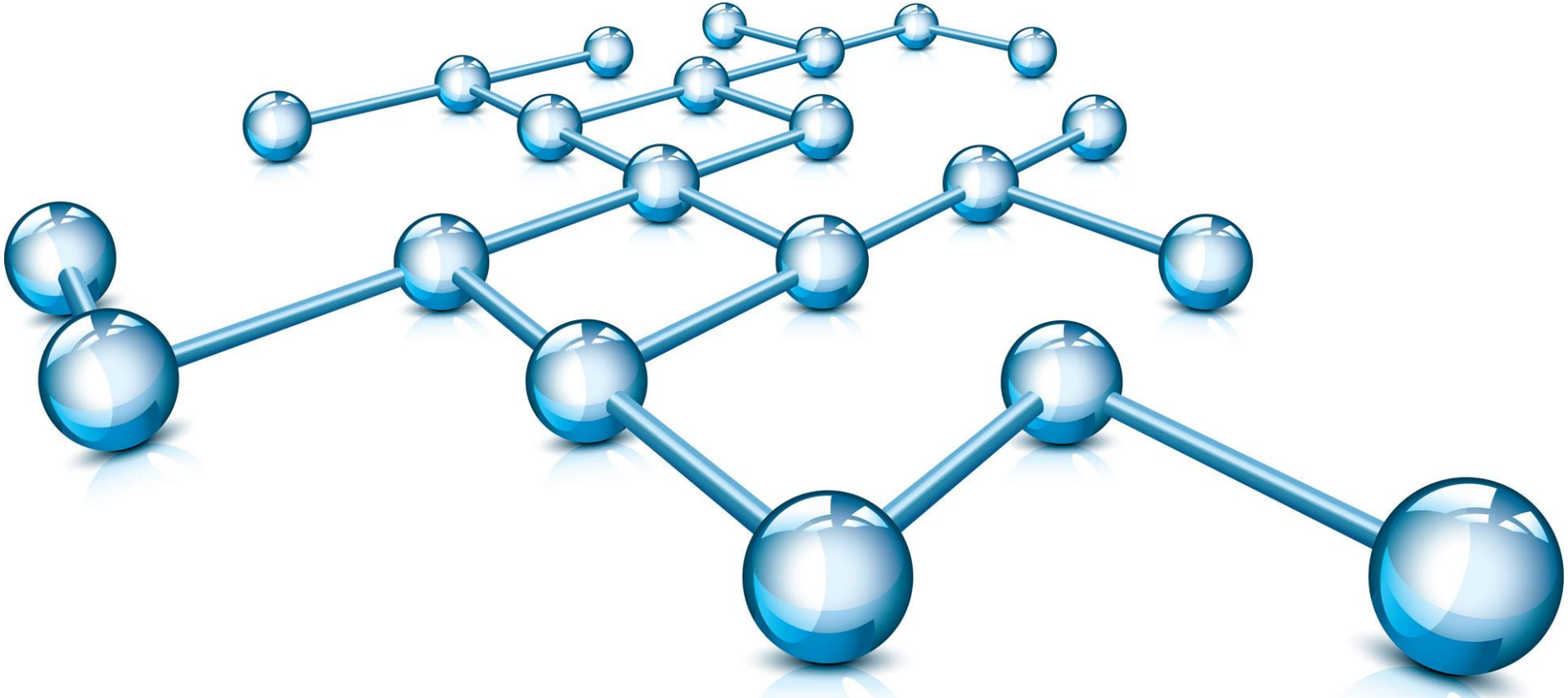


Modelos y algoritmos para logística y transporte

Agustín Pecorari

Rodrigo Maranzana

Localización

Criterios de clasificación en la localización.

- **Planar o continuo:** los sitios pueden estar en todas partes en el plano.
- **Discreto:** número finito de sitios, y conocemos la matriz "distancias" en el sentido amplio entre sitios o distancia en km, tiempo, coste, etc.
- **En una red:** ubicación discreta en los nodos de una red.

Localización

Puntuaciones de distancia-carga:

- **Datos**
  - *m* sitios posibles para una nueva instalación,
  - *n* entidades (clientes, proveedores, etc.) para servir,
  - medida simple de "carga" L_j para cada entidad j por definir: tonelaje entrante o saliente, número de viajes / semana, pacientes que acuden a consultar, etc.
  - distancias entre cualquier sitio i y cualquier entidad j.
- **Objetivo:** elegir un sitio i* minimizando una estimación de puntaje el trabajo de cargas móviles.

Localización

Método:

- cálculo para cualquier sitio *i* de una puntuación de *distancia-carga ld(i)*,
- luego la elección del sitio de puntuación mínima.

$$ld(i) = \sum_{j=1,n} L_j . d_{ij}$$

Ejemplo: Localización de un dispensario en una ciudad.

- Entidades: barrios.
- Cargas: número de habitantes (pacientes potenciales).
- Distancias: desde el centro de los sectores hasta los posibles sitios.

Localización

Centro de gravedad

Para localización continua (= no discreta).

- Sabemos las coordenadas x_j y y_j de la entidad *j*.
- Buscamos las coordenadas *(x*, y*)* de la instalación.
- Elección obvia: centro de gravedad de cargas.

El lugar obtenido a menudo debe ser ajustado. La localización planar de varias instalaciones es más difícil.

$$x^* = \frac{\sum_{j=1,n} L_j . x_j}{\sum_{j=1,n} L_j}, \quad y^* = \frac{\sum_{j=1,n} L_j . y_j}{\sum_{j=1,n} L_j}$$

Localización. Cobertura.

Problema de cobertura total

Datos:

- m puntos de demanda o "clientes" (indexados por i).
- n sitios potenciales (indexados por j).
- distancia de cobertura d_c (distancia máxima de servicio).
- booleanos $a_{ij} = 1$ ssi cubiertos por el sitio j (distancia ≤ d_c).
- c_j el costo de abrir el sitio j.

Objetivo: determinar los sitios a abrir para cubrir todas las solicitudes, con un coste total mínimo.

Localización. Cobertura.

- NP-difícil. Formulación por un PL en 0-1:

$$(1)\ \text{Min} \sum_{j=1,n} c_j x_j$$

$$(2)\ \forall i = 1 \ldots m : \sum_{j=1,n} a_{ij} x_j \geq 1$$

$$(3)\ \forall j = 1 \ldots n : x_j \in \{0,1\}$$

Localización. Cobertura.

Resolviendo problemas de cobertura

- Con software de programación lineal.
  - PL en 0-1 → NP-difícil.
  - Algoritmo simplex inutilizable.
  - métodos de árbol ($x_j = 0$ o $x_j = 1$) + reducciones.
  - Ejemplo de problema manejable: 100 clientes, 20 sitios.

Nota: Los problemas de partición (A.x = 1) son aún más difíciles porque no siempre son factibles.

Ejemplo: división electoral, sectores comerciales.

Localización. Cobertura.

PL demasiado grande para resolver grandes problemas. Las heurísticas son entonces necesarias.

- Heurística glotona (la más simple):
  - Decisiones finales (sin retroceder).
  - La elección más ventajosa en cada etapa según un criterio determinado (por definir)
  - Ejemplo, **Vecino Más Cercano** para el TSP.

Localización. Cobertura.

Heurística golosa de Chvàtal (cobertura total)

Abra el sitio de menor costo por cada nuevo cliente cubierto.

```
Costo: = 0  // costo total de sitios abiertos
Abrir: = ∅  // todos los sitios abiertos
repeat
   j := sitio de costo promedio mínimo por nuevo punto
   cubierto: c(j) / nb nuevos puntos cubiertos.
   Agrega j a Abrir
   Costo: = Costo + c(j)
   Eliminar el sitio j y nuevos puntos cubiertos del
   problema
until (cobertura completa).
```

Localización. Cobertura.

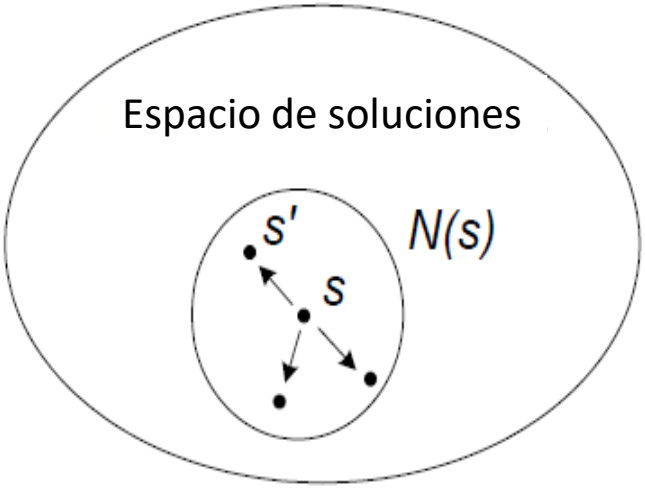
Búsquedas locales

Principios:

- Heurística para la optimización combinatoria.
- Mejora gradual de una solución inicial S, por ejemplo, dado por una heurística golosa.
- Basado en un subconjunto "pequeño" de soluciones V(S), llamado vecindad de S.
- V(S) se define implícitamente por una transformación simple de S llamado movimiento.

Localización. Cobertura.

```
calcular una solución inicial
repeat
   buscar una mejor solución S' en N(S)
   si S' encontrada entonces S := S'
until que no se encuentra S'
```

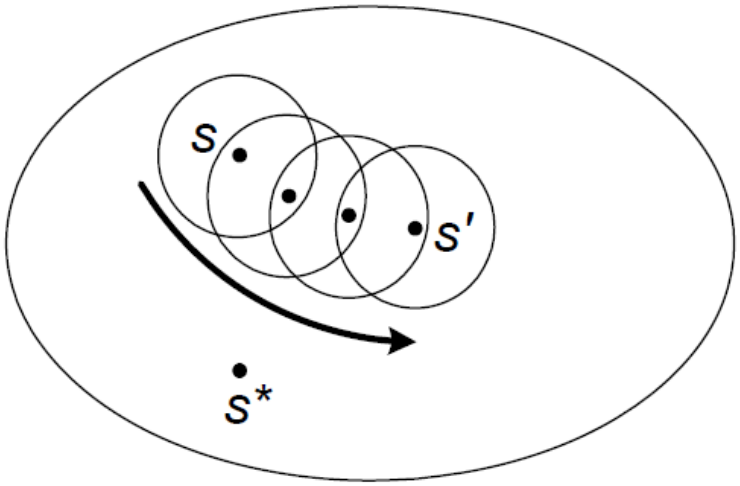


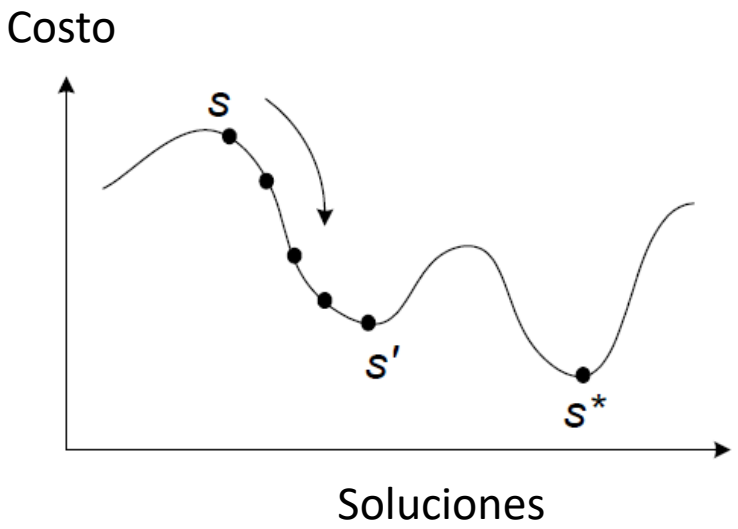
<u>Dos versiones:</u> En cada iteración, toma

- Ya sea la primera solución S' encontrada que mejora S.
- La mejor solución S' en la vecindad.

Al final, S **óptimo local** para el tipo de vecindad elegido.

Localización. Cobertura.

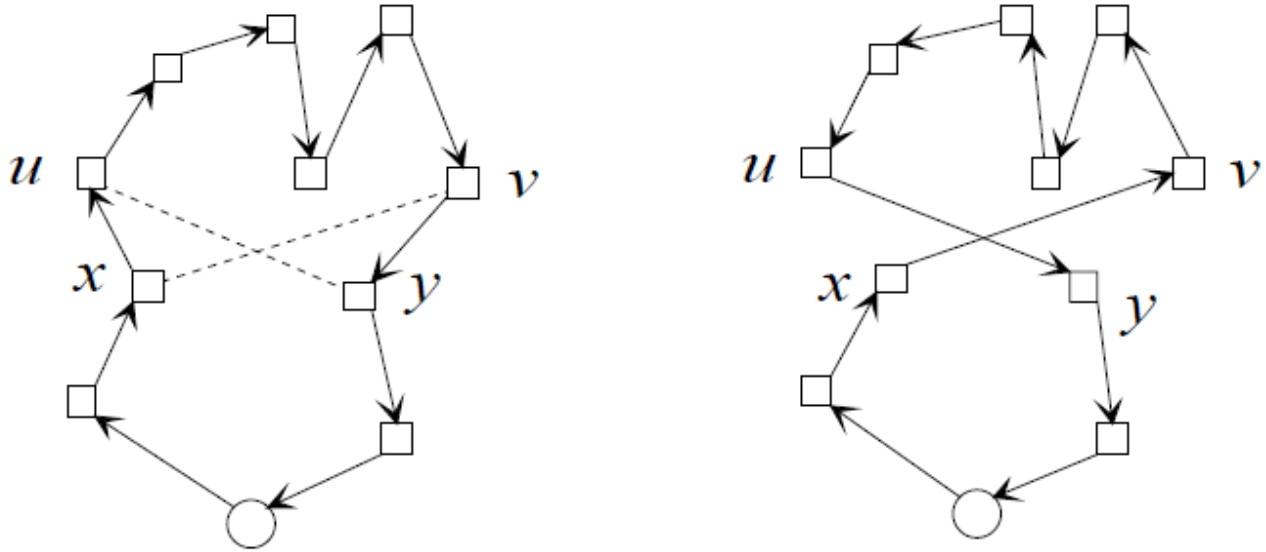




- solución inicial de S'.
- final de S'.
- S * óptimo.

Localización. Cobertura.

Búsqueda local 2-OPT para el TSP: V(S) es el conjunto de soluciones de $O(n^2)$ obtenidas cruzando 2 arcos de S.



P-centros y p-medianas

Si todos los clientes son considerados accesibles (sin distancia de cobertura) pero las distancias son tenidas en cuenta.

Problema de los p-centros (NP-difícil)

Frecuente en logística de emergencias. Datos:

- *m* clientes para servir (índice *i*), sin distancia de cobertura.
- *n* sitios (índice *j*) pero podemos abrir p sitios al máximo.
- Distancias d_{ij} en el sentido amplio entre nodos y sitios.

Objetivo:

- Coloque sitios minimizando la distancia máxima a los clientes.

Ejemplo: colocar 4 estaciones de bomberos en una ciudad para

- Minimizar el tiempo de intervención en un incendio.

P-centros y p-medianas

(1) $\text{Min } z$

(2) $\forall i = 1 \ldots m : \sum_{j=1,n} y_{ij} = 1$

(3) $\sum_{j=1,n} x_j = p$

(4) $\forall i = 1 \ldots m, \forall j = 1 \ldots n : y_{ij} \leq x_j$

(5) $\forall i = 1 \ldots m : z \geq \sum_{j=1,n} d_{ij} . y_{ij}$

(6) $\forall j = 1 \ldots n : x_j \in \{0,1\}$

(7) $\forall i = 1 \ldots m, \forall j = 1 \ldots n : y_{ij} \in \{0,1\}$

(8) $z \geq 0$

Variables enteras y reales.

Variables de decisión:

- x_j = 1 si el sitio elegido,
- y_{ij} = 1 si estoy asignado al sitio j.
- Variable real z: distancia máxima de intervención.

P-centros y p-medianas

¡PL más complicado que los anteriores!

Fácil: (1) objetivo de minimizar, (2) cada nodo se asigna a un sitio, (3) p sitios abiertos, (6) (7) (8) definición de variables.

Cada restricción (4) refleja la implicación:

si el sitio j está cerrado, no se le puede asignar ningún cliente.

Restricciones (5) o minimizar la distancia máxima de intervención:

- cálculo del máximo de d_{ij} tal que $y_{ij} = 1$.
- Linealizamos limitando las distancias de intervención por z.
- Este límite z será comprimido por minimización.
- En el óptimo, z será igual a la distancia máxima de intervención.

P-centros y p-medianas

Preguntas:

- ¿Podemos poner ≤ o ≥ en restricción (3)?
- para cada restricción (5), ¿por qué hacer una suma en j en lugar de escribir una restricción $z \geq d_{ij}.y_{ij}$ para cada sitio j?
- ¿Podemos hacer tal suma en j en las restricciones (4)?

P-centros y p-medianas

Problema de las p-medianas (NP-difícil)

- El problema de los p-centros favorece la distancia máxima, puede dar una distancia promedio excesiva debido a algunos nodos distantes pero no importantes.
- El problema de las p-medianas busca minimizar la *distancia de intervención* media. Estas distancias pueden ser ponderadas por la demanda q_i de los nodos. Ejemplo: colocar depósitos para minimizar el tiempo promedio para entregar un cliente.

P-centros y p-medianas

$$(1)\quad Min \sum_{i=1,m} \sum_{j=1,n} q_i . d_{ij} . y_{ij}$$

$$(2)\quad \forall i = 1 \ldots m : \sum_{j=1,n} y_{ij} = 1$$

$$(3)\quad \sum_{j=1,n} x_j = p$$

$$(4)\quad \forall i = 1 \ldots m, \forall j = 1 \ldots n : y_{ij} \leq x_j$$

$$(5)\quad \forall j = 1 \ldots n : x_j \in \{0,1\}$$

$$(6)\quad \forall i = 1 \ldots m, \forall j = 1 \ldots n : y_{ij} \in \{0,1\}$$

PL en 0-1 más simple:

- iguales variables binarias
- (1) distancia promedio
- (2) clientes asignados a un sitio
- (3) número de sitios para abrir
- (4) no hay clientes para un sitio cerrado.

P-centros y p-medianas

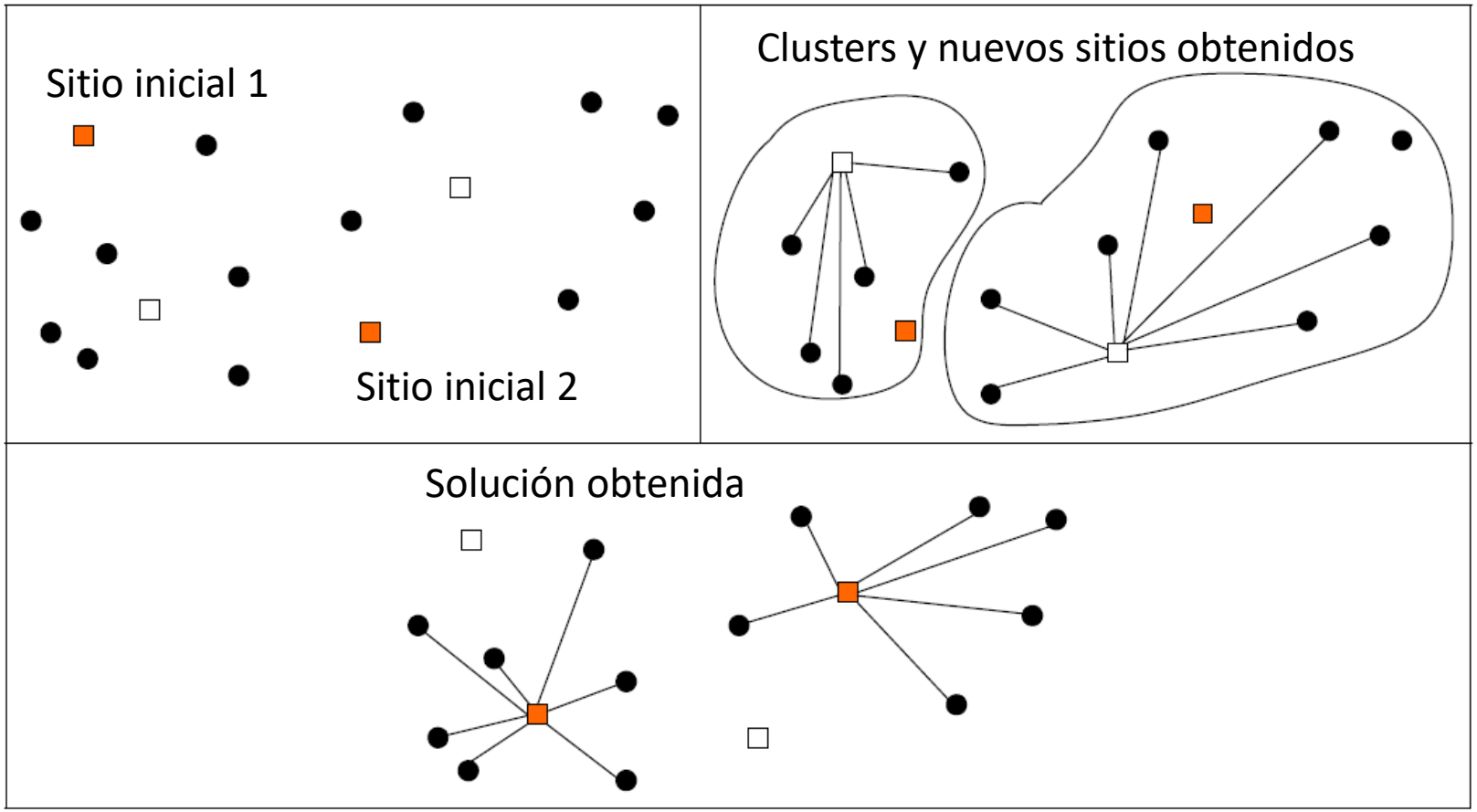
Una heurística muy efectiva que también sirve en el caso continuo:

```
Abrir sitios p al azar o con una heurística
repeat
   asignar a cada cliente al sitio abierto más cercano
   obtenemos clusters (un sitio + sus clientes)
   calcular el mejor sitio en cada cluster.
until todos los sitios no cambien.
```

El cálculo del mejor sitio en cada grupo se realiza:

- Con el método de puntajes, si es finita la lista de sitios.
- mediante el cálculo de un centro de gravedad, si la localización es continua.

P-centros y p-medianas



Localización de depósitos

- Determinar qué almacenes abrir.
- Asignar clientes a almacenes.
- Un cliente puede ser abastecido por varios almacenes (≠ asignar)
- Minimizar costes de almacenes + costes de transporte.

Caso con capacidades de almacén "infinitas" y producto " único":

- m sitios (índice i) y n clientes (índice j).
- f_i costo fijo de apertura del sitio i.
- d_j demanda cliente j, c_{ij} costo unitario de transporte de i a j.
- y_i variable binaria de apertura.
- x_{ij} cantidad entregada por el sitio i al cliente j.

Localización de depósitos

$$(1)\quad \text{Min} \sum_{i=1,m} f_i . y_i + \sum_{i=1,m} \sum_{j=1,n} c_{ij} . x_{ij}$$

$$(2)\quad \forall j = 1 \ldots n : \sum_{i=1,m} x_{ij} = d_j$$

$$(3)\quad \forall i = 1 \ldots m : \sum_{j=1,n} x_{ij} \leq M \cdot y_i$$

$$(4)\quad \forall i = 1 \ldots m : y_i \in \{0,1\}$$

$$(5)\quad \forall i = 1 \ldots m, \forall j = 1 \ldots n : x_{ij} \geq 0$$

PLEM

- (1) costo total, para ser minimizado.
- (2) demandas.
- (3) M constante grande> 0.
- Un depósito cerrado no entrega nada.

Localización de depósitos

Problema estratégico (largo plazo) y macroscópico (agregado):

- Un "cliente" puede ser una ciudad con una demanda anual agregadas y estimadas.
- Combinación de costos de apertura y costos de transporte por única vez. todos los días? De hecho, puede ser un costo operativo diario.
- ¿Sitios de capacidad infinita? Los sitios pueden ser tierra gratis: construiremos sitios abiertos según cantidades.
- Entregado en la solución. Capacidad límite de los sitios.
- ¿Un producto único? De hecho, nos referimos a que los productos son indistinguibles para el transporte (paletas, contenedores).

Localización de depósitos

- Gracias a las infinitas capacidades, cada cliente siempre puede ser asignado a un sitio (el menos costoso).
- Podemos extender el modelo a capacidades limitadas: algunos clientes deben ser enviados desde múltiples sitios.
- También podemos generalizar a varios productos.

Ruteo. Camino más corto.

Transporte de cargas, entre fuentes (con disponibilidades conocidas) y destinos con demandas dadas.

Los arcos de la red tienen costos y posiblemente capacidades (máximo rendimiento).

- problemas de la ruta óptima.
- problemas de distribución sin capacidad.
  - "problema de transporte" (red de 2 capas).
  - "problema de asignación" (caso con cantidades = 1).
  - "pb de transbordo" (cualquier red).

Ruteo. Camino más corto.

Problemas de distribución con capacidades.

- pb del flujo máximo.
- pb del flujo de costo mínimo.
- pb de multi-commodity.

A diferencia de los problemas de ubicación, estos problemas (excepto el último) tienen algoritmos rápidos (polinomiales).

También se pueden resolver mediante programación lineal.

Ruteo

El problema

- Grafo orientado G = (X, U, C).
- X conjunto de n nodos, U conjunto de m arcos.
- C_{ij} es el "peso" o "costo" del arco (i, j).

El "costo" puede ser un costo real, una distancia, una duración, etcétera. Si G no es orientado, reemplazamos cada borde [i, j] con dos arcos (i, j) y (j, i) del mismo costo. El coste de un camino, se suele definir como el coste total de sus arcos.

Ruteo

Queremos calcular una ruta de costo mínimo (o ruta más corta o RMC) entre 2 nodos dados de G.

El problema tiene sentido si G no contiene un circuito de costo negativo (llamado circuito absorbente).

Entonces podemos limitarnos a los caminos elementales (sin nodos repetidos), con un máximo de n - 1 arcos. Esta propiedad es explotada por todos los algoritmos.

Ruteo

Algoritmos de ruta óptima

Considere el caso donde el costo de un camino es la suma de los costos del arco. Hay algoritmos para 3 tipos de problemas:

- **Pb A:** encuentre un RMC entre dos nodos s y t.
- **Pb B:** encontrar un RMC de un nodo a todos los otros nodos, por lo que un árbol de n - 1 caminos.
- **Pb C:** encuentre un RMC entre cualquier par de nodos, bajo forma de una matriz D n x n llamada distancia.

Ruteo

Un algoritmo para un problema se puede utilizar para resolver los otros dos, a veces utilizándolo varias veces.

Dijkstra y Bellman, para el pb B. Ellos calculan para cada nodo *x* una etiqueta *V(x)*, costo de RMC entre el nodo inicial *s* y el nodo *x*.

Algorito de Dijkstra es del tipo de etiquetado: en cada iteración, calcula el valor definitivo de una etiqueta *V(x)*.

Bellman's es un algoritmo de corrección de etiquetas: la etiqueta puede cambiar hasta la última iteración de cada nodo.

Ruteo. Dijkstra.

Algoritmo de etiqueta Dijkstra

a) Principio

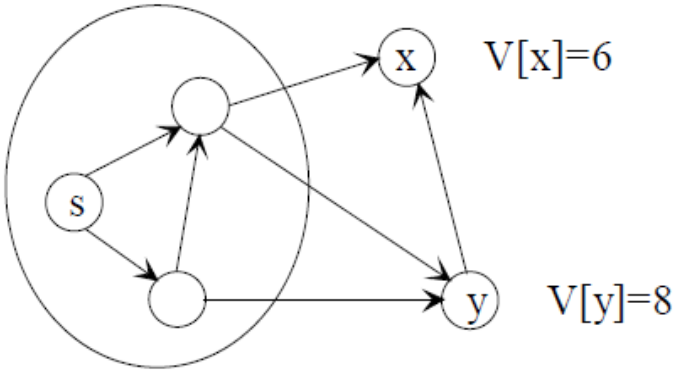
Atención: el algoritmo supone costes no negativos.

Al principio, el nodo de inicio tiene una etiqueta nula, los otros una etiqueta $+\infty$. El conjunto F de los nodos fijos (de etiqueta definitivo) está vacío.

En cada iteración, buscamos el nodo no fijado *x* de etiqueta mínima y se lo fija.

Ruteo. Dijkstra.

De hecho, *V(x)* ya no puede disminuir si los costos son no negativo. Ejemplo:



F (nodos fijos)

El nodo *x*, de etiqueta mínima, puede ir a *F*. De hecho, si *V(x)* no fuera definitivo, habría un camino negativo de *y* a *x*, contradicción.

Ruteo. Dijkstra.

Luego, para cualquier sucesor *y* de *x*, actualizamos *V(y)* si el camino a través de *x* mejora *V(y)*, es decir, si

$$V(x) + C(x, y) < V(y).$$

Luego notamos que llegamos desde *x*, gracias a un tabla *P* de los predecesores: P(y): = x.

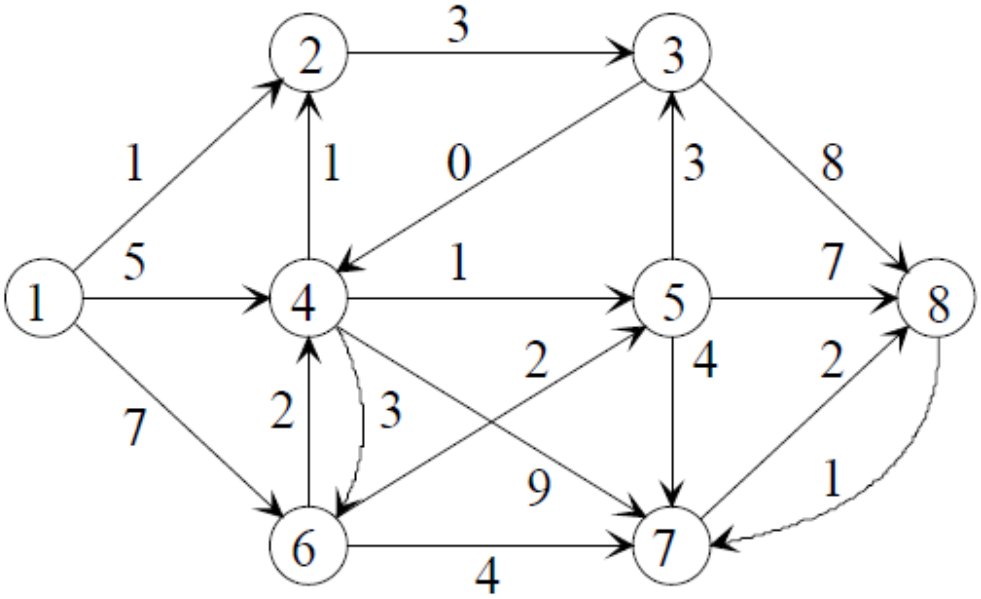
Si queremos un RMC de *s* a un solo nodo *t*, paramos cuando *t* es fijo. De lo contrario, debemos hacer *n* iteraciones para fijar todos los nodos.

Al final, *V* y *P* definen un RMC de *s* hacia cualquier otro nodo.

Ruteo. Dijkstra.

Ejemplo de ejecución (salida *s* = 1)

La tabla nos da las etiquetas al comienzo de las iteraciones y al fin del algoritmo. El nodo fijado a cada iteración tiene un etiqueta enmarcada. Los guiones recuerdan a los nodos ya fijados.



Ruteo. Dijkstra.

V[1]	V[2]	V[3]	V[4]	V[5]	V[6]	V[7]	V[8]	Nodo fijo
[0]	∞	∞	∞	∞	∞	∞	∞	1
-0-	[1]	∞	5	∞	7	∞	∞	2
-0-	-1-	[4]	5	∞	7	∞	∞	3
-0-	-1-	-4-	[4]	∞	7	∞	12	4
-0-	-1-	-4-	-4-	[5]	7	13	12	5
-0-	-1-	-4-	-4-	-5-	[7]	9	12	6
-0-	-1-	-4-	-4-	-5-	-7-	[9]	12	7
-0-	-1-	-4-	-4-	-5-	-7-	-9-	[11]	8
-0-	-1-	-4-	-4-	-5-	-7-	-9-	-11-	FIN

Ruteo. Dijkstra.

Notas:

- Una etiqueta puede disminuir varias veces.
- Para pequeños grafos, en lugar de la tabla, se pueden calcular las etiquetas directamente en el dibujo.
- Obtenemos el camino al ver de dónde viene la etiqueta. Nodo 8: V(8) = 11 = V(7) + 2: venimos del nodo 7.
- Computacionalmente, con la tabla P: P[8] = 7.
- A veces hay varios caminos óptimos.

Ruteo. Dijkstra.

```
V := +∞, V[s]:=P :=0, F=∅    // inicialización
for k := 1 a n do          // bucle principal (k es un contador)
   Vmin:= +∞               // buscar x no conjunto de etiquetas min
   for y:= 1 a n con y no en F and V[y]<Vmin do
      x:= y
      Vmin:= V[y]
   endfor
   agrega x a F                        // establece x
   for each sucesor de x hacer         // actualizar sucesores allí
     if Vmin+C[x, y]<V[y] entonces //si etiqueta de y es mejorable
         V[y]:= Vmin + C[x, y]
         P[y]:= x                      //memoriza de dónde venimos
     endif
   endfor
endfor
```

Ruteo

Para calcular una ruta entre dos nodos *s* y *t*, reemplazar el bucle principal por: repita ... hasta que *t* pertenezca a *F*.

Para tener todas las distancias *D*, n x n, llamar al algoritmo para *s* variando de 1 a *n* y copie *V* en la línea *s* de *D*.

Análisis del algoritmo.

Para un algoritmo de optimización combinatoria, es:

- probar la convergencia (¿el algo se detiene?),
- probar optimalidad (¿es el algo óptimo?),
- evaluar la complejidad (¿es eficaz?).

Ruteo

Nuestro algo converge: fija un nodo por iteración.

Es óptimo: las siguientes propiedades son verdaderas para cada iteración (prueba por recurrencia):

- Si un nodo *z* es fijo (*z* pertenece a *F*), entonces *V(z)* es óptimo.
- Si *z* no es fijo, *V(z)* es el costo del RMC de *s* a *z*, de forma tal que todos los nodos están fijos excepto el último (*z*).

Ruteo

Complejidad. El 1er **for** tiene *n* iteraciones. Contiene un **for** de *n* iteraciones y otro de como máximo *n* iteraciones (porque *x* tiene a lo sumo *n* sucesores): el algoritmo es de $O(n^2)$.

Hay una versión en $O(m.\log_2 n)$, basada en estructuras de datos complejas:

- Si G está completo ($m = n^2$), esta versión es más lenta.
- Si G es del tipo de rutero ($m \approx 4n$), es más rápido: unos pocos segundos para un ejemplo de 1000x1000.

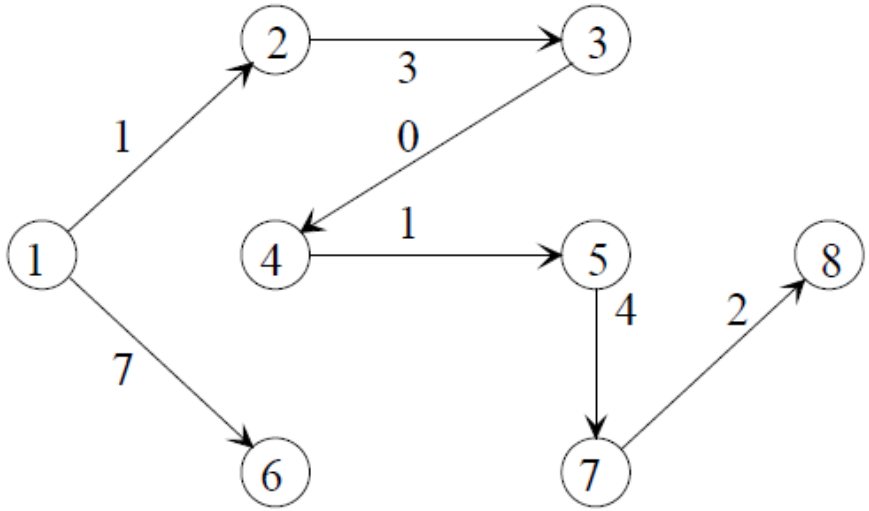
Ejercicio: compara las 2 complejidades para n = 1000.

Ruteo

Almacenamiento y recuperación de caminos.

P[x]: predecesor de x en la ruta óptima de s a x.

La tabla P describe (a la inversa) la estructura de árbol de los RMC:



Ruteo

Para recuperar el camino desde s hasta x (al revés) se remonta hacia s con un algoritmo muy simple:

```
// camino de s a x (al revés)
repeat
    escribe x
    x := P[x]
until x = 0
```

Ruteo

Algoritmo de Bellman (corrección de etiquetas)

Principio

Acepta costos <0. Método de programación dinámica, definida por relaciones de recurrencia:

$$\begin{cases} V_0(s) = 0 \\ \forall y \neq s : V_0(y) = +\infty \\ \forall k > 0, \forall y : V_k(y) = \underset{x \text{ préd. de } y}{Min} \{V_{k-1}(y), V_{k-1}(x) + C(x,y)\} \end{cases}$$

Ruteo

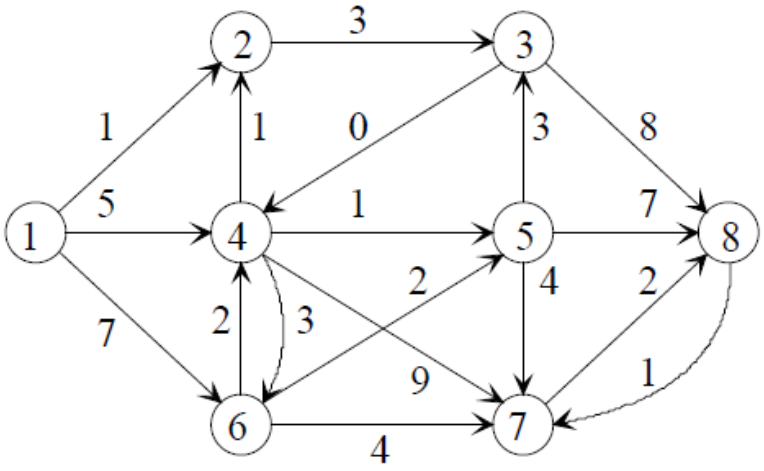
Principio:

- cálculo de la RMC de como máximo k arcos, k = 1,2,3 ...
- V_0 matriz inicial de etiquetas, etiquetas V_k en el paso k.
- 3ª relación: una RMC de k arcos de s a y extiende un RMC de k-1 arcos a un predecesor x de y.
- En la práctica, calculamos las tablas sucesivas V_k con la tercera relación.
- Paramos cuando las etiquetas ya no cambian.

Nota: para un grafo sin circuito (gestión de proyectos), alcanza con aplicar la 3ª relación nivel por nivel.

Ruteo

Ejecución en el gráfico de muestra (s = 1)



k	Labels dans V_k							
	1	2	3	4	5	6	7	8
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	1	$+\infty$	5	$+\infty$	7	$+\infty$	$+\infty$
2	0	1	4	5	6	7	11	$+\infty$
3	0	1	4	4	6	7	10	12
4	0	1	4	4	5	7	10	12
5	0	1	4	4	5	7	9	12
6	0	1	4	4	5	7	9	11
7	0	1	4	4	5	7	9	11

Ruteo

```
inicialice V := +∞, V[s] y P a 0 // inicialización
k := 0
repeat                          // bucle principal
   k := k + 1                   // calcula W(V_k) a partir de V(V_k-1)
   inicialice W a +∞ y q a 0 // número de etiquetas modificadas
   for y := 1 a n do            // calcula W(y) = V_k(y)
      for cada x predecesor de y
         if V[x] + C(x, y) < W[y] then
            W[y] := V[x] + C(x, y)
            P[y] := x; q:= q + 1
         endif
      endfor
   endfor
   V := W    // W se convierte en V para la siguiente iteración
until q = 0. // detener si no se modifica ninguna etiqueta
```

Ruteo

Análisis del algoritmo.
Convergencia. El algoritmo no se detiene si hay un circuito absorbente.
De lo contrario, las etiquetas son estables a más tardar para k = n-1 (arcos máximos de una RMC elemental). El algoritmo hace una iteración de más para verificar que q = 0.
Optimalidad. Por inducción, demostramos que V define la RMC de no más de k arcos al final de la iteración k.
Complejidad. Ambos **for** traducen la tercera relación y consultan todos los arcos (x, y) de G: una iteración cuesta O (m). Como hay como máximo n iteraciones, el algo está en O (mn), es decir O (n^3) si G está completo.

Ruteo

Para comparar con $O(n^2)$ para Dijkstra.

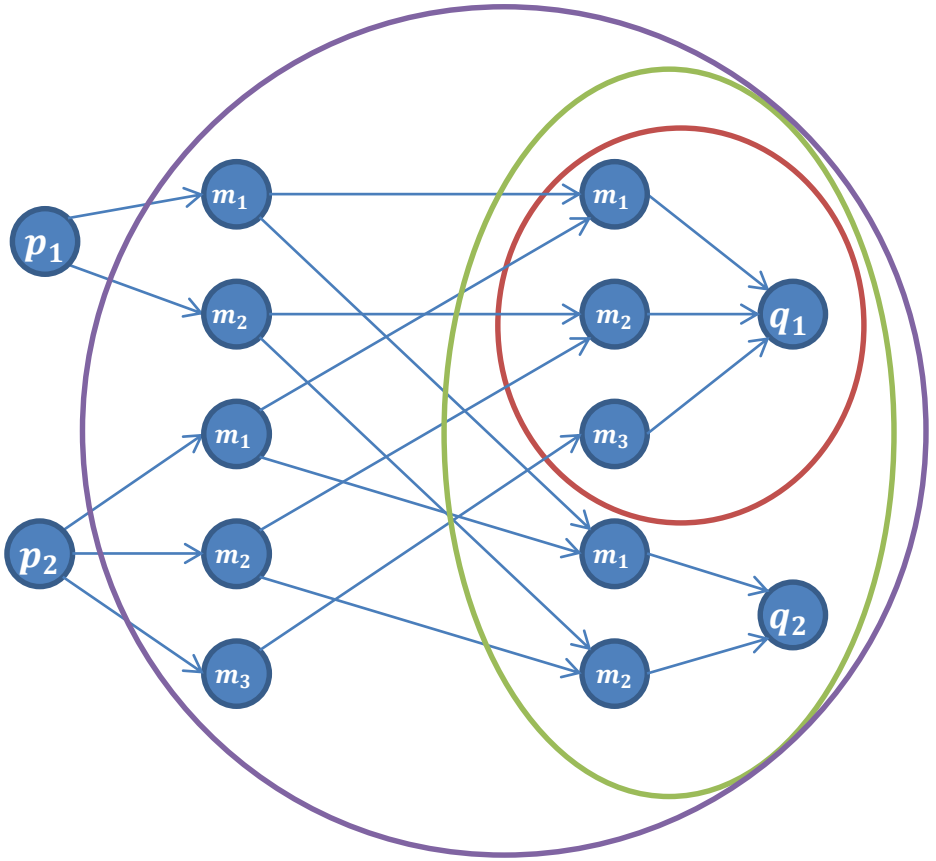
En la práctica, el algoritmo es rápido: por ejemplo $m \approx 4n$ para un grafo rutero. Las etiquetas a menudo se estabilizan en
$k < n-1$ iteraciones ($k = 1$ si es una red en estrella!).

Para detectar un posible circuito de absorción, es necesario reemplace el **until** por **until** $(q = 0)$ o $(k = n)$. Al final, si
$k = n$ pero $q \neq 0$ es que las etiquetas no son estables: G contiene un circuito absorbente.

Si uno quiere calcular RMC de a lo más e pasos (arcos e), alcanza con detenerse cuando $k = e$.

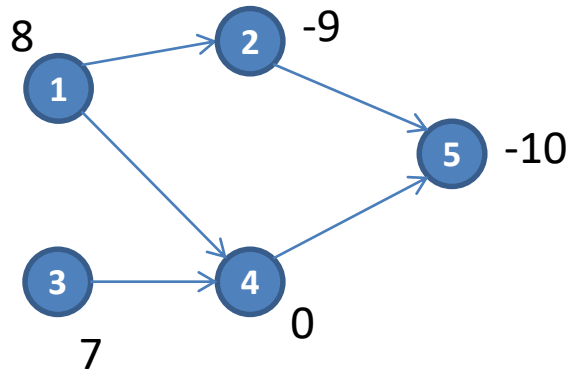
Problemas de clausura óptima

- Una clausura de un grafo $G(N, A)$ está dada por un subconjunto $N_1 \subset N$ tal que no existen arcos que salgan de N_1.



Clausuras

- Ejemplo: Hallar todas las clausuras de

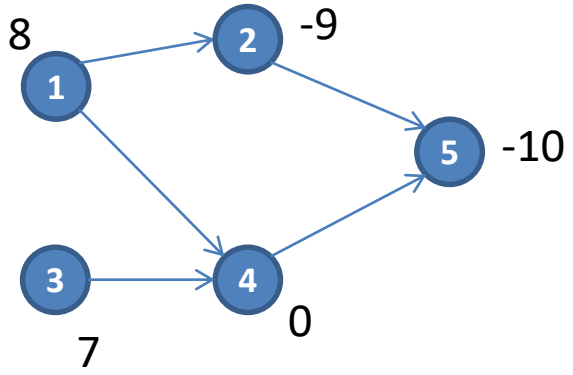


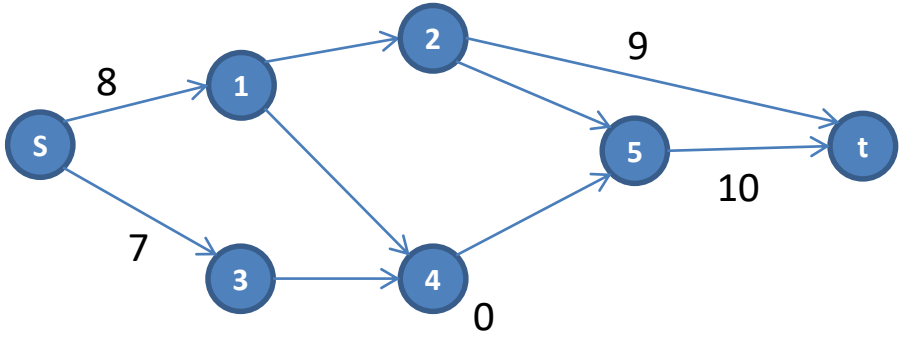
- Si los nodos tienen pesos asignado, se puede hablar de una ***clausura óptima***.

Clausuras óptimas

Teorema: Una clausura óptima (max) en G es equivalente a un problema de Flujo Máximo de s-t en G'.

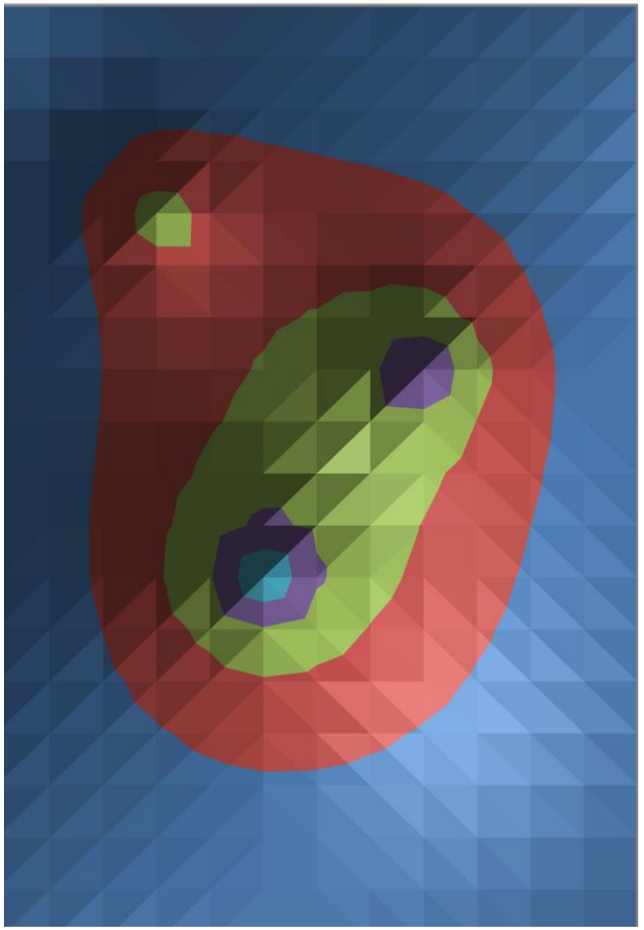
G' se obtiene agregando un nodo s y un nodo t, uniendo con s los nodos de peso positivo y con t los nodos de peso negativo. El peso asignado a los arcos es el valor absoluto del peso de los nodos.





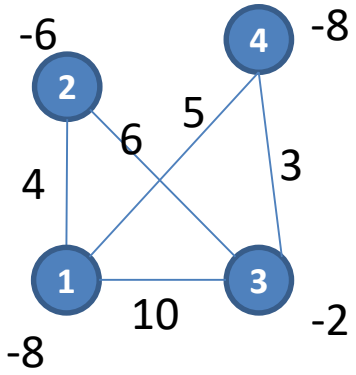
Ejemplo del pozo Minero

- Para operar un pozo minero se debe decidir qué bloques excavar conocidos los materiales en profundidad.
- La excavación tiene un costo que es contrarrestado por ganancia si se extrae minerales útiles.
- El problema consiste en dado el costo de extracción de los bloques y el valor de ganancia de cada uno decidir qué bloques extraer teniendo en cuenta un ángulo de 45 grados para la extracción.



Ejemplos de Clausura óptima

- Una Cía logística, está considerando la instalación de centros de operaciones. Tiene que elegir entre un conjunto S de ubicaciones posibles donde cada una tiene un costo de instalación y además cada par de ubicaciones tiene un beneficio extra al atraer parte del mercado.
- Ejemplo:



Ejemplo del kit de reparación

- Una Cía distribuida geográficamente tiene un equipo de reparación que se desplaza a cada sitio para eventuales reparaciones.
- Dicho equipo lleva un kit de reparación constituido por un conjunto de partes de reemplazo.
- Dichas partes tienen un costo de transporte y stock, pero su ausencia en el kit implica un costo por no poder arreglar un desperfecto que la necesite.
- Para calcular el kit óptimo se deben ponderar dichos costos y se puede formular un PCM.

Ejemplo del kit de reparación

- Si consideramos r partes en el kit, un kit posible es un subconjunto $M \subset \{1, 2, 3, \ldots, r\}$ y tiene asociado un costo de kit dado por

$$\sum_{i \in M} h_i$$

- Donde h_i es el costo de la pieza i.
- Por otro lado la lista de posibles desperfectos es $J_1, J_2, \ldots, J_k$ sabiendo que piezas requiera cada uno y además su costo promedio anual.
- Cómo calcular el kit óptimo ?