Entrega numero 1: Analisis de algoritmos

Pablo Robayo Rodriguez, Orlando Abaunza

**Abstract**—En este documento se presenta la investigación y análisis respectivo de tres problemas de diferente índole asociados a algoritmos computacionales y su respectivo rendimiento y desempeño. Los problemas analizados, son de tres tipos diferente: Programacion lineal (LP), minimo árbol de expasion (MST), y cliques en grafos. Para cada uno podrá encontrar su respectiva descripción, análisis, y un breve acercamiento a los tipos de soluciones para dicha clase de problemas.

**Index Terms**—Analisis, algoritmos, computacion, rendimiento, desempeño, problema, LP, MST, cliques, solucion.

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

————————————————

—————————— ◆ ——————————

# 1 Introduccion

Este documento tiene como propósito documentar y analizar problemas de diferentes tipos que requieren del diseño de un algoritmo computacional único como solución.

El propósito de la investigación realizada y presente en este documento, es poder entender de fondo cada uno de los problemas planteados para el posterior análisis y desarrollo de algún algoritmo matemático óptimo en cuanto desempeño y rendimiento computacional.

Para cada uno de los problemas, encontrara la explicación correspondiente, la definición del tipo problema a resolver, e instacias que permitan entender como se solucionan dichos problemas de manera general.

# 2 Definicion de los problemas

## 2.1 Programacion lineal (LP)

La programación lineal, también conocida como optimización lineal, es un método que tiene como objetivo encontrar el mejor resultado (Maximo o minimo) en un modelo matemático cuyas restricciones están representadas por relaciones lineales. [1]

Un modelo de programación lineal (LP) considera que las variables de decisión tienen un comportamiento lineal, tanto en la función objetivo como en las restricciones del problema. En este sentido la programación lineal es una de las herramientas mas utilizadas en la investigación operativa debido a que por su naturaleza se facilitan los cálculos y representan una buena aproximación de la realidad. [2]

## 2.1.1 Tipos de modelos (LP)

Los modelos se dividen básicamente en modelos deterministas (MD) o modelos estocásticos (ME). Los modelos deterministas son aquellos en los que se considera que las variables asociadas al problema son conocidos en su totalidad. En cuanto a los modelos estocásticos son aquellos en donde los parámetros del modelo tienen una distribucion de probabilidad asociada.

## 2.1.2 Ejemplos problemas (LP)

**1**. Problema de la Dieta: (Stigler, 1945). Consiste en determinar una dieta de manera eficiente, a partir de un conjunto dado de alimentos, de modo de satisfacer requerimientos nutricionales. La cantidad de alimentos a considerar, sus características nutricionales y los costos de éstos, permiten obtener diferentes variantes de este tipo de modelos. Por ejemplo:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Leche**  **(lt)** | **Legumbre**  **(1 porción)** | **Naranjas**  **(unidad)** | **Requerimientos**  **Nutricionales** |
| **Niacina** | 3,2 | 4,9 | 0,8 | 13 |
| **Tiamina** | 1,12 | 1,3 | 0,19 | 15 |
| **Vitamina C** | 32 | 0 | 93 | 45 |
| **Costo** | 2 | 0,2 | 0,25 |  |

**Variables de Decisión:**

* **X1:** Litros de Leche utilizados en la Dieta
* **X2:** Porciones de Legumbres utilizadas en la Dieta
* **X3:** Unidades de Naranjas utilizadas en la Dieta

**Función Objetivo:** (Minimizar los Costos de la Dieta) **Min 2X1 + 0,2X2 + 0,25X3**

**Restricciones:** Satisfacer los requerimientos nutricionales

* Niacina: **3,2X1 + 4,9X2 + 0,8X3 >= 13**
* Tiamina: **1,12X1 + 1,3X2 + 0,19X3 >=15**
* Vitamina C: **32X1 + 0X2 + 93X3 >= 45**
* No Negatividad: **X1>=0; X2>=0; X3>=0**

2. [Problema de Dimensionamiento de Lotes](http://www.gestiondeoperaciones.net/programacion_lineal/problema-de-produccion-e-inventario-resuelto-con-solver-de-excel/): (Wagner y Whitin, 1958). Consiste en hallar una polìtica óptima de producción para satisfacer demandas fluctuantes en el tiempo, de modo de minimizar los costos de producción e inventario, considerando la disponibilidad de recursos escasos.

Considere que una fábrica puede elaborar hasta 150 unidades en cada uno de los 4 periodos en que se ha subdividido el horizonte de planificación y se tiene adicionalmente la siguiente información:

|  |  |  |  |
| --- | --- | --- | --- |
| Periodos | Demandas  (unidades) | Costo Prod.  (US$/unidad) | Costo de Inventario  (US$/unidad) |
| 1 | 130 | 6 | 2 |
| 2 | 80 | 4 | 1 |
| 3 | 125 | 8 | 2.5 |
| 4 | 195 | 9 | 3 |

Adicionalmente considere que se dispone de un Inventario Inicial de 15 unidades y no se acepta demanda pendiente o faltante, es decir, se debe satisfacer toda la demanda del período.

Variables de Decisión:

* Xt: Unidades elaboradas en el período t (Con t =1,2,3,4)
* It: Unidades en inventario al final del período t (Con t =1,2,3,4)

Función Objetivo: (Minimizar los Costos de Producción e Inventarios) Min 6X1 + 4X2 + 8X3 + 9X4 + 2I1 + 1I2 + 2,5I3+ 3I4

Restricciones:

* Capacidad de Producción por Período: Xt <= 150 (Con t =1,2,3,4)
* Satisfacer Demanda Período 1: X1 + I0 - I1 = 130 (I0 = 15)
* Satisfacer Demanda Período 2: X2 + I1 - I2 = 80
* Satisfacer Demanda Período 3: X3 + I2 - I3 = 125
* Satisfacer Demanda Período 4: X4 + I3 - I4 = 195
* No Negatividad: Xt >=0, It >=0

## 2.1.3 Metodo simplex

El método simplex, es un algoritmo iterativo iterativo publicado por George Dantzig en 1947. El algoritmo a través de iteraciones se va aproximando a la solución deseada en caso de existir una.

La primera vez que se implemento el método simplex fue en el año 1952 para la solución de un problema de 71 variables y 48 ecuaciones.

El Método Simplex hace uso de la propiedad de que la solución óptima de un problema de Programación Lineal se encuentra en un vértice o frontera del dominio de puntos factibles (esto último en casos muy especiales), por lo cual, la búsqueda secuencial del algoritmo se basa en la evaluación progresiva de estos vértices hasta encontrar el óptimo. Cabe destacar que para aplicar el Método Simplex a un modelo lineal, este debe estar en un formato especial conocido como **formato estándar** el cual se ilustra a continuacion. [2]

**FORMA ESTÁNDAR DE UN MODELO DE PROGRAMACIÓN LINEAL**

Consideremos un modelo de Programación Lineal en su forma estandar, que denotaremos en lo que sigue por:

* **Min          c1x1  + c2x2  + ... + cnxn**
* **sa            a11x1 + a12x2 + ... + a1nxn = b1**
* **a21x1 + a22x2 + ... + a2nxn = b2**
* **...          ...                  ...**
* **am1x1 + am2x2 + ... + amnxn = bm**
* **xi >=  0,   i = 1, 2, ..., n    y    m <= n**

Matricialmente escrito como:

**Min**    **cTx**   
**s.a      Ax = b**   
**x >=  0**

No existe pérdida de generalidad en asumir que un modelo de PL viene dado en su forma estándar:

* **EJEMPLO**
* **P)            Max        9u + 2v + 5z**
* **sa            4u + 3v + 6z <=  50**
* **u + 2v - 3z >=  8**
* **2u - 4v + z = 5**
* **u,v >=  0**
* **z e IR**

1. Siempre es posible llevar un problema de maximización a uno de minimización. Si **f(x)** es la función objetivo a maximizar y **x\*** es la solución óptima **f(x\*) >= f(x)**, para todo x factible. **-f(x\*) <= - f(x)**, para todo x factible. En consecuencia:  **x\*** es también mínimo de  **-f(x)**
2. Cada restricción del tipo <= puede ser llevada a una ecuación de igualdad usando una (nueva) **variable de** **holgura** no negativa, con coeficiente nulo en la función objetivo.
3. Cada restricción del tipo >= puede ser llevada a una ecuación de igualdad usando una (nueva) **variable de** **exceso** no negativa, con coeficiente nulo en la función objetivo.
4. Siempre es posible escribir una variable
5. libre de signo como la diferencia de dos variables no negativas.

Considerando la siguiente notación: **u = x1, v = x2, z = x3 - x4, s1 = x5 (holgura), s2 = x6 (exceso)**, el problema P) puede ser escrito en forma equivalente como:

* **Min         - 9x1 - 2x2 - 5x3 + 5x4**
* **+ 0x5 + 0x6**
* **sa:              4x1 + 3x2 + 6x3 - 6x4 +    x5          = 50**
* **x1 + 2x2 - 3x3 + 3x4             - x6  =  8**
* **2x1 - 4x2 +  x3   -   x4                     =  5**
* **xi >=  0,    i=1,2,3,4,5,6.**

## 2.1.4 Algoritmos para la solución de problemas LP

**1. Algoritmo de Seidel [3]**

Seidel da un algoritmo para para problemas de baja dimension lineal que pueden ser adaptados a la forma LP para su solución. El algoritmo de Seidel toma como entrada un conjunto S y un conjunto X (vacio inicialmente) de elementos que se sabe pertenecen al cojunto base optimo. El algoritmo después analiza los elementos restantes uno por uno aleatoriamente, realizando pruebas de segmentación para cada una y dependiendo el resultado realiza una llamada recursiva al mismo algoritmo con un número mayor de elementos base. El pseudocodigo es el siguiente:

def seidel(S,f,X):

R = empty set

B = X

for x in a random permutation of S:

if f(B) ≠ f(B ∪ {x}):

B = seidel(R,f, basis(X ∪ {x}))

R = R ∪ {x}

return B

**2. Algoritmo de Clarkson [3]**

Clarkson define dos algoritmos, uno recursivo y otro iterativo, para programación lineal basada en técnicas de pruebas al azar y sugiere una combinación de llamadas de los dos algoritmos de tal manera que el algoritmo iterativo sea llamado desde el algoritmo recursivo. El algoritmo recursivo selecciona muestras aleatorias continuamente cuyo tamaño es aproximado a la raíz cuadrada del tamaño de entrada, y luego usa pruebas de segmentación para encontrar el subconjunto restante de elementos que deben ser incluidos en al menos un elemento base. El pseudocodigo es el siguiente:

def recursive(S,f):

X = empty set

repeat:

R = a random subset of S with size d√n

B = basis for R ∪ X, computed recursively

V = {x | f(B) ≠ f(B ∪ {x})}

X = X ∪ V

until V is empty

return B

**3. Matousek, Sharir and Welzl [3]**

Matousek, Sharir and Welzl describen un algoritmo que utiliza una propiedad adicional the la programación lineal que no siempre es utilizada por otros problemas del tipo LP: Que todas las bases tienen la misma cardinalidad. Si un problema LP no tiene esta propiedad, puede lograrse adicionando un conjunto d de elementos basura y modificando la función f para retornar ña pareja ordenada de su viejo valor f(A) y del numero minimo d.

El algoritmo remueve elementos uno por uno. En cada paso mantiene la base C que puede inicialmente ser el conjunto de elementos basura. EL pseudocodigo es el siguiente:

def msw(S,f,C):

if S = C:

return C

choose a random element x of S \ C

B = msw(S \ x, f, C)

if f(B) ≠ f(B ∪ {x}):

B = basis(B ∪ {x})

B = msw(S, f, B)

return B

## 2.2 Minimo arbol de expansion (MST)

Un árbol minimo de expasion es un subconjunto de vértices de un grafo no dirigido que conecta todos los vértices juntos, sin ningún ciclo y con el peso minimo para cada vértice dentro de el. [4]

## 2.2.1 Propiedades de un MST [4]

**1. Posible multiplicidad**

Si hay n vértices en un grafo, entonces cada árbol de expansión tiene n-1 vertices.

**2. Unicidad**

Si cada vértice tiene un peso distinto entonces solo habrá un único árbol minimo de expansión.

**3. Subgrafo de minimo costo**

Si los pesos son positivos, entonces el minimo árbol de expansión es el subgrafo con menor costo que conecta todos los vértices, ya que los subgrafos que contienen ciclos tienen necesariamente mas peso.

**4. Propiedad de ciclos**

Para cualquier corte C en el grafo, si el peso de un vértice e de C es mayor que el peso individual de todos los otros vértices de C, entonces ese vértice no puede perteneces al minimo árbol de expansión.

**5. Propiedad de corte**

Para cualquier corte C en el grafo, si el peso de un vértice e en el conjunto de corte C es estrictamente menor que los pesos de todos los otros ejes del conjunto de corte, entonces este vértice pertenece a todos los arboles de menor expansión del grafo.

**6. Vertice de minimo coste**

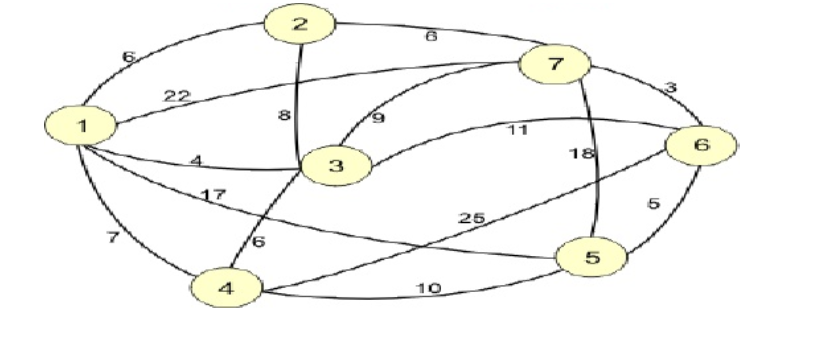
Si el costo minimo de un vértice e en el grafo es único, entonces este eje esta incluido en el árbol de minima expansión.

**7. Contraccion**

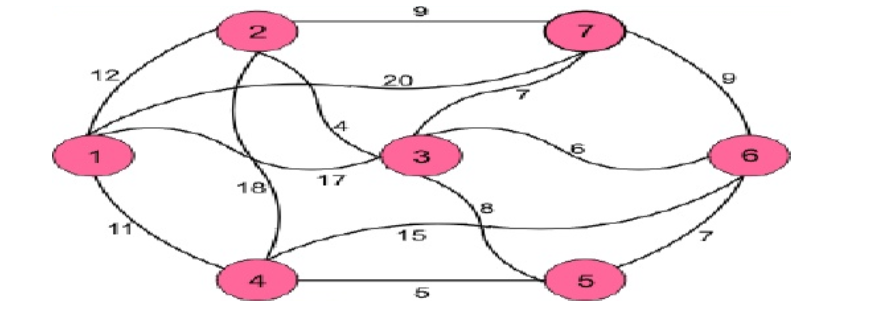
Si T es un árbol de vértices de minima expansión, entonces no se puede contraer T en un simple vértice mientras que se mantiene la invariante de que el árbol minimo de expansión del grafo contraído más T da el árbol minimo de expansión antes de contraerlo.

## 2.2.2 Ejemplos problemas MST [5]

**1.** La siguiente red representa una serie de poblados que se encuentran comunicados a través de caminos rurales o empedrados. El Gobernador del Estado al que pertenecen ha aprobado se pavimenten los caminos que permitan unir a todos los poblados, buscando que la distancia a pavimentar sea la mínima posible. ¿Cuáles caminos son los que se deben de pavimentar? 2. La siguiente red representa una serie de nuevas colonias que se han establecido en una localidad, la compañía de Luz desea suministrar el servicio correspondiente, para ello se requiere instalar el cableado eléctrico. ¿Determine la cantidad de km de cable mínimo que debe de instalarse de tal forma que se proporcione el servicio a todas las colonias?



**2.** La siguiente red representa una serie de nuevas colonias que se han establecido en una localidad, la compañía de Luz desea suministrar el servicio correspondiente, para ello se requiere instalar el cableado eléctrico. ¿Determine la cantidad de km de cable mínimo que debe de instalarse de tal forma que se proporcione el servicio a todas las colonias?



## 2.2.3 Algoritmos para la resolución de problemas MST

**1. Algoritmo de Kruskal [5]**

El algoritmo de Kruskal permite hallar el árbol de minima expansión de cualquier grafo valorado. El algoritmo consta de los siguientes pasos:

* Marcar la arista con menor valor. Si hay más de una, se elige cualquiera de ellas.
* De las aristas restantes, se marca la que menor valor tenga, si hay mas de una se elige cualquiera de ellas.
* Repetir el paso anterior siempre que la arista elegida no forme un ciclo con las ya marcadas.
* El proceso termina cuando todos los nodos del grafo en alguna de las aristas marcadas, es decir, cuando tenemos marcados n-1 arcos, siendo n el número de nodos del grafo.

El pseudocodigo es el siguiente:

**Función** Kruskal(*G*)

**Para cada** *v* en V[G] **hacer**

Nuevo conjunto C(*v*) ← {*v*}.

Nuevo heap Q que contiene todas las aristas de G, ordenando por su peso.

Defino un arbol T ← Ø

// n es el número total de vértices

**Mientras** T tenga menos de n-1 aristas y !Q.vacío() **hacer**

(*u*,*v*) ← Q.sacarMin()

// Previene ciclos en T. agrega (*u*,*v*) si *u* y *v* están diferentes componentes en el conjunto.

// Nótese que C(*u*) devuelve la componente a la que pertenece *u*.

**Si** C(*v*) ≠ C(*u*) **hacer**

Agregar arista (*v*,*u*) a T.

Merge C(*v*) y C(*u*) en el conjunto

**Responder** arbol T

**2. Algoritmo de Prim [5]**

El algoritmo de Kruskal permite hallar el árbol de minima expansión de cualquier grafo valorado. El algoritmo consta de los siguientes pasos:

* Se marca un nodo cualquiera, será el nodo de partida.
* Relacionamos las aristas de menor valor incidente en el nodo marcado anteriormente, y marcamos el otro nodo en el que incide.
* Repetir el paso anterior, siempre que la arista elegida enlace un nodo marcado y otro que no lo este.
* El proceso termina cuando tenemos todos los nodos marcados del árbol.

El pseudocodigo del algoritmo es el siguiente:

*Prim* (Grafo *G*)

*/\* Inicializamos todos los nodos del grafo.*

La distancia la ponemos a infinito y el padre de cada nodo a NULL

Encolamos, en una cola de prioridad

donde la prioridad es la distancia,

todas las parejas <nodo,distancia> del grafo\*/

**por cada** *u* en *V[G]* **hacer**

distancia[*u*] = INFINITO

padre[*u*] = NULL

Añadir(cola,<*u*,distancia[*u*]>)

distancia[*u*]=0

**mientras** !esta\_vacia(cola) **hacer**

*// OJO: Se entiende por mayor prioridad aquel nodo cuya distancia[u] es menor.*

*u* = extraer\_minimo(cola) //devuelve el minimo y lo elimina de la cola.

**por cada** **v** adyacente a '**u'** **hacer**

**si** ((*v* ∈ cola) **&&** (distancia[*v*] > peso(u, v)) **entonces**

padre[*v*] = *u*

distancia[*v*] = peso(u, v)

Actualizar(cola,<*v*,distancia[*v*]>)

## 2.3 Cliques en grafos

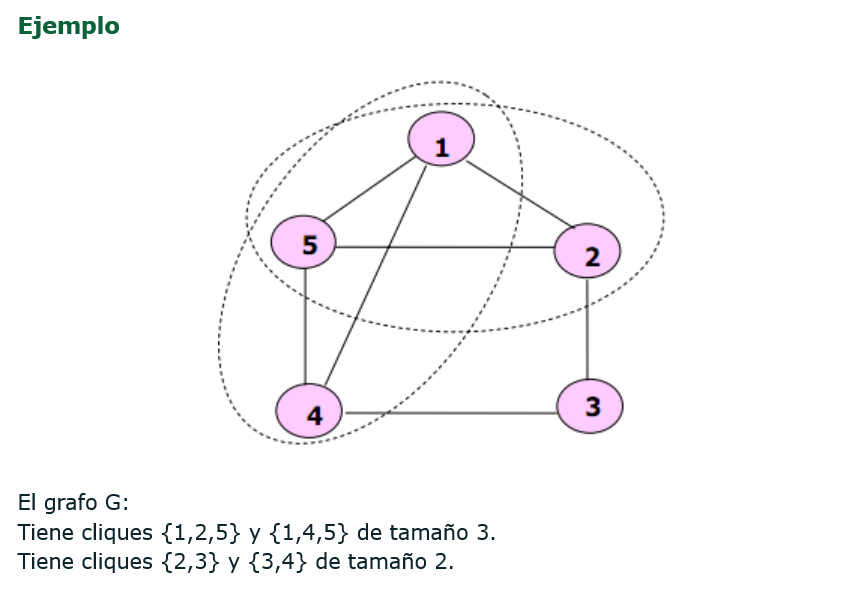
## 2.3.1 ¿Qué es un clique?

En teoría de grafos, un clique en un grafo no dirigido G es un conjunto de vértices V tal que para todo par de vértices de V, existe una arista que las conecta. En otras palabras, un clique es un subgrafo en que cada vértice está conectado a cada otro vértice del subgrafo, es decir, todos los vértices del subgrafo son adyacentes. Esto equivale a decir que el subgrafo inducido por V es un grafo completo. El tamaño de un clique es el número de vértices que contiene. [6]

## 2.3.2 El problema del clique

El problema de clique es un problema de decisión para determinar cuándo un grafo contiene un clique de al menos un tamaño k. Una vez que tenemos k o más vértices que forman un clique, es trivial verificar que lo son, por eso es un problema NP. El correspondiente problema de optimización, consiste en encontrar un clique de tamaño máximo en un grafo (un subgrafo completo de tamaño máximo). Este problema se puede enunciar como un problema de decisión si la pregunta que se hace es saber si existe un clique de tamaño k en el grafo. [7]

## 2.3.3 Ejemplo



## 2.3.4 Algoritmos para la resolución de cliques

**1. Algoritmo de fuerza bruta [8]**

Un ejemplo de algoritmo de fuerza bruta para encontrar una clique en un grafo consiste en listar todos los subconjuntos de vértices V y verificar para cada uno de ellos si forma una clique.

**2. Algoritmo Bon-Kerbosch [8]**

Este algoritmo consiste en arrancar con cliques de un solo elemento e intentar mezclar cliques para obtener otras más grandes, hasta que no queden más mezclas por intentarse. Dos cliques pueden ser mezcladas si cada nodo de la primera es adyacente a cada nodo de la segunda.  
  
Es eficiente en el peor de los casos por un resultado de Moon and Moser, donde un grafo de n vértices tiene a lo sumo 3^(n/3) cliques máximos, y el tiempo de ejecución del peor caso del algoritmo Bron-Kerbosch, con una estrategia dinámica que reduce al mínimo el número de llamadas recursivas realizadas en cada paso, es O(3^(n/3)), que coincidan con este límite.

# Referencias

[1] «Linear programming», *Wikipedia*. 09-feb-2017.

[2] «Programación Lineal - Ejemplos de Programación Lineal, Método Simplex, Dualidad y Análisis de Sensibilidad». [En línea]. Disponible en: http://www.programacionlineal.net/. [Accedido: 18-feb-2017].

[3] «LP-type problem», *Wikipedia*. 12-dic-2015.

[4] «Minimum spanning tree», *Wikipedia*. 16-feb-2017.

[5] «2.3 Problema Árbol Expandido Mínimo», *INVESTIGACION DE OPERACIONES*, 09-abr-2010. .

[6] «Clique», *Wikipedia, la enciclopedia libre*. 06-ene-2016.

[7] «Problema de la clique», *Wikipedia, la enciclopedia libre*. 17-may-2016.

[8] R. González, «Esteban González: Problema de Clique», *Esteban González*, 05-jul-2011. .

.