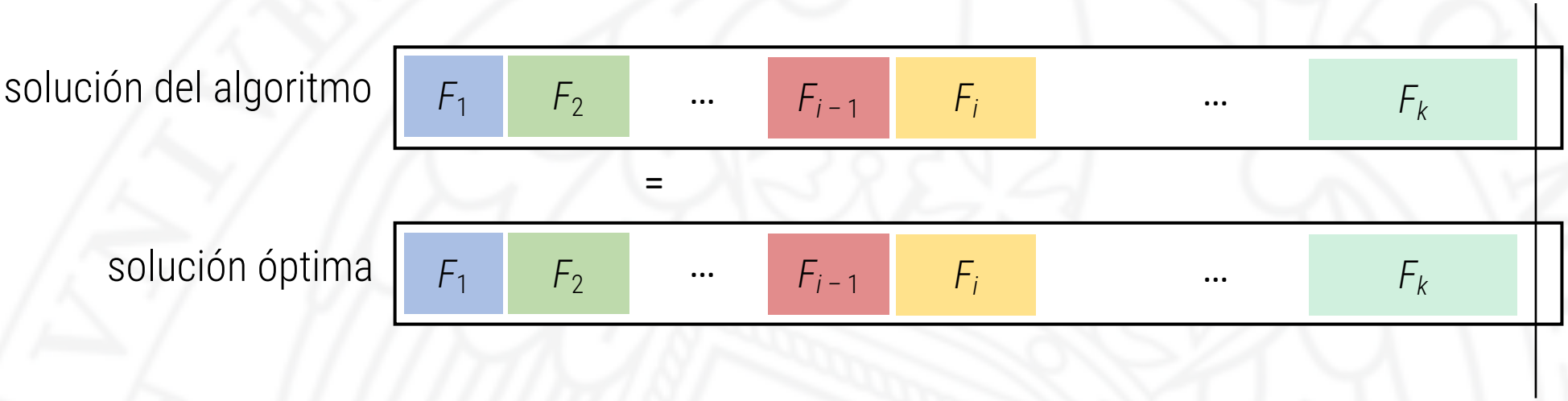


Maximizar número de ficheros en disco, corrección



- F_i no está en la solución óptima
- $j > i \implies s_i \leq s_j$

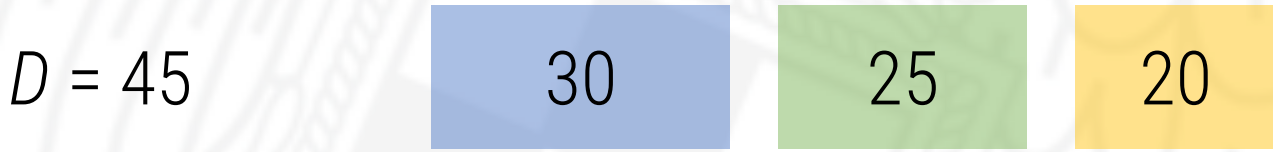
Maximizar número de ficheros en disco, corrección



- F_i no está en la solución óptima
- $j > i \implies s_i \leq s_j$
- Podemos seguir haciendo transformaciones con los k ficheros de la solución del algoritmo

Maximizar ocupación del disco

- No existe estrategia voraz para este problema.
- Buscar contraejemplos que demuestren que una estrategia no funciona en todos los casos.
- Contraejemplo para la estrategia que considera los programas de mayor a menor tamaño:



Maximizar ocupación del disco

- No existe estrategia voraz para este problema.
- Buscar contraejemplos que demuestren que una estrategia no funciona en todos los casos.
- Contraejemplo para la estrategia que considera los programas de menor a mayor tamaño: