Nuevas Tendencias en Optimización

Universidad Tecnológica Nacional
Facultad Regional San Rafael
Grupo de Sistemas Complejos

Prof. Dr. Ricardo R. Palma <rpalma@uncu.edu.ar>
Instituto de Ingeniería Industrial
Universidad Nacional de Cuyo

May 29, 2019

Poner orden cuando los datos ... están desorganizados

Una de las cosas notables que el advenimiento de la Industria 4.0 ha tenido en los últimos años, es el impacto en la formación de los ingenieros industriales. Muchos de los empleadores "tradicionales" de los egresados de estas carreras se están preguntado por qué no los encuentran y la respuesta es que existen otros nuevos centros de industria que capturan a los egresados con más y mejores posibilidades de trabajo en este nuevo paradigma.

Los tradicionales titanes de la industria metalmecánica y siderúrgica como Tenaris, Pascarmona, y tantos otros están siendo lenta, pero inexorablemente reemplazados por Mercado Libre, Pedidos Ya, Eventbrite o Facebook. En una reunión de los directores de carrera que se desarrolló en Buenos Aires junto a interesados en las TIC's organizada por la UBA y el ITBA se les preguntó a estas empresas ¿Por qué contratan a un Ingeniero Industrial en lugar de un Ingeniero de Software? y la respuesta fue unánime.

"Allí donde los programadores, ingenieros de software y administradores de bases de datos parecen tener un diálogo que no logra entenderse en el resto de los procesos de la empresa, el único que puede poner orden en estos datos desorganizados es el Ingeniero Industrial."

Una cosa notable en las carreras de II del ITBA y la UBA es que desde primer año se ha comenzado a enseñar algoritmia y programación en Pythom o R. Esto no quiere decir que se ha incorporado una nueva cátedra, sino que desde Análisis Matemático I en todas y cada una de las cátedras hay que resolver los problemas de antes pero con el uso de las herramientas modernas. Si se observa el Libro Rojo de la CONFEDI y las competencias que se piden para el II se puede ver claramente que esta competencia ha sido olvidada, pero estas universidades han incorporado junto a la Gestión de la Innovación como "metacompetencias".

Para regiones como la nuestra, en las que los sistemas regionales de innovación, pagan la demora que la academia impone para desarrollar competencias o carreras nuevas como la Mecatrónica, la Producmática, la Productrónica o la Infranómica, es que se desarrolla este material y el curso asociado.

Mucho de este trabajo está inspirado en la Conferencia de Planes de Estudios que a nivel Mundial se realizó en Viena en 2017 y que finalizó en una publicación que se ha transformado en mi motivación (por los aportes que ha sumado a mi visión) y mi preocupación al mismo tiempo (por ver cuanto nos falta en la formación de grado y posgrado en nuestro país)

Con la humilde esperanza se que podamos recoger frutos de este esfuerzo y encender la motivación que esto colegas de todo el mundo han inspirado en mi los invito a recorrer este material y recibir sus críticas y ayudas en la mejora del mismo.

Lectura Obligatoria

Industrial Engineering in the Industry 4.0 Era Selected papers from the Global Joint Conference on Industrial Engineering and Its Application Areas, GJCIE 2017, July 20–21, Vienna, Austria Editors: Calisir, Fethi, Camgoz Akdag, Hatice (Eds.) ISBN 978-3-319-71225-3

Pescando la Información en mares tormentosos

Las compañías que integran sus sistemas locales con servicios en la nube híbrida pueden tener que lidiar con información duplicada, registros de clientes desprotegidos y datos incoherentes. Sin un plan orientado, la consecuencia es una compleja integración y datos desorganizados del negocio. La información crítica termina propagándose entre diferentes sistemas, infraestructura virtual y servicios en la nube. Sin embargo, hay estrategias que pueden ayudarlo a evitar la propagación de estos datos. Sin dudas Windows es el sistema operativo más ampliamente utilizado en el mundo, y su suite de ofimática (Word Excel Power Point) es la herramienta que hoy no podría prescindir ningún II en su escritorio.

Hace unos cinco años en el Simposio de Ingeniería Industrial que organiza la SADIO, hubo una conferencia sobre el uso de aplicaciones en la nube. En particular se habló el uso del procesador de texto en la nube y como escribiríamos nuestros papers en la nube. El descreimiento fue inconmensurable. Todos los asistentes decían que no sería posible tener los datos de un documento en la nube. El ancho de banda disponible en ese momento (aún dial-up) no era seguro y menos aún tener el procesador de texto en la nube. Pues bien, ¿en dónde escribiste tu último contrato ?. Si lo has escrito en Ofice 365, lamento informarte que has usado aplicaciones en la nube.

¿Que pasaría si tu sistema operativo viviese en la nube?

Del mismo modo que en el congreso de la JAIIO inquietó a los asistentes, esta pregunta inquieta a cualquiera. ¿Podría el Windows 11 o 12 no ser suministrado con un medio de instalación? , ¿Es posible que mi máquina arranque un sistema operativo que no está siquiera en mi disco? . La respuesta ya no está en potencial, sino es presente del indicativo. SI Windows al igual que Amazon o Oracle o Google ya tienen sus sistemas operativos en la nube. Para Microsoft esto se llama AZURE.

Ahora, ¿ Que forma de explotación está pensando la empresa de Redmon para este sistema operativo? La respuesta es simple y justifica este curso. Se llama R-Server.

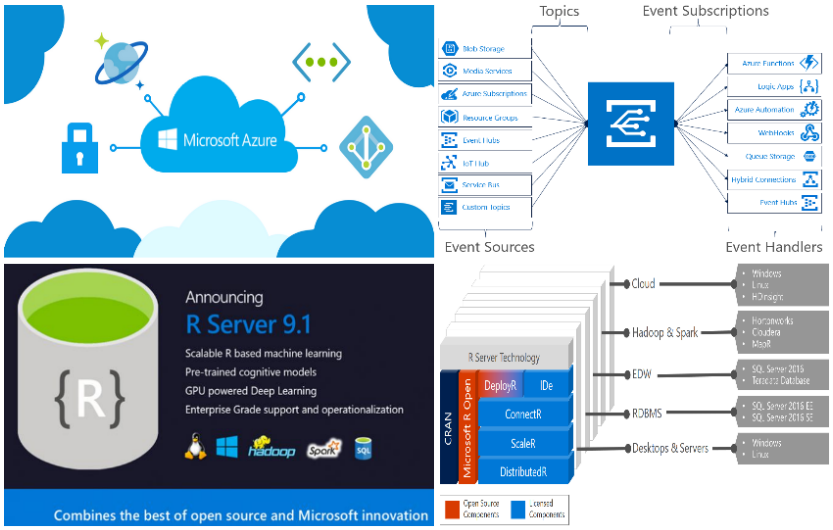


Figure 1: Suite Azure - R-Server

Tecnologías Emergentes

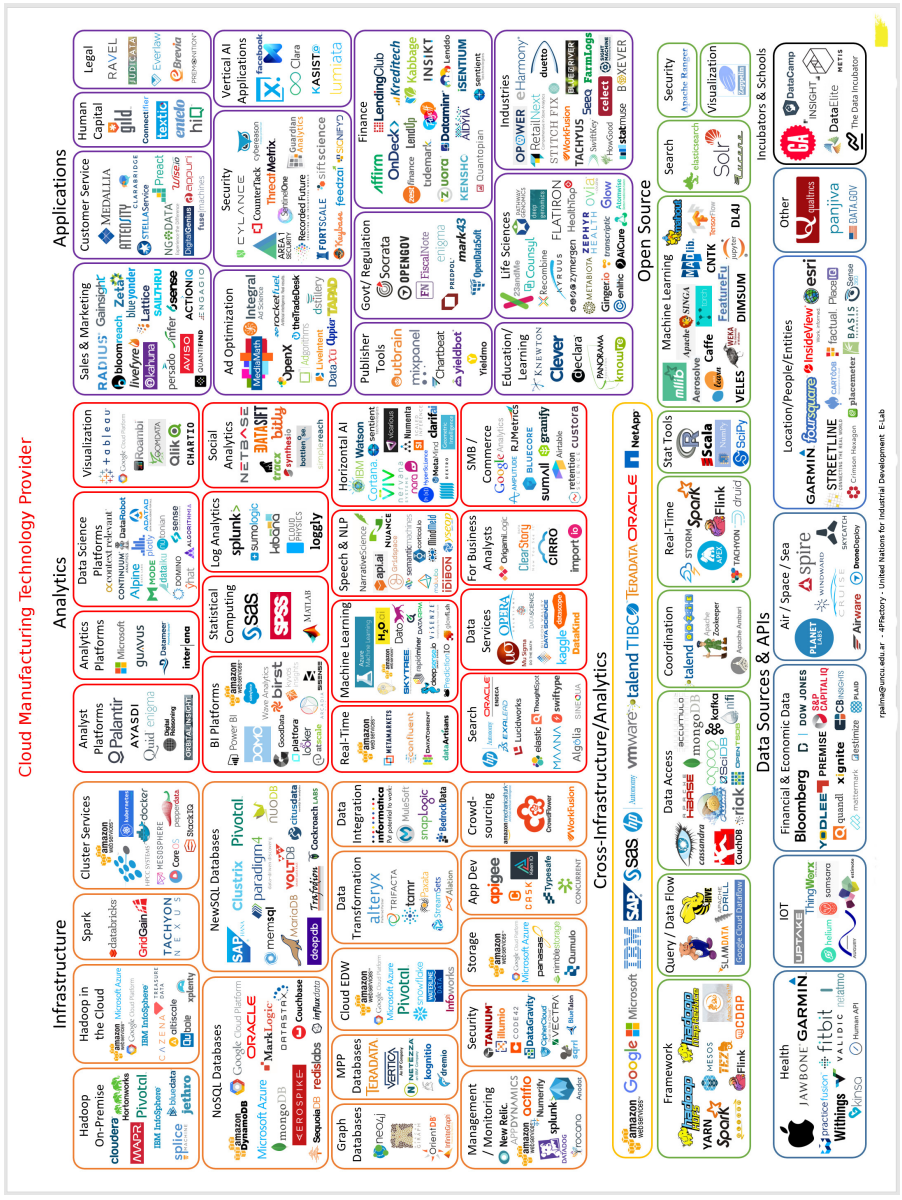


Figure 2: Las Tecnologias Emergentes

Impedir los datos desorganizados es "enfoque sistémico"

Organizar y controlar datos exige recursos significativos y orientados junto con un conjunto de habilidades que muchas organizaciones simplemente no tienen. Las herramientas para lidiar con sus datos pueden ser variadas y confusas a la vez. Cada plataforma local puede tener su propio conjunto de herramientas, mientras que los proveedores de servicios en la nube a menudo proporcionan un conjunto de funciones personalizado creado por el proveedor. Esto reduce la visibilidad del estado de los datos de su compañía. La adopción de la infraestructura de nube híbrida adecuada puede marcar la diferencia. Casi dos tercios de las compañías ya han adoptado el camino híbrido, mientras que otro 18 % implementa una estrategia de nube híbrida sin advertirlo, según la encuesta Estado de la nube híbrida 2017 (State of the Hybrid Cloud 2017) de Microsoft.1

La recuperación del orden a partir de datos caóticos comienza con los siguientes pasos:

1. Crear una estrategia para las peores situaciones
2. Proporcionar inteligencia empresarial con confianza a través de un motor de análisis comprobado
3. Permitir el análisis, el aprendizaje profundo y las aplicaciones inteligentes en datos locales y en la nube con plantillas comunes y compatibilidad con el lenguaje R reutilizable
4. Centrarse en la coherencia de los datos para obtener beneficios de seguridad Continúe leyendo para explorar la aplicación de los pasos anteriores en su entorno híbrido.

¿Qué es un problema de optimización?

Existen cientos de casos en los que el uso de las herramientas corrientes defraudan al experto al tratar de optimizar algo.

Lo más decepcionante es cuando luego de mucho esfuerzo en aprender y conseguir la información para el modelo todo falla, y lo peor es que no sabemos por que. Aún peor es encontrarnos que luego de fallar un núcleo de nuestro procesador se quemó.

R-CRAN tiene recursos de sobra para hacer cosas como el balance de carga en los núcleos.

```
>
> # library(multicore)
> # load the package
> # x1=1:5;x2=5:10
> # create 2 objects
> # p1=parallel(factorial(x1)) # run in parallel
```

```
> # p2=parallel(factorial(x2)) # run in parallel
> # collect(list(p1,p2))
> # collect results
>
> # $‘8995‘ [1]  1  2  6 24 120
> # $‘8996‘ [1]  120  720  5040  40320  362880 3628800
>
```

Dentro de la investigación operativa, las técnicas de optimizan se enfocan en determinar la política a seguir para maximizar o minimizar la respuesta del sistema. Dicha respuesta, en general, es un indicador del tipo “Costo”, “Producción”, “Ganancia”, etc., la cual es una función de la política seleccionada. Dicha respuesta se denomina objetivo, y la función asociada se llama función objetivo.

Pero, ¿qué entendemos cómo política? Una política es un determinado conjunto de valores que toman los factores que podemos controlar a fin de regular el rendimiento del sistema. Es decir, son las variables independientes de la función que la respuesta del sistema.

Por ejemplo, supongamos que deseamos definir un mix de cantidad de operarios y horas trabajadas para producir lo máximo posible en una planta de trabajos por lotes.

En este ejemplo, nuestro objetivo es la producción de la planta, y las variables de decisión son la cantidad de operarios y la cantidad de horas trabajadas. Otros factores afectan a la producción, como la productividad de los operarios, pero los mismos no son controlables por el tomado de decisiones. Este último tipo de factores se denominan parámetros.

En general, el quid de la cuestión en los problemas de optimización radica en que estamos limitados en nuestro poder de decisión. Por ejemplo, podemos tener un tope máximo a la cantidad de horas trabajada, al igual que un rango de cantidad de operarios entre los que nos podemos manejar.

Estas limitaciones, llamadas restricciones, reducen la cantidad de alternativas posibles, definiendo un espacio acotado de soluciones factibles ,y complicando, de paso, la resolución del problema. Observen que acabamos de hablar de soluciones factibles.

Y es que, en optimización, cualquier vector con componentes apareadas a las variables independientes que satisfaga las restricciones, es una solución factible, es decir, una política que es posible de implementar en el sistema. Dentro de las soluciones factibles, pueden existir una o más soluciones óptimas, es decir, aquellas que, además de cumplir con todas las restricciones, maximizan o minimizan, según sea el problema a resolver el valor de la función objetivo.

Resumiendo, un problema de optimización está compuesto de los siguientes elementos:

1. Un conjunto de restricciones
2. Un conjunto de soluciones factibles, el cual contiene todas las posibles combinaciones de valores de variables independientes que satisfacen las

restricciones anteriores.

3. Una función objetivo, que vincula las soluciones factibles con la performance del sistema.

Modelado de sistemas Vs. Resolución de modelos

Dentro de la práctica de la optimización, existen dos ramas relacionadas pero diferenciadas entre sí. Por un lado, nos encontramos con el problema de cómo modelar adecuadamente el sistema bajo estudio, y por otro como resolver el modelo. En general, la rama matemática de la IO se ocupa de buscar mejores métodos de solución, mientras que la rama ingenieril analiza formas de modelar sistemas reales.

Un ingeniero, en su actividad normal, se ocupa en general de modelar los sistemas, seleccionar un método de resolución y dejar que la computadora haga el resto. Sin embargo, es conveniente conocer el funcionamiento de cada metodología, a fin de determinar cuál es la más adecuada para cada situación, y modelar el problema de una forma amigable a dicho método.

Primeros pasos en R

R se puede usar como una calculadora avanzada en el escritorio de windows linux o android.

R se puede usar como una calculadora avanzada en el escritorio de windows / android.

```
> 3+5

[1] 8

> a<-3
> b<-5
> a+b

[1] 8

> sin(pi/2)

[1] 1

> sqrt(9)

[1] 3
```

Filosofía Matricial

- Estructura de Datos - Variables Indexadas - Definición de Funciones

Estructura de Datos

Variables como a <-3 pueden ser valores numéricos, pero R los interpreta como matrices. a en este caso es una matriz de 1x1 con el valor guardado 3 . Se emplea la flecha y no el signo igual para asignar a efectos de no confundir con la operación de comparación que sí usa =.

Existen tres estructuras matriciales y son objetos distintos

* vectores (matriz de una fila o columna) * listas (matriz con caracteres alfabéticos) * data-frame (matriz con elementos numéricos y alfabéticos)

Vectores

Son las estructuras de datos más simples y están obligadas a contener datos de la misma especie.

```
> car_name <-c("Honda","BMW","Ferrari")
> car_color =c("Black","Blue","Red")
> car_cc =c(2000,3400,4000)
> car_name

[1] "Honda"   "BMW"     "Ferrari"
```

Listas

Son más versátiles que los vectores, pero pueden tener elementos numéricos y alfabéticos que provienen de vectores disimiles. Piense en ellos como vector de vectores.

```
> cars <-list(name =c("Honda","BMW","Ferrari"),
+ color =c("Black","Blue","Red"),
+ cc =c(2000,3400,4000))
> cars

$name
[1] "Honda"   "BMW"     "Ferrari"

$color
[1] "Black" "Blue"  "Red"

$cc
[1] 2000 3400 4000
```

Cada operación queda registrada en el historial en pantalla.

Matrices

Son estructuras como las listas, pero limitadas sólo a datos numéricos. Pueden venir de aparatos que emiten constantemente valores (caja registradora) o fuentes

de web (Servicio Meteorológico), incluso pueden venir del teclado usando la función scan()

```
> mdat <-matrix(c(1,2,3, 11,12,13), nrow =2, ncol =3, byrow =TRUE,
+ dimnames =list(c("Fila1", "Fila2"),
+ c("Col1", "Col2", "Client3")))
> mdat

      Col1 Col2 Client3
Fila1    1    2       3
Fila2   11   12      13
```

DataFrame

Los dataframe extienden las matrices con la capacidad adicional de contener tipos de datos heterogéneos. En un dataframe, puede almacenar variables de caracteres, numéricas y de factores en diferentes columnas del mismo df. En casi todas las tareas de análisis de datos, con filas y columnas de datos, df se presenta como una opción natural para almacenar los datos y maneja mucho mejor la memoria y recursos del hardware

```
> L3 <-LETTERS[1:3]
> fac <-sample(L3, 10, replace =TRUE)
> df <-data.frame(x =1, y =1:10, fac = fac)
```

DataFrame 2

```
> class(df$x)

[1] "numeric"

> class(df$y)

[1] "integer"

> class(df$fac)

[1] "factor"
```

Variables Indexadas o Subseting

R tiene uno de los operadores de subconjunto más avanzados, potentes y rápidos en comparación con cualquier otro lenguaje de programación. Es poderoso hasta el punto de que, salvo en algunos casos que discutiremos en la siguiente sección, no existe una construcción de bucle for o while sino que se trabaja todo maticialmente, aunque R explícitamente proporciona estas palabras reservadas si es necesario.

Subseting 2

Aunque es muy poderoso, sintácticamente podría llegar a ser una pesadilla o podría aparecer un error grave si no se presta atención cuidadosa al colocar el número requerido de paréntesis, corchetes y comas. Los operadores [, [[y $ se usan para subconjuntos, dependiendo de qué estructura de datos contiene los datos. También es posible combinar el subconjunto con la asignación para realizar una función realmente complicada con muy pocas líneas de código.

Subseting en Vectores

Para los vectores, la subconjunto podría hacerse haciendo referencia al índice respectivo de los elementos almacenados en un vector. Por ejemplo, car name [c (1,2)] devolverá los elementos almacenados en el índice 1 y 2, y car name [-2] devolverá todos los elementos excepto el segundo. También es posible utilizar operadores binarios para indicar al vector que recupere o no un elemento.

```
> carname <-c ("Honda", "BMW", "Ferrari")
> #Selecciona el 1 ° y el 2 ° elemento del vector
> carname [c (1,2)]

[1] "Honda" "BMW"

>
```

Subseting Vectores 2

```
> #Seleciona todo, salve el 2do elemento
> carname[-2]

[1] "Honda"   "Ferrari"
```

Otra forma de seleccionar el 2do elemento

```
> carname[c(FALSE,TRUE,FALSE)]

[1] "BMW"
```

Subseting en Listas

En listas es similar al subseting en un vector; sin embargo, dado que una lista es una colección de muchos vectores, se debe usar corchetes dobles para recuperar un elemento de la lista. Por ejemplo, cars [2] recupera el segundo vector completo de la lista y $cars[[c(2,1)]]$ recupera el primer elemento del segundo vector.

```
> cars <-list(name =c("Honda","BMW","Ferrari"),
+ color =c("Black","Blue","Red"),
+ cc =c(2000,3400,4000))
> cars[2]$color
```

```
[1] "Black" "Blue"  "Red"
```

```
> cars[[c(2,1)]]
```

```
[1] "Black"
```

Subseting en Matrices

Las matrices tienen un subconjunto similar como vectores. Sin embargo, en lugar de especificar un índice para recuperar los datos, necesitamos dos índices aquí: uno que indique la fila y el otro para la columna.

Por ejemplo, mdat [1: 2,] recupera todas las columnas de las primeras dos filas, mientras que *mdat*[1 : 2,"*C*.1"] recupera las primeras dos filas y C.1 column.mn.

Subseting en Matrices 2

```
> mdat <-matrix(c(1,2,3, 11,12,13), nrow =2, ncol =3,
+ byrow =TRUE, dimnames =list(c("F.1", "F.2"),
+ c("C.1", "C.2", "C.3")))
```

Subseting en Matrices 3

```
> mdat[1:2,] #Select first two rows and all columns
```

```
    C.1 C.2 C.3
F.1   1   2   3
F.2  11  12  13
```

```
> mean(mdat[1,])
```

```
[1] 2
```

```
> mdat[2,3]
```

```
[1] 13
```

Sumario o información sumaria

```
> summary(cars)
```

```
      Length Class  Mode
name  3      -none- character
color 3      -none- character
cc    3      -none- numeric
```

Cuidado ! , existe un dataset llamado cars. Si definimos una variable con el nombre de un dataset podemos tener problemas de confusión.

Cargar Paquetes

Como se ha señalado existen más de veinte mil paquetes oficiales a la fecha y cada paquete suele venir con un set de datos de prueba que sirven para ver como trabaja el modelo.

ejecute el comando *data*() para ver que tiene disponible ejecute *data*(*package* = "*datasets*") para ver el contenido del datos del paquete datasets

- *tienes que instalar antes el paquete con:
- *install.packages("dataset")
- *y cargarlo con
- *library(dataset)

Ver paquetes instalados

```
> head(installed.packages())
```

```
         Package    LibPath                                                Version
abind    "abind"    "/home/rpalma/R/x86_64-pc-linux-gnu-library/3.4" "1.4-5"
acepack  "acepack"  "/home/rpalma/R/x86_64-pc-linux-gnu-library/3.4" "1.4.1"
adabag   "adabag"   "/home/rpalma/R/x86_64-pc-linux-gnu-library/3.4" "4.2"
alabama  "alabama"  "/home/rpalma/R/x86_64-pc-linux-gnu-library/3.4" "2015.3-1"
alluvial "alluvial" "/home/rpalma/R/x86_64-pc-linux-gnu-library/3.4" "0.1-2"
askpass  "askpass"  "/home/rpalma/R/x86_64-pc-linux-gnu-library/3.4" "1.1"
         Priority Depends                              Imports
abind    NA       "R (>= 1.5.0)"                       "methods, utils"
acepack  NA       NA                                   NA
adabag   NA       "rpart, caret, foreach, doParallel" NA
alabama  NA       "R (>= 2.10.1), numDeriv"            NA
alluvial NA       NA                                   NA
askpass  NA       NA                                   "sys (>= 2.1)"
         LinkingTo Suggests
abind    NA        NA
acepack  NA        "testthat"
adabag   NA        "mlbench"
alabama  NA        NA
alluvial NA        "devtools, testthat, reshape2, knitr, rmarkdown, dplyr"
askpass  NA        "testthat"
         Enhances License              License_is_FOSS License_restricts_use
abind    NA       "LGPL (>= 2)"        NA              NA
acepack  NA       "MIT + file LICENSE" NA              NA
adabag   NA       "GPL (>= 2)"         NA              NA
alabama  NA       "GPL (>= 2)"         NA              NA
alluvial NA       "MIT + file LICENSE" NA              NA
askpass  NA       "MIT + file LICENSE" NA              NA
```

```
         OS_type MD5sum NeedsCompilation Built
abind    NA      NA     "no"             "3.4.4"
acepack  NA      NA     "yes"            "3.4.4"
adabag   NA      NA     "no"             "3.4.4"
alabama  NA      NA     "no"             "3.4.4"
alluvial NA      NA     "no"             "3.4.4"
askpass  NA      NA     "yes"            "3.4.4"
```

Ver descripción de paquetes

Si esto te da error debes instalar el paquete alluvial.

```
> packageDescription("alluvial")
```

```
Package: alluvial
Type: Package
Title: Alluvial Diagrams
Version: 0.1-2
Date: 2016-09-09
Authors@R: c( person("Michal", "Bojanowski", role=c("aut", "cre"),
        email="michal2992@gmail.com"), person("Robin", "Edwards",
        role="aut", email="robin.edwards@ucl.ac.uk") )
Description: Creating alluvial diagrams (also known as parallel sets
        plots) for multivariate and time series-like data.
URL: https://github.com/mbojan/alluvial
BugReports: https://github.com/mbojan/alluvial/issues
Suggests: devtools, testthat, reshape2, knitr, rmarkdown, dplyr
License: MIT + file LICENSE
LazyLoad: yes
LazyData: yes
VignetteBuilder: knitr
RoxygenNote: 5.0.1
NeedsCompilation: no
Packaged: 2016-09-09 09:58:05 UTC; mbojan
Author: Michal Bojanowski [aut, cre], Robin Edwards [aut]
Maintainer: Michal Bojanowski <michal2992@gmail.com>
Repository: CRAN
Date/Publication: 2016-09-09 13:08:51
Built: R 3.4.4; ; 2019-04-05 02:31:51 UTC; unix

-- File: /home/rpalma/R/x86_64-pc-linux-gnu-library/3.4/alluvial/Meta/package.rds
```

Ayuda de uso de paquete

```
> help(package = "alluvial")
```

Ver ventana de ayuda abajo a la derecha ...

Muestreo

Generar un muestreo entre 10 individuos

Primero generamos los individuos

```
> x <- sample(1:10)
> x

 [1]  4  8  3  6  9  2  7  1  5 10

> match(c(4,8),x)

[1] 1 2

>
```

Elegir individuos entre el cuatro y el octavo

```
> match(c(4,8),x)

[1] 1 2
```

Muestreo con repetición

Generamos un muestreo de 30 muestras con 7 individuos y repetición

```
> x <- sample(1:7,30,replace=TRUE)
> x

 [1] 1 4 4 4 4 4 5 6 7 7 3 5 2 2 2 1 3 5 3 6 5 6 6 1 4 1 6 4 7 3
```

¿En que casos aparecieron los individuos 2 o 4?

```
> which(x %in% c(2,4))

 [1]  2  3  4  5  6 13 14 15 25 28
```

Captura de datos desde el teclado

Generaremos un vector llamado productividad obtenido de un estudio realizado sobre 25 empresas.

```
productividad <- scan()
```

Tipear estos números. Al final de cada número presione entre e ingrese el que sigue. En el último caso tipear dos veces enter.

Se puede ejecutar este comando desde el teclado para cargar un archivo local

```
library(car)
    library(readr)
    partners <- read.table("Socios.csv",header=TRUE,sep=";")
    partners2 <- read-xlsx ("D:/Datos/Socios2.xls")
```

o leerlo desde la web

```
> library(car)
> # Buscamos la tabla http://ceal.fing.uncu.edu.ar/r-cran/Socios.csv
> partners <-read.table("https://bit.ly/2WcBcdZ",header=TRUE,sep=";")
> names(partners)

[1] "TIR"    "VAN"    "Invent" "Asset"  "Type"

> head(partners)

  TIR VAN Invent Asset Type
1 5.1 3.5    1.4   0.2   Co
2 4.9 3.0    1.4   0.2   Co
3 4.7 3.2    1.3   0.2   Co
4 4.6 3.1    1.5   0.2   Co
5 5.0 3.6    1.4   0.2   Co
6 5.4 3.9    1.7   0.4   Co
```

¿Cuántas columnas tiene la tabla partners?

```
> length(partners)

[1] 5
```

¿Cuantas filas tiene?
Podemos saberlo examinando la longitud de cualquier columna

```
> length(partners$VAN)

[1] 150
```

Muestreo Estadístico

Para ello debemos pasar la tabla a memoria (attach)

```
> attach(partners)
> str(partners)

'data.frame':        150 obs. of  5 variables:
 $ TIR   : num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ VAN   : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Invent: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Asset : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Type  : Factor w/ 3 levels "Co","Da","Pi": 1 1 1 1 1 1 1 1 1 1 ...
```

```
>
```

Separaremos a la población en una muestra que tiene el 30% y reservaremos el resto (70%) de la población para otros cálculos o verificación de modelos

```
> muestra_2 <- sample(2, nrow(partners), replace=TRUE, prob=c(0.7, 0.3))
```

Ejecutar View muestra 2

```
> muestra_2

  [1] 1 2 1 2 2 2 2 2 1 1 2 1 2 1 2 2 1 2 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 2 1 2
 [38] 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 2 1 2 1 1 1 1 2
 [75] 1 1 1 1 1 2 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 2 2 1 1 1 2 1 1 1 1 2 1 2 1 2 2
[112] 1 1 1 1 1 2 1 2 2 1 1 2 1 2 2 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 2 2 2
[149] 1 2
```

Mostrar los datos de los casos usados en la muestra

```
> head( partners [muestra_2==1,])
```

```
   TIR VAN Invent Asset Type
1  5.1 3.5    1.4   0.2   Co
3  4.7 3.2    1.3   0.2   Co
9  4.4 2.9    1.4   0.2   Co
10 4.9 3.1    1.5   0.1   Co
12 4.8 3.4    1.6   0.2   Co
14 4.3 3.0    1.1   0.1   Co
```

```
>
```

Hoja de Ruta

Tenemos un centro de manufactura automatizada que tiene siete clientes que le envían trabajos a sus tres CNC. Según el BOM cada operación ha sido identificada y se conoce que secuencia se debe seguir en cada máquina. Se ha identificado la secuencia de máquinas a seguir por los pedidos de cada cliente. Todos los pedidos pasan por las tres máquina sin importar en que orden lo hagan. Se conocen los tiempos de mecanizado y las demoras que se generarán en el transporte desde uno a otro más el tiempo de espera estimado en la cola de entrada del centro de mecanizado.

Generar las hojas de ruta.

El tiempo de mecanizado sigue una ley beta (pert) de promedio 0.6 hs con apertura 2 y excentricidad 0.1 + 0.3 En tanto el transit time más la espera se han aproximado con una weibull de promedio 0.5 min con spread de 1

Graficar histogramas de ambos tiempos para el período de trabajo (130 operaciones) que en promedio se realizan en un día.

- ¿Cual es el tiempo máximo de manufactura? wtime
- Cual es el tiempo máximo de traslado / espera? qtiem histograma
- Realizar los mismos histogramas con 20 cajas
- Realizar el gráfico de densidad
- Determinar las horas a facturar a cada cliente
- Hacer un análisis sumario del tiempo que requiere cada cliente
- ¿Que probabilidad tengo de que el tiempo de manufactura sea menor o igual a 2 horas?

```
> x <- sample(1:7,130,replace=TRUE)
> HojaRuta <- list (cliente = LETTERS[x] ,
+                   op_1 = round(runif(130,1,4)),
+                   op_2 = round(runif(130,1,4)),
+                   op_3= round(runif(130,1,4)) ,
+                   wtime= 0.3+rbeta(130,0.6,2,0.1),
+                   queuet= (1+rweibull(130,0.5,1))/60 )

> summary(HojaRuta$wtime)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.3000  0.3438  0.4469  0.5269  0.6160  1.1283
```

Tiempo máximo de manufactura wtime

Convertiremos la lista en matriz

```
>  matrizHR <- matrix(unlist(HojaRuta), ncol = 6, byrow = FALSE)
```

View matrizHR

```
> library(nnet)
> which.is.max(matrizHR)

[1] 94

> matrizHR[74, ]

[1] "B"                "2"                "4"
[4] "3"                "0.421019338957121" "0.188563706338783"
```

Cual es
ver el comando class matrizHR

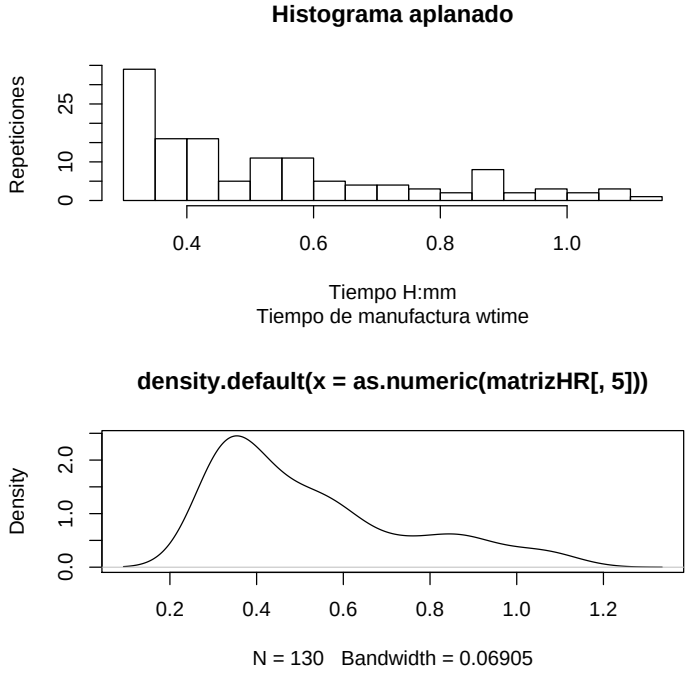
```
> density( as.numeric(matrizHR[ ,5]))
```

```
Call:
        density.default(x = as.numeric(matrizHR[, 5]))

Data: as.numeric(matrizHR[, 5]) (130 obs.);        Bandwidth 'bw' = 0.06905

       x                y
 Min.   :0.09287   Min.   :0.0006342
 1st Qu.:0.40350   1st Qu.:0.2392225
 Median :0.71414   Median :0.5929315
 Mean   :0.71414   Mean   :0.8038416
 3rd Qu.:1.02477   3rd Qu.:1.3017146
 Max.   :1.33541   Max.   :2.4519257
```

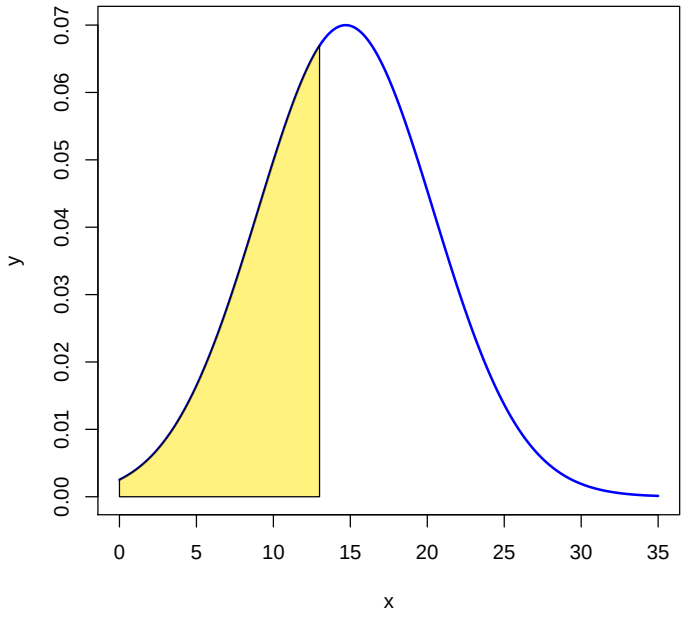
```
> par(mfrow=c(2,1))
> hist( as.numeric(matrizHR[ ,5]),
+       breaks = 25,
+       main = "Histograma aplanado",
+       sub = "Tiempo de manufactura wtime",
+       ylab= "Repeticiones",
+       xlab="Tiempo H:mm" )
> plot(density( as.numeric(matrizHR[ ,5])))
>
```



Trucos

Para resolver estos puntos se recomienda inspirarse en este ejemplo

```
> x=seq(0,35,length=350)
> x1=seq(0,13,length=130)
> x2=seq(13,35,length=(350-130))
> y=dnorm(x,mean=14.7,sd=5.7)
> y1=dnorm(x1,mean=14.7,sd=5.7)
> y2=dnorm(x2,mean=14.7,sd=5.7)
> plot(x,y,type="l", lwd=2, col="blue")
> polygon(c(0,x1,13),c(0,y1,0),col=rgb(1, 0.9, 0,0.5) )
```



Distribución Normal

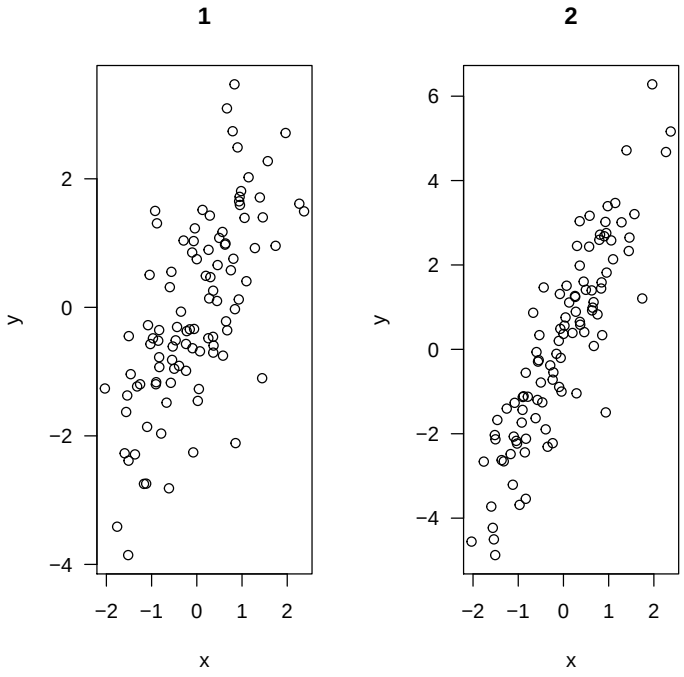
- R tiene una colección importante de distribuciones de probabilidades.
- Sin lugar a dudas la Distribución Normal es la que más se usa (acepta)
- Es posible obtener ayuda sobre el comando que trabaja con esta distribución con :

```
> ?pnorm
```

Mire en la ventana de ayuda lo que aparece y explique que hacen los comando dnorm, pnorm, qnorm y rnorm.

Ejemplos de Normales

```
> n <- 100
> x <- rnorm(n)
> par(mfrow=c(1,2), las=1)
> for(i in 1:8) {
+   y <- i*x + rnorm(n)
+   plot(x, y, main=i)
+ }
```



Uso de tablas

```
> n <- 100
> w <- rnorm(n)
> y <- 2*w + rnorm(n)
> out <- lm(y ~ w)
> library(knitr)
> kable(summary(out)$coef, digits=2)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.04	0.10	0.38	0.7
w	2.14	0.09	22.74	0.0

Otras Distribuciones

Distrib.	Comandos
Beta	pbeta qbeta dbeta rbeta
Binomial	pbinom qbinom dbinom rbinom
Cauchy	pcauchy qcauchy dcauchy rcauchy
Chi-Square	pchisq qchisq dchisq rchisq
Exponential	pexp qexp dexp rexp
F	pf qf df rf
Gamma	pgamma qgamma dgamma rgamma
Geometric	pgeom qgeom dgeom rgeom

Otras distribuciones (2)

Distrib.	Comandos
Hypergeometr	phyper qhyper dhyper rhyper
Logistic	plogis qlogis dlogis rlogis
Log Normal	plnorm qlnorm dlnorm rlnorm
Neg.Binomial	pnbinom qnbinom dnbinom rnbinom
Normal	pnorm qnorm dnorm rnorm
Poisson	ppois qpois dpois rpois
Student t	pt qt dt rt
Studentized	Range ptukey qtukey dtukey rtukey

Otras distribuciones (3)

Distrib.	Comandos
Uniform	punif qunif dunif runif
Weibull	pweibull qweibull dweibull rweibull
Wilcox Rank	pwilcox qwilcox dwilcox rwilcox
WilcoxSigned	psignrank qsignrank dsignrank rsignrank

Timeline

```
> library(timevis)
> data1 <- data.frame(
```

```
+   id      = 1:4,
+   content = c("Inicio", "Reunión_1",
+               "Rev.Papers", "Modelado"),
+   start   = c("2018-01-10", "2018-01-11",
+               "2018-01-20", "2018-02-14 15:00:00"),
+   end     = c(NA, NA, "2018-02-04", NA)
+ )
>
>
```

Timeview 2

Ejecutar desde la línea de comando
timevis(data1)

```
> timevis(data1)
```

Ejecutar desde la línea de comando
timevis(data1)

Clasificación de problemas de optimización

En base a su naturaleza, hay varias formas de clasificar un problema de optimización. Analizar en qué categoría entra es importante para definir el método de solución a utilizar, ya que no hay un método único para todos los posibles problemas. Para comenzar, una primera distinción la podemos realizar en base a la continuidad o no de las variables de decisión. Se dice que estamos frente a un problema de "Optimización Continua" cuando todas las variables de decisión pueden tomar cualquier valor perteneciente al conjunto de los reales. Dentro de este tipo de problemas, son de particular importancia los problemas de "Optimización Convexa", en los cuales se debe minimizar una función convexa sujeta a un conjunto solución convexo.

Cuando tanto la función objetivo como las restricciones son lineales, hablamos de un problema de "Optimización Convexa Lineal" o "Programación Lineal". En el caso de trabajar con variables discretas (es decir, que solo puedan tomar valores enteros) nos enfrentamos a un problema de "Optimización Combinatoria". Por raro que pueda parecer, en general un problema de optimización combinatoria es más complicado de resolver que uno de optimización continua. En el medio tenemos los problemas de "Optimización Mixta" es los cuales algunas variables son continuas y otras son discretas. En la práctica, estos problemas se resuelven en forma más parecida a los problemas combinatorios que a los continuos. Un caso particular de optimización combinatoria es la "Optimización Binaria", aquella en la cual todas sus variables están restringidas a tomar uno de dos valores (en general, 0 y 1).

Este caso es bastante raro de encontrar en la práctica, siendo más habitual encontrar problemas combinatorios con algunas variables binarias (optimización mixta).

Otra clasificación la podemos hacer en base a la naturaleza probabilística del problema. Cuando podemos considerar que todas las variables son determinísticas, estamos ante un problema determinista, en caso contrario nos enfrentamos a un problema estocástico. El modelado y resolución de un problema estocástico es mucho más complejo que el modelado de un problema determinístico

Clasificación de métodos de resolución

Los métodos de resolución de problemas de optimización se pueden clasificar en tres tipos diferentes:

- Resolución mediante cálculo
- Resolución mediante técnicas de búsquedas
- Resolución mediante técnicas de convergencia de soluciones

Resolución mediante cálculo

Los métodos de resolución por cálculo apelan al cálculo de derivadas para determinar para qué valores del dominio la función presenta un máximo o un mínimo. Son métodos de optimización muy robustos, pero que requieren mucho esfuerzo de cómputo y que tanto la función objetivo como las restricciones presenten determinadas condiciones (por ejemplo, que estén definidas, que sean continuas, etc.). En general, no se suele apelar a estos métodos en el mundo de la ingeniería, ya que los problemas no se ajustan a las restricciones de continuidad y tienen demasiadas variables como para que su tratamiento pueda ser eficiente.

Resolución mediante técnicas de búsquedas

Dentro de este apartado, podemos encontrar un gran abanico de técnicas, desde el viejo método de prueba y error hasta las modernas técnicas de programación matemática. En forma genérica, consisten en el siguiente algoritmo: Seleccionar una solución inicial y hacerla la solución actual:

1) Hacer Mejor Solución = Solución Actual
2) Buscar n soluciones cercanas a la Solución Actual
3) Para cada una de las n soluciones cercanas hacer

- a. Si el valor de la función objetivo de la solución a verificar es mayor (o menor) al valor generado por la solución actual, hacer Mejor Solución = Solución Evaluada
- b. Si Mejor Solución = Solución Actual, Finalizar del procedimiento, en caso contrario Hacer Solución Actual = Mejor Solución y volver a 2)

Dentro de estos métodos tenemos técnicas para abarcar una gran variedad de problemas. Desde técnicas exactas, como la "Programación Lineal" (que se limita solo a problemas con un conjunto solución convexo y función objetivo y restricciones lineales) hasta las técnicas metaheurísticas de solución aproximada como la "Búsqueda Tabú".

Algoritmos para programación lineal

Introducción a la Programación Lineal

Los problemas de Programación Lineal son problemas de optimización convexa en el cual tanto la función objetivo como las restricciones son funciones lineales. Si bien su definición parece un tanto restrictiva, en realidad muchos de los problemas estudiados en el mundo real entran dentro de esta categoría, por ejemplo:

- Asignación de rutas de transporte
- Balance de líneas de producción
- Asignación de procesos a equipos de procesamiento
- Definición de mix de productos
- Definición de mezcla de componentes para productos
- Nivelación de recursos

Un problema de programación lineal se caracteriza porque su conjunto de soluciones factibles forma un politopo (o sea, la generalización de un polígono para n dimensiones). Por su propia naturaleza, al estar los gradientes definidos en una solo dirección, los puntos extremos se encuentran en la frontera del politopo, más específicamente en algunos de sus vértices. Esto permitió el desarrollo del muy eficiente Método Simplex para su resolución, que se aprovecha de esta particularidad para buscar soluciones solo en dichos puntos.

Cómo resolver problemas de PL con R-Cran

Dentro de los paquetes "estándares" que se distribuyen con R, tenemos el paquete "boot". Entre otras, este paquete tiene una función denominada **simplex** que se encarga de aplicar el algoritmo de dicho nombre. Para usarla, primero carguemos el paquete:

```
> library(boot)
```

La función "simplex" se encarga de resolver problemas de programación lineal. Permite únicamente el uso de solo variables continuas y fuerza las soluciones no negativas. Veamos un ejemplo de uso:

0.1 Resolver el siguiente problema

$$
\begin{array}{lllllll}
Max.\ Z & = & 1x_1+ & 2x_2+ & 3x_3 & & \\
s.t & & 0x_1+ & 2x_2+ & 3x_3 & \leq & 100 \\
 & & 5x_1+ & 0x_2+ & 3x_3 & \leqslant & 100 \\
x_j \geqslant 0 & j=1,2,3 & & i \in N & & & \\
1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}
$$

El código para escribir este tipo de problema en LaTeXes el siguiente:

```
\[ \begin{array}{lllllllr}
Max. \phantom{1} Z & = & 1x_1 +& 2x_2 + & 3x_3   & \phantom{1} \\[\jot]
s.t & \phantom{5} & 0x_1  +& 2x_2 + & 3x_3  & \leq & 100  \\
\phantom{5} & \phantom{5} & 5x_1 +& 0x_2 + & 3x_3  & \leqslant & 100\\[\jot]
x_j \geqslant 0 & j=1,2,3 & \quad & i \in N & \quad & \quad & \phantom{5} \\
% linea guia para ver la matriz
1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}\]
```

Este problema se puede resolver en R-cran invocando la biblioteca anterior como sigue:

```
> coefFuncionZ <- c(1,2,3)
> matrizRestriccionesMenorOIgual <-
+ matrix(c(0,2,3,5,0,3),2,3,byrow=TRUE)
> vectorTermIndep <- c(100,100)
> mySimplex <- simplex(coefFuncionZ, A1 =
+ matrizRestriccionesMenorOIgual, b1= vectorTermIndep,
+ maxi=TRUE )
> mySimplex

Linear Programming Results

Call : simplex(a = coefFuncionZ, A1 = matrizRestriccionesMenorOIgual,
    b1 = vectorTermIndep, maxi = TRUE)

Maximization Problem with Objective Function Coefficients
x1 x2 x3
 1  2  3


Optimal solution has the following values
x1 x2 x3
20 50  0
The optimal value of the objective  function is 120.
```

simplex() es una función compleja para utilizar debido al gran número de parámetros que requiere. Veamos su ayuda:

```
> ?simplex
```

Como vemos, amén del vector "a" de los coeficientes de la función objetivo, tenemos parámetros del tipo matrix para los coeficientes de las restricciones (A1, A2 y A3, dependiendo si son del tipo $\leq$, $\geq$ o $=$, respectivamente), coeficientes para los términos independientes de las restricciones (b1, b2 y b3, con la misma lógica que los An), un parámetro maxi que si vale TRUE el solver se ejecuta maximizando (si se omite o vale FALSE se ejecuta minimizando), y parámetros de cálculo algorítmicos como "n.iter" para el número de iteraciones y "eps" para la precisión decimal al momento de comparar valores. Entonces, como mínimo, para utilizar esta función, debemos definir un vector de coeficientes para la función objetivo, una o más matrices de coeficientes de restricciones (A1 para las restricciones $\leq$, A2 para las $\geq$, y A3 para las $=$) y el término independiente respectivo para cada tipo de restricción, y opcionalmente una indicación acerca de si queremos minimizar. Como objeto hecho y derecho, podemos acceder a los atributos del objeto generado por simplex:

```
> class(mySimplex)

[1] "simplex"

> mySimplex$soln # Soluciones

x1 x2 x3
20 50  0

> mySimplex$soln[2] # Solución de la segunda variable

x2
50

> mySimplex$value # Valor de la función objetivo

  b
120

> mySimplex$slack # Holgura de cada restricción menor igual

[1] 0 0

> mySimplex$surplus # Excedente de cada restricción mayor igual

NULL
```

Este objeto tiene, además, otros atributos relativos al cálculo:

```
> mySimplex$a # Coeficientes de Z con variables básicas

[1] 0.0 0.0 0.6 1.0 0.2
```

```
> #             con coeficientes igual a cero
> mySimplex$basic # Los índices de las variables básicas

[1] 2 1

> #                 en la solución encontrada
> mySimplex$A # Matriz de restricciones expresada en

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1 -1.5 -0.5  0.0
[2,]    1    0 -0.6  0.0 -0.2

> #             términos de variables no básicas
>
```

Esta función es de las más básicas para trabajar con problemas de optimización. No trae incorporado análisis de sensibilidad (aunque pueden calcularse), ni permite declarar variables enteras. Además presupone no negatividad, por lo cual si queremos que nuestras variable puedan tomar negativas, tenemos que utilizar el artilugio de separar nuestra variable en dos 37 . Sin embargo, es muy intuitiva en su operación con matrices. La contra más grande que tiene es que es lenta, tarda un tiempo promedio proporcional al cubo de la cantidad de restricciones para hallar una respuesta.

Otros paquetes de PL: lpSolve

Vamos a trabajar con otro paquete para resolver problemas de optimización lineal. Como no es estándar, tenemos que descargarlo desde CRAN:

install.packages("lpSolve")

Ahora, seleccionamos la ubicación de algún servidor y listo, ya queda instalado.

En RStudio podría haber ido al menú "Tools", seleccionar el ítem "Install Packages" y escribir "lpSolve" en el cuadro de texto:

Una vez instalado, tenemos que cargarlo al workspace 38 actual con la función "library":

```
> library(lpSolve)
```

lpSolve encapsula el acceso al programa también llamado lpSolve . El mismo tiene licencias LGPL y está disponible en forma de bibliotecas. El paquete lpSolve implementa algunas de las funcionalidades del programa lpSolve. ¿Se acuerdan cómo funcionaba la función "simplex" del paquete "boot"?. Teníamos tres matrices de coeficientes de restricciones y tres vectores de términos independientes, uno por cada tipo de restricción ($\leq$, =, $\geq$). En este paquete las cosas funcionan un poco más simples. Bueno, resolvamos un problema sencillo con lpSolve.

Primero, definamos los coeficientes de la función objetivo Z:

```
> problem.coef.obj <- c(1, 9, 3)
```

Como vemos, es una función de tres variables. Supongamos que tenemos dos restricciones, una del tipo ≥ y otra del tipo ≤:

```
> problem.coef.restr <- matrix (c(1, 2, 3, 3, 2, 2), nrow=2,
+   byrow=TRUE)
>         problem.coef.restr

     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    3    2    2
```

En una única matriz defino mis dos restricciones. Como el parámetro "byrow" 40 vale "TRUE", tenemos la primera restricción de coeficientes (1, 2, 3) y la segunda de coeficientes (3,-2,2). Y ahora defino los signos de mis restricciones en un vector del tipo "character":

```
> problem.dir.restr <- c(">=", "<=")
> problem.dir.restr

[1] ">=" "<="
```

Seguimos con los términos independientes:

```
> problem.term.restr <- c(1, 15)
```

Maximicemos la función objetivo:

```
> lp ("max", problem.coef.obj, problem.coef.restr,
+ problem.dir.restr, problem.term.restr)

Success: the objective function is 67.5
```

En cuanto a sintaxis, mucho mejor que usar la función simplex. Además, el algoritmo simplex que implementa la función lp() es mucho más veloz. También a diferencia de solver en Excel tiene capacidad de administrar problemas mucho más grandes, incluso en máquinas de pequeño porte sin provocar desbordamientos de memoria "la temida pantalla azul de windows"

Como ya sabemos, todo en R es un objeto:

```
> mylp <- lp ("max", problem.coef.obj, problem.coef.restr,
+ problem.dir.restr, problem.term.restr)
> mylp

Success: the objective function is 67.5
```

Y como objeto, podemos acceder a sus miembros. Primero, las soluciones del problema:

```
> mylp$solution

[1] 0.0 7.5 0.0
```

También el valor de Z alcanzado:

```
> mylp$objval

[1] 67.5
```

¿Queremos hacer análisis de sensibilidad?, bueno, hagamos valer TRUE el parámetro "compute.sens" y volvamos a resolver:

```
> mylp <- lp ("max", problem.coef.obj, problem.coef.restr,
+ problem.dir.restr, problem.term.restr, compute.sens=TRUE)
```

¿Queremos hacer análisis de sensibilidad?, bueno, hagamos valer TRUE el parámetro "compute.sens":

```
> mylp$sens.coef.from

[1] -1e+30  3e+00 -1e+30
```

Y accedamos a los atributos "sens.coef.from" y "sens.coef.from" para ver la sensibilidad de los coeficientes de la función objetivo:

```
> mylp$sens.coef.from

[1] -1e+30  3e+00 -1e+30
```

"compute.sens=TRUE" nos permite hacer análisis de dualidad también:

```
> mylp$duals

[1]   0.0   4.5 -12.5   0.0  -6.0
```

Donde sí tengo "n" variables y "m" restricciones, los primeros m valores corresponden a los valores duales de las restricciones y los siguientes n valores a los duales de las actuales variables de decisión. Puedo ver la sensibilidad de los duales también:

```
> mylp$duals.from

[1] -1.0e+30  1.0e+00 -1.0e+30 -1.0e+30 -1.4e+01

> mylp$duals.to

[1] 1.0e+30 1.0e+30 5.0e+00 1.0e+30 7.5e+00
```

No me devuelve las holguras, pero las puedo calcular a mano. Primero devuelvo la matriz de las restricciones:

```
> mylp$constraints

              [,1] [,2]
                 1    3
                 2    2
                 3    2
const.dir.num    2    1
const.rhs        1   15
```

La matriz se devuelve transpuesta (es la forma en que lpSolve trabaja internamente). La primera columna es la primera restricción, la segunda es la segunda restricción. La última fila representa a los términos independientes, la anteúltima son referencias al sentido de las restricciones (2 es $\leq$, 1 es $\geq$ y 3 =) y los primeros valores los coeficientes de las restricciones. Entonces, podemos hacer:

```
> sum(mylp$constraints[1:3,1] * mylp$solution) -
+ mylp$constraints[5,1]

const.rhs
       14
```

Luego

```
> sum(mylp$constraints[1:3,2] * mylp$solution) -
+ mylp$constraints[5,2]

    const.rhs
-1.776357e-15
```

/subsection*Problema de Almacenes y Clientes El paquete lpSolve tiene algunas funcionalidades adicionales. Viene con dos funciones que implementan los algoritmos de asignación y transporte. Veamos el caso de transporte. Definamos la matriz de costo:

```
> matrizCosto <- matrix (10000, 8, 5)
> matrizCosto

      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] 10000 10000 10000 10000 10000
[2,] 10000 10000 10000 10000 10000
[3,] 10000 10000 10000 10000 10000
[4,] 10000 10000 10000 10000 10000
[5,] 10000 10000 10000 10000 10000
[6,] 10000 10000 10000 10000 10000
[7,] 10000 10000 10000 10000 10000
[8,] 10000 10000 10000 10000 10000
```

Este problema podría enunciarse del siguiente modo: Tengo 8 almacenes (filas) que tienen que abastecer a 5 Clientes. Genero una matriz con los costos de transportar desde cada almacén a cada cliente. En principio he tomado un valor muy alto para ese costo, con el objeto de poner luego costos reales y que al minimizar el costo total jamás se tome ese camino de costo alto.

En esta matriz cada elemento [i,j] nos dice cual es costo unitario de transporte desde i a j. La modificaremos un poco para hacerla más real:

Podría usar scan() para cargar los datos a mano, pero quiero que vean como puedo hacerlo por código. Incluso puedo armar un bucle for / while para hacer que el costo crezca según una ley dada

```
> matrizCosto[4,1]<- matrizCosto[-4,5]<- 0
> matrizCosto

      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] 10000 10000 10000 10000     0
[2,] 10000 10000 10000 10000     0
[3,] 10000 10000 10000 10000     0
[4,]     0 10000 10000 10000 10000
[5,] 10000 10000 10000 10000     0
[6,] 10000 10000 10000 10000     0
[7,] 10000 10000 10000 10000     0
[8,] 10000 10000 10000 10000     0

> matrizCosto[1,2]<- matrizCosto[2,3]<- matrizCosto[3,4] <- 7
> matrizCosto[1,3]<- matrizCosto[2,4]<- 7.7
> matrizCosto

      [,1]  [,2]    [,3]    [,4]  [,5]
[1,] 10000     7     7.7 10000.0     0
[2,] 10000 10000     7.0     7.7     0
[3,] 10000 10000 10000.0     7.0     0
[4,]     0 10000 10000.0 10000.0 10000
[5,] 10000 10000 10000.0 10000.0     0
[6,] 10000 10000 10000.0 10000.0     0
[7,] 10000 10000 10000.0 10000.0     0
[8,] 10000 10000 10000.0 10000.0     0

> matrizCosto[5,1]<- matrizCosto[7,3]<- 8
> matrizCosto

      [,1]  [,2]    [,3]    [,4]  [,5]
[1,] 10000     7     7.7 10000.0     0
[2,] 10000 10000     7.0     7.7     0
[3,] 10000 10000 10000.0     7.0     0
[4,]     0 10000 10000.0 10000.0 10000
[5,]     8 10000 10000.0 10000.0     0
```

```
[6,] 10000 10000 10000.0 10000.0     0
[7,] 10000 10000     8.0 10000.0     0
[8,] 10000 10000 10000.0 10000.0     0
```

Modifiquemos aún más

```
> matrizCosto[1,4] <- 8.4
> matrizCosto[6,2] <- 9
> matrizCosto[8,4] <- 10
> matrizCosto[4,2:4] <- c(.7, 1.4, 2.1)
> matrizCosto

      [,1]  [,2]    [,3]    [,4]  [,5]
[1,] 10000 7e+00     7.7     8.4     0
[2,] 10000 1e+04     7.0     7.7     0
[3,] 10000 1e+04 10000.0     7.0     0
[4,]     0 7e-01     1.4     2.1 10000
[5,]     8 1e+04 10000.0 10000.0     0
[6,] 10000 9e+00 10000.0 10000.0     0
[7,] 10000 1e+04     8.0 10000.0     0
[8,] 10000 1e+04 10000.0    10.0     0
```

Armemos el vector de restricciones. Estos vectores indican que en ninguno caso la suma de las filas debe ser superior a la cantidad que tiene cada almacén. Por otro lado la suma de las columnas tienen que ser exactamente lo que cada cliente pide.

```
> row.dir <- rep ("<", 8)
> row.term <- c(200, 300, 350, 200, 100, 50, 100, 150)
> col.dir <- rep (">", 5)
> col.term <- c(250, 100, 400, 500, 200)
```

a primer línea dice que todos los 8 almacenes tiene una restricción del tipo ”menor que”. Fijarse como con el comando rep repito la dirección de la restricción La segunda línea señala la cantidad que cada almacén tiene La tercer es la restricción que dice que no podemos enviar más que lo que el cliente pidió Finalmente tenemos un vector que dice cuanto han pedido los clientes. Para resolver el problema Corramos hora el comando lp.transport():

```
> myTransport <- lp.transport(matrizCosto, "min", row.dir,
+ row.term, col.dir, col.term)
> myTransport

Success: the objective function is 7790
```

Veamos cuanta carga se ha asignado para obtener este costo mínimo

```
> myTransport$solution
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    0  100  100    0    0
[2,]    0    0  200  100    0
[3,]    0    0    0  350    0
[4,]  200    0    0    0    0
[5,]   50    0    0    0   50
[6,]    0    0    0    0   50
[7,]    0    0  100    0    0
[8,]    0    0    0   50  100
```

Muy "inteligentemente", R nos devuelve una matriz de asignación de orígenes a destinos, en vez de una lista de valores de variables de decisión.

0.2 El problema de asignación

Definir un problema de asignación es más fácil, ya que al ser todas las restricciones ≤ con variables binarias, lpSolve me automatiza la carga, por lo cual solo me ocupo de la matriz de costo:

Este problema de asignación es típico en casos de problemas de pallets sobre piso en posiciones flotantes, también se usa para diseño de layouts, casos simples de ruteo, etc. En la biografiara existe un método llamado "Método Húngaro" pero la heurística de lpsolve es mucho más poderosa.

Generaremos la matriz de costos de asignación, que debe interpretarse así. Cada elemento A(i,j) representa el costo de poner el elemento que se señala en la fila i en la posición de piso j.

```
> matrizCostoAsignacion <- matrix (c(2, 7, 7, 2, 7, 7, 3, 2,
+ 7, 2, 8, 10, 1, 9, 8, 2), 4, 4)
> matrizCostoAsignacion

     [,1] [,2] [,3] [,4]
[1,]    2    7    7    1
[2,]    7    7    2    9
[3,]    7    3    8    8
[4,]    2    2   10    2
```

En el método Húngaro, de debía proporcionar una solución inicial. Esta matriz solución llamada H era una matriz diagonal unitaria. Se establecían restricciones del tipo: La suma de toda las filas debe ser igual a 1 La suma de todas las columnas debe ser igual a 1 Esto se puede interpretar como "un lugar para cada cosa" y "cada cosa en su lugar" Sin embargo no es necesario usar esto en lpsolve. Se invoca el problema como de asignación y se resuelve.

```
> myAssign <- lp.assign(matrizCostoAsignacion, "min")
> myAssign

Success: the objective function is 8
```

Este es el costo mínimo. Luego el plan de uso de las posiciones o asignaciones de productos al almacén serán las que se exponen a continuación. La cantidad de problemas que se pueden resolver por este método es realmente impresionante. Este es uno de los terrenos en los que R-Cran corre con ventajas frente a otros productos. En la Universidad Nacional de Cuyo existía un problema serio de aisgnación de aulas por franja horaria, con vaientntes que complicaban más aún el problema, tales como la necesidad para algunos profesores ślo en algunas clases al año de alta conectividad en el aula o necesidad de un proyectos de mayor resolución de pantalla que los que se utilizan a diario. Este método ha demostrado ser muy eficaz para resolverlo.

```
> myAssign$solution
```

```
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    1
[2,]    0    0    1    0
[3,]    0    1    0    0
[4,]    1    0    0    0
```

Otras alternativas de Optimización con R

En los ejemplos anteriores hemos visto la importancia de modelar (antes que resolver) problemas Hemos visto también los distintos tipoos de probleas de optimización y las limitaciones que podemos encontrar al trabajarlos con hojas de calculo. En este apartado nos sentraremos un una serie de bilbiotecas (library) que pueden ayudarnos cuando la potencia de las hojas de cálculo no nos dejan tener certeza de los resultados obtenidos.

Otros paquetes de Programación Lineal: lpSolveAPI

lpSolve no es la única implementación de "lpSolve" para R, tenemos también lpSolveAPI:

```
> #install.packages("lpSolveAPI")
> library(lpSolveAPI)
```

Los modelos con lpSolveAPI se construyen de otra forma. Para empezar, no son objetos R hechos y derechos, así que es muy parecido su uso a como se usaría desde C. Con lpSolveAPI comienzo así:

```
> lprec <- make.lp(0, 4)
> lprec
```

```
Model name:
            C1    C2    C3    C4
Minimize     0     0     0     0
```

```
Kind       Std   Std   Std   Std
Type      Real  Real  Real  Real
Upper      Inf   Inf   Inf   Inf
Lower        0     0     0     0

>
```

Con esto creo un objeto "modelo lineal" con 4 variables de decisión. Este es un objeto vacio, pero que tiene la estructura para que podamos cargar nuestro modelo (o problema a resolver) Definamos ahora los coeficientes:

```
> set.objfn(lprec, c(1, 3, 6.24, 0.1))
> lprec

Model name:
            C1    C2    C3    C4
Minimize     1     3  6.24   0.1
Kind       Std   Std   Std   Std
Type      Real  Real  Real  Real
Upper      Inf   Inf   Inf   Inf
Lower        0     0     0     0

>
```

Agrego las restricciones:

```
> add.constraint(lprec, c(0, 78.26, 0, 2.9), ">=", 92.3)
> add.constraint(lprec, c(0.24, 0, 11.31, 0), "<=", 14.8)
> add.constraint(lprec, c(12.68, 0, 0.08, 0.9), ">=", 4)
> lprec

Model name:
             C1     C2     C3     C4
Minimize      1      3   6.24    0.1
R1            0  78.26      0    2.9  >=  92.3
R2         0.24      0  11.31      0  <=  14.8
R3        12.68      0   0.08    0.9  >=     4
Kind        Std    Std    Std    Std
Type       Real   Real   Real   Real
Upper       Inf    Inf    Inf    Inf
Lower         0      0      0      0

>
```

En caso de restricciones cuya única función sea acotar una única variable, las agrego separadamente como "Bounds": Esto suele ser casi siempre una limitación del tipo presupuestaria (no puedo superar cierto gasto, cierno número de camiones, etc)

```
> set.bounds(lprec, lower = c(28.6, 18), columns = c(1, 4))
> set.bounds(lprec, upper = 48.98, columns = 4)
> lprec

Model name:
             C1     C2     C3     C4
Minimize      1      3   6.24    0.1
R1            0  78.26      0    2.9  >=  92.3
R2         0.24      0  11.31      0  <=  14.8
R3        12.68      0   0.08    0.9  >=     4
Kind        Std    Std    Std    Std
Type       Real   Real   Real   Real
Upper       Inf    Inf    Inf  48.98
Lower      28.6      0      0     18

>
```

Normalmente el modelo como variables puede ser claro para quien modela el problema, pero puede ser confuso para quien toma decisiones. Tengo ponerle nombre:

```
> rowNames <- c("Restricción_01", "Restricción_02", "Restricción_03")
> colNames <- c("Var_X1", "Var_X2", "Var_X3", "Var-X4")
> dimnames(lprec) <- list(rowNames, colNames)
> lprec

Model name:
                Var_X1  Var_X2  Var_X3  Var-X4
Minimize             1       3    6.24     0.1
Restricción_01       0   78.26       0     2.9  >=  92.3
Restricción_02    0.24       0   11.31       0  <=  14.8
Restricción_03   12.68       0    0.08     0.9  >=     4
Kind               Std     Std     Std     Std
Type              Real    Real    Real    Real
Upper              Inf     Inf     Inf   48.98
Lower             28.6       0       0      18

>
>
```

Listo, ahora podemos pasar a resolverlo: Sorprende la facilidad con la que lo hacemos, solamente llamamos a un comando invocando el objeto creado.

```
> solve(lprec)

[1] 0

>
```

Si deseo ver el valor optimal de Z (valor alcanzado por la función a optimizar) invoco el comando siguiente.

```
> get.objective(lprec)

[1] 31.78276
```

Obtener las soluciones o valores de la variables en el óptimo:

```
> get.variables(lprec)

[1] 28.60000  0.00000  0.00000 31.82759

>
```

Valor de las restricciones asumidos o proporcionados

```
> get.constraints(lprec)

[1]  92.3000   6.8640 391.2928

>
```

Programación Lieal Entera

Resulta comprensible el por qué es más conveniente esta bilioteca que las otras (nativas) que trae el paquete R.

Trnasformar un problema de LP en ILP (Integer Linear Programin) resulta un juego de niños. Supongamos que tenemos el mismo problema anterior y queremos que la variable 4 sea etntera. Esta variable puede represetar el número de containers que tendré en inventario, la cantidad de choferes a asignar a cada ruta u otra variable que me interese manter entera.

```
> set.type(lprec, 4, "integer")
> lprec

Model name:
                Var_X1  Var_X2  Var_X3  Var-X4
Minimize             1       3    6.24     0.1
Restricción_01       0   78.26       0     2.9  >=  92.3
Restricción_02    0.24       0   11.31       0  <=  14.8
Restricción_03   12.68       0    0.08     0.9  >=     4
Kind               Std     Std     Std     Std
Type              Real    Real    Real     Int
Upper              Inf     Inf     Inf   48.98
Lower             28.6       0       0      18

>
```

Veamos ahora las souciones obtenidas y coparemos con el caso anterior.

```
> solve(lprec)

[1] 0

> get.objective(lprec)

[1] 31.792

> get.variables(lprec)

[1] 28.60000000  0.03066701  0.00000000 31.00000000

> get.constraints(lprec)

[1]  92.300   6.864 390.548

>
>
```

Otro Problema de Asignación

Este tipo de problema se llama bounded, quiere decir que además de la matrz de unos y ceros, tenemos variables que solamente pueden tomar valores entre ciertos límites que pueden ser superiores o inferiores o ambos. Podemos intuir que siguiendo la síntaxis anterior sería posible declarar una variable binaria (en lugar de entera). Esto, tal como se vio en los primeros ejercicios, no lleva a tener la base para armar un problema de asignación.

Usando el ejemplo anterior veamos como declaro una variable binaria.

Si la segunda variable debiera ser binaria:

```
> set.type(lprec, 2, "binary")
> lprec

Model name:
                Var_X1  Var_X2  Var_X3  Var-X4
Minimize             1       3    6.24     0.1
Restricción_01       0   78.26       0     2.9  >=  92.3
Restricción_02    0.24       0   11.31       0  <=  14.8
Restricción_03   12.68       0    0.08     0.9  >=     4
Kind               Std     Std     Std     Std
Type              Real     Int    Real     Int
Upper              Inf       1     Inf   48.98
Lower             28.6       0       0      18
```

Luego repasemos las soluciones Z, valores de X y restricciones:

```
> solve(lprec)

[1] 0

> get.objective(lprec)

[1] 31.8

> get.variables(lprec)

[1] 28.6  0.0  0.0 32.0

> get.constraints(lprec)

[1]  92.800   6.864 391.448

>
>
```

Ejercicios para resolver Examen Final

Estos ejercicios deben ser resuletos con R-Cran y lpsolve_api

Plan de producción

Plantear este ejercicio en R

Consideremos la siguiente situación: Una pequeña empresa vende dos productos, denominados Producto 1 y Producto 2. Cada tonelada del Producto 1 consume 30 horas de trabajo, y cada tonelada del producto 2 consume 20 horas de trabajo. El negocio tiene un máximo de 2.700 horas de trabajo durante el período considerado. En cuanto a las horas de máquina, cada tonelada de Productos 1 y 2 consume 5 y 10 horas de máquina respectivamente. Hay 850 horas de máquina disponibles.

Cada tonelada del Producto 1 produce 20 Me de ganancia, mientras que el Producto 2 produce 60 Me por cada tonelada vendida. Por razones técnicas, la empresa debe producir un mínimo de 95 toneladas en total entre ambos productos. Necesitamos saber cuántas toneladas de Producto 1 y 2 deben ser producidas para maximizar el beneficio total.

Esta situación es apta para ser modelada como un modelo PL. Primero, necesitamos definir las variables de decisión. En este caso tenemos:

- P 1 número de toneladas producidas y vendidas del Producto 1
- P 2 número de toneladas producidas y vendidas del producto 2

Los coeficientes de costo de estas variables son 20 y 60, respectivamente. Por lo tanto, la función objetivo se define multiplicando cada variable por su Coeficiente de coste correspondiente.

Las limitaciones de este LP son:

1. Una limitación de Horas de Trabajo que haga que la cantidad total de horas de trabajo utilizadas en el Producto 1 y Producto 2, que sea igual a 30P 1 + 20P 2, sea inferior o igual a 2.700 horas.

2. Una restricción similar Horas de máquina que hacen que el total de horas de máquina 5P 1 + 10P 2 sea menor o igual que 850.

3. Una restricción PM que hace que el total de unidades producidas y vendidas P1 + P2 son mayores o iguales que 95.

Juntando todo esto, y considerando que las variables de decisión son no negativas, el LP que maximiza el beneficio es:

$$\begin{array}{lrrrrr} Max.\ Z & = & 20P_1+ & 60P_2 & & \\ s.t & & WH) & 30P_1+ & 20P_2 & \leq & 2700 \\ & & MH) & 5P_1+ & 10P_2 & \leqslant & 850 \\ & & PM) & P_1+ & P_2 & \geq & 95 \end{array}$$

Un problema de transporte

Consideremos un problema de transporte de dos orígenes a y b, y tres destinos 1, 2 y 3. En la Tabla se presenta el costo c(ij) de transportar una unidad desde el origen i al destino j, y la capacidad máxima de los orígenes y Demanda requerida en los destinos. Necesitamos saber cómo debemos cubrir la demanda de los destinos en Un costo mínimo.

Algoritmo Genético

Problema tipo PnakSack

El algoritmo genético (GA) es una técnica de optimización basada en la búsqueda basada en los principios de genética y selección natural. Con frecuencia se utiliza para encontrar soluciones óptimas o casi óptimas a problemas difíciles que de otra manera podrían llevar una vida para resolverse. Se utiliza con frecuencia para resolver problemas de optimización, cálculo variacional, simulación y análisis de escenarios, así como en la investigación, y en el aprendizaje automático (Deep Learning).

GA se basa en la generción de una población, los individuos en esta población (a menudo representdos como cromosomas) tienen un estado dado. Una vez que se genera la población, el estado de estos individuos se evalúa y se clasifica en su valor o desempeño. Se debe crear una función de desempeña para que a partir de estado cromosómico se obtenga una cifra de preformance. Las mejores candidatos son tomadas y cruzadas -en la esperanza de generar una mejor descendencia, para formar la nueva población. En algunos casos se conservan los mejores individuos de la población con el fin de garantizar ”buenos candidatos”

en la nueva generación (esto se llama elitismo, entendido como procentaje de padres que prevalecen frente a hijos).

Para explicar el ejemplo se utilizarar una versión simplificada de la problema mochila (pnacksack).

Te proponen pasar un mes en el desierto. Partirás llevando una mochila y tu supervivencia depende de lo que cargues, sin embargo, el peso máximo que puede transportar es de 20 kilogramos. Tienes una serie de artículos de supervivencia disponibles, cada uno con su propio número de "puntos de supervivencia". El objetivo es maximizar el número de puntos de supervivencia.

En la siguiente tabla se muestran los elementos que puede elegir.

ITEM	SURVIVAL POINTS	peso kg
navaja	10,00	1,00
conserva	20,00	5,00
papas	15,00	10,00
mosquetones	2,00	1,00
bolsa dormir	30,00	7,00
soga	10,00	5,00
brújula	30,00	1,00

```
> library(genalg)
> library(ggplot2)
> dataset <- data.frame(item = c("pocketknife", "beans", "potatoes", "unions",
+     "sleeping bag", "rope", "compass"), survivalpoints = c(10, 20, 15, 2, 30,
+     10, 30), weight = c(1, 5, 10, 1, 7, 5, 1))
> weightlimit <- 20
>
>
```

Veamos como ha quedado conformado el set de datos

```
> dataset
```

```
          item survivalpoints weight
1  pocketknife             10      1
2        beans             20      5
3     potatoes             15     10
4       unions              2      1
5 sleeping bag             30      7
6         rope             10      5
7      compass             30      1
```

Configuración de la Función de Evaluación

Antes de crear el modelo tenemos que configurar una función de evaluación. La función de evaluación evaluará a los diferentes individuos (cromosomas) de la población sobre el valor de su configuración genética.

Por ejemplo, un individuo puede tener la siguiente configuración de genes: 1001100. Esto representa algo así como que llevo los itmes señalados con 1 en el cromosoma, tomando como referencia la columna 2 del dataset. Podemos elegir a esta como una solución inicial, independientemente de cumplir o no con la restricción de 20 kg.

Cada número de esta cadena binaria representa si desea o no llevar un elemento con usted. Un valor de 1 se refiere a poner el artículo específico en la mochila mientras que un 0 se refiere a dejar el elemento en casa. Dado el ejemplo de configuración de genes, tomaríamos los siguientes elementos;

Podemos comprobar a qué cantidad de puntos de supervivencia de esta configuración. Esto le dará un valor a la configuración génica de un cromosoma dado. Esto es exactamente lo que hace la función de evaluación.

```
> chromosome = c(1, 0, 0, 1, 1, 0, 0)
> dataset[chromosome == 1, ]

          item survivalpoints weight
1  pocketknife             10      1
4       unions              2      1
5 sleeping bag             30      7
```

Mutiplico cada uno y cadad cero del cromosoma por el survivalpoint

```
> cat(chromosome %*% dataset$survivalpoints)

42
```

Crearemos una función de desempeño para no tener que repetir en pada paso los cálculo.

Funcicón para nuestro problema de mochila.

En GAs, tenemos una uestra o una población inicial de posibles soluciones para el problema dado, no nos interesa si cumplen o no con la restricción. Estas soluciones se someten a la recombinación y mutación (como en la genética natural), produciendo nuevos niños, y el proceso se repite a lo largo de varias generaciones. A cada individuo (o solución candidata) se le asigna un valor de aptitud (basado en su valor de función objetivo) y a los individuos más "aptos" se les da una mayor probabilidad de aparearse y producir más individuos hijos. Esto está en consonancia con la teoría darwiniana de la "supervivencia del más apto" (que en realidad es una falacia).

El algoritmo genalg intenta optimizar siempre hacia el valor mínimo. Por lo tanto, el valor se calcula por la función que construimos, y se multiplica por-1 si se quiere maximizar. Una configuración que lleva a exceder la restricción de peso devuelve un valor de 0 (también se puede dar un valor más alto o distinto). Esto último se conoce como tunnear el algoritmo.

```
> evalFunc <- function(x) {
+     current_solution_survivalpoints <- x %*% dataset$survivalpoints
+     current_solution_weight <- x %*% dataset$weight
+
+     if (current_solution_weight > weightlimit)
+         return(0) else return(-current_solution_survivalpoints)
+ }
```

nota: observe que en R el símbolo asterisco se utiliza para señalar multiplicación. En este caso si lo utilizo tendría el producto vectorial de dos matrices y obtendría error si no coinciden la cantidad de filas de la primerca con el número de columnas de la segunda. Para poder obtener la multiplicación de cada valor de una matriz (o como en este caso un vector) por su homólogo en la otra entidad se ha adoptado el signo asterísco pero rodeado por el de porcentaje. Esto hace el producti de las componentes una a una.

Corrida del modelo

A continuación, elegimos el número de iteraciones, diseñamos y ejecutamos el modelo.

```
> iter = 100
> GAmodel <- rbga.bin(size = 7, popSize = 200, iters = iter,
+       mutationChance = 0.01,  elitism = T, evalFunc = evalFunc)
>       plot(GAmodel)
```

Interpretación gráfica

Generamos un histograma de la población elitista que mejor resuleve el problema

```
> plot(GAmodel,type = "hist")
```

Vemos que es una población de comportamento discreto.

Encontrar la mejor solución

Qué significa la mejor solución?

```
> solution = c(1, 1,0, 1, 1, 1, 1)
> dataset[solution == 1, ]
```

```
          item survivalpoints weight
1  pocketknife            10      1
2        beans            20      5
4       unions             2      1
5 sleeping bag            30      7
6         rope            10      5
7      compass            30      1
```

La mejor solución se encuentra en 1111101. Esto nos lleva a tomar los siguientes artículos con nosotros en nuestro viaje a la naturaleza.

```
> cat(paste(solution %*% dataset$survivalpoints, "/", sum(dataset$survivalpoints)))

102 / 117
```

Resolver problemas difíciles

En Ciencias de la computación, hay un gran conjunto de problemas, que son NP-Hard. Lo que esto significa esencialmente es que, incluso los sistemas de computación más poderosos toman un tiempo muy largo (¡ incluso años!) para resolver ese problema. En tal escenario, GAs demuestra ser una herramienta eficiente para proporcionar soluciones utilizables casi óptimas en un corto período de tiempo.