CI2613: Algoritmos y Estructuras III

Blai Bonet

Universidad Simón Bolívar, Caracas, Venezuela

Enero-Marzo 2015

Aplicaciones

Aplicaciones

Existen diversas aplicaciones de los algoritmos vistos hasta ahora

Veremos 4 aplicaciones distintas:

- Agrupamiento ó "clustering" de objetos en **Machine Learning**
- Cálculo de secuencias de arbitraje en **finanza**
- Solución de sistemas de diferencias en **optimización combinatoria**
- Alineación de secuencias de ADN en **bioinformática**

Clustering

k-clustering simple

Dada un universo $U = \{p_1, p_2, \ldots, p_n\}$, se quiere **particionar** los objetos en k **grupos o clusters** de forma que cada grupo contenga "objetos similares"

Los objetos representan entidades abstractas; estos pueden ser imágenes, documentos, páginas web, vectores, etc.

La **similitud** entre objetos se mide con una función de (di)similitud o distancia $d : U \times U \to \mathbb{R}^{\geq 0}$

La similitud entre $p, q \in U$ es $d(p, q)$

Las uńicas suposiciones son:

- $d(p, p) = 0$ para todo objeto $p \in U$
- $d(p, q) = d(q, p)$ para todo par de objetos $p, q \in U$ (simetría)

k-clustering simple: Formulación

Un k-clustering es una partición $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ de U

Decimos que dos puntos p y q son $\mathcal{C}$-similares si ambos pertenecen a un mismo cluster $C_i \in \mathcal{C}$

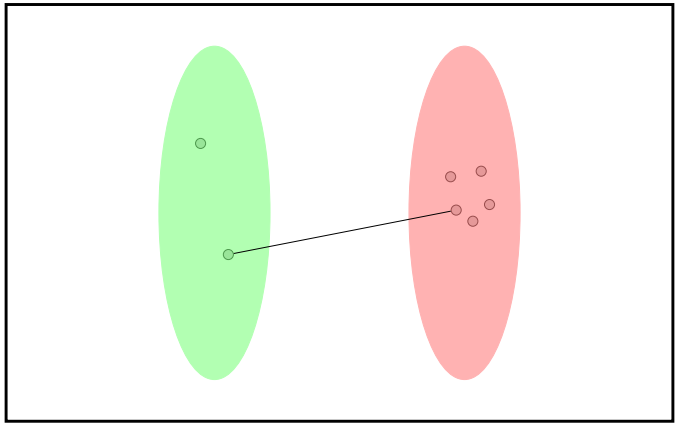
La **separación de $\mathcal{C}$** es la mínima separación entre puntos no $\mathcal{C}$-similares; i.e.

$$sep(\mathcal{C}) = \min\{\ d(p, q) : p, q \in U \text{ tal que } p \text{ y } q \text{ no son } \mathcal{C}\text{-similares }\}$$

Problema de k-clustering: conseguir un k-clustering $\mathcal{C}$ con máxima separación; i.e. un k-clustering $\mathcal{C}$ tal que para todo k-clustering $\widehat{\mathcal{C}}$, $sep(\mathcal{C}) \geq sep(\widehat{\mathcal{C}})$

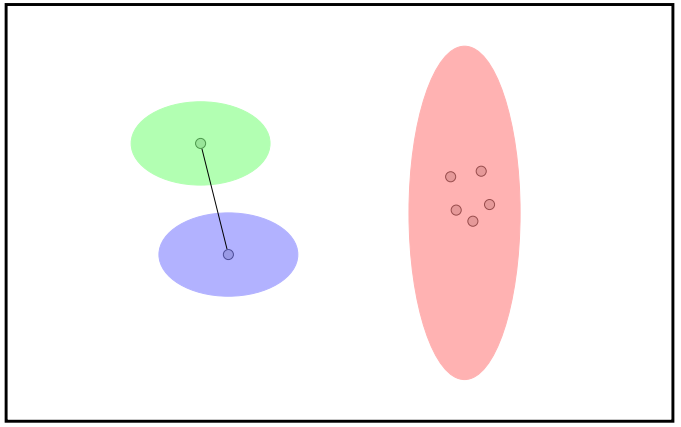
k-clustering simple: Ejemplo

Universo de 7 objetos: 2-**clustering**



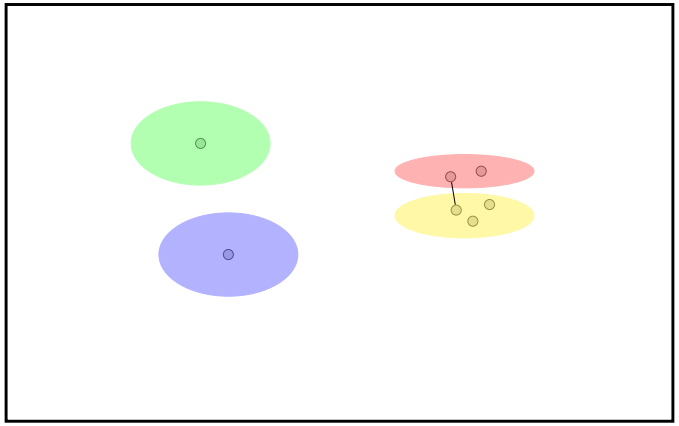
k-clustering simple: Ejemplo

Universo de 7 objetos: 3-**clustering**



k-clustering simple: Ejemplo

Universo de 7 objetos: 4-**clustering**



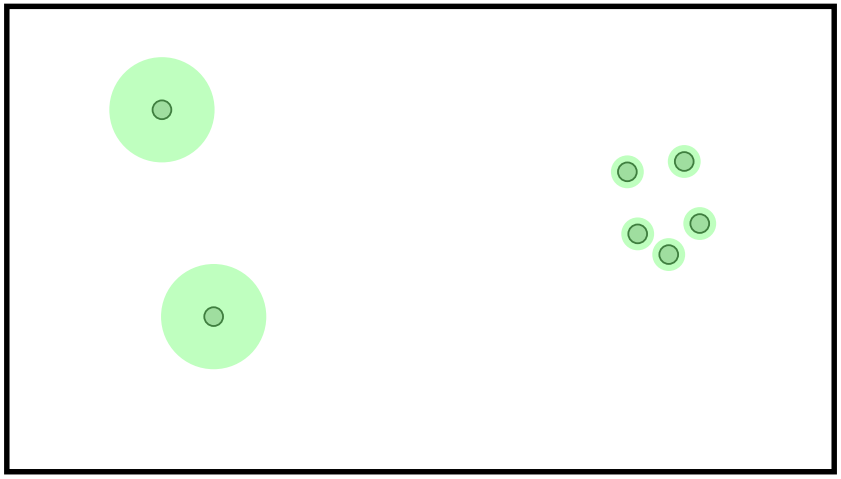
Algoritmo greedy para k-clustering

Comenzamos con la **partición más fina** en n bloques donde cada objeto es su propio cluster $\mathcal{C} = \{\{p_1\}, \{p_2\}, \ldots, \{p_n\}\}$

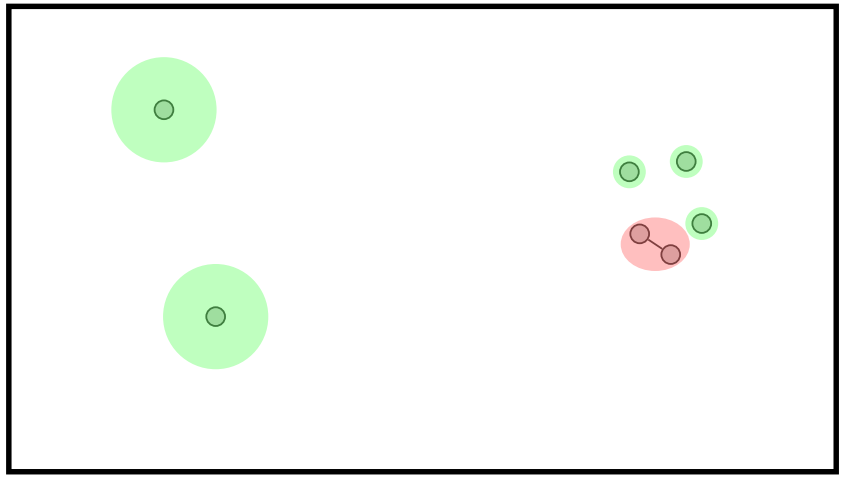
De forma iterativa:

- Seleccionamos dos puntos p y q no $\mathcal{C}$-similares con mínima distancia $d(p, q)$
- Unimos los clusters C_i y C_j para p y q en un nuevo cluster C que reemplaza C_i y C_j
- Iteramos hasta que lleguemos a k clusters

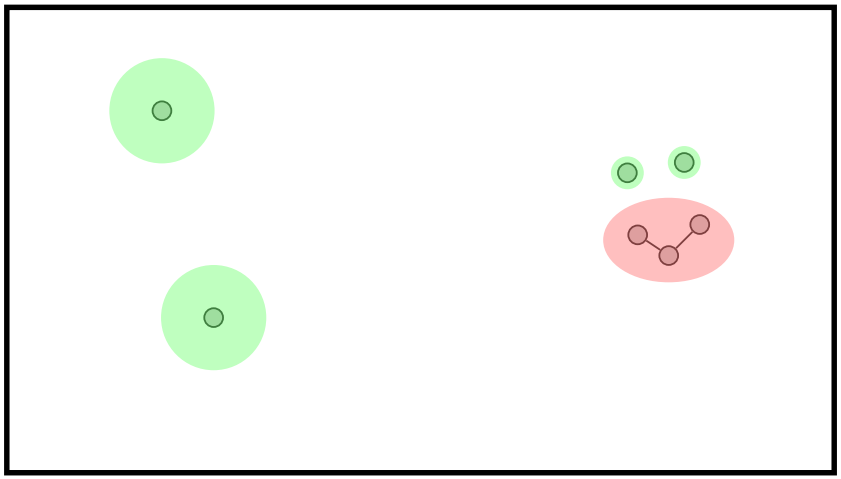
Algoritmo greedy para k-clustering: Ejemplo $k = 4$



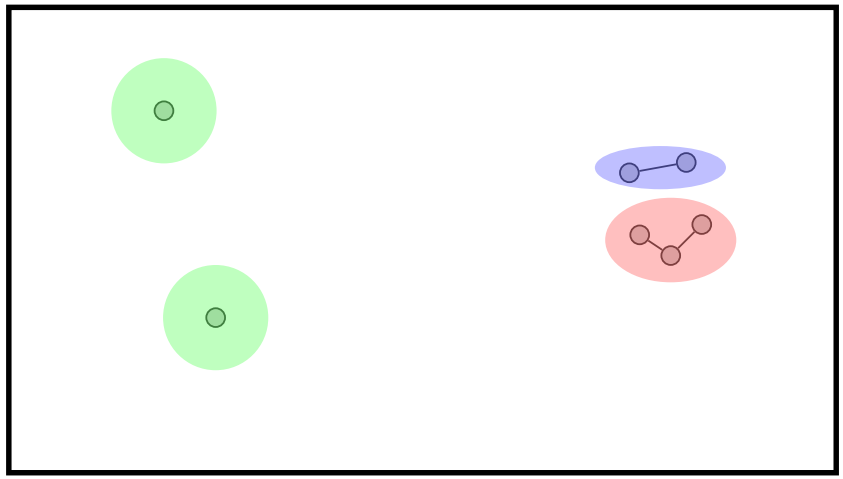
Algoritmo greedy para k-clustering: Ejemplo $k = 4$



Algoritmo greedy para k-clustering: Ejemplo $k = 4$



Algoritmo greedy para k-clustering: Ejemplo $k = 4$



Algoritmo greedy para k-clustering: Pseudocódigo

```
1  bool Clustering(U={p1,p2,...,pn}, d):
2
3      % inicialización
4      foreach Objeto p ∈ U: make-set(p)
5
6      % algoritmo de greedy
7      while |C| > k
8          Seleccionar dos objetos p y q tal que d(p,q)
9          es mínima y find(p) != find(q)
10
11         Union(p,q)
```


Correctitud del algoritmo de clustering

Teorema

Dado un universo U y función de similitud $d : U \times U \to \mathbb{R}^{\geq 0}$, el algoritmo de clustering calcula un k-clustering $\mathcal{C}$ de separación máxima

Prueba: sea $\mathcal{C}$ el k-clustering calculado por el algoritmo con $sep(\mathcal{C}) = S$. Mostraremos $S \geq sep(\widehat{\mathcal{C}})$ para todo k-clustering $\widehat{\mathcal{C}}$

Observar:

1. Si $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \ldots$ son los clusterings calculados en cada iteración:

$$sep(\mathcal{C}_1) \;\leq\; sep(\mathcal{C}_2) \;\leq\; sep(\mathcal{C}_3) \;\leq\; \cdots$$

2. Si p y q son "unidos" durante el clustering, $d(p, q) \leq S$

3. Si p y q son $\mathcal{C}$-similares, existen $p = p_1, \ldots, p_j = q$ tal que (p_i, p_{i+1}) fueron unidos durante el clustering, para $1 \leq i < j$

Correctitud del algoritmo de clustering

Sea $\widehat{\mathcal{C}}$ un k-clustering cualquiera diferente de $\mathcal{C}$

Existen puntos $p, q \in U$ que son $\mathcal{C}$-similares pero no $\widehat{\mathcal{C}}$-similares (¿por qué?)

Sean $p = p_1, \ldots, p_j = q$ tales que (p_i, p_{i+1}) fueron unidos durante el clustering

Si "caminamos" de p_1 a p_j, comenzamos en un cluster C en $\widehat{\mathcal{C}}$ y terminanos en otro cluster C' en $\widehat{\mathcal{C}}$

Por lo tanto, existe i tal que $p_i \in C$ y $p_{i+1} \notin C$

Como p_i y p_{i+1} fueron unidos, tenemos $d(p_i, p_{i+1}) \leq sep(\mathcal{C}) = S$

Sin embargo, p_i y p_{i+1} no son $\widehat{\mathcal{C}}$-similares y por lo tanto

$$sep(\widehat{\mathcal{C}}) \;\leq\; d(p_i, p_{i+1}) \;\leq\; S$$

□

Relación entre algoritmo de clustering y MST

Veremos que el algoritmo de k-clustering no es más que una versión del algoritmo de Kruskal para MST

Formulamos el problema de clustering como un grafo $G = (V, E)$:

- Los vértices V son el universo de objetos U
- Existe arista (p, q) para cada par de objetos p, q con peso $d(p, q)$

Reformulación de algoritmo para k-clustering

```
1  bool Clustering(U={p1,p2,...,pn}, d):
2
3      % inicialización
4      foreach Objeto p ∈ U: make-set(p)
5
6      % ordenar aristas
7      Ordenar aristas (p,q) de forma no-decreciente segun peso d(p,q)
8
9      % algoritmo de greedy
10     foreach arista (p,q) en orden de peso
11         if |C| <= k: break
12         if find(p) != find(q)
13             Union(p,q)
```

La única diferencia entre los algoritmos de Kruskal y k-clustering es la línea 11 que Kruskal no tiene!

Cálculo de k-clustering óptimo en tiempo $O(E \log V) = O(n^2 \log n)$

□

Arbitraje

Arbitraje

Wikipedia:

> *Arbitraje es la práctica de tomar ventaja de una diferencia de precio entre dos o más mercados: realizar una combinación de transacciones complementarias que capitalizan el desequilibrio de precios. La utilidad se logra debido a la diferencia de precios de los mercados. Por medio de arbitraje, los participantes en el mercado pueden lograr una utilidad instantánea libre de riesgo. El término es comúnmente aplicado a las transacciones de instrumentos financieros, como bonos, acciones, derivados financieros, mercancías y monedas.*
>
> *Si los precios de mercado no permiten la ejecución de arbitraje rentable, se dice que los precios constituyen un equilibrio de arbitraje. El equilibro de arbitraje es una precondición para un equilibrio económico general.*

Arbitraje de tipos de cambio

Considere un conjunto de monedas $M_1, \ldots, M_2, M_K$ con tasas de conversión $R[i, j]$ para $1 \leq i, j \leq K$

Una secuencia $M_{i_1}, M_{i_2}, \ldots, M_{i_n}$ de cambios es una **secuencia de arbitraje** si y solo sí al

- cambiar 1 unidad de M_{i_1} a la moneda M_{i_2}
- cambiar lo obtenido en M_{i_2} a la moneda M_{i_3}
- $\ldots$
- cambiar lo obtenido en $M_{i_{n-1}}$ a la moneda M_{i_n}
- finalizar cambiando lo obtenido en M_{i_n} a la moneda M_1

se obtiene más de 1 unidad de M_{i_1}

Arbitraje de tipos de cambio

Lema

Una secuencia $M_{i_1}, M_{i_2}, \ldots, M_{i_n}$ es de arbitraje si y solo sí $R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{n-1}, i_n] \cdot R[i_n, i_1] > 1$

Prueba: inmediato de la defición □

Corolario

Una secuencia $M_{i_1}, M_{i_2}, \ldots, M_{i_n}$ es de arbitraje si y solo sí $-\log R[i_1, i_2] - \log R[i_2, i_3] - \cdots - \log R[i_{n-1}, i_n] - \log R[i_n, i_1] < 0$

Algoritmo para arbitraje de tipos de cambio

Considere un conjunto de monedas $M_1, M_2, \ldots, M_K$ con tasas de conversión $R[i, j]$

Considere el digrafo $G = (V, E)$ donde $V = \{1, 2, \ldots, K\}$, y aristas (i, j) para todo $1 \leq i, j \leq K$ con $i \neq j$

Considere los pesos $w : E \to \mathbb{R}$ dados por $w(i, j) = -\log R[i, j]$

Teorema

Existe una secuencia de arbitraje si y sólo si el digrafo $G = (V, E)$ con pesos $w : E \to \mathbb{R}$ tiene un ciclo de costo negativo. El ciclo de costo negativo constituye una secuencia de arbitraje

Algoritmo para arbitraje de tipos de cambio

El algoritmo de Bellman-Ford detecta la existencia de un ciclo de peso negativo alcanzable desde un vértice fuente s

En el problema de arbitraje no existe un vértice fuente y queremos encontrar un ciclo de costo negativo en el grafo

Ejercicios:

- Sea $G = (V, E)$ un grafo dirigido con pesos $w : E \to \mathbb{R}$. Consiga un algoritmo eficiente que detecta si G tiene un ciclo de peso negativo. El algoritmo debe correr en tiempo $O(VE)$
- Modifique el algoritmo anterior para que devuelva un ciclo de costo negativo si G contiene al menos un tal ciclo

Solución de sistemas de diferencias

Sistema de diferencias: Ejemplo

El siguiente es un sistema sobre las variables $\{x_1, x_2, x_3, x_4, x_5\}$:

$$\begin{aligned}
x_1 - x_2 &\leq 0 \\
x_1 - x_5 &\leq -1 \\
x_2 - x_5 &\leq 1 \\
x_3 - x_1 &\leq 5 \\
x_4 - x_1 &\leq 4 \\
x_4 - x_3 &\leq -1 \\
x_5 - x_3 &\leq -3 \\
x_5 - x_4 &\leq -3
\end{aligned}$$

Se quiere una solución que satisfaga las restricciones del sistema

Sistema de diferencias: Formulación

Un sistema de diferencias sobre las variables $X = \{x_1, x_2, \ldots, x_n\}$, es un conjunto de m desigualdades sobre diferencias del tipo $x_i - x_j \leq b$

El sistema puede expresarse de forma compacta como

$$Ax \leq b$$

utilizando **notación matricial** donde A es una matrix $m \times n$, x es un vector columna de forma $(x_1, x_2, \ldots, x_n)^\top$, y b es un vector columna

La matrix A tiene coeficientes en $\{0, 1, -1\}$ tal que cada fila de A contiene exactamente un 1 y un -1

Sistema de diferencias: Ejemplo

El ejemplo anterior puede expresarse de forma $Ax \leq b$:

$$\underbrace{\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}_{x} \leq \underbrace{\begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}}_{b}$$

$$\begin{aligned} x_1 - x_2 &\leq 0 \\ x_1 - x_5 &\leq -1 \\ x_2 - x_5 &\leq 1 \\ x_3 - x_1 &\leq 5 \\ x_4 - x_1 &\leq 4 \\ x_4 - x_3 &\leq -1 \\ x_5 - x_3 &\leq -3 \\ x_5 - x_4 &\leq -3 \end{aligned}$$

La matriz A transpuesta

La matriz A tiene coeficientes en $\{0, 1, -1\}$ donde cada file tiene exactamente un 1 y un -1

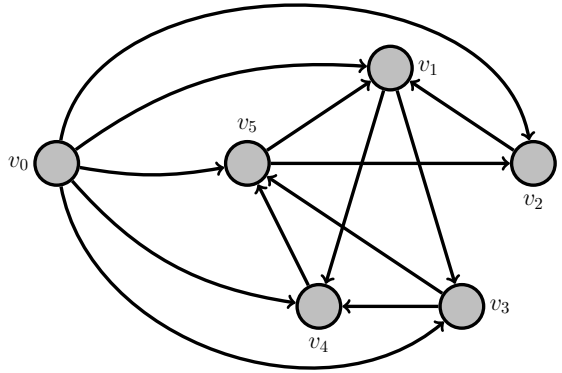
La **transpuesta** es una matriz donde cada columna tiene exactamente un 1 y un -1

$$\begin{pmatrix} 1 & 1 & 0 & -1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 \\ 0 & -1 & -1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

La transpuesta es la **matrix de incidencia** de un grafo $G = (V, E)$

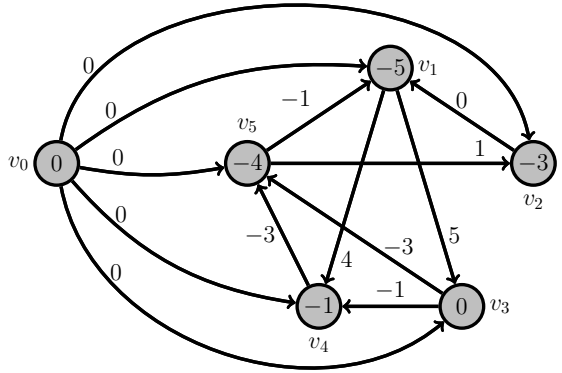
Matriz de incidencia $A^\top$

$$\begin{pmatrix} 1 & 1 & 0 & -1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 \\ 0 & -1 & -1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$



Algoritmo para sistemas de diferencias

1. Construir el grafo de restricciones $G = (V, E)$
2. Definir pesos $w : E \to \mathbb{R}$ usando vector b y $w(v_0, \cdot) = 0$
3. Correr Bellman-Ford para calcular distancias $\delta(v_0, \cdot)$
4. Solución $x_i = \delta(v_0, v_i)$: $x_1 = -5$, $x_2 = -3$, $x_3 = 0$, $x_4 = -1$, $x_5 = -4$



Algoritmo para sistemas de diferencias: Correctitud

Teorema

Sea $Ax \leq b$ un sistema de diferencias y $G = (V, E)$ el grafo de restricciones. Si G no contiene un ciclo de costo negativo, entonces $x = (\delta(v_0, v_1), \ldots, \delta(v_0, v_n))$ es una solución. Si G contiene un ciclo de costo negativo, entonces el sistema no tiene solución.

Prueba: Considere el primer caso que G no contiene ciclos de costo negativo. Entonces, $\delta(v_0, v_i)$ es finito para todo vértice v_i

Sea $x_i - x_j \leq b$ una desigualdad. Ella está asociada a la arista $v_j \to v_i$ con peso $w(v_j, v_i) = b$

Entonces:

$$x_i - x_j \;=\; \delta(v_0, v_i) - \delta(v_0, v_j) \;\leq\; w(v_j, w_i) = b$$

Por des. triangular $\delta(v_0, v_i) \leq \delta(v_0, v_j) + w(v_j, v_i)$ para la arista (v_j, v_i)

Algoritmo para sistemas de diferencias: Correctitud

Teorema

Sea $Ax \leq b$ un sistema de diferencias y $G = (V, E)$ el grafo de restricciones. Si G no contiene un ciclo de costo negativo, entonces $x = (\delta(v_0, v_1), \ldots, \delta(v_0, v_n))$ es una solución. Si G contiene un ciclo de costo negativo, entonces el sistema no tiene solución.

Prueba: Considere el segundo caso y sea $c = (v_1, \ldots, v_k)$, con $x_1 = x_k$, un ciclo de costo negativo. Suponga que $x = (x_1, \ldots, x_n)$ es una solución:

$$\begin{aligned} x_2 - x_1 &\leq w(x_1, x_2) \\ x_3 - x_2 &\leq w(x_2, x_3) \\ &\ldots \\ x_{k-1} - x_{k-2} &\leq w(x_{k-2}, x_{k-1}) \\ x_k - x_{k-1} &\leq w(x_{k-1}, x_k) \end{aligned}$$

Sumando, $x_k - x_1 \leq w(c) \implies 0 \leq w(c)$ imposible ya que $w(c) < 0$ □

Algoritmo para sistemas de diferencias: Mejora

Bellman-Ford resuelve un sistema de m diferencias sobre n variables en tiempo $O((n+1)(m+n)) = O(n^2 + nm)$ ya que el grafo de restricciones tiene $n+1$ vértices y $m+n$ aristas

Ejercicio:

– Explotar la estructura conocida del grafo $G = (V, E)$ para modificar Bellman-Ford y obtener un algoritmo que corra en tiempo $O(nm)$

□

Alineación de secuencias de ADN

Ejemplo: Alineación de secuencias

Considere las sequencias $\sigma_1 = \texttt{AGTGTTCCAG}$ y $\sigma_2 = \texttt{AATCGTTACAG}$

Una **alineación** de σ_1 con σ_2 es una correspondencia entre los símbolos de ambas secuencias y el símbolo especial '-'

Las siguientes son alineaciones:

```
Alineación 1:  A-GTGTTCCAG
               |   ||| |||
               AATCGTTACAG

Alineación 2:  -AGT-GTTCCAG
                | | ||| |||
               AA-TCGTTACAG

Alineación 3:  ---------AGTGTTCCAG
                        ||
               AATCGTTACAG--------
```


Ejemplo: Alineación de secuencias

El símbolo '-' representa una inserción o eliminación (relativo a σ_1) dependiendo si aparece en σ_1 o σ_2 respectivamente

Una alineación se representa con una palabra en el alfabeto $\{\texttt{i}, \texttt{e}, \texttt{s}\}$

```
Alineación 1:  A-GTGTTCCAG
               |   ||| |||
               AATCGTTACAG
               sisssssssss

Alineación 2:  -AGT-GTTCCAG
                | | ||| |||
               AA-TCGTTACAG
               isesisssssss

Alineación 3:  ---------AGTGTTCCAG
                        ||
               AATCGTTACAG--------
               iiiiiiiiisseeeeeeee
```


Ejemplo: Alineación de secuencias

```
Alineación 1:  A-GTGTTCCAG
               |   ||| |||
               AATCGTTACAG
               sisssssssss

Alineación 2:  -AGT-GTTCCAG
                | | ||| |||
               AA-TCGTTACAG
               isesisssssss

Alineación 3:  ---------AGTGTTCCAG
                        ||
               AATCGTTACAG--------
               iiiiiiiiisseeeeeeee
```

¿Cuál es la mejor alineación?

Respuesta: dependerá del **costo** asociado a las inserciones, eliminaciones y sustituciones

Costos de edición

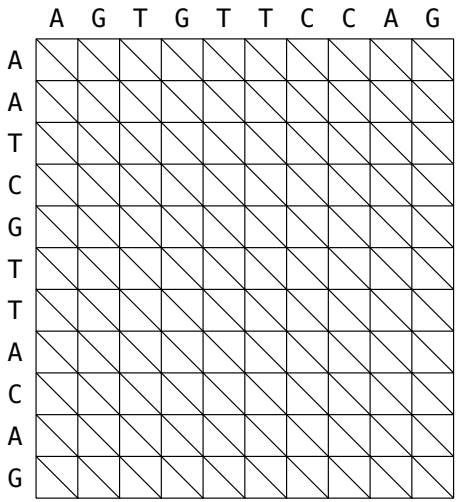
Utilizamos los siguientes costos o distancias de edición:

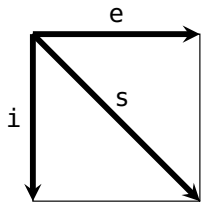
$$i(x) = \text{“costo de insertar símbolo } x \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}\text{”}$$

$$e(x) = \text{“costo de eliminar símbolo } x \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}\text{”}$$

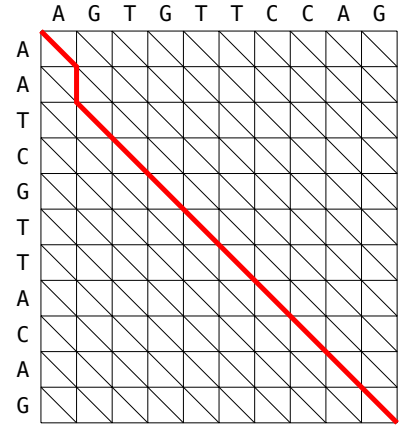
$$s(x, y) = \text{“costo de sustituir símbolo } x \text{ por } y\text{, con } x, y \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}\text{”}$$

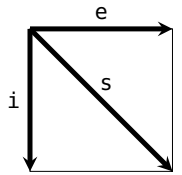
Ejemplo: Grafo de edición





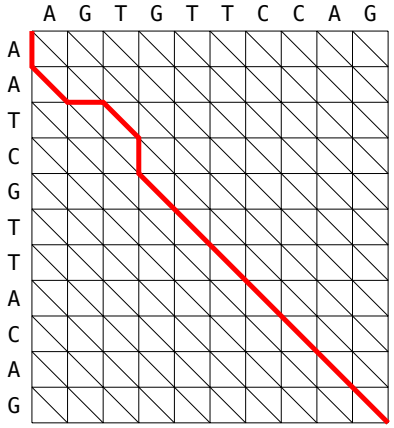
Ejemplo: Grafo de edición

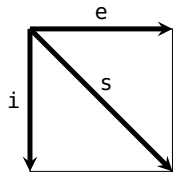





```
Alineación 1:  A-GTGTTCCAG
               |   ||| |||
               AATCGTTACAG
               sisssssssss
```

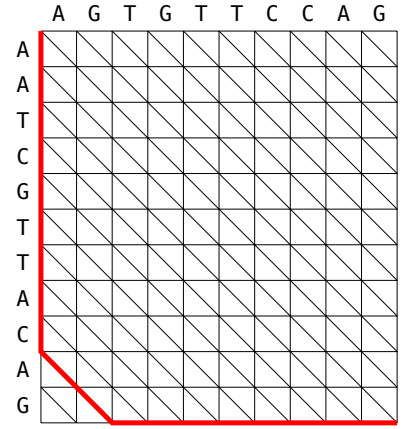

Ejemplo: Grafo de edición

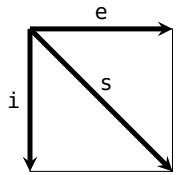





```
Alineación 2:  -AGT-GTTCCAG
                | | ||| |||
               AA-TCGTTACAG
               isesisssssss
```


Ejemplo: Grafo de edición






```
Alineación 3:  ---------AGTGTTCCAG
                        ||
               AATCGTTACAG--------
               iiiiiiiiisseeeeeeee
```

Grafo de edición

Dadas dos secuencias $\sigma_1, \sigma_2 \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}^*$ con $|\sigma_1| = n$ y $|\sigma_2| = m$

El grafo de edición $G(n, m) = (V, E)$ es un reticulado dirigido donde esquina superior izquierda es $(0, 0)$ y esquina inferior derecha es (n, m)

Para cada punto (x, y), existen las siguientes aristas:

$$((x, y), (x, y + 1)) \in E \qquad \text{si } y < m$$
$$((x, y), (x + 1, y)) \in E \qquad \text{si } x < n$$
$$((x, y), (x + 1, y + 1)) \in E \qquad \text{si } x < n \text{ y } y < m$$

Pesos $w(e)$ para aristas $e \in E$ dados por costos $i(x), e(x), s(x, y)$

Tenemos $|V| = (n + 1)(m + 1)$ y $|E| = 3(n + 1)(m + 1) - 2(n + m)$

Alineación óptima

La alineación óptima se define como la alineación de costo mínimo

La alineación óptima corresponde a un **camino de costo mínimo** desde $(0, 0)$ a (n, m) en el grafo $G(n, m)$ de edición

Si asumimos costos no-negativos, podemos utilizar algoritmo de Dijkstra para encontrar alineación óptima de σ_1 y σ_2 en tiempo

$$O(E + V \log V) = O(3nm - nm \log(nm)) = O(nm)$$

donde $|\sigma_1| = n$ y $|\sigma_2| = m$

Es decir, **tiempo cuadrático en el tamaño de la entrada**

Alineación de múltiples secuencias

Ejercicios:

- ¿Que debe hacerse para alinear 3 secuencias σ_1, σ_2 y σ_3 de forma óptima?
- ¿En cuánto tiempo puede hacerse esto?
- ¿En cuánto tiempo y cómo pueden alinearse de forma óptima n secuencias $\{\sigma_i\}_{i=1}^n$?

□

