PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Apuntes Investigación de operaciones Avanzadas

GONZALO TELLO VALENZUELA

Docente: Guillermo Cabrera

2do Semestre, 2020

Índice

Lista de Figuras IV

Lista de Tablas V

1 Sistemas complejos y Complejidad computacional **1**
1.1 Los inicios: Reduccionismo 1
1.2 ¿Qué es un sistema complejo? 1
1.2.1 Definición 2
1.2.2 Ejemplos 2
1.2.3 Características 2
1.2.4 Medidas 2
1.3 Complejidad Computacional 3
1.3.1 ¿Qué es? 3
1.3.2 Computabilidad 3
1.3.3 Russell's paradox 5
1.3.4 Hilbert Problems 5
1.3.5 Halting problem 6
1.3.6 Problemas de decisión 6
1.3.7 Clasificación de los problemas 7

2 Introducción a la programación Linea: Modelado de problemas **8**
2.1 Optimización 8
2.2 Tipos de problemas de optimización lineal 8
2.3 Definición del problema 9
2.3.1 Ejemplo 1 9
2.3.2 Ejemplo 2 9
2.3.3 Ejemplo 3 10
2.4 Solving problem 10
2.4.1 Validación / Implementación 11
2.5 Procedimiento General para la construcción de modelos matemáticos 11
2.5.1 Identificar las variables de decisión del problema . . . 12
2.5.2 Identificar los datos del problema 12
2.5.3 Formular la función objetivo del problema 12
2.5.4 Formular las restricciones del problema. 13
2.6 ¿Qué es la programación lineal? 13
2.6.1 ¿Por qué estudiar Linear Programming? 14

2.6.2 Notación en Linear Programming 14
2.6.3 Modelo General 14
2.6.4 Terminología 15
2.6.5 Ejemplo 15
2.6.6 General 16
2.6.7 Reformulación de restricciones 16

3 Introducción a AMPL 19
3.1 Mathematical programming 19
3.2 Lenguaje de modelado algebraico 20
3.3 Lenguaje de modelado AMPL 21
3.4 Maximizar ganancias - Minimizar costos 22
3.4.1 Problema de programación lineal de dos variables . . . 22
3.4.2 Elementos del modelado en AMPL 22
3.5 Modelo de programación lineal 24
3.5.1 Consideraciones 27
3.5.2 Modelo mejorado 27
3.5.3 Prueba 28
3.5.4 Manejo de errores 28
3.5.5 Añadir limites inferiores 29
3.5.6 Uso de etapas 30

4 Introducción a JuMP 33
4.1 Lenguaje de modelado JuMP 33
4.2 Problema de programación lineal de dos variables 34
4.3 Modelo 34
4.4 Variables de decisión 34
4.5 Restricciones 35
4.6 Función objetivo 35
4.7 Resolver Modelo 35

5 Método de Nelder Mead 36
5.1 Historia 36
5.2 Clasificación 36
5.2.1 Derivate Free Methods 37
5.3 Nelder-Mead Method 37
5.3.1 Pseudocodigo 41
5.3.2 Ejemplo 42

6 Métodos en programación matemática **44**
6.1 Condiciones de optimalidad 44
6.1.1 Ejemplo 45
6.2 Constrained Optimisation 46
6.2.1 Definiciones 46
6.3 Ejemplo (extraído de Nocedal & Wright) 48
6.3.1 Lagrange 49

7 Relajación Lagrangeana. **52**
7.1 Método de Lagrange con restricciones de desigualdad 53

8 Dualidad Lagraniana **56**
8.1 Saddle Point and Duality 56
8.2 Duality 56
8.2.1 Ejemplo 57

9 Optimización Combinatorial **59**
9.1 Optimización Continua vs Optimización Discreta 59
9.1.1 Ejemplo - CFLP 60
9.2 Branch & Bound 61
9.2.1 Ejemplo 63

Lista de Figuras

1 Diagrama Metodología en Investigación de operaciones 8
2 Representación ejemplo 3 11
3 The triangle $\triangle BGW$ and midpoint M and reflected point R for the Nelde-Mead method 39
4 The contraction point C_1 or C_2 for Nelder-Mead method . . . 40
5 Shrinking the triangle toward B 41
6 Schematic representation of saddle point solution to PP . . . 57
7 Branch & Bound - Nivel 1 64
8 Branch & Bound - Nivel 2 64

Lista de Tablas

1. Sistemas complejos y Complejidad computacional

1.1. Los inicios: Reduccionismo

Descartes, Newton, entre otros promovieron el reduccionismo:

- 'Dividir todas las dificultades bajo examinación en tantas partes como sea posible, y tantas como sean necesarias para resolverlas de mejor manera'
- 'Conducir mis pensamientos en un determinado orden, empezando con los objetos más simples y fáciles de entender, ascendiendo gradualmente, paso a paso, hacia el conocimiento de lo más complejo.

A finales de los 1800's, la mayoría de los cientificos estaba de acuerdo con que 'los principios más básicos han sido establecidos y que, a partir de ellos, todos los fenómenos que podemos observar pueden ser explicados'.

El reduccionismo no es capaz de explicar los sistemas complejos con los que vivimos.

'El todo es más que la suma de todas las partes'.

1.2. ¿Qué es un sistema complejo?

'How Can the Study of Complexity Transform OurUnderstanding of the World?'

Primero que todo, se debe recordar que un **sistema** es un conjunto de elementos o partes que interactuan entre sí a fin de alcanzar un objetivo concreto.

Bajo esta definición, un sistema complejo es aquel que no puede ser estudiado a partir de sus unidades individualmente, se requiere algo más que una suma de la contribución de cada elemento. Como sistema, estos entregan información, sin embargo, esta es impredescible e imposible de obtener a partir del análisis de sus componentes.

En un sistema complejo, las interacciones dan lugar a fenómenos emergentes que no se pueden derivar sólo del estudio del comportamiento de cada parte por separado.

1.2.1. Definición

Es un sistema en el que grandes conexiones de componentes sin control central ni reglas simples de operación, dan lugar a comportamientos colectivos complejos, sofisticados procesos de información y adaptación, mediante la vía del aprendizaje o evolución.

1.2.2. Ejemplos

- Colonias de insectos: hormigas, abejas.
- Cerebro, sistema Inmune
- Economía (la mano invisible)

1.2.3. Características

- Complex Collective Behavior
- Signalling and information processing
- Adaptation

Otros conceptos comunmente usados para referirse a estos términos son:

- Complex Adaptative Systems
- Self Orginising Systems
- Emergent Systems

1.2.4. Medidas

Seth Lloyd en 2001 propone 3 dimensiones para la medición de la complejidad de un objeto o proceso

- ¿Qué tan dificil es de describir?
- ¿Qué tan dificil es de crear?
- ¿Cuál es su grado de organización?

Buenas medidas de la complejidad de un proceso/sistema/objeto.

- Tamaño

- Entropía (de Shannon)
- Algorithmic Information Content
- Capacidad computacional (Turing machine)

Otros: Fractal, Jerarquía, Estadística.

1.3. Complejidad Computacional

1.3.1. ¿Qué es?

- Computación:
  - Problemas para ser resueltos
  - Algoritmos que lo resuelvan (modelo computacionales).
- Complejidad de un problema:
  - Cuánto recurso es necesario/suficiente para resolverlo

Complejidad computacional de un problema en modelos computacionales $\Longrightarrow$ Computabilidad.

1.3.2. Computabilidad

El objetivo es determinar 'qué puede' y 'qué no puede' ser resuelto por un programa computacional.

Un programa es computable si puede ser resuelto por un algoritmo, sin embargo, un problema no es computable si no puede ser resuelto por algún algoritmo.

Tratando de mecanizar el razonamiento:

- Silogismos-deducciones lógicas (Aristoteles, 350 BC)
- Axiomas - Reglas de ingerencias (Euclides, the elements).

Euclides

- Postulados o Axiomas
  - Dos puntos cualesquiera determinan un segmento de recta.
  - Un segmento de recta se puede extender indefinidamente en una línea recta.

    - Se puede trazar una circunferencia dados un centro y un radio cualquiera.
    - Todos los ángulos rectos son iguales entre sí.
    - Si una línea recta corta a otras dos, de tal manera que la suma de los ángulos interiores del mismo lado sea menor que dos rectos, las otras dos rectas se cortan al prolongarlas, por el lado en el que están los ángulos menores que dos rectos (axioma de las paralelas).

- Reglas de inferencia

    - Things which equal the same thing also equal one another.
    - If equals are added to equals, then the wholes are equal.
    - If equals are subtracted from equals, then the remaindersare equal.
    - Things which coincide with one another equal one another.
    - The whole is greater than the part.

A partir de éstos axiomas y reglas de inferencia, una serie de proposiciones fueron propuestas. Una proposición es una afirmación suficientemente precisa para ser determinada como verdadera o falsa.

La primera proposición de Euclides es: Dada cualquier linea, un triangulo equilatero puede ser construido con los borde que tengan la longitud de esa línea.

Las proposiciones deben ser probadas.

Definición

Secuencia de pasos que termina con la proposición. Cada paso debe seguir desde los axiomas usando las reglas de inferencia.

- Por construcción
- Por contradicción

Un sistema axiomático puede ser:

- Consistente
- Completo

1.3.3. Russell's paradox

El fin del siglo XIX trajo consigo el desarrollo de sistemas axiomáticos completos para algún área de las matemáticas.

Gottlob Frege en 1893, mediante su obra 'Grundgestze der Arithmetik' (Leyes Básicas de la Aritmética), planteó el 'Sistema axiomático para toda la ciencia matemática a partir de la lógica simple'.

De lo anterior, Bertrand Russell descubre un problema con el sistema propuesto por Frege, conocido como *Russell's Paradox (paradoja de Russell)*.

Supongamos que R es definida como un conjunto de todos los conjuntos que no se contienen a si mismos como elementos.

Por ejemplo, el conjunto de los números primos no se contiene a si mismo como miembro y, por lo tanto, es miembro de R. Por otra parte, el conjunto de elementos no-primos también es miembro de R. Este conjunto contiene a todos los conjuntos ya que un conjunto no es un número primo (Debe contenerse a si mismo).

En *la paradoja de Russell*, la pregunta (problema de decisión) es:

Es $R \in R$?

- Si $R \in R$, luego, R no puede contenerse a si mismo y la sentencia $R \in R$ debe ser falsa.
- No, $R \notin R$, luego R no se contiene a si mismo y, por definicion $R \in R$, una contradicción. Por lo tanto, la sentencia $R \notin R$ debe ser falsa.

La pregunta en la paradoja de Russell es perfectamente clara y precisa, sin embargo, ninguna de las dos posibles respuestas 'Si' o 'No' tienen sentido. Formalmente resumimos la paradoja como:

$For\ any\ set\ S,\ S \in R \iff S \notin S$
$If\ S = R, then\ R \in R \iff R \notin R$ contradicción

El filósofo Cretado Epimenides dijo que *'Todos los cretados son mentirosos'*. Si la sentencia es verdadera, entonces Epimenides, un cretano, no es mentiroso y, luego, la sentencia es falsa.

1.3.4. Hilbert Problems

En el 1900, David Hilbert sacude a la comunidad cientifica con un set de preguntas que podemos resumir en 3:

1. Is Mathematics complete?
2. Is mathematics consistent?
3. Is every statement in mathematics decidable? (Entsheidungsproblem).

Estas preguntas no pudieron ser resueltas sino hasta 1930.
Gödel demostró que la matemática debía ser inconsistente o incompleta, pero que no podía ser ambas.

'This statement is no provable'

1.3.5. Halting problem

La tercera pregunta de Hilbert fue resuelta 5 años más tarde por Alan Turing. La pregunta aquí es, ¿Existe algún problema para el cual no existe un algoritmo (finito) que lo resuelva?
Definición

- Problema: Descripción de una entrada y una salida deseada.
- Procedimiento: Una especificación de una serie de acciones
- Algoritmo: Un procedimiento fínito (garantía de término)

Un procedimiento soluciona un problema si ese procedimiento produce una salida correcta de todas las posibles salidas.
Un problema P es computable si existe un algoritmo que lo resuelva.

- Finito
- Produce la salida correcta para todos los posibles inputs de P.

Sin embargo, un problema P no es computable si no existe un algoritmo que lo resuelva. Alan Turing demostró que los problemas no computables existen, y que se denominan **Halting problem**.
Como corolario, demostró que el *Entsheidungsproblem* también es no computable.

1.3.6. Problemas de decisión

Dado un input, obtener un output 'satisfactorio':

- Evaluación: Sólo un output es correcto.

- Búsqueda: Encontrar uno de entre varios (si existe)
- Decisión: Encontrar uno si existe

Los problemas que veremos en esta parte del curso son los problemas de decisión

- Input:
$$String\ binario \Longrightarrow \mathbb{N}_0{}^+$$
- Output:
$$\{Si, No\}$$
- Ejemplo:
$$L = \{x | xtiene\ el\ mismo\ numero\ de\ 0\ y\ 1\}$$

Problema de decisión: dado un input decidir si esta o no en L.

Sabemos que una secuencia infinita de bits representa un número en $\mathbb{R}$

$$\mathbb{N} << \mathbb{R}\ (\mathbb{R}\ no\ es\ numerable)$$

Dado que cada secuencia de strings responde a un problema en particular, no hay suficientes strings para todos los posibles problemas. Luego, la gran mayoría de los problemas no se pueden resolver.

1.3.7. Clasificación de los problemas

- P = Problemas que se pueden resolver en tiempo polinomial
- Exp = Problmas que se pueden resolver en tiempo exponencial
- R = Problemas que se pueden resolver en tiempo finito
- Problemas NP = Problemas solucionables en tiempo vía 'lucky algorithms'
  - Siempre le apuntan a las mejores opciones.
  - No necesitan explorar todo el espacio de búsqueda.
  - Decimos que son **no deterministas**
  - Output siempre es 'SI', si existe la solución ('NO' en caso contrario).

2. Introducción a la programación Linea: Modelado de problemas

2.1. Optimización

Cuando hablamos de optimización, nos referimos a el proceso de maximizar o minimizar alguna función objetivo. Una función objetivo es una expresión matemática que representa ya sea un costo (a minimizar) o una ganancia (a ser maximizada).

Para esto, necesitamos variables de decisión que nos permitan medir dicha función objetivo. Los valores que estas variables de decisión pueden tomar estarán restringidos por condiciones lógicas, físicas o de gestión, entre otras.

Estas restricciones también deben representadas matemáticamente e incluidas en el modelo.

2.2. Tipos de problemas de optimización lineal

- Problemas lineales
- Problemas enteros lineales
- Problemas binarios lineales

Depende de el tipo de variable de decisión y de la estructura de la función objetivo y restricciones incorporadas al modelo.

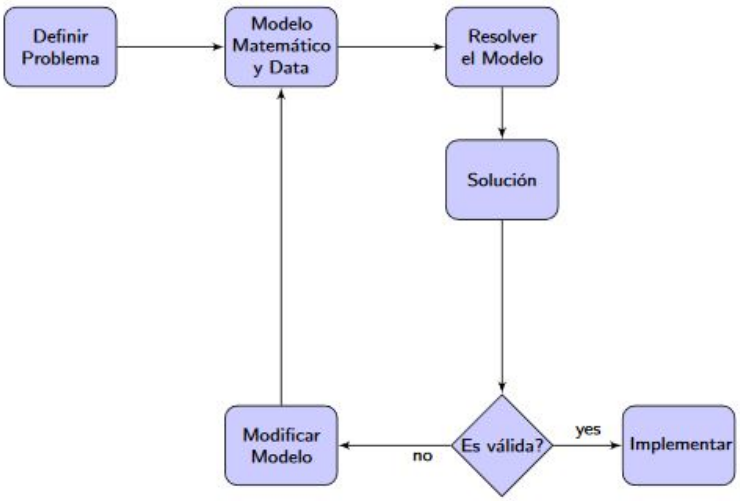


Figura 1: Diagrama Metodología en Investigación de operaciones

2.3. Definición del problema

La gran mayoría de los problemas tienen 3 características principales que deben ser identificadas.

- ¿Qué necesitamos decidir?
- ¿Cuándo decidimos, qué queremos lograr? (a menudo minimizar o maximizar algo).
- ¿Qué dificultades encontramos para lograr nuestro objetivo?

2.3.1. Ejemplo 1

Sarah has just won the first prize in a Christmas Shopping Draw. She is allowed to spend up to 12 minutes in the "Mega Discount" store and up to 12 mins in the "Super Buy"store, taking whatever she can fit into a single large trolley. It is expected that for each minute, she can grab on average 4 kg of goods worth $ 400 from Mega Discount and 3 kg of goods worth $250 from Super Buy. The maximum capacity of the trolley is 72kg. What should Sarah's strategy be?

- Decision
- Objetivo
- Restricciones

2.3.2. Ejemplo 2

Mark has to decide how to split his retirement funds between a stock fund and a bond fund. The Stock Fund has typically given an annual return of 10 %, while the Bond Fund has given a 6 % return. Mark wants to ensure he invests his money wisely,and so doesn't want to put all his eggs in one basket. In particular, he wants no more than 75 % of his money invested in either fund, and wants no more than twice as much in stocks as in bonds.

- Decision
- Objetivo
- Restricciones

2.3.3. Ejemplo 3

Case Chemicals produces two solvents, CS-01 and CS-02, at its Cleveland plant. The businesses that buy these solvents use them to dissolve certain toxic substances that arise during particular manufacturing processes. Case Chemical's production plant has two departments - Blending and Purification. The Blending department operates 40 hours per week and employs five full-time and two part-time workers, who work 15hours per week. These people run the seven machines that blend certain chemicals to produce each solvent.Each thousand gallons of CS-01 takes 2 hours of staff time to blend; the same quantity of the lighter CS-02takes half this time to blend. The products leave the Blending Department to be refined in the Purification Department which currently has seven purifier machines and employs six full-time workers and one part-time worker who puts in 10 hours per week. Sixty minutes of staff time can be used to purify either one thousand gallons of CS-01 or 500 gallons of CS-02.Case Chemicals has a nearly unlimited supply of the raw materials it needs to produce the two solvents. Case Chemicals can sell any amount of CS-01, but the demand for the more specialized product CS-02 is limited to, at the most, 120,000 gallons per week. Finance estimates a profit of $0.30 per gallon for CS-01, and $0.50 per gallon for CS-02. As Production Manager, you want to determine the optimal weekly production plan for Case Chemicals. How much of each solvent should Case Chemicals produce to maximise profit?

2.4. Solving problem

Tenemos (al menos) dos caminos para resolver estos problemas los que pueden ser identificados:

- Garantía de optimalidad
- Buenas (?) aproximaciones

Nuestra capacidad de formular problemas es mucho mayor que nuestra capacidad de resolverlos. De hecho, programas como MS-Excel, Gurobi, CPlex o IpOpt, entre otrosm sólo pueden resolver una fracción de los problemas que podemos formular. Muchas investigaciones alrededor del mundo trabajan por disminuir esta distancia entre lo que formulamos y lo que podemos resolver.

2.4.1. Validación / Implementación

- **Validación**.

  El concepto de **validación** implica asegurar que la solución obtenida tiene sentido y que puede ser implementada.

- **¿Por qué validar?**.

  El modelo matemático podría no haber capturado todas las limitaciones del problema real. Ciertos aspectos del problema podrían haber sido pasadas por alto, omitidos (intencionalmente) o simplificados. Los datos utilizados podrían haber sido incorrectamente estimados.

2.5. Procedimiento General para la construcción de modelos matemáticos

Tomaremos como referencia el ejemplo expuesto en el punto 2.3.3.

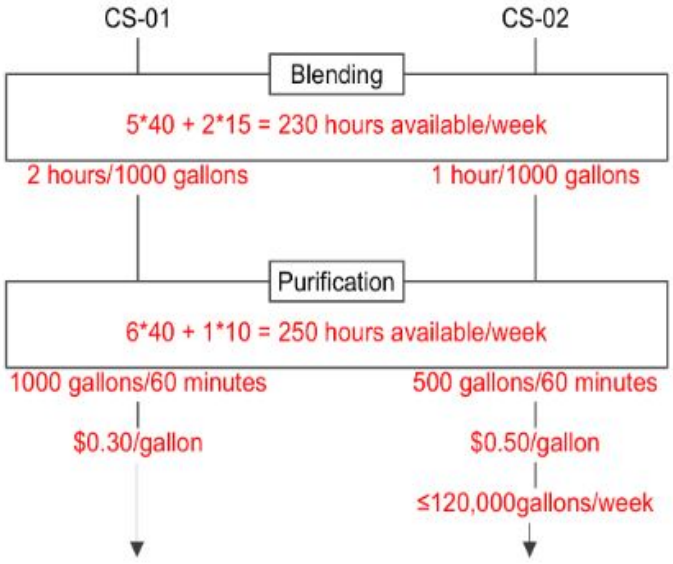


Figura 2: Representación ejemplo 3

Formulación del problema: El paso desde una descripción cualitativa del problema (escrita o verbal) a una formulación matemática:

2.5.1. Identificar las variables de decisión del problema

Algunas preguntas útiles que podemos hacer para identificar variables de decisión son:

- ¿Qué variables afectan los costos/utilidades del problema?
- ¿Sobre qué aspectos tenemos control y sobre cuales no?
- ¿Qué decisiones se deben tomar?
- ¿Qué valores, una vez determinados, constituyen una solución al problema?

Algunas notas respecto a este punto derivan a las siguientes premisas:

1. Siempre traten de encontrar las decisiones más relevantes que deben ser tomadas y; asegúrense que no hay más decisiones que necesiten ser modeladas.
2. Asegúrense de especificar la(s) unidad(es) de medida de su(s) variable(s) de decisión!!

2.5.2. Identificar los datos del problema

Los datos relevantes del problema son aquellos que deben conocerse para escribir las ecuaciones del modelo.

Algunas notas respecto a este punto derivan a la siguiente consideración

- Asegúrense de no mezclar distintas unidades (minuto con horas, litros con galones, etc).

2.5.3. Formular la función objetivo del problema

- Expresar el objetivo de manera verbal: '*Maximizar la utilidad semanal de la producción de CS-01 y CS-02*'
- Si es necesario, descomponer el objetivo en sumatorias, diferencias, productos de los términos individuales. Ejemplo:

$$MaxProfit = (CS - 01profit) + (CS - 02profit)$$

- Definir los terminos individuales identificados en el paso anterior.

2.5.4. Formular las restricciones del problema.

Las restricciones buscan representar:

- Límites físicos
- Restricciones operativas o de Gestión
- Regulaciones externas (ej: leyes laborales)
- Relaciones implícitas entre variables
- Restricciones de no-negatividad de las variables (ej: tiempo).

Por cada restricción:

- Expresar la restricción de manera verbal: *'Las horas usadas en la mezcla (blending) no puede exceder los 230 horas por semana'.*
- Si es necesario, descomponer la restricción en sumatorias, diferencias, productos de los terminos individuales. Ej:

  $staffHoursBlendingCS - 01 + staffHoursBlendingCS - 02 \leq 230$

- Expresar las cantidades individuales matemáticamente usando las variables de decisión y los datos encontrados en las fases anteriores.

2.6. ¿Qué es la programación lineal?

A typical optimization problem is to find the best element from a given set. In order to compare elements, we need a criterion, which we call an objetive function.

The given set is called the feasible set which is usually defined by.

$$X = \{x \in \mathbb{R}^n | g_i \leq 0, \forall i = 1, ..., m\}$$

Este problema de optimización puede ser formulado como:

$$\max_{x \in X} F(x)$$

Donde X corresponde al hiper-espacio definido por el conjunto de restricciones:

$$g_i \leq 0 \in \mathbb{R}$$

2.6.1. ¿Por qué estudiar Linear Programming?

- **Es simple**: Puede ser resueeelto de manera eficiente.
- Es la base para el desarrollo de técnicas en otros problemas más complejos (Integer programming, noin-linear programming, etc).

2.6.2. Notación en Linear Programming

1. Para una matriz A, denotaremos su transpuesta como A^t
2. Un vector n-dimensional $x \in \mathbb{R}^n$ se denota por un vector columna

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_{n-1} \\ x_n \end{bmatrix}$$

3. Dado dos vectores $x = (x_1, x_2, ..., x_n)^t$ e $y = (y_1, y_2, ..., y_n)^t$, decimos que la multiplicación $x^t y$ se denota por:

$$x^t y = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + x_2 y_2 + ... + x_n y_n$$

2.6.3. Modelo General

En un LP general, el vector de costo $c = (c_1, c_2, ..., c_n)^t$ es dado. Podemos escribir la función objetivo como la minimización (o maximización) de una función de costo lineal:

$$c^t x = \sum_{i=1}^{n} c_i x_i$$

Esta función estará sujeta a un conjunto finito de restricciones de igualdad y desigualdad. En efecto:

$$\max_{x \in \mathbb{X}} c^t x$$

s.t.

$$a_i^t x \geq b_i, i \in \mathbb{M}_+$$

$$a_i^t x \leq b_i, i \in \mathbb{M}_-$$
$$a_i^t x = b_i, i \in \mathbb{M}_0$$
$$x_j \geq 0, j \in \mathbb{N}_+$$
$$x_j \leq 0, j \in \mathbb{N}_-$$

donde $a_i = (a_{i1}, a_{i2}, ..., a_{in})^t$ es un vector en $\mathbb{R}^n$ y b_i es un escalar.

2.6.4. Terminología

En el modelo anterior, x_i son nuestras variables de decisión. Usualmente, x no es un escalar, sino un vector de largo n. De ahí que $x \in \mathbb{X} \subseteq \mathbb{R}^n$, es decir, $x = x_1, x_2, ..., {x_n}^t$, donde $x_i \in \mathbb{R}, \forall i = 1, 2, ..., n$.

Cada restricción es bien una ecuación o una inecuación de la forma $\leq$ o $\geq$. Restricciones de la forma $a_i^t x(\leq, =, \geq) b_i$ son, a veces, llamadas restricciones funcionales (fuctional contraints). En LP, todas las funciones son funciones lineales.

Si $j \notin N_+$ y $j \notin N_-$ entonces decimos que no hay restricciones de signo sobre las variables de decisión. En este caso, decimos que la variable x_j es 'no restringida' (unrestricted) o 'sin límites' (unbounded).

Decimos que un vector $x = (x_1, x_2, ..., x_n)^t$ que satisface todas las restricciones es **factible**. El conjunto de todos los vectores factibles es llamado 'región factible' (fesible region) o 'conjunto factible' (feasible set).

Decimos que la función $c^t x$ es la función objetivo o función de costo (cost function). Un vector factible x^* que minimiza (maximiza) la función objetivo, es decir,

$$(c^t x^* \leq c^t x; \forall \mathbf{x} \in \mathbb{X})$$
$$(c^t x^* \geq c^t x; \forall \mathbf{x} \in \mathbb{X})$$

es una solución del problema o vector óptimo (solución óptima). El valor de $c^t x^*$ corresponderá al valor óptimo (optimal objective value).

Diremos que la función de costo 'no tiene límites' (unbounded), es decir $c^t x^* = -inf(+inf)$ si por cada número real k, podemos encontrar un vector factible $\mathbf{x'}$ tal que su costo $c^t x' < (>)k$. Maximizar $c^t x$ es equivalente a minimizar $-c^t x$

2.6.5. Ejemplo

Recordemos el ejemplo propuesto en la sección 2.3.1, un modelo acorde a ese problema está dado por:

$$\text{máx}\, 400x_{md} + 250x_{sd}$$

s.t

$$4x_{md} + 3x_{sd} \leq 72$$

$$x_{md} \leq 12$$

$$x_{sd} \leq 12$$

$$x_{md}, x_{sd} \geq 0$$

2.6.6. General

Los problemas de programación lineal no admiten funciones y/o restricciones del tipo

$$x_1^2 + 3x_2^5$$

$$x1x2 \geq 10$$

$$\frac{1}{x}$$

$$\log x$$

Debemos notar que algunas restricciones no-lineales pueden ser convertidas (o reformuladas) en restricciones lineales.

2.6.7. Reformulación de restricciones

Asuma la siguiente restricción

$$\frac{5x_1 + 7x_2 + 2x_3 + x_4}{1 + x_1 + 4x_2 + 2x_3 + 5x_4} \leq 3$$

$$x_i \geq 0, \forall i = 1, 2, 3, 4$$

Si el denominador es siempre positivo, entonces decimos (**Estrategia 1**)

$$5x_1 + 7x_2 + 2x_3 + x_4 \leq 3(1 + x_1 + 4x_2 + 2x_3 + 5x_4)$$

$$2x_1 - 5x_2 - 4x_3 - 14x_4 \leq 3$$

Si el denominador es siempre negativo, también podemos transformar, teniendo cuidado de cambiar el sentido de la inecuación.

Por otro lado, existe una **Estrategia 2**

$$D = 1 + x_1 + 4x_2 + 2x_3 + 5x_4 \Longrightarrow 1 + x_1 + 4x_2 + 2x_3 + 5x_4 - D = 0$$

$$1 + x_1 + 4x_2 + 2x_3 + 5x_4 - D = 0$$

$$5x_1 + 7x_2 + 2x_3 + x_4 - 3D \leq 0$$

$$D \geq 0(opcional)$$

Si el denominador es siempre negativo, también podemos transformar, teniendo cuidado de cambiar el sentido de la inecuación.

Sin embargo, ¿que hacemos con un caso como este?

$$\text{mín máx}(x_1, x_2)$$

Agregar una variable M y asegurarse de que corresponda al máximo entre x_1 y x_2.

$$minM$$

s.t

$$M - x_1 \geq 0$$

$$M - x_2 \geq 0$$

Si $x_1 \geq 0$ y $x_2 \geq 0$, entonces podemos agregar una restricción $M \geq 0$ (está ímplicito)

Por otro lado, para un caso de valor absoluto, como el siguiente:

$$min|x|$$

s.t

$$x \geq -80$$

$$x \leq 100$$

$$x \in \mathbb{R}$$

Incluir dos variables $x_1 \geq 0$ y $x_2 \geq 0$ y hacer $x = x_1 - x_2$

$$minx_1 - x_2$$

s.t

$$x \geq -80$$

$$x \leq 100$$

$$x = x_1 - x_2$$

$$x_1 \leq 0$$

$$x_2 \leq 0$$

Finalmente, si regresamos al problema propuesto en la sección 2.3.1:

$$\text{máx}\, 400x_{md} + 250x_{sd}$$

s.t

$$4x_{md} + 3x_{sd} \leq 72$$

$$x_{md} \leq 12$$

$$x_{sd} \leq 12$$

$$x_{md}, x_{sd} \geq 0$$

Dado que tenemos sólo 2 variables (x_{md}, x_{sd}) nuestro espacio de decisión es un espacio bidimensional, por lo tanto, es posible graficarlo.

De manera general, decimos que los multiplicadores de lagrange nos indican 'como cambia la solución a medida que cambian las restricciones'.

3. Introducción a AMPL

3.1. Mathematical programming

- El termino 'programming' fue usado por primera vez en el año 1940 para describir un plan o secuencia de actividades con una gran organización.
- Los programadores determinaron que las actividades podían definirse como variables cuyo valor debía ser determinado.
- De esta forma, se pueden describir matemáticamente restricciones inherentes al problema como un set de ecuaciones o inecuaciones relacionadas a las variables.
- Una solución que cumple todas las restricciones es considerado como un plan aceptable.

La experiencia rápidamente demostró que es díficil modelar un problema solamente especificando restricciones. Con pocas restricciones, soluciones inferiores podrían satisfacer el problema, y en caso de muchas restricciones se vuelve complicado encontrar alguna solución.

Como respuesta a esta dificultad se planteo el uso de un objetivo aplicado a las variables del problema. De esta forma, mediante una función se podría determinar cuando una solución es mejor que otra en base a un criterio especifico, por ejemplo: minimizar costos o maximizar beneficios.

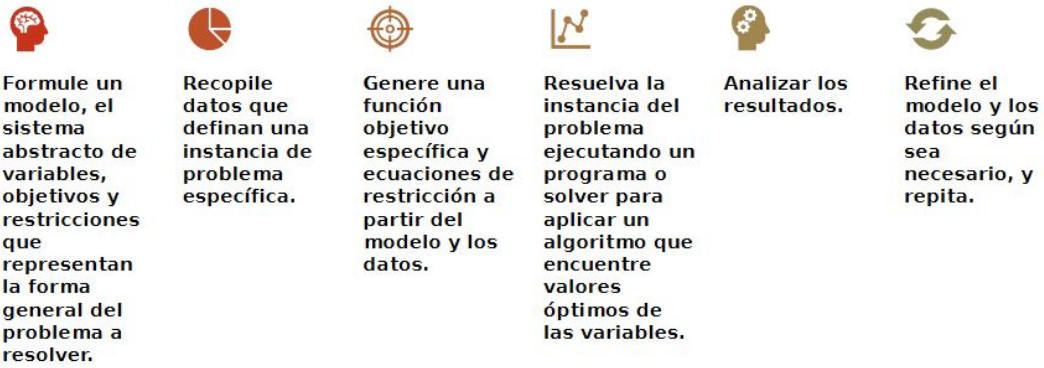
Así ya no es un problema el tamaño del espacio de soluciones factibles, pues solo se necesita encontrar una solución que minimice o maximice la función objetivo.

Existen diferentes tipos de mathematical programming los cuales reciben su nombre en base a la complejidad matemática de los problemas que modelan, por ejemplo:

- **Linear programming**: El objetivo es una función lineal y restricciones son ecuaciones e inecuaciones lineales.
- **Integer programming**: Las variables solo pueden tomar valores enteros.
- **Nonlinear programming**: El objetivo o al menos una de las restricciones es no lineal.

Existen más clasificaciones y cada una de ella tiene diferente complejidad matemática y computacional donde los problemas lineales son los menos complejos.

La programación matemática práctica rara vez es tan simple como ejecutar algún método algoritmico en una computadora e imprimir la solución óptima. La secuencia completa de eventos es más como:



Existen muchas diferencias entre la forma en que los modeladores humanos entienden un problema y la forma en que los algoritmos solver trabajan con él. La conversión de la 'forma del modelador' a la 'forma del algoritmo' es, en consecuencia, un procedimiento lento, costoso, a menudo propenso a errores, difícil de debuggear y mantener.

Para enfrentar este problema, se crearon lenguajes de modelado, los cuales están diseñados para expresar la forma del modelador de manera que pueda servir como entrada directa a un sistema informático. Luego, la traducción al algoritmo se puede realizar completamente por computadora, sin la etapa intermedia de la programación de la computadora.

3.2. Lenguaje de modelado algebraico

Como hay más de una forma que usan los modeladores para expresar programas matemáticos, hay más de un tipo de lenguaje de modelado. Un lenguaje de modelado algebraico es una variedad popular basada en el uso de la notación matemática tradicional para describir funciones objetivo y de restricción. Un lenguaje algebraico proporciona equivalente legibles por computadora de notaciones como:

$$x_i + y_i \sum_{j=1}^{n} a_{ij} x_{ij} \ , x_i > 0 \ and \ i \in S$$

Que sería familiar para cualquiera que haya estudiado algebra o cálculo. La familiaridad es una de las principales ventajas de los lenguajes de modelado algebraico; otro es su aplicabilidad a una variedad particularmente amplia de modelos de programación lineal, no lineal y entera.

3.3. Lenguaje de modelado AMPL

AMPL, un lenguaje de modelado algebraico para programación matemática; fue diseñado e implementado por los autores alrededor de 1985, y ha ido evolucionando desde entonces. AMPLes notable por la similitud de sus expresiones aritméticas con la notación algebraica habitual, y por la generalidad y el poder de su conjunto y expresiones de subíndice. AMPL también extiende la notación algebraica para expresar estructuras de programación matemática comunes, como restricciones de flujo de red y linealidad por partes (piecewise linear).

AMPL ofrece un entorno de comando interactivo para configurar y resolver problemas de programación matemática. Una interfaz flexible permite que varios solvers estén disponibles a la vez para que un usuario pueda cambiar entre solver y seleccionar opciones que puedan mejorar el rendimiento. Algunos solvers que se pueden integrar a AMPL son Knitro, Gurobi, CPLEX por mencionar algunos.

3.4. Maximizar ganancias - Minimizar costos

3.4.1. Problema de programación lineal de dos variables

Identificar la función objetivo y las restricciones del problema.

$$\text{máx}\, 25x_b + 30x_c$$

s.t

$$\frac{1}{200}x_b + \frac{1}{140}x_c \leq 40$$
$$0 \leq x_b \leq 6000$$
$$0 \leq x_b \leq 4000$$

Para modelar el problema propuesto creamos un archivo llamado *prod0.mod* en el cual especificaremos los comandos en AMPL que representen el problema propuesto.

```
var XB;
var XC;
maximize Profit: 25 * XB + 30 * XC;
subject to Time: (1/200) * XB + (1/140) * XC <= 40;
subject to B_limit: 0 <= XB <= 6000;
subject to C_limit: 0 <= XC <= 4000;
```

Como se puede apreciar, los comandos de AMPL que modelan el problema son bastante similares al modelo algebraico del mismo. La mayor diferencia se encuentra en la regularidad y formalidad de los comandos de AMPL lo cual facilita el procesamiento para el.

3.4.2. Elementos del modelado en AMPL

Podemos escribir una descripción compacta de la forma general del problema, que llamamos modelo, usando notación algebraica para el objetivo y las restricciones. Los componentes fundamentales para cualquier modelo son:

- Set o Conjuntos, como los productos
- Parámetros, como las tasas de producción y ganancias
- Variables, cuyos valores debe determinar el solver.

- Un objetivo, para maximizar o minimizar.
- Restricciones o limitaciones que la solución debe satisfacer.

Cada elemento del modelo algebraico tiene su equivalente en el modelo de AMPL

```
var XB;
var XC;
maximize Profit: 25 * XB + 30 * XC;
subject to Time: (1/200) * XB + (1/140) * XC <= 40;
subject to B_limit: 0 <= XB <= 6000;
subject to C_limit: 0 <= XC <= 4000;
```

- **Variables:** Son las variables de decision del problema y se definen mediante la palabra reservada 'var'.
- **Función objetivo:** El resultado de esta función es el valor que se desea optimizar. En AMPL podemos utilizar 'maximize' para problemas de maximización y 'minimize' para aquellos de minimización.
- **Restricciones**: Son las restricciones propais del problema y las definimos utilizando 'subject to' y un nombre propio definido a las restricción.
- **Operadores**: En AMPL, al igual que en cualquier lenguaje de programación se utilizan una variedad de operadores lógicos y matemáticos como $+$, $*$, $<$, $=$, $\leq$ entre otros.

```
ampl: model prod0.mod;

ampl: solve;
MINOS 5.5: optimal solution found.
2 iterations, objective 192000

ampl: display XB, XC;
XB = 6000
XC = 1400

ampl: quit;
```

Para probar el modelo creado accedemos a la consola de AMPL y ejecutamos los comandos necesarios, dentro de los más básicos podemos mencionar:

- **model**: Seguido de este comando se debe ingresar el nombre del modelo que queramos cargar
- **solve**: Ejecuta el modelo cargado y arroja como resultado el nombre del solver que fue utilizado para resolver el modelo, y si fue capaz de encontrar una solución óptima, seguido de la cantidad de iteraciones necesarias para llegar al óptimo y el valor de la función objetivo.
- **display**: Seguido de este comando ingresamos el nombre de las variables a las cuales queremos saber su valor final, separadas por una coma.
- **quit**: el comando finaliza AMPL.

3.5. Modelo de programación lineal

En el ejemplo dado se resolvió en AMPL un problema de programación lineal de 2 variables a modo de ejemplo básico, pero ¿qué sucede si se trata de resolver problemas más realistas y complejos con mayor cantidad de elementos como variables y restricciones? ¿Y si el problema está sujeto a cambios continuos en forma o en los valores de los datos?

Para enfrentar estas dificultades AMPL permite representar un programa lineal (LP) en dos partes, modelo matemático y data. Realizar esta separación es fundamental para describir LP complejos y facilitar su entendimiento y depuración.

Given: P, a set of products
a_j = tons per hour of product j, for each $j \in P$
b = hours available at the mill
c_j = profit per ton of product j, for each $j \in P$
u_j = maximum tons of product j, for each $j \in P$

Define variables: X_j = tons of product j to be made, for each $j \in P$

Maximize: $\sum_{j \in P} c_j X_j$

Subject to: $\sum_{j \in P} (1/a_j) X_j \leq b$

$0 \leq X_j \leq u_j$, for each $j \in P$

Este modelo describe una gran cantidad de problemas de optimización. Si se provee una data específica, entonces estamos refiriendonos a una instancia del modelo, donde cada data diferente se define una instancia.

```
set P;

param a {j in P};
param b;
param c {j in P};
param u {j in P};

var X {j in P};

maximize Total_Profit: sum {j in P} c[j] * X[j];

subject to Time: sum {j in P} (1/a[j]) * X[j] <= b;

subject to Limit {j in P}: 0 <= X[j] <= u[j];
```

En AMPL existen constructores para todos los componentes fundamentales de un modelo, lo cual nos permite traducir el modelo matemático a un lenguaje fácil de entender y procesar para un computador. Al observar, el código se aprecia que visualmente no dista mucho de la notación matemática original.

Si descomponemos el modelo anterior nos encontramos con la notación de AMPL de los componentes de un modelo, debemos considerar que los valores para el set y los parámetros son dados en un archivo separando que incluye solo la data del problema:

- La palabra reservada 'set' declara el nombre de un conjunto.

```
set P;
```

- 'param' declara un parámetro el cual puede ser un escalar o una colección de valores indexados por un set.

```
param P;
param a {j in P};
```

- 'var' declara una colección de variables, una por cada elemento de P. El solver se encarga de encontrar un valor a estas variables.

```
var X {j in P};
```

- El objetivo se declara con las palabras reservadas 'maximize' o 'minimize' acompañado de un nombre cualquiera.

```
maximize Total_Profit: sum{j in P}c[j]*x[j];
```

  Este caso el objetivo se define como una sumatoria de un producto dado que la precedencia del operador suma es menor que *

- Las restricciones son declaradas con las palabras reservadas 'subject to' seguidas por un nombre elegido por el usuario para dicha restricción

```
subject to Time: sum {j in P} (1/a[j]) * X[j] <= b;
subject to Limit: {j in P}: 0 <= X[j] <= u[j];
```

  La construcción {j en P} se llama expresión en indexación y se usa no solo para declarar parámetros y variables, sino en cualquier contexto en el que el modelo algebraica haga algo 'para caja j en P'.

```
set P := bands coils;

param:     a     c     u  :=
  bands   200   25   6000
  coils   140   30   4000 ;

param b := 40;
```

La figura anterior proporciona datos para el modelo de producción básico en esa forma. Una instrucción de conjunto proporciona los miembros

(band y coils) del conjunto P, y una tabla de parámetros proporciona los valores correspondientes para a, c y u. Una simple declaración 'param' da el valor de b.

3.5.1. Consideraciones

- Los parámetros, conjuntos y variables deben ser declarados antes de ser usados, sin embargo, estos pueden ingresarse en cualquier orden.
- Todas las sentencias finalizan con ';' y se pueden añadir espacios para mejorar el entendimiento del modelo
- Mayúsculas y minúsculas son consideradas diferentes al declarar alguna variable o nombre
- Los comentarios inician con '#' y terminan al final de la línea.
- Como en cualquier lenguaje de programación, el uso de nombres descriptivos, comentarios y formatos significativos ayuda a hacer que los modelos AMPL sean más legibles y comprensibles.

3.5.2. Modelo mejorado

```
set PROD;  # products

param rate {PROD} > 0;      # tons produced per hour
param avail >= 0;           # hours available in week

param profit {PROD};        # profit per ton
param market {PROD} >= 0;   # limit on tons sold in week

var Make {p in PROD} >= 0, <= market[p]; # tons produced

maximize Total_Profit: sum {p in PROD} profit[p] * Make[p];

               # Objective: total profits from all products

subject to Time: sum {p in PROD} (1/rate[p]) * Make[p] <= avail;

               # Constraint: total of hours used by all
               # products may not exceed hours available
```

Figure 1-4a: Steel production model (`steel.mod`).

```
set PROD := bands coils;

param:    rate  profit  market :=
  bands    200    25     6000
  coils    140    30     4000 ;

param avail := 40;
```

- Uso de nombres descriptivos
- Limites definidos juntos con la variable y no como restricción adicional.
- Uso de comentarios explicativos.

3.5.3. Prueba

```
ampl: model steel.mod;
ampl: data steel.dat;
ampl: solve;
MINOS 5.5: optimal solution found.
2 iterations, objective 192000

ampl: display Make;
Make [*] :=
bands  6000
coils  1400
;
```

- Se cargan los archivos que contienen el modelo y la data.
- Se usa el comando 'solve' para resolver el modelo utilizando algún solver
- Finalmente, se muestran los resultados obtenidos y los valores asignados a las variables que logran optimizar el objetivo.

3.5.4. Manejo de errores

AMPL notifica mediante consola los errores de programación presentes en el modelo, la data o los comandos ingresados señalando la línea en que se detecto el error y rodeando la posible causa del error con '>>> error <<<'. Sin embargo, no detecta errores en el planteamiento del modelo por lo que se considera una buena practica el modelar problemas incluyendo validaciones que permitan identificar si existe algún error de diseño o planteamiento.

3.5.5. Añadir limites inferiores

Para añadir un poco más de realismo al modelo se le pueden agregar límites inferiores a la cantidades que se deben producir, lo cual se traduce en valores mínimos iniciales para las variables de decisión. En este ejemplo, el límite inferior viene dato por 'commit' mientras que el superior lo define 'market'

```
set PROD;  # products

param rate {PROD} > 0;     # produced tons per hour
param avail >= 0;          # hours available in week
param profit {PROD};       # profit per ton

param commit {PROD} >= 0;  # lower limit on tons sold in week
param market {PROD} >= 0;  # upper limit on tons sold in week

var Make {p in PROD} >= commit[p], <= market[p]; # tons produced

maximize Total_Profit: sum {p in PROD} profit[p] * Make[p];
               # Objective: total profits from all products

subject to Time: sum {p in PROD} (1/rate[p]) * Make[p] <= avail;
               # Constraint: total of hours used by all
               # products may not exceed hours available
```

Figure 1-5a: Lower bounds on production (`steel3.mod`).

```
set PROD := bands coils plate;

param:    rate  profit  commit  market :=
  bands    200    25     1000    6000
  coils    140    30      500    4000
  plate    160    29      750    3500 ;

param avail := 40;
```

Figure 1-5b: Data for lower bounds on production (`steel3.dat`).

Se resuelve el modelo, y se muestra el valor de la función objetivo, junto con los valores de los límites inferiores, el valor óptimo y el limite superior para cada variable.

```
ampl: model steel3.mod; data steel3.dat; solve;
MINOS 5.5: optimal solution found.
2 iterations, objective 194828.5714

ampl: display commit, Make, market;
:     commit    Make    market    :=
bands   1000    6000      6000
coils    500     500      4000
plate    750   1028.57    3500
;
```

3.5.6. Uso de etapas

Siguiendo con la idea de generar un modelo genérico que aplique a una amplia variedad de problemas, se da el caso donde es necesario separar el problema en distintas etapas, donde se utilizan valores de data.

En este caso se divide el problema en 2 etapas, reheat y roll, cada una con sus propios valores de rate. A su vez, cada etapa tiene su propio limite establecido en el parámetro avail.

```
set PROD := bands coils plate;
set STAGE := reheat roll;

param rate:  reheat  roll :=
  bands        200    200
  coils        200    140
  plate        200    160 ;

param:    profit  commit  market :=
  bands     25     1000    6000
  coils     30      500    4000
  plate     29      750    3500 ;

param avail :=  reheat 35   roll   40 ;
```

El modelo modificado para trabajar etapas contempla la modificación de la restricción time, la cual ahora es una familia de restricciones asignando una a cada etapa del modelo.

```
set PROD;   # products
set STAGE;  # stages

param rate {PROD,STAGE} > 0; # tons per hour in each stage
param avail {STAGE} >= 0;    # hours available/week in each stage
param profit {PROD};         # profit per ton

param commit {PROD} >= 0;    # lower limit on tons sold in week
param market {PROD} >= 0;    # upper limit on tons sold in week

var Make {p in PROD} >= commit[p], <= market[p]; # tons produced

maximize Total_Profit: sum {p in PROD} profit[p] * Make[p];

              # Objective: total profits from all products

subject to Time {s in STAGE}:
   sum {p in PROD} (1/rate[p,s]) * Make[p] <= avail[s];

              # In each stage: total of hours used by all
              # products may not exceed hours available
```

El comando **reset** elimina los modelos anteriormente cargados y permite la carga de un nuevo modelo. Al resolver el problema con el modelo propuesto se pueden apreciar diferentes valores obtenidos por el solver, adicionales al valor objetivo.

```
ampl: reset;
ampl: model steel4.mod; data steel4.dat; solve;
MINOS 5.5: optimal solution found.
4 iterations, objective 190071.4286

ampl: display Make.lb, Make, Make.ub, Make.rc;
:     Make.lb    Make   Make.ub     Make.rc       :=
bands   1000    3357.14   6000     5.32907e-15
coils    500     500      4000     -1.85714
plate    750    3142.86   3500     3.55271e-15
;

ampl: display Time;
Time [*] :=
reheat  1800
  roll  3200
;
```

Con el comando **display** se muestran los límites inferiores, el valor óptimo, los limites superiores y el costo reducido para las variables de make, que tiene el mismo significado de los valores marginales con respecto a las restricciones.

```
ampl: reset;
ampl: model steel4.mod; data steel4.dat; solve;
MINOS 5.5: optimal solution found.
4 iterations, objective 190071.4286

ampl: display Make.lb, Make, Make.ub, Make.rc;
:     Make.lb    Make   Make.ub     Make.rc       :=
bands   1000    3357.14   6000     5.32907e-15
coils    500     500      4000    -1.85714
plate    750    3142.86   3500     3.55271e-15
;
ampl: display Time;
Time [*] :=
reheat  1800
  roll  3200
;
```

Al comparar esta sesion con ña anterior, vemos que la restricción adicional del tiempo de recalentamiento o 'reheat' reduce las ganancias en aproximadamente '$4750' y obliga a un cambio sustancial en la solución óptima: una producción mucho mayor de 'plate' y una menor producción de 'bands'. Además, la lógica subyacente al óptimo ya no es tan obvia. Es la dificultad de resolver LP's solo por razonamiento lógico lo que requiere sistemas basados en computadora como AMPL.

4. Introducción a JuMP

4.1. Lenguaje de modelado JuMP

Jump ('Julia for mathematical Programming') es un lenguaje de modelado de código libre embebido de Julia. Entre sus principales caracterísitcas están su facilidad de trasladar un problema matemático en uno computacional, y que permite utilizar solver externos (tanto open source como comerciales). De acuerdo a la documentación de JuMP, los solvers que éste incluyen son:

- Artelys
- Bonmin
- Cbc
- Clp
- Couenne
- CPLEX
- ECOS
- FICO Xpress
- GLPK
- Gurobi
- Ipopt
- MOSEK
- NLOPT
- SCS

Aquí se presenta JuMP usando algunos ejemplos y algunas capacidades elementales.

4.2. Problema de programación lineal de dos variables

$$\text{máx}\, 25x_b + 30x_c$$

s.t

$$\frac{1}{200}x_b + \frac{1}{140}x_c \leq 40$$
$$0 \leq x_b \leq 6000$$
$$0 \leq x_b \leq 4000$$

Para esto, decimos pues que $\text{máx}\, 25x_b + 30x_c$ es lo que denominaremos como función objetivo, la cantidad exacta que queremos calcular a partir de un conjunto de variables. Por consiguiente $\frac{1}{200}x_b + \frac{1}{140}x_c \leq 40$ es una restricción del problema, un set de condiciones que nos describe una característica propia que nuestra solución debe cumplir para ser considerada como factible. Y finalmente tanto $0 \leq x_b \leq 6000$ como $0 \leq x_b \leq 4000$ representan las correspondientes variables del problema junto a su respectivo dominio, el espacio general por el cual se moverá nuestras variables.

4.3. Modelo

Un modelo es un objeto que contiene variables, restricciones, opciones de solver, entre otras cosas. En JuMP, los modelos son creados usando la función **Model()**. Estos modelos pueden ser creados pasando por argumento el solver a usar como se muestra en la siguiente figura.

```
using JuMP
using GLPK
model = Model(GLPK.Optimizer)
```

4.4. Variables de decisión

Una variable es modelada usando el comando

```
@variable(nombre del modelo, nombre de la variable, tipo de variable)
```

Los limites pueden ser inferiores, superiores o ambos. Si el tipo de variable no es definida, por defecto se considerará como una variable del tipo real.

```
@variable(model, 0 <= xb <= 6000)
@variable(model, 0 <= xc <= 4000)
```

4.5. Restricciones

Las restricciones se modelan usando el siguiente comando

```
@constraint(nombre del modelo, nombre de la restriccion, restriccion)
```

No es obligatorio ingresar el nombre a una restricción, por lo que se puede utilizar el comando

```
@constraint(nombre del modelo, restriccion)
```

```
@constraint(model, con, (1/200)xb + (1/140)xc <= 40)
```

4.6. Función objetivo

La función objetivo se define con el comando

```
@objective(nombre del modelo, Min/Max, funcion a optimizar)
```

```
@objective(model, Max, 25xb + 30 * xc)
```

4.7. Resolver Modelo

Para resolver el modelo cargado, se llama a la siguiente función

```
@optimize!(nombre del modelo)
```

Además, utilizaremos el comando **show** (también se puede utilizar la función print), para imprimir en pantlla los valores obtenidos para las variables en la función objetivo, añadido a su respectivo valor.

```
optimize!(model)
@show value(xb);
@show value(xc);
@show objective_value(model);
```

```
value(xb) = 6000.0
value(xc) = 1400.0
objective_value(model) = 192000.0
```

5. Método de Nelder Mead

5.1. Historia

Después del desarrollo del algoritmo simplex en la década de los 40's, durante la segunda mitad del siglo pasado, una serie de métodos de programación matemática (o 'exactos') fueron propuestos con el fin de resolver, de manera eficiente, los problemas que aparecerían reportados en las revistas científicas de la época.

En particular, resolver problemas 'grandes' se volvió el foco principal de los investigadores, dado las dificultades que los métodos propuestos anteriormente tenían cuando se enfrentaban a este tipo de problemas.

Es fácil ver que esta 'necesidad' por resolver problemas cada vez más complejos fue de la mano con el desarrollo del hardware necesario para poder probar los algoritmos que se proponían

Muchos buenos algoritmos (desde el punto de vista teórico) quedaron olvidados por mucho tiempo dado que, cuando fueron propuestos, los recursos computacionales que requería su implementación no estaban disponibles.

A pesar de la proliferación de muchos algoritmos, no existe aquel que resuelva 'todos' los problemas de optimización que existen.

Según Nocedal & Wright (1999), '... la elección de un algoritmo u otro para resolver un problema dado es de **responsabilidad del usuario** y determinará que tan rápido o lento el problema será resultado'.

Es así como una larga lista de estrategias exactas fueron propuestas durante esta época:

- Line Search Method
- Gradient Descent Method
- Newton's Method
- Decomposition Methods (Dantzig-Wolfe, Benders, Column Generation).
- Primal-Dual Methods (Lagrange's Relaxation).
- Entre muchos otros.

5.2. Clasificación

Estos métodos pueden clasificarse de distintas maneras dependiendo de distintas características. Una forma de clasificarlos es dependiendo de si requiere o no el uso de derivadas. Es así como tenemos:

- Derivate-Free Methods (e.g. line search)
- First Order Methods (e.g. gradient descent)
- Second Order Methods (e.g. Newton's Methods)

5.2.1. Derivate Free Methods

Cuando hablamos de técnicas 'libres' de derivadas, lo primero que se viene a la mente es un conjunto de algoritmos (meta)-heurísticos tales como:

- Algoritmos evolutivos (Genéticos, culturales, etc)
- Algoritmos de Enjambre (PSO, Abejas, etc).
- Búsqueda Local (Tabu Search, VNS, Simulated Annealing, etc).

Sin embargo, existen algoritmos exactos muy eficientes que no requieren del uso de derivadas.

5.3. Nelder-Mead Method

Conocido también como *downhill simplex method*, es un método numérico para optimización. El algoritmo está basado en la idea de 'triangulos' presente en el algoritmo Simplex de Dantzig.

Fue propuesto por John Nelder y Roger Mead en 1965. Aunque el algoritmo tiene varias falencias (e.g. puede converger a puntos no estacionarios), varias variantes del algoritmo han sido propuestos para superar estos problemas.

Formalmente, el metodo usa el concepto de *simplex*, el cual es el polytope de $n + 1$ dimensiones con n igual al número de variables (e.g. para dos variables, se forma un triángulo). El método se aproxima a un óptimo local. Si el problema es **convexo**, entonces la solución encontrada es también el óptimo global.

- **INITIAL TRIANGLE BGW**

  Let $f(x, y)$ be the function that is to be minimized. To start, we are given three vertices of a triangle:

  $$V_k = (x_k, y_k) \ ; \ k = 1, 2, 3$$

  The function $f(x, y)$ is then evaluated at each of the three points:

  $$z_k = f(x, y) \ ; \ k = 1, 2, 3$$

The subscripts are the reordered so that $z_1 \leq z_2 \leq z_3$. We use the notation:

$$B = (x_1, y_1)$$

$$G = (x_2, y_2)$$

$$W = (x_3, y_3)$$

to help remember that B is the best vertex, G is good (next to best), and W is the worst vertex.

- **Midpoint of the Good Side**

  The construction process uses the midpoint of the line segment joining B and G. It's found by averaging the coordinate:

$$M = \frac{B+G}{2} = \left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right)$$

- **Reflection Using the point R**

  The function decreases as we move along the side of the triangle from W to B. and it decreases as we move along the side from W to G. Hence it's feasible that $f(x, y)$ take on smaller values at points that lie away from W on the opposite side of the line between B and G.

  We choose a test point R that is obtained by 'reflecting' the triangle through the side $\overline{BG}$. To determine R, we first find the midpoint M of the side $\overline{BG}$. then draw the line segment from W to M and call its lenth d. This last segment is extended a distance d through M to locate the point R. The vector formula for R is:

$$R = M + (M - W) = 2M - W$$

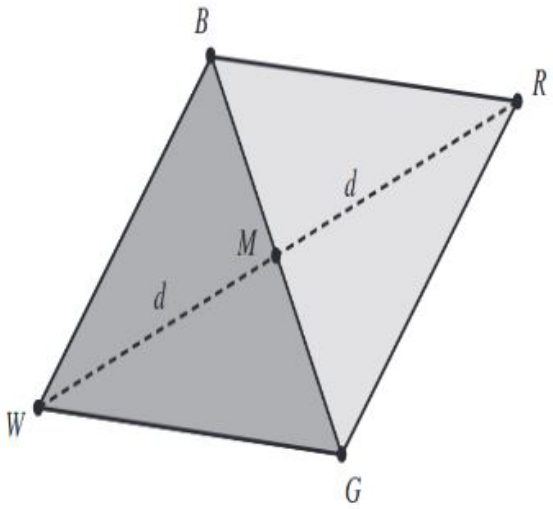


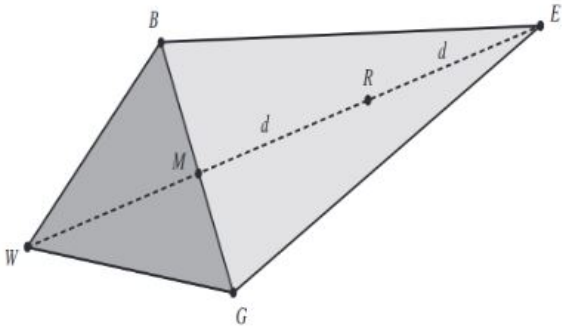
Figura 3: The triangle $\triangle BGW$ and midpoint M and reflected point R for the Nelde-Mead method

- **Expansion using the point E**

  If the function value at R is smaller than the function value at W, then we have moved in the correct direction toward the minimum. Perhaps the minimum is just a bit further that the point R. So, we extend the line segment through M and R to the point E. This forms and expanded triangle BGE.

  The point E is found by moving and additional distance d along the line joining M and R. If the function value at E is less than the function value at R, then we have found a better vertex than R. The vector formula for E is:

$$E = R + (R - M) = 2R - M$$



- **Contraction using the point C**

  If the function values to R and W are the same, another point must be tested. Perhaps the function is smaller at M, but we cannot replace W with M because we must have a triangle. Consider the two midpoint C_1 and C_2 of the line segments $\overline{WM}$ and $\overline{MR}$, respectively- The point with the smaller function value is called C, and the new triangle is BGC.

  The choice between C_1 and C_2 might seem inaproppiate for the two-dimensional case, but it is important in higher dimensions.

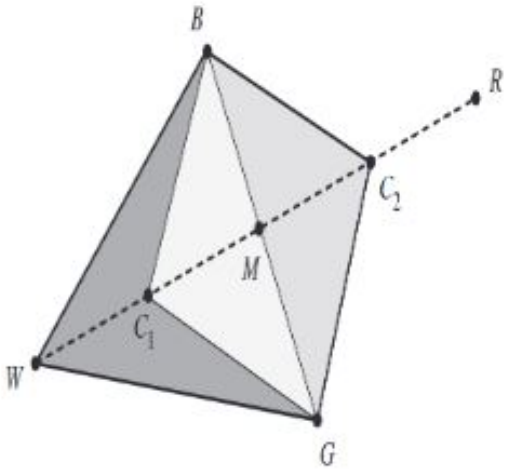


Figura 4: The contraction point C_1 or C_2 for Nelder-Mead method

- **Shrink toward B**

If the function value at C is not less than the value at W, the points G and W must be shrunk toward B. The point G is replaced with M, and W is replaced with S, wich is the midpoint of the line segment joining B with W.

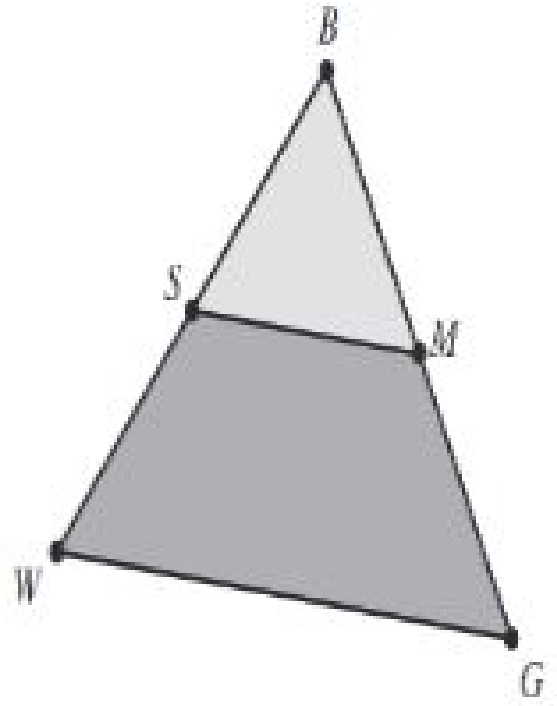


Figura 5: Shrinking the triangle toward B

5.3.1. Pseudocodigo

```
IF f(R) < f(G) THEN Perform Case (i) (either reflect or extend)
ELSE Perform Case (ii) (either contract or shrink)

BEGIN {Case (i)}
IF f(B) < f(R) THEN
    replace W with R
ELSE
    Compute E and f(E)
    IF f(E) < f(B) THEN
```

```
        replace W with E
    ELSE
        replace W with R
    ENDIF
ENDIF
END {Case (i)}
```

```
BEGIN {Case (ii)}
IF f(R) < f(W) THEN
    replace W with R
compute C = (W + M)/2
or C = (M + R)/2 and f(C)

IF f(C) < f(W) THEN
    replace W with C
ELSE
    compute S and f(S)
replace W with S
replace G with M
ENDIF
END{Case (ii)}
```

5.3.2. Ejemplo

Tomando como ejemplo la siguiente función objetivo:

$$\text{mín}\, f(x, y) = x^2 - 4x + y^2 - y - xy$$

Tomemos como solución inicial los siguientes 3 vertices

$$V_1 = (0, 0),\ V_2 = (1{,}2, 0{,}0),\ V_3 = (0{,}0, 0{,}8)$$

El valor de la función objetivo en esos puntos es:

- $f(0, 0) = 0{,}0$
- $f(1{,}2, 0{,}0) = -3{,}36$
- $f(0{,}0, 0{,}8) = -0{,}16$

Luego,

- $W = (0, 0)$

- $B = (1{,}2, 0{,}0)$
- $G = (0{,}0, 0{,}8)$

Ahora, debemos reemplazar el vertice W. Para ello calculamos M y R

$$M = \frac{B+G}{2} = \left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right) = (0{,}6, 0{,}4)$$

$$R = 2M - W = (1{,}2, 0{,}8)$$

El valor de la función objetivo en R es

$$f(1{,}2, 0{,}8) = -4{,}8 \leq f(G)$$

entonces, estamos en el caso (i). Dado que $f(R) \leq f(B)$, sabemos que nos movemos en la dirección correcta y, por lo tanto, el vértice E debe ser calculado:

$$E = 2R - M = 2(1{,}2, 0{,}8) - (0{,}6, 0{,}4) = (1{,}8, 1{,}2)$$

Donde $f(E) = -5{,}88 \leq f(B)$. El nuevo triángulo tendrá vertices.

$$V_1 = (1{,}8, 1{,}2),\ V_2 = (1{,}2, 0{,}0),\ V_3 = (0{,}0, 0{,}8)$$

Sin embargo, esto puede continuar iterativamente.

6. Métodos en programación matemática

6.1. Condiciones de optimalidad

Consideremos una función $f : \mathbb{R}^n \to \mathbb{R}$ con $x \in \mathbb{R}^n$ diferenciable. Note que el problema no tiene restriccciones.

- **Condiciones necesarias**

  Los óptimos locales tienen $\bigtriangledown f(x^*) = 0$ y $\bigtriangledown^2 f(x^*)$ positiva semi-definida.

- **Condiciones suficientes**

  Cualquier punto x^* para el cual $\bigtriangledown f(x^*) = 0$ y $\bigtriangledown^2 f(x^*)$ definida positiva es un óptimo local estricto.

$\bigtriangledown f(x^*)$ el vector gradiente de f que está dado por las derivadas parciales de cada variable independiente.

$$\bigtriangledown f(x) \equiv g(x) \equiv \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n} \right]^T$$

Mientras el gradiente de $f : \mathbb{R}^n \to \mathbb{R}$ corresponde a un vector de largo n, la segunda derivada de f es una matriz de n^2 derivadas parciales (Hessiano). Si las derivadas parciales $\frac{\partial f}{\partial x_i}, \frac{\partial f}{\partial x_j}$ son continuas y f es *single-valued*, entonces $\frac{\partial^2 f}{\partial x_i \partial x_j}$ existe y $\frac{\partial^2 f}{\partial x_j \partial x_i} = \frac{\partial^2 f}{\partial x_i \partial x_j}$

$$\bigtriangledown^2 f(x) \equiv H(x) \equiv \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial^2 x_n} \end{bmatrix}$$

El cual contiene $\frac{n(n+1)}{2}$ elementos independientes. Si f es doblemente diferenciable, diremos que f es convexo (en el espacio X), si su Hessiano $\mathbb{H}f$ es positivo semi-definido.

- La matriz $H \in \mathbb{R}^{nxn}$ es **definida positiva** si $p^T Hp > 0$ para todos los vectores $p \in \mathbb{R}^n$, $p \neq 0$
- La matriz $H \in \mathbb{R}^{nxn}$ es **semi-definida positiva** si $p^T Hp \geq 0$ para todos los vectores $p \in \mathbb{R}^n$, $p \neq 0$
- La matriz $H \in \mathbb{R}^{nxn}$ es **indefinida** si existe $p, q \in \mathbb{R}^n$, tal que $p^T Hp > 0$ y $q^T Hq < 0$

- La matriz $H \in \mathbb{R}^{nxn}$ es **indefinida negativa** si $p^T Hp < 0$ para todos los vectores $p \in \mathbb{R}^n$, $p \neq 0$

6.1.1. Ejemplo

Condiciones necesarias vs Suficientes

$$f(x) = 1{,}5x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + 0{,}5x_1^4$$

Encuentre todos los puntos estacionarios ($\bigtriangledown f(x) = 0$) y clasifiquelos.

Desarrollo

- Se igualan las derivadas parciales a cero.
- Se resuelven las ecuaciones anteriores y se obtienen las coordenadas de los puntos críticos.
- Se construye la matriz hessiana (segundas derivadas parciales).

$$f(x) = 1{,}5x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + 0{,}5x_1^4$$

$$\frac{\partial f}{\partial x_1} = 3_x 1 + 2x_2 + 6x_1^2 + 2x_1^3$$

$$\frac{\partial f}{\partial x_2} = 2x_2 - 2x_1$$

$$\bigtriangledown f(x) \equiv H(x) \equiv \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial^2 x_2} \end{bmatrix} \equiv \begin{bmatrix} 6x_1 + 12x_1 + 3 & -2 \\ -2 & 2 \end{bmatrix}$$

En el caso particular de $f(x) = z$ (superficie en $\mathbb{R}^3$), con segundas derivadas continuas, se pueden estudiar los puntos críticos evaluando la matríz hessiana en ellos, y luego utilizando el criterio de determinación de extremos. Si (a, b) es un punto crítido de f:

$$\frac{\partial f}{\partial x_1}(a, b) = 0$$

$$\frac{\partial f}{\partial x_2}(a, b) = 0$$

Definición

- $|H(a, b)| > 0$ y $h_{11}(a, b) < 0$, decimos que f alcanza su máximo en (a,b)

- $|H(a,b)| > 0$ y $h_{11}(a,b) > 0$, decimos que f alcanza su mínimo en (a,b)
- $|H(a,b)| < 0$, decimos que $f(a,b)$ es un punto silla
- $|H(a,b)| = 0$, el criterio no concluye resultado alguno

Por otro lado, para 3 variables

$$\bigtriangledown^2 f(x) \equiv H(x) \equiv \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial^2 x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_3} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial^2 x_3} \end{bmatrix}$$

Se deben calcular todos los menores positivos $|H_1|$, $|H_2|$ y $|H_3|$

Definición

- Si $|H_1| > 0, |H_2| > 0$ y $|H_3| > 0$ entonces hay un mínimo.
- Si $|H_1| < 0, |H_2| > 0$ y $|H_3| < 0$ entonces hay un máximo.
- Si $|H_1| \neq 0, |H_2| \neq 0$ y $|H_3| \neq 0$ entonces no es máximo o mínimo, sino que es un punto silla.
- Sino, el criterio no decide

Cuando agregamos restricciones a nuestro problema, por ejemplo, $X \subset \mathbb{R}^n$, pueden pasar dos situaciones.

- El problema se vuelve más sencillo, al tener un espacio de búsqueda acotado
- El problema se vuelve más complejo, al tener un espacio de búsqueda que es díficil de recorrer (non-smooth, non-convex, etc).

Todo dependerá de cómo está definido el espacio de búsqueda (o espacio factible) X.

6.2. Constrained Optimisation

6.2.1. Definiciones

Sea $f : \mathbb{R}^n \to \mathbb{R}$ la función de minimizar, donde $X \subset \mathbb{R}^n$ and $X \neq \emptyset$

- Un punto x se llama punto factible si $x \in X$

- Si $\hat{x} \in X$ y $f(\hat{x}) \leq f(x)$ para cada $x \in X$ con $x \neq \hat{x}$, entonces $\hat{x}$ es una solución óptima u **óptimo global**.
- El conjunto de todos los óptimos globales $\hat{x} \in X$, denotado por $\hat{X}$, es el conjunto de soluciones optimas alternativas. Es claro que $\hat{X} \subseteq X$.
- Si $\hat{x} \in X$ y existe un vecindario $N_\epsilon(\hat{x}) \subset X, \epsilon > 0$, alrededor de $\hat{x}$, tal que $f(\hat{x}) \leq f(x)$ para todo $x \in N_\epsilon(\hat{x})$, entonces $\hat{x}$ es un optimo global. Si $N_\epsilon(\hat{x}) = X$ entonces $\hat{x}$ es un optimo global.
- Si $\hat{x} \in X$ y $f(\hat{x}) < f(x)$ para todo $x \in N_\epsilon(\hat{x})$, $x \neq \hat{x}$, entonces $\hat{x}$ es un optimo local estricto.

Un problema convexo será aquel donde la función f y el espacio factible X son convexos.

Theorem 6.1 *Óptimos locales en problemas convexos son óptimos globales*
Supongamos que $X \neq \emptyset$ y que $f : X \to \mathbb{R}$ es convexa en X. Si, $\hat{x}$ es un óptimo local, entonces es un óptimo global. Además, si $\hat{x}$ es un óptimo local estricto o bien f es estrictamente convexa, entonces $\hat{x}$ es un óptimo global único.

Por ejemplo, si f tiene multiples mínimos locales en el espacio factible $X \subset \mathbb{R}^n$, encontrar el óptimo global puede volverse una tarea muy complicada. Sin embargo, si X es convexo, y f es convexo en X, entonces decimos que cualquier mínimo local x^* es también, un óptimo global.

Diremos que $X \subseteq \mathbb{R}^n$ es un espacio convexo si:

$$\forall \lambda \in [0,1], x, y \in X \Longrightarrow \lambda x + (1-\lambda)y \in X$$

Diremos que también la función f es una función convexa si:

$$\lambda f(x) + (1-\lambda) f(y) \geq f(\lambda f(x) + (1-\lambda) f(y))$$

Diremos que la función f es concava si $-f$ es convexa.

Una condición suficiente para conseguir un óptimo local será

Theorem 6.2 *Supongamos que $f : \mathbb{R}^n \to \mathbb{R}$ es doblemente diferenciable en $\hat{x}$. Si $\hat{x}$ es un óptimo local, entonces*

$$\bigtriangledown f(\hat{x}) = 0$$

y $H(\hat{x})$ es positivo semi-definido.

Theorem 6.3 *Supongamos que $f : \mathbb{R}^n \to \mathbb{R}$ es doblemente diferenciable en $\hat{x}$. Si $\bigtriangledown f(\hat{x}) = 0$ y $H(\hat{x})$ es positivo definido, entonces $\hat{x}$ es un mínimo local estricto.*

6.3. Ejemplo (extraído de Nocedal & Wright)

El desafío será entonces que dado un punto $x \in X \subset \mathbb{R}^n$, podamos determinar si x^* es o no un óptimo local o global.

$$\text{mín } f(x) = x_1 + x_2$$

s.t.

$$x_1^2 + x_2^2 = 2$$

En el ejemplo, es simple ver que hay un movimiento que nos permite ir minimizando el valor de f mientras que cumple con la restricción c_1.

También,nos damos cuenta que para la solución óptima $x^* = (-1, -1)$, el vector normal de c_1, $\bigtriangledown c_1$ es paralelo al vector gradiente $\bigtriangledown f(x^*)$. Luego, existe un escalar λ_1^* tal que:

$$\bigtriangledown f(x^*) = \lambda_1^* \bigtriangledown c_1(x^*)$$

en el ejemplo $\lambda_1^* = -1/2$.

Del mismo modo, podemos decir que se requiere de un movimiento pequeño s que satisfaga la restricciónc_1, esto es $c_1(x + s) = 0$. A través de una aproximación de series de Taylor, decimos:

$$0 = c_1(x + s) \approx c_1(x) + \bigtriangledown c_1(x)^T s = \bigtriangledown c_1(x)^T s$$

$$\bigtriangledown c_1(x)^T s = 0$$

Igualmente, este movimiento pequeño debe permitir una reducción en nuestra función objetivo f. Luego, decimos:

$$0 > f(x + s) - f(x) \approx \bigtriangledown f(x)^T s$$

$$\bigtriangledown f(x)^T s < 0$$

Por otra parte, la no existencia de un vector d de descenso en el punto x^*, implica que no existe un s que satisfaga las condiciones en x^* y, por ende, decimos que x^* es un óptimo (mínimo) local.

Es importante notar que estas condiciones son necesarias pero no suficientes. Por ejemplo, el punto $x = (1, 1)$ cumple con $\bigtriangledown f(x) = \lambda_1 \bigtriangledown c_1(x)$ para $\lambda_1 = \frac{1}{2}$. Sin embargo, $x = (1, 1)$ claramente no es un óptimo local del problema (de hecho, lo máximiza).

6.3.1. Lagrange

Supongamos que sólo tenemos restricciones del tipo

$$g(x) = c$$

donde $g : \mathbb{R}^n \to \mathbb{R}$ y c es una constante. Luego, el óptimo de la función estará en algún punto donde las curvas de nivel de f sean tangentes a g.

Dado que el espacio de búsqueda está restringido, la dirección de descenso definida antes ya no (necesariamente) es válida. Por lo mismo, debemos encontrar otras técnicas que nos aseguren llegar al óptimo de la función teniendo en cuenta las restricciones de igualdad al problema.

Usando el gradiente:

- Sabemos que los puntos críticos estarán en aquellos puntos donde las líneas de contorno de f y g son tangentes.
- También sabemos que, cuando dos vectores tienen la misma dirección, podemos encontrar una constnate λ tal que la multiplicación de uno de los vectores por λ tendrá como resultado el otro vector.

$$\bigtriangledown f(x) = \lambda \bigtriangledown g(x)$$

Siguiendo con el ejemplo anterior, calculemos $\bigtriangledown f(x)$ y $\bigtriangledown g(x)$

$$\bigtriangledown f(x) = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\bigtriangledown g(x) = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix}$$

y por último

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix}$$

tres variables, tres ecuaciones, por lo tanto lo podemos resolver.

Del ejemplo anterior, podemos identificar dos condiciones generales:

- La restricción se debe cumplir ($g(x) = c$).
- Condición de la tangente ($\bigtriangledown f(x) = \lambda g(x)$)

La condición de tangente se puede separar en dos componentes:

$$\frac{\partial f}{\partial x_1} = \lambda \frac{\partial g}{\partial x_1}$$

y

$$\frac{\partial f}{\partial x_2} = \lambda \frac{\partial g}{\partial x_2}$$

Lagrange resumió estas condiciones en una simple ecuación que llamamos **Lagrangeano** o función lagrangeana L.

$$L(x, \lambda) = f(x) - \lambda(g(x) - c)$$

De la función Lagrangeana, podemos obtener lo siguiente:

$$\frac{\partial L}{\partial \lambda} = 0 - (g(x) - c)$$

y luego, diremos que la restricción del problema original puede ser escrita como :

$$\frac{\partial L}{\partial \lambda} : -g(x) - c = 0$$

Es decir, la función lagrangeana cumple con la primera condición general. De la función lagrangeana, podemos obtener lo siguiente:

$$\frac{\partial L}{\partial x_1} = 0$$

$$\frac{\partial f}{\partial x_1} - \lambda \frac{\partial g}{\partial x_1} = 0$$

$$\frac{\partial f}{\partial x_1} = \lambda \frac{\partial g}{\partial x_1}$$

De la misma forma:

$$\frac{\partial L}{\partial x_2} = 0$$

$$\frac{\partial f}{\partial x_2} - \lambda \frac{\partial g}{\partial x_2} = 0$$

Luego, estas condiciones son equivalentes a decir:

$$\bigtriangledown f(x) = \lambda \bigtriangledown g(x)$$

Por lo tanto, podemos encontrar x y λ haciendo

$$\bigtriangledown L = 0$$

El largo del vector λ corresponderá al número de restricciones del problema. Cada elemento del vector es llamado **multiplicador de Lagrange**. El procedimiento general es el siguiente:

- Fijar el gradiente de L igual a 0 ($\bigtriangledown L(x, \lambda) = 0$)
- Evaluar los puntos críticos encontrados en f (sin considerar λ)

7. Relajación Lagrangeana.

Hasta ahora, hemos aprendido a optimizar problemas sin restricciones y problemas con restricciones de igualdad (funciones afines). Sin embargo, como vimos anteriormente, muchas veces las restricciones que debemos enfrentar son inequidades, las cuales requieren de un tratamiento distinto. Por ejemplo:

Para revisar cómo resolver este tipo de problemas de optimización, usaremos el siguiente ejemplo:

$$\text{mín } f(x) = x^2$$

$$\text{s.t } 1 \leq x \leq 2$$

Es claro que podemos transformar la restricción del problema en dos restricciones independientes

$$x - 1 \geq 0$$

$$2 - x \geq 0$$

Como sabemos, el método de Lagrange requiere que todas las restricciones sean restricciones de igualdad. Luego, necesitamos construir una restricción de igualdad que sea equivalente a la original, como lo hacemos? (TIP: Qué hacemos con ese tipo de restricciones en el tableau del método Simplex?).

- Para poder transformar las restricciones originales del problema en restricciones de igualdad, requerimos del uso de variables auxiliares (también conocidas como variables de holgura o slack variables).
- Luego, dado que tenemos dos restricciones de desigualdad, tendremos que incorporar 2 variables de slack, s_1 y s_2. Naturalmente que usaremos una variable por cada restricción.
- Entonces, nuestras nuevas restricciones de igualdad quedan como sigue:

$$g_1(x, s_1, s_2) = x - 1 - s_1^2 = 0 \tag{1}$$

$$g_2(x, s_1, s_2) = 2 - x - s_2^2 = 0 \tag{2}$$

¿Por qué las variables de slack entran como s_1^2 y s_2^2 y no simplemente como s_1 y s_2? Entonces nuestro problema será:

$$\text{mín } f(x) = x^2$$

$$s.t\ g_1(x, s_1, s_2) = x - 1 - s_1^2 = 0$$

$$g_2(x, s_1, s_2) = 2 - x - s_2^2 = 0$$

Este tipo de problemas los sabemos resolver!.

7.1. Método de Lagrange con restricciones de desigualdad

Entonces, siguiendo con el método de Lagrange tal como lo conocemos, lo primero es hacer:

$$\nabla f(x) = \lambda_1 \nabla g_1(x, s_1, s_2) + \lambda_2 \nabla g_2(x, s_1, s_2)$$

El gradiente operará sobre las tres variables $x,\ s_1,\ y\ s_2$,.
$\nabla = < \partial_x, \partial_{s_1}, \partial_{s_2} >$
En efecto:

$$\nabla = < \partial_x, \partial_{s_1}, \partial_{s_2} >$$

Luego:

$$\nabla f = < 2x, 0, 0 >$$

$$\nabla g_1 = < 1, -2s_1, 0 >$$

$$\nabla g_2 = < -1, 0, -2s_2 >$$

Es decir, ∇L es:

$$\frac{\partial L}{\partial x} = 2x - \lambda_1 + \lambda_2$$

$$\frac{\partial L}{\partial s_1} = -2s_1\lambda_1$$

$$\frac{\partial L}{\partial s_2} = -2s_2\lambda_2$$

Dado que ∇L, tenemos que:

$$2x = \lambda_1 - \lambda_2$$

$$-2s_1\lambda_1 = 0$$

$$-2s_2\lambda_2 = 0$$

Una vez calculado ∇L, agregamos a nuestro sistema de ecuaciones las restricciones que calculamos anteriormente.

$$g_1(x, s_1, s_2) = x - 1 - s_1^2 = 0$$

$$g_2(x, s_1, s_2) = 2 - x - s_2^2 = 0$$
$$\nabla f = \ 2x - \lambda_1 + \lambda_2 = 0$$
$$\nabla g_1 = \ -2s_1\lambda_1 = 0$$
$$\nabla g_2 = \ -2s_2\lambda_2 = 0$$

Para cada una de las restricciones anteriores, podemos determinar si ellas son *Activas* o *Inactivas*.

Decimos que una restricción estará activa, si esta en su valor límite (la expresión del lazo izquierdo toma el mismo valor que el del lado derecho).Decimos que una restricción es inactiva si no está activa.

Es importante notar que si las restricciones están activos entonces su variable de slack será cero. Del ejemplo, si:

$$x - 1 - s_1^2$$

es activa, entonces

$$x_1 = 0 \ \wedge \ s = 0$$

Es decir, la restricción de desigualdad actúa como una restricción de igualdad. Más aún, en este caso decimos que el multiplicador de Lagrange, λ_1 es nonzero.

En caso de que las restricciones están inactivas, entonces no nos preocupa demasiado. Su variable de slack será nonzero y el multiplicador de Lagrange será 0.

De

$$\nabla g_1 = \ -2s_1\lambda_1 = 0$$

Tenemos que

1. $s_1 \neq 0$ y $\lambda_1 = 0$, entonces ....

   $$x = 1 + s_1^2$$

   (de la primera ecuación)... Ven algún problema?

   $$1 + s_1^2 > 1 \ \forall s_1 \neq 0$$

   ...o

2. $s_1 = 0$ y $x = 1$

De

$$\nabla g_2 = \ -2s_2\lambda_2 = 0$$

Tenemos que:

1. $s_2 \neq 0$ y $\lambda_2 = 0$, entonces...

   $$x = 2 - s_2^2$$

   (de la segunda ecuación)... Ven algún problema?

   $$2 - s_2^2 < 2 \ \forall s_2 \neq 0$$

   ...o

2. $s_2 = 0$ y $x = 2$

Luego, de acuerdo con las ecuaciones del sistema, tenemos que solo uno de estos casos es posible

- Caso I: $x = 1$
- Caso II: $x = 2$
- Caso III: $1 < x < 2$ y $\lambda_1 = 0 \ \wedge \ \lambda_2 = 0$

Concentremonos en el caso III, ven algo que no les haga sentido?. Si evaluamos el caso III, tenemos que la ecuación $2x - \lambda_1 + \lambda_2 = 0$ obliga que

$$x = \frac{(\lambda_1 - \lambda_2)}{2}$$

Pero en el caso III, también decimos que $\lambda_1 = 0 \ \wedge \ \lambda_2 = 0$, luego, $x = 0$, violando la condición de que $1 < x < 2$.

Luego, una vez que ya hemos chequeado las distintas posibilidades, evaluamos los puntos que son candidatos a ser puntos extremos. $f(1)$ y $f(2)$. ¿Cuál de los dos minimiza la función?

8. Dualidad Lagraniana

8.1. Saddle Point and Duality

En el método de Lagrange, aprendimos a resolver con restricciones (de igualdad y/o desigualdad) y aprendimos a interpretar el valor de los multiplicadores de Lagrange. Ahora veremos que otras propiedades tiene este resultado.

Lo primero entonces es que decimos que si (x^*, λ^*), con $\lambda^* \geq 0$, es un punto silla de la función Lagrangeana L asociaco al problema primal, entonces x^* es una solución del problema primal. Este teorema nos será de utilidad en algunos momentos más.

8.2. Duality

Sea

$$h(\lambda) = \text{mín}_x L(x, \lambda)$$

La *función dual*. Note que:

- Una solució óptima $x^*(\lambda)$ no necesariamente cumple con $x \in X$.
- El óptimo puede incluso no existir para todo λ

Si ahora definimos el set:

$$D = \{\lambda \mid h(\lambda) \; \exists \text{ and } \lambda \; \geq 0\}$$

Entonces tenemos el siguiente problema dual

$$\text{máx}_{\lambda \in D} h(\lambda)$$

Que es equivalente a

$$\text{máx}_{\lambda \in D}(\text{mín}_x L(x, \lambda))$$

Decimos que el punto (x^*, λ^*), con $\lambda^* \geq 0$, es un punto silla de la función lagrangeana L asociada al problema primal, si y solo si:

- x^* es una solución del problema primal
- λ^* es una solución del problema dual
- $f(x^*) = h(\lambda^*)$

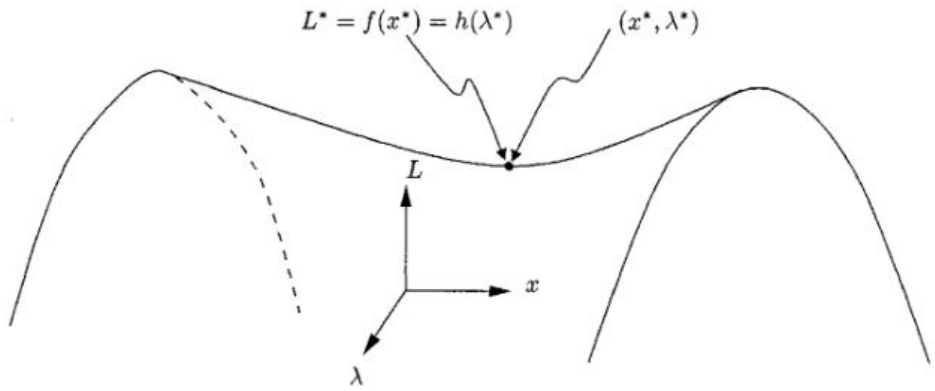


Figura 6: Schematic representation of saddle point solution to PP

Lo anterior tiene una importancia practica. Podríamos encontrar una solución al problema primal sin necesidad de resolverlo directamente. Si seguimos el siguiente procedimiento:

1. Resuelva el problema dual para obtener $\lambda^* \geq 0$
2. Una vez conocido λ^*, resuelva el problema $\text{mín}_x L(x, \lambda^*)$ para obtener $x^* = x^*(\lambda^*)$
3. Chequear si el punto (x^*, λ^*) cumple con las condiciones de optimalidad (KKT conditions).

8.2.1. Ejemplo

Veamos como se puede aplicar lo anterior en un ejemplo concreto.

$$\text{mín } f(x) = x_1^2 + 2x_2^2$$

$$\text{s.t. } g(x_1, x_2) = x_1 + x_2 \geq 1$$

Para este problema, el Lagrangeano es:

$$L(x, \lambda) = x_1^2 + 2x_2^2 + \lambda(1 - x_1 - x_2)$$

Para un λ dado, las condiciones necesarias para encontrar $\text{mín}_x L(x, \lambda)$ en $x^*(\lambda)$ son:

$$\frac{\partial L}{\partial x_1} = 2x_1 - \lambda = 0 \Longrightarrow x_1^*(\lambda) = \frac{\lambda}{2}$$

$$\frac{\partial L}{\partial x_2} = 4x_2 - \lambda = 0 \Longrightarrow x_2^*(\lambda) = \frac{\lambda}{4}$$

Note que $x^*(\lambda)$ es un mínimo ya que H_L (Heassiano de la función Lagrangeana):

$$H_L = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

Es positiva-definida. Luego, lo que hacemos es reemplazar x_1^* y x_2^* en L.

$$L(x^*(\lambda), \lambda) = h(\lambda) = \left(\frac{\lambda}{2}\right)^2 + 2\left(\frac{\lambda}{4}\right)^2 + \lambda\left(1 - \frac{\lambda}{2} - \frac{\lambda}{4}\right)$$

Lo que nos entrega que nuestra función dual es:

$$h(\lambda) = -\frac{3}{8}\lambda^2 + \lambda$$

Una vez con nuestra función dual, resolvemos el problema dual correspondiente $\max_\lambda h(\lambda)$. Sabemos que la condición necesaria para un máximo es:

$$\frac{\partial h}{\partial \lambda} = 0$$

Es decir:

$$-\frac{3\lambda}{4} + 1 = 0 \Longrightarrow \lambda^* = \frac{4}{3} > 0$$

Note que, dado $\frac{\partial^2 h}{\partial \lambda^2} < 0$, $\lambda^* = \frac{4}{3}$ es un máximo del problema dual.

Ahora, sólo nos queda reemplazar λ^* en $x^*(\lambda)$. Con eso obtenemos:

$$x_1^* = \frac{2}{3}x_2^* = \frac{1}{3} \Longrightarrow f(x_1^*, x_2^*) = \frac{2}{3} = h(\lambda^*)$$

Finalmente, dado que $\lambda = \frac{4}{3} \geq 0$ y las condiciones de optimalidad en (x^*, λ^*) se cumplen, entonces x^* es el mínimo del problema primal.

9. Optimización Combinatorial

9.1. Optimización Continua vs Optimización Discreta

En muchos problemas de optimización, las variables sólo tienen sentido si éstas toman un valor entero. Por ejemplo, si queremos optimizar el número de trabajadores en una planta, no tendría sentido hablar de valores que no fueran entero: ¿Cómo implementaríamos 7,5 trabajadores por turno?

Para ello, lo que usualmente se hace es reemplazar las variables continuas $x \in \mathbb{R}^n$ propias de los problemas de optimización (no-) lineales, por variables enteras (o binarias) del tipo $x \in \mathbb{Z}^m$ ($x \in \{0, 1\}$ para el caso binario).

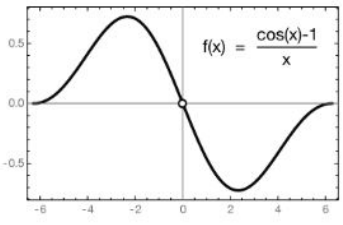
Cuando todas las variables de decisión son de tipo entero, entonces decimos que estamos frente a un problema de programación o *Problema Combinatorial*.

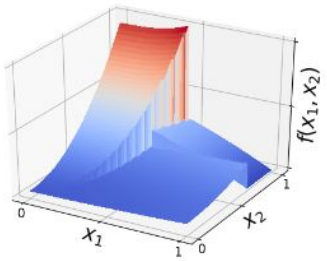
El nombre 'combinatorial' surge de la naturaleza 'finita' de este tipo de problemas, en contraste con la naturaleza 'infinita' de los problemas continuos.

Lo anterior se refiere al número de puntos factibles que podemos encontrar en el espacio de cada problema. Mientras que en los problemas continuos el número de vectores factibles es infinito, en el caso de los problemas combinatoriales, el número de vectores factibles es finito (aunque usualmente muy grande). A pesar de la diferencia de 'tamaño' en el espacio factible, es normal que los problemas de optimización continuos sean mucho más fáciles de resolver que los problemas combinatoriales.

Algunas razones para lo anterior son:

- Normalmente, se consideran funciones 'suaves' que permiten explotar la información de la función objetivo y las restricciones en un punto particular x para deducir lo que sucede en los puntos alrededor de dicho punto x.





A diferencia de lo que pasa en optimización continua, en optimización discreta el comportamiento de la función objetivo y de las restricciones puede variar significativamente en la meddia que nos movemos de un punto factible hacia el otro, incluso si esos dos puntos están 'cerca' el uno del otro, según alguna medida de distancia.

9.1.1. Ejemplo - CFLP

Un problema ampliamente estudiado en la literatura es el **Facility Location Problem**. En particular, en este documento nos enfocaremos en la versión 'capacitada' del problema (**Capacitated Facility Location Problem**), es decir, la versión del FLP que considera restricciones de capacidad.

El CFLP puede ser descrito informalmente como sigue: Dado un conjunto de potenciales centros de distribución cuyo costo de instalación/operación es conocido y unaa red de clientes cuyas demandas y distancias a cada uno de los potenciales centros también son conocidos, se busca determinar:

1. Qué centros de distribución deberían ser abiertos
2. La asignación de cada cliente respecto de un centro abierto

Naturalmente, el problema debe asegurar que:

1. La capacidad del centro no sea superada (es decir que la demanda total de clientes abastecidos por ese centro no debe ser superior a la capacidad de dicho centro).
2. Qué los clientes ....

Formalizando un poco más: Sea

- J el conjunto de todos los potenciales centros de la red
- I el conjunto de clientes de la red.
- d_i de la demanda del cliente i
- f_j el costo de instalación del centro j
- c_{ij} el costo de abastecer al cliente i desde el centro j
- I^j_{cap} Capacidad del centro j

A partir de acá, como quedaría el modelo? Cuáles serían sus variables de decisión? En efecto:

- $x_j \in \{0,1\}$, tal que si $x_j = 0$ el centro j no está seleccionado y $x_j = 1$ el centro si fue seleccionado.
- $y_{ij} \in \{0,1\}$ tal que $y_{ij} = 0$ si el cliente i no está asignado al centro j y $y_{ij} = 1$ si el cliente i está asignado al centro j

De esta forma, el modelo del (sigle source) CFLP será:

$$\text{mín}\, CFLP(x,y) = \sum_{j \in J}(f_j * x_j) + \sum_{i \in I}\sum_{j \in J}(c_{ij} * y_{ij})$$

s.t.

$$\sum_{j \in J} y_{ij} = 1 \ \forall i \in I$$

$$\sum_{i \in I} d_i * y_{ij} \leq I^j_{cap} \ \forall j \in J$$

$$y_{ij} \geq x_j \ \forall j \in J, \ i \in I$$

$$x, y \in \{0,1\}$$

9.2. Branch & Bound

Uno de los métodos exactos más usados para la resolución de problemas IP es **Branch & bound**. Un problema etero puro (IP) es sólo un caso particular de un problema entero mixto (MILP) donde el número de variables continuas es igual a 0.

Sea:

$$MILP(x,y) = max\{cx + hy : (x,y) \in S\}$$

Donde:

$$S := \{(x,y) \in \mathbb{Z}^n_+ \times \mathbb{R}^p_+ : Ax + Gy \leq b\}$$

Sea (x^*, y^*) una solución óptima al problema MILP y z^* el calor de MILP evaluado en (x^*, y^*). Ahora hacemos que P_0 sea la relajación lineal de S (es decir, que $S \subseteq P_0$) y diremos que (x^0, y^0) es la solución de dicho LP, que con z_0 el valor del LP en (x^0, y^0).

Dado que $S \subseteq P_0$, sabemos que $z^* \leq z_0$ (recuerden que estamos **MAXIMIZANDO**). Más aún, si $(x^0, y^0) \in S$ (es decir, si x^0 es entero), entonces $z^* = z_0$. Esto implica que el MILP está solucionado. A continuación, damos una vista general del algoritmo base de B&B.

Lo primero que hacemos es elegir un índice $1 \leq j \leq n$ tal que x_j^0 es no entero. Para no complejizar más la notación diremos que $f := x_j^0$ y que

$$S_1 := S \cap \{(x, y) : x_j \leq \lfloor f \rfloor\}$$

y

$$S_1 := S \cap \{(x, y) : x_j \leq \lceil f \rceil\}$$

S_1 y S_2 serán nuestras particiones. Considere ahora los siguientes MILP, por cada participación

$$MILP_1(x, y) = max\{cx + hy : (x, y) \in S_1\}$$

y

$$MILP_2(x, y) = max\{cx + hy : (x, y) \in S : 2\}$$

La solución óptima del MILP original saldrá de las soluciones óptimas de $MILP_1$ y $MILP_2$. Ahora, denotemos por P_1 y P_2 las relajaciones lineales de S_1 y S_2.

$$P_1 := P_0 \cap \{(x, y) : x_j \leq \lfloor f \rfloor\}$$

y

$$P_2 := P_0 \cap \{(x, y) : x_j \leq \lfloor f \rfloor\}$$

Y considere los problemas lineales

$$LP_1(x, y) = max\{cx + hy : (x, y) \in P_1\}$$

y

$$LP_2(x, y) = max\{cx + hy : (x, y) \in P_2\}$$

A partir de estas premisas, tenemos que:

1. Si alguno de los LP_i es infactible (i.e $P_i = \emptyset$), entonces $MILP_i$ también es infactible ya que $S_i \subseteq P_i$. Por lo mismo, el $MILP_i$ no necesita seguir siendo considerado. En este caso, decimos que el problema es *podado por infactibilidad*.

2. Sea (x^i, y^i) la solución óptima de LP_i y z_i su valor objetivo, $i = 1, 2$. Entonces:

   a) Si $x^i \in \mathbb{Z}_+^n$, entonces (x^i, y^i) es la solución óptima de $MILP_i$ y una solución factible de $MILP$. Dado que $MILP_i$ ya fue que $S_i \subseteq S$, sabemos que $z_i \leq z^*$, es decir que z_i es un *lower bound* del $MILP$.

b) Si $x^i \notin Z_+^n$ y z_i es menor o igual que el mejor lower bound conocido de MILP, entonces S_i no puede contener una mejor solución para $MILP$ y decimos que el problema se poda por bound.

c) Si $x^i \notin Z_+^n$ y z_i es mayor que el mejor lower bound conocido del $MILP$, entonces S_i podría aún contener una solución óptima del $MILP$. Luego, sea $x_j^i \notin \mathbb{Z}_+$ un componente del vector x^i. Sea $f' := x_j^i$. Definimos los sets:

$$S_{i1} := S_i \cap \{(x, y) : x_j' \leq \lfloor f' \rfloor\}$$

y

$$S_{i2} := S_i \cap \{(x, y) : x_j' \leq \lfloor f' \rfloor\}$$

y se repite el proceso anterior.

9.2.1. Ejemplo

$$\text{máx}\, MILP(x_1, x_2) = 5{,}5x_1 + 2{,}1x_2$$

$$s.t \;\; -x_1 + x_2 \leq 2$$

$$8x_1 + 2x_2 \leq 17$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \;\; \text{integer}$$

Raminicando por el lado de la variable x_1

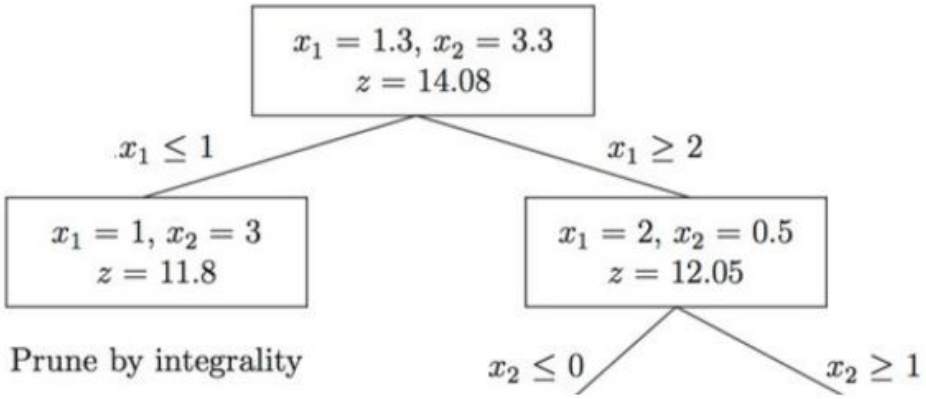


Figura 7: Branch & Bound - Nivel 1

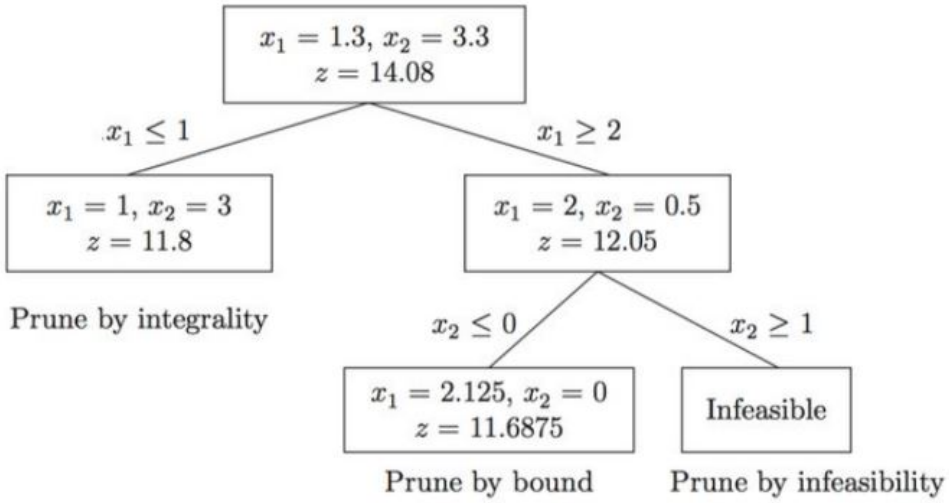


Figura 8: Branch & Bound - Nivel 2