

# **Ferramentas en para a resolución de problemas de Investigación de Operacóns**

**Alejandro Saavedra-Nieves**

**Ano 2022**

# **Ferramentas en para a resolución de problemas de Investigación de Operacíons**

Ferramentas en  para a resolución de problemas  
de Investigación de Operaciones

Elaborado por:  
Alejandro Saavedra-Nieves

I.S.B.N.: 978-84-09-38447-1  
D.L.: C 242-2022

# Contidos

O paquete lpSolveAPI de 	6
O paquete nloptr de 	10
O paquete igraph de 	14
O paquete CoopGame de 	19
O paquete InventoryModel de 	28
O paquete ProjectManagement de 	36



## O paquete lpSolveAPI de

### A optimización con restricións

Na optimización matemática, a optimización con restricións é o proceso de optimización dunha función objetivo con respecto a algunas variables con restricións nas mesmas.

- ▶ A función obxectivo é, ou ben unha función de custo ou unha función de enerxía, ou unha función de utilidade, que ha de ser maximizada.
- ▶ As restricións establecen condicións para as variables baixo estudo e para as que se require o seu cumprimento.

$$\begin{aligned} \max / \min \quad & f(x_1, \dots, x_n) \\ \text{suxeto a} \quad & g_k(x_1, \dots, x_n) = 0, \text{ con } k = 1, \dots, m \\ & h_k(x_1, \dots, x_n) \leq 0, \text{ con } k = 1, \dots, s \end{aligned}$$

## Pasos na modelización dun problema de optimización

---

A modelización axeitada nun problema de investigación de operacións, aínda que é difícil, garante unha aproximación realista das solucións que propón.

- ▶ **Definición do problema.** Definición dos obxectivos e dos elementos identificativos do problema: descripción das variables de decisión, determinación do obxectivo e das súas restricións.
- ▶ **Construcción do modelo.** Traducir o problema a linguaxe matemática.
- ▶ **Solución do modelo.** Usando algoritmos de optimización, obter unha solución. Cobra interese o comportamento dos parámetros do modelo (análise de sensibilidade).
- ▶ **Validación do modelo.** Chequeo do bo comportamento do modelo en situacións reais.
- ▶ **Implementación.** Tradución do modelo e asumir os seus resultados como solución.

## A programación lineal

---

A **programación lineal** (*Linear programming, LP*) é unha técnica de investigación de operacións usada cando a función obxectivo e as restricións son lineais (nas variables) e cando as variables son de tipo continuo.

- ▶ Xerarquicamente, a programación lineal pode considerarse a técnica de investigación operativa más sinxela.

O paquete `lpSolveAPI` de R contén funcións específicas para resolución de problemas de programación lineal.

```
library(lpSolveAPI)
```

## Construindo un problema de PL con lpSolveAPI

Un obxecto tipo “problema de programación lineal” con  $m$  restriccións e  $n$  variables de decisión pode crearse coa función de `make.lp()` de `lpSolveAPI`. Por exemplo,

```
omeulp <- make.lp(3, 2)
```

permitiría crear un obxecto deste tipo con 3 restricións e 2 variables de decisión. A función `resize.lp()` permitiría a redimensión de tal obxecto (de ser necesario).

```
> omeulp
Model name:
          C1    C2
Minimize   0    0
R1          0    0 free  0
R2          0    0 free  0
R3          0    0 free  0
Kind       Std   Std
Type      Real  Real
Upper     Inf   Inf
Lower     0    0
```

## Construindo un problema de PL con lpSolveAPI

Os próximos pasos requieren a definición explícita do problema de programación lineal. Supoñamos que desexamos resolver o seguinte problema.

$$\begin{aligned} \min \quad & -2x_1 - x_2 \\ \text{suxeto a} \quad & \\ & x_1 + 3x_2 \leq 4 \\ & x_1 + x_2 \leq 2 \\ & 2x_1 \leq 3 \\ & \text{con } x_1 \geq 0 \text{ e } x_2 \geq 0. \end{aligned}$$

```
set.column(omeulp, 1, c(1, 1, 2))      # coef. da columna 1
set.column(omeulp, 2, c(3, 1, 0))      # coef. da columna 2
set.objfn(omeulp, c(-2, -1))          # coef. da func. obx.
set.constr.type(omeulp, rep("<=", 3)) # sentido das desigualdades
set.rhs(omeulp, c(4, 2, 3))           # lados dereitos
```

## Resolvendo un problema de PL con lpSolveAPI

```
> omeulp
Model name:
      C1     C2
Minimize -2    -1
R1       1     3 <= 4
R2       1     1 <= 2
R3       2     0 <= 3
Kind     Std   Std
Type     Real  Real
Upper    Inf   Inf
Lower    0     0
> solve(omeulp)           # resolvo o meu lp
[1] 0
> get.objective(omeulp)   # canto vale a func. obxectivo?
[1] -3.5
> get.variables(omeulp)   # e as variables no óptimo?
[1] 1.5 0.5
> get.constraints(omeulp) # e as restricións?
[1] 3 2 3
```

## Construindo un problema de PL con lpSolveAPI

Por defecto, as variables son continuas. Sen embargo, a función `set.type()` permite cambiar a tipoloxía desta variable.

```
set.type(omeulp, 2, "integer") # A segunda variable é enteira
```

O rango das variables tamén pode ser indicado. Para iso, utilizamos a función `set.bounds()`

```
set.bounds(omeulp, lower = c(-5), columns = 1) # cota inferior
set.bounds(omeulp, upper = c(4), columns = 1)   # cota superior
```

## O paquete nlopt de

## A optimización con restricións: o caso non lineal

---

Na optimización matemática, a optimización con restricións é o proceso de optimización dunha función objetivo con respecto a algunas variables con restricións nas mesmas.

- ▶ A función obxectivo é, ou ben unha función de custo ou unha función de enerxía, ou unha función de utilidade, que ha de ser maximizada.
- ▶ As restricións establecen condicións para as variables baixo estudo e para as que se require o seu cumprimento.

$$\begin{aligned} \max / \min \quad & f(x_1, \dots, x_n) \\ \text{suxeto a} \quad & g_k(x_1, \dots, x_n) = 0, \text{ con } k = 1, \dots, m \\ & h_k(x_1, \dots, x_n) \leq 0, \text{ con } k = 1, \dots, s \end{aligned}$$

**Que pasa cando a función obxectivo ou as restricións son non lineais?**

## Resolvendo un problema de PNL con nloptr

O paquete `nloptr` de R contén funcións específicas para resolución de problemas de programación non lineal.

```
library(nloptr)
```

$$\begin{array}{ll} \min & x_1x_4(x_1 + x_2 + x_3) + x_3 \\ \text{suxeto a} & \\ & 25 - x_1x_2x_3x_4 \leq 0 \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 - 40 = 0 \end{array}$$

## Resolvendo un problema de PNL con nloptr

## Introducindo a función obxectivo

## Resolvendo un problema de PNL con nloptr

---

### Introducindo as restricciones

```
# constraint functions
# inequalities
eval_g_ineq <- function(x) {
    constr <- c(25-x[1]*x[2]*x[3]*x[4] )
    grad <- c(-x[2]*x[3]*x[4], -x[1]*x[3]*x[4],
              -x[1]*x[2]*x[4], -x[1]*x[2]*x[3])
    return(list("constraints"=constr, "jacobian"=grad))
}
# equalities
eval_g_eq <- function(x) {
    constr <- c(x[1]^2+x[2]^2+x[3]^2+x[4]^2-40)
    grad <- c(2.0*x[1], 2.0*x[2], 2.0*x[3], 2.0*x[4])
    return(list("constraints"=constr, "jacobian"=grad))
}
```

## Resolvendo un problema de PNL con nloptr

---

### Introducindo as opciones

```
# initial values
x0 <- c( 1, 5, 5, 1 )
# lower and upper bounds of control
lb <- c( 1, 1, 1, 1 )
ub <- c( 5, 5, 5, 5 )
local_opts <- list("algorithm" = "NLOPT_LD_MMA", "xtol_rel" = 1.0e-7)
opts <- list("algorithm"="NLOPT_LD_AUGLAG", "xtol_rel"=1.0e-7,
            "maxeval"=1000, "local_opts"=local_opts )
```

### Resolvendo o problema

```
res<-nloptr(x0=x0, eval_f=eval_f, lb=lb, ub=ub, eval_g_ineq=eval_g_ineq,
             eval_g_eq=eval_g_eq, opts=opts)
print(res)
```

## Outros paquetes de R para a optimización

- ▶ `minqa`: algoritmos de optimización mediante aproximación cuadrática.
- ▶ `ompr`: implementación de modelos de programación lineal e enteira de forma alxebraica.
- ▶ `Rglpk`: R/GNU interface.
- ▶ `ROI`: estrutura específica para o manexo de problemas de optimización en R.

## A función optimize de R

A función `optimize()` busca no intervalo marcado o mínimo/máximo da función  $f$ .

### **Exemplo 1**

```
optimize(function(x) x^2-1, lower = 0, upper = 10)
```

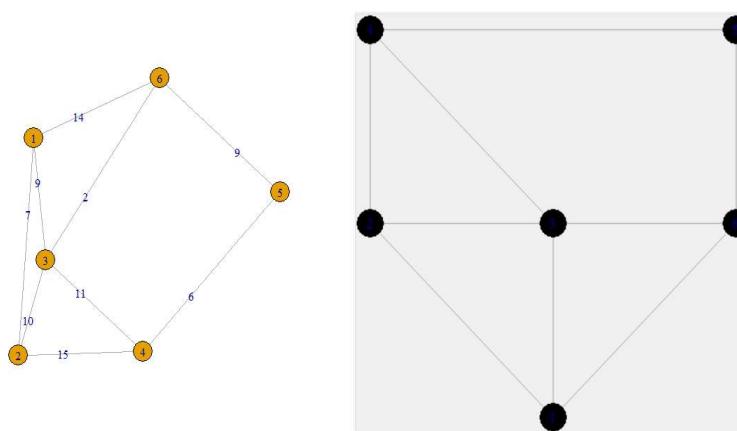
### **Exemplo 2**

```
f <- function (x, a) (x - a)^2
optimize(f, c(0, 1), tol = 0.0001, a = 1/3)
```

## O paquete `igraph` de

### A análise de redes

- ▶ Un dos paquetes que nos permite realizar análisis de redes usando R é o paquete `igraph`.
- ▶ Proporciona rutinas e funcións para crear e manipular grafos con sinxeleza.



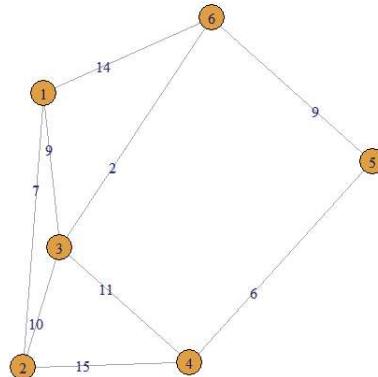
## A análise de redes con igraph

- ▶ Como fío conductor, imos crear un grafo non dirixido ponderado.
- ▶ Un dos obxectivos será calcular o camiño máis corto entre dous vértices.
- ▶ Para calcular esta ruta, igraph emprega o algoritmo de Dijkstra.

### Construcción dun grafo con igraph

- ▶ Para obter un grafo non dirixido ponderado necesitamos crear un listado de arestas (edgelist) cos pesos de cada aresta (ou distancias entre nodos).
- ▶ Creamos tres vectores, os dous primeiros representan a conexión (as arestas) entre os vértices e o terceiro os pesos de cada aresta.

```
var1<-c(1,1,2,2,3,3,4,5,6)
var2<-c(2,3,3,4,4,6,5,6,1)
weight<-c(7,9,10,15,11,2,6,9,14)
data<-data.frame(var1,var2,weight)
```



## A análise de redes con igraph

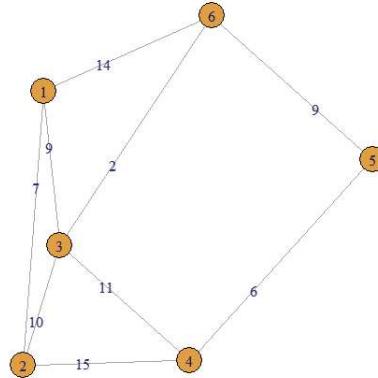
### Construcción dun grafo con igraph

```
g <- graph.data.frame(data, directed = FALSE)      # Crea igraph
class(g)                                            # Clase do obxecto
V(g)$name                                           # Nomes dos vértices
E(g)$weight                                         # Peso das arestas
tkplot(g)                                            # Gráfico dinámico
plot(g, edge.label = paste(E(g)$weight, sep = "")) # Gráfico
```

## O camiño más corto con igraph

Calculamos o camiño más corto en función da distancia total percorrida (suma de pesos) e a secuencia de vértices do mesmo.

```
> sp <- shortest.paths(g, v = "1", to = "5")
> sp[]
5
1 20
> gsp <- get.shortest.paths(g, from = "1"
+ to = "5")
> V(g)[gsp$xpath[[1]]]
+ 4/6 vertices, named, from 5c53cbb:
[1] 1 3 6 5
```



## Obtendo información do grafo...

### A matriz de adxacencia (?)

A función `get.adjacency()` devolve a matriz de adxacencia asociada o grafo (`attr=NULL`).

```
> adj <- get.adjacency(g, attr='weight', sparse = FALSE)
> adj
1 2 3 4 5 6
1 0 7 9 0 0 14
2 7 0 10 15 0 0
3 9 10 0 11 0 2
4 0 15 11 0 6 0
5 0 0 0 6 0 9
6 14 0 2 0 9 0
```

## Obtendo información do grafo...

---

### A matriz de distancias

A función shortest.paths() devolve a matriz de distancias co camiño más corto entre cada par de nodos.

```
> distMatrix <- shortest.paths(g, weights = E(g)$weight)
> distMatrix
 1 2 3 4 5 6
1 0 7 9 20 20 11
2 7 0 10 15 21 12
3 9 10 0 11 11 2
4 20 15 11 0 6 13
5 20 21 11 6 0 9
6 11 12 2 13 9 0
```

## Obtendo información do grafo...

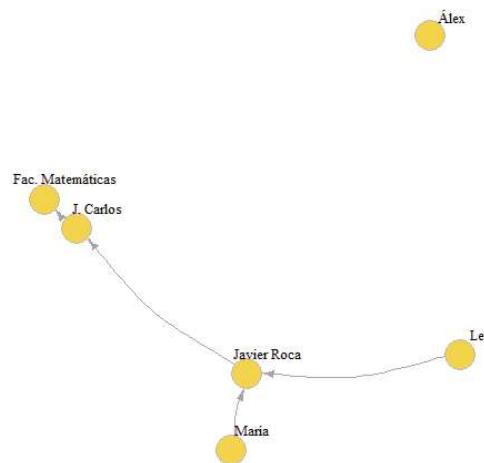
---

### Tódolos camiños más cortos

A función get.all.shortest.paths() devolve tódolos camiños más cortos desde un nodo.

```
allsp <- get.all.shortest.paths(g, from = "1")
```

## E se o grafo é dirixido...?



```
g4 <- graph(c("Leti","Javier Roca","María","Javier Roca",
              "Javier Roca","J. Carlos","J. Carlos","Fac. Matemáticas"),
              isolates=c("Álex"))
```

## E se o grafo é dirixido...?

```
plot(g4, edge.arrow.size=.5, vertex.color="gold", vertex.size=15,
      vertex.frame.color="gray", vertex.label.color="black",
      vertex.label.cex=0.8, vertex.label.dist=2, edge.curved=0.2)

V(g)$name                                # Nomes dos vértices
E(g)$weight                               # Peso das arestas
```

## O paquete CoopGame de

## Xogos TU

### Teoría de Xogos: teoría matemática de problemas de decisión

Un **xogo TU** é un par  $(N, v)$ , onde:

- ▶  $N$  é o conxunto de xogadores e,
- ▶  $v : 2^N \rightarrow \mathbb{R}$  é a súa función característica con  $v(\emptyset) = 0$ .

- ▶ 3 grupos de investigación: Milán (grupo 1), Génova (group 3), e Santiago de Compostela (grupo 3).

- ▶ Organizar a viaxe dun profesor visitante minimizando os custos.

Entón,  $N = \{1, 2, 3\}$ . O custo de visitar cada posible grupo é:

$$c(\{1\}) = 1500, c(\{2\}) = 1600, c(\{3\}) = 1900,$$

$$c(\{1, 2\}) = 1600, c(\{1, 3\}) = 2900, c(\{2, 3\}) = 3000, \text{ e } c(N) = 3000.$$

$c(S)$  é o custo mínimo por visitar as cidades en  $S$ .

## Xogos TU

Alonso-Meijide e Bowles (2005) estudaron o Fondo Monetario Internacional (Xaneiro, 2002).



País	Dereitos de voto	Índice de poder
United States	17.11	0.6471
Japan	6.14	0.1752
Germany	6.00	0.1712
France	4.95	0.1411
United Kingdom	4.95	0.1411
Austria	0.87	0.0248
Belarus	0.19	0.0054
Belgium	2.13	0.0608
Czech Republic	0.39	0.0111
Hungary	0.49	0.0140
Kazakhstan	0.18	0.0051
Luxembourg	0.14	0.0040
Slovak Republic	0.18	0.0051
Slovenia	0.12	0.0034
Turkey	0.46	0.0131
Armenia	0.05	0.0014
Bosnia and Herzegovina	0.09	0.0026
Bulgaria	0.31	0.0088
...	...	...



Alonso-Meijide, J. M., Bowles, C. (2005). Generating functions for coalitional power indices: An application to the IMF. Annals of Operations Research, 137(1), 21-44.

## Xogos TU

	(H) Mrs. Hewitt	(I) Mr. Isaacs	(J) Mrs. Jones	(K) Mr. Kent
A printer				
	1 hora 1 u.m. por hora	5 horas 10 u.m. por hora	6 horas 8 u.m. por hora	2 horas 3 u.m. por hora

**Cales son os costes de procesado?**

(H)-(I)-(J)-(K):  $1 \cdot 10 + 10 \cdot 6 + 8 \cdot 12 + 3 \cdot 14 = 199$  u.m.  
(I)-(K)-(J)-(H):  $10 \cdot 5 + 3 \cdot 7 + 8 \cdot 13 + 1 \cdot 14 = 189$  u.m.

Que orde minimiza o custo total?



Curiel, I. (1997). Cooperative Game Theory and Applications: Cooperative Games Arising from Combinatorial Optimization Problems (Vol. 16). Springer Science & Business Media.

## Xogos TU



The collage includes:

- 20 minutos** (top left): "Santander y Fridman acuerdan que evita el acuerdo con los acreedores de Día". EP 20.05.2019 - 18:19H. Botín considera un "tratamiento justo" tras eliminar la disc.
- Expansión** (top right): "Quién es quién en el laberinto legal de Thomas Cook". A. GÁLISSEO | MADRID 10 OCT. 2019 - 00:29. TRANSPORTE Y TURISMO.
- Pescanova en la EMV** (bottom left): "Guerra abierta en el consejo de la compañía". La empresa quiere despedir al auditor y que una comisión de expertos investigue las irregularidades. Pescanova está ahogando a Pescanova. En un consejo que duró más de tres horas, los vocales pidieron exolicaciones avar al presidente de la
- ÚLTIMA HORA** (bottom right): Dos muertos en un atentado contra una sinagoga en Alemania. Renault relevará a su CEO para dejar atrás la era Ghosn y acercarse a Noto. La planta de Valladolid.

**seis conceptos que aprendimos con la gran crisis financiera**

Formaron parte de una de las mayores crisis económicas tras el colapso de un sistema financiero que ya se germinaba antes de 2008 debido a una burbuja inmobiliaria que tuvo graves consecuencias.

## Xogando con CoopGame

O paquete CoopGame permite:

- ▶ xerar xogos TU con estrutura especial, coma os xogos de bancarrota, xogos de custos, xogos de maioría ponderada ou xogos de unanimidade.
- ▶ chequear propiedades dos xogos, como por exemplo a súa superaditividade, convexidade ou o seu carácter equilibrado.
- ▶ computar solucións de xogos tipo conxunto, coma o core e outros conceptos relacionados.
- ▶ obter solucións tipo punto, coma o valor de Shapley, o nucleolus ou números índice.
- ▶ representación gráfica das solucións para xogos de 3 e 4 xogadores.

## Definindo os xogos TU

---

A función `createBitMatrix(n,v)` permite construir unha matriz 0-1, onde cada fila representa unha coalición de  $N$ .

```
> (Maschler <- c(0,0,0,60,60,60,72))
[1] 0 0 0 60 60 60 72
> createBitMatrix(n=3,Maschler)
      cVal
[1,] 1 0 0     0
[2,] 0 1 0     0
[3,] 0 0 1     0
[4,] 1 1 0     60
[5,] 1 0 1     60
[6,] 0 1 1     60
[7,] 1 1 1     72
```

## Algunhas funcionalidades de CoopGame

---

Sexa  $(N, v) \in G^N$  un xogo TU. A súa 0-normalización vén dada polo xogo  $(N, w) \in G^N$  definido por:

$$w(S) = v(S) - \sum_{i \in S} v(i), \text{ para cada } S \subseteq N.$$

```
> (w <- getZeroNormalizedGameVector(Maschler))
[1] 0 0 0 60 60 60 72
```

Ó dividir o xogo 0-normalizado por  $v(N)$ , obtemos a 0,1-normalización de  $(N, v)$ .

```
> (w01 <- getZeroOneNormalizedGameVector(Maschler))
[1] 0.0000000 0.0000000 0.0000000 0.8333333 0.8333333 0.8333333 1.0000000
```

## Estudando os xogos TU

---

Sexa  $(N, v) \in G^N$  un xogo TU. Para cada  $i \in N$ , defínese a contribución marxinal dese xogador a unha certa coalición  $S \subseteq N \setminus \{i\}$  como:

$$v(S \cup \{i\}) - v(S)$$

```
> MC <- getMarginalContributions(Maschler)
> names(MC)
[1] "A"                  "combinations"      "marginal_values"
> MC$marginal_values
[,1] [,2] [,3]
[1,]    0   60   12
[2,]    0   12   60
[3,]   60    0   12
[4,]   12    0   60
[5,]   60   12    0
[6,]   12   60    0
```

## Estudando os xogos TU

---

Sexa  $(N, v) \in G^N$  un xogo TU. O vector de utopía vén dado por

$$M_j = v(N) - v(N \setminus \{j\}), \text{ con } j \in N.$$

```
> (M <- getUtopiaPayoff(Maschler))
[1] 12 12 12
```

O vector de dereitos minimais vén dado por

$$m_j = \max_{S:j \in S} R(S, j), \text{ para cada } j \in N,$$

onde  $R(S, j) = v(S) - \sum_{i \in S, i \neq j} M_i$ .

```
> (m <- getMinimalRights(Maschler))
[1] 48 48 48
```

## Estudando os xogos TU

Sexa  $(N, v) \in G^N$  un xogo TU. O exceso dunha certa coalición  $S \subseteq N$  con respecto a unha asignación  $x \in \mathbb{R}^N$  obtense como

$$e(S, x) = v(S) - \sum_{i \in S} x_i.$$

```
> x<-c(2,30,40)
> (ec <- getExcessCoefficients(Maschler,x))
[1] -2 -30 -40  28  18 -10   0
```

O exceso per capita, calculado a partir dos excesos, dividíndoos polo cardinal de cada coalición, obtense coa función `getPerCapitaExcessCoefficients`.

## Estudando as propiedades dos xogos TU

O paquete `CoopGame` dispón de 17 funcións para o chequeo de propiedades ben coñecidas dos xogos TU.

Como argumento, as devanditas funcións só requiren a definición do xogo TU como da maneira xa mencionada.

Estas proporcionan como resposta, unha variable lóxica (`TRUE` ou `FALSE`) indicando (ou non) o cumprimento da propiedade baixo estudo.

## Estudando algunas propiedades dos xogos TU

Propiedad		Función de R
Non negatividade	$v(S) \geq 0$ , para todo $S \subseteq N$	isNonnegativeGame()
Esencial	$v(N) > \sum_{i=1}^n v(\{i\})$	isEssentialGame()
Monotonía	$S \subseteq T \subseteq N \implies v(S) \leq v(T)$	isMonotonicGame()
Superaditivididade	$v(S \cup T) \geq v(S) + v(T)$ , para $S, T \subseteq N$ , con $S \cap T = \emptyset$ .	isSuperadditiveGame()
Equilibrado	$i C(N, v) \neq \emptyset?$	isBalancedGame()
Convexo	$v(S \cup T) + v(S \cap T) \geq v(S) + v(T)$ , con $S, T \subseteq N$	isConvexGame()

```
> isConvexGame(Maschler)
[1] FALSE
> isBalancedGame(Maschler)
[1] FALSE
```

## Solucións de xogos TU

Un dos obxectivos fundamentais dos xogos TU é o de [repartir os beneficios/custos que resultan da cooperación](#) de todos aqueles xogadores involucrados.

Na literatura, a definición de moitos destes repartos vén xustificada por criterios como a da xustiza, da equidade ou da estabilidade, entre outros. De maneira xeral, estes poden agruparse en dous grandes grupos dacordo á súa cardinalidade.

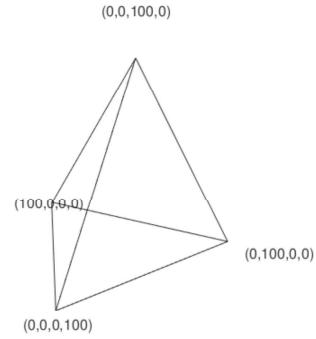
- ▶ **Solucións tipo conxunto:** o conxunto de imputacións, o core, o core-cover, o conxunto de Weber, entre outros.
- ▶ **Solución tipo punto:** o valor de Shapley, o valor de Banzhaf, o nucleolus, o core-center, entre outros.

## Solucións de xogos TU: o conxunto de imputacións

Sexa  $(N, v) \in G^N$ . O **conxunto de imputacións** de  $(N, v)$  vén determinado por:

$$I(v) = \{x \in \mathbb{R}^N : \sum_{i \in N} x_i = v(N) \text{ e } x_i \geq v(i)\}.$$

```
> v<-c(0,0,0,0,3,5,7,5,
4,6,20,30,40,50,100)
> imputationsetVertices(v)
[,1] [,2] [,3] [,4]
[1,]    0    0    0   100
[2,]    0    0   100    0
[3,]    0   100    0    0
[4,]   100    0    0    0
> belongsToImputationset(c(10,25,25,40),v)
[1] TRUE
> drawImputationset(v)
```

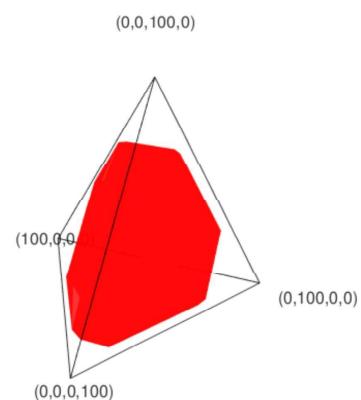


## Solucións de xogos TU: o núcleo

Sexa  $(N, v) \in G^N$ . O **núcleo** (*core*) de  $(N, v)$  vén determinado por:

$$C(v) = \{x \in \mathbb{R}^N : \sum_{i \in N} x_i = v(N) \text{ e } \sum_{i \in S} x_i \geq v(S) \text{ para todo } S \subset N\}.$$

```
v<-c(0,0,0,0,3,5,7,5,
4,6,20,30,40,50,100)
coreVertices(v)
belongsToCore(c(10,25,25,40),v)
drawCore(v)
```



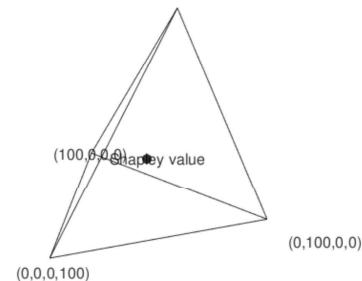
## Solucións de xogos TU: o valor de Shapley

Sexa  $(N, v) \in G^N$ . O **valor de Shapley**  $\Phi$  asigna a cada xogador  $i \in N$

$$\Phi_i(v) = \frac{1}{n!} \sum_{\sigma \in \Pi(N)} m_i^\sigma(v),$$

onde  $\Pi(N)$  é o conxunto de permutacións en  $N$ ,  $P_i^\sigma = \{j \in N : \sigma(j) < \sigma(i)\}$  e  $m_i^\sigma(v) = v(P_i^\sigma \cup i) - v(P_i^\sigma)$  para cada  $\sigma \in \Pi(N)$ .

(0,0,100,0)



```
> shapleyValue(v)
[1] 20.00000 22.83333 26.83333 30.33333
> drawShapleyValue(v)
```

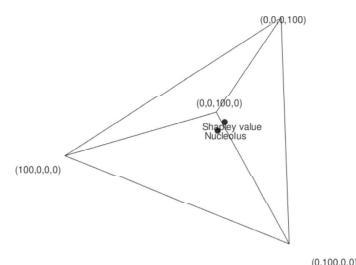
## Solucións de xogos TU: o nucleolus

Sexa  $(N, v) \in G^N$ . O **nucleolus** é aquela imputación que minimiza o vector de excesos ordenados en forma decrecente dacordo coa orde lexicográfica.

$x \succeq_L y$  se e soamente se

$x = y$  ou existe  $i \in \{1, \dots, n\}$  tal que, para cada  $j < i$ ,  $x_j = y_j$  e  $x_i > y_i$ .

```
> nucleolus(v)
[1] 25 25 25 25
> drawNucleolus(v, holdOn=TRUE)
```



## O paquete InventoryModel de

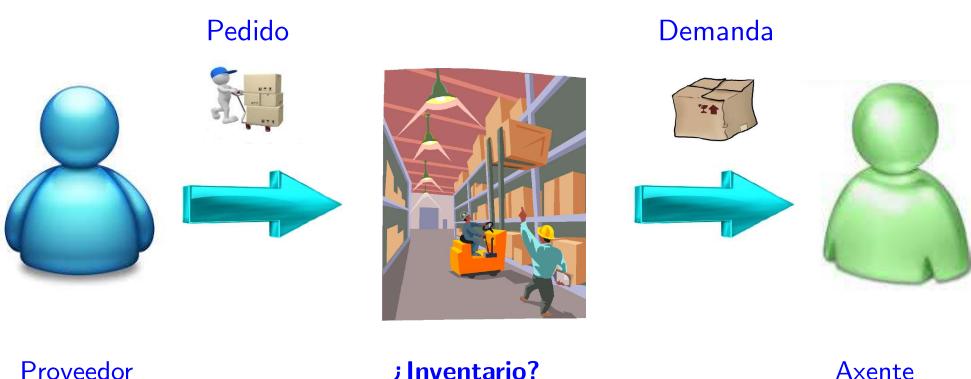


Saavedra-Nieves, A. (2018). Inventorymodel: an R Package for Centralized Inventory Problems. *The R Journal*, 10(1), 200.

## A Teoría de Inventarios

Que é o inventario?

É a cantidade de un ben ou recurso económico almacenado nun instante de tempo.



**Teoría de Inventarios**

Deseño e optimización dos sistemas de almacenaxe de bens para minimizar custos.

## Aplicacións da Teoría de Inventarios



## Os problemas de inventario centralizados

**Problema.** Algunxs axentes que se enfrentan a problemas de inventario individuais, deciden cooperar na realización de pedidos conxuntos para reducir custos.

### Pasos

- ▶ Modelización matemática da situación de inventarios.
- ▶ Determinación da política de inventario óptima para o grupo.
- ▶ Reparto dos custos conxuntos asociados á colaboración.

### Teoría de Xogos Cooperativos

- ▶ Elección dunha regra de asignación dos custos da colaboración entre os axentes.

## Os xogos de custo

### A cooperación

- ▶ A formación de subgrupos de xogadores, **coalicións**, fundamentan a definición dos xogos TU.

Un **xogo de custo TU** é un par  $(N, c)$ :

- ▶  $N := \{1, \dots, n\}$  é o conxunto de xogadores,
- ▶  $c : 2^N \rightarrow \mathbb{R}$  é a **función característica** do xogo, con  $c(\emptyset) = 0$ .

Para cada coalición  $S \subseteq N$ ,  $c(S)$  denota os custos asociados á formación de  $S$ .

## A cooperación nos problemas de inventario centralizados

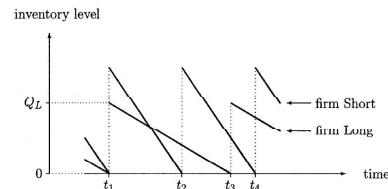
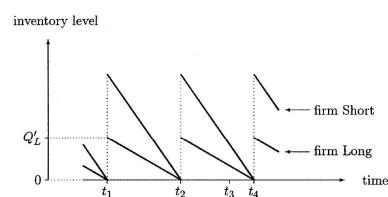
### Baixo a cooperación...

- ▶ Os axentes deben coordinarse na realización dun **único pedido conxunto**.
- ▶ Deben **facer pedidos ó mesmo tempo**, e polo tanto, o mesmo número deles.

Denótase por  $d_i$  a demanda de produto e  $Q_i$  a cantidad de inventario do axente  $i$ .

Se  $i$  e  $j \in N$  cooperan, a igualdade no número de pedidos é equivalente a

$$\frac{d_i}{Q_i} = \frac{d_j}{Q_j}.$$



## O modelo EOQ



A. Meca, J. Timmer, I. García-Jurado, P. Borm (2004). Inventory games. European Journal of Operational Research, **156**(1), 127–139.

Para cada axente  $i \in N$ ,

- ▶ A **demanda** a cubrir é  $d_i > 0$  unidades por unidad de tempo.
- ▶ O **custo de pedido**, abonado con cada novo pedido e igual a  $a > 0$ .
- ▶ O **custo de almacenamiento** dunha unidade de producto é  $h_i > 0$ .
- ▶ A **cantidad de inventario** necesaria é  $Q_i > 0$  ( $i \in N$ ).

## O modelo EOQ

O **custo da colaboración** queda determinado por:

- ▶ O custo medio asociado á realización de  $ad_1/Q_1$  pedidos.
- ▶ A suma dos custos individuais do almacenamento do producto.

$$C(Q_1, \dots, Q_n) = a \frac{d_1}{Q_1} + \sum_{i \in N} h_i \frac{Q_i}{2}.$$

A cantidad óptima de pedido para  $i \in N$ , é  $Q_i^* = \sqrt{\frac{2ad_i^2}{\sum_{j \in N} h_j d_j}}$ , con un custo igual a

$$c(Q_1^*, \dots, Q_n^*) = a \frac{d_1}{Q_1^*} + \sum_{i \in N} h_i \frac{Q_i^*}{2}.$$

## O modelo EOQ

---

### O xogo de custo TU

O xogo de custo  $(N, c)$  asociado a este modelo asínalle a cada  $S \subset N$ ,  $c(S) = c(Q_1^*, \dots, Q_s^*)$ .

A **regra de reparto SOC** (*Share the Ordering Costs*) propón, para cada axente  $i \in N$ ,

$$\sigma_i(c) = \frac{c^2(i)}{c(N)}.$$

## O modelo EOQ en InventoryModel

---

```
EOQcoo(n=3,a=600,d=c(500,300,400),h=c(9.6,11,10))
SOC(n=3,a=600,d=c(500,300,400),h=c(9.6,11,10),model="EOQ",cooperation=1)
```

$S$	Política óptima				Regla SOC		
	$Q_1^*$	$Q_2^*$	$Q_3^*$	$C(Q_i^*)_{i \in S}$	$\sigma_1(c)$	$\sigma_2(c)$	$\sigma_3(c)$
$\emptyset$	-	-	-	0	-	-	-
$\{1\}$	250	-	-	2400	2400	-	-
$\{2\}$	-	180.91	-	1989.98	-	1989.98	-
$\{3\}$	-	-	219.09	2190.89	-	-	2190.89
$\{1, 2\}$	192.45	115.47	-	3117.69	1847.52	1270.17	-
$\{1, 3\}$	184.64	-	147.71	3249.66	1772.58	-	1477.10
$\{2, 3\}$	-	121.63	162.18	2959.73	-	1337.96	1621.77
$\{1, 2, 3\}$	157.46	94.48	125.98	3810.51	1511.61	1039.23	1259.67

## Os sistemas de inventario e transporte



M.G. Fiestras-Janeiro, I. García-Jurado, A. Meca, M.A. Mosquera (2012). Cost allocation in inventory transportation systems. *Top*, **20**(2), 397–410.

Para cada axente  $i \in N$ ,

- ▶ A **demand**a a cubrir é  $d_i > 0$  unidades por unidad de tempo.
- ▶ O **custo fixo de pedido**, abonado con cada novo pedido e igual a  $a > 0$ .
- ▶ O **custo variable de pedido**,  $a_i > 0$ , vinculado á distancia do axente  $i$  ó proveedor.
- ▶ O **custo de almacenamiento** dunha unidade de producto é  $h_i > 0$ .
- ▶ A **cantidade de inventario** necesaria é  $Q_i > 0$  ( $i?$ ).

## Os sistemas de inventario e transporte

O **custo da colaboración** queda determinado por:

- ① O custo por un novo pedido é  $a + a_S$ , con  $a_S = \max\{a_i : i \in S\}$ .
- ② O custo medio pola realización de  $(a + a_S)d_1/Q_1$  pedidos.
- ③ A suma dos custos individuais do almacenamento do producto.

$$C(Q_1, \dots, Q_n) = (a + a_S) \frac{d_1}{Q_1} + \sum_{i \in N} h_i \frac{Q_i}{2}. \quad (1)$$

A cantidade óptima de pedido para  $i \in N$ , é  $Q_i^* = \sqrt{\frac{2(a+a_N)d_i^2}{\sum_{j \in N} h_j d_j}}$ , con un custo igual a

$$c(Q_1^*, \dots, Q_n^*) = (a + a_N) \frac{d_1}{Q_1^*} + \sum_{i \in N} h_i \frac{Q_i^*}{2}.$$

## Os sistemas de inventario e transporte

---

### O xogo de custo TU

O xogo de custo  $(N, c)$  asociado a este modelo asignalle a cada  $S \subset N$ ,  $c(S) = c(Q_1^*, \dots, Q_s^*)$ .

A **regra da liña** propón, para cada axente  $i \in N$ ,

$$L_i(N, \mathcal{I}) = \frac{1}{|\Pi(N, \mathcal{I})|} \sum_{\sigma \in \Pi(N, \mathcal{I})} c(P_i^\sigma \cup i) - c(P_i^\sigma),$$

con  $\Pi(N, \mathcal{I})$  o conxunto das permutacións que invirten a orde dos  $\{a_j\}_{j \in N}$  e  $P_i^\sigma$  os predecesores de  $i$  na permutación  $\sigma$ .

## Os sistemas de inventario e transporte en R

---

```
> STIcoo(n=3,a=200,av=c(300,300,900),d=c(90,80,20),h=c(0.06,0.06,0.1))
> linerulecoalitional(n=3,a=200,av=c(300,300,900),d=c(90,80,20),
+ h=c(0.06,0.06,0.1))
```

$S$	$\emptyset$	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	$N$
$c(S)$	0	73.49	69.28	66.33	101.00	127.59	122.31	163.83
Ax. 1	-	1224.75	-	-	891.13	1551.81	-	1208.58
Ax. 2	-	-	1154.70	-	792.12	-	1438.95	1074.29
Ax. 3	-	-	-	663.33	-	344.85	359.74	268.57

## Outros modelos de inventario en R

- ▶ Modelos con déficits.
- ▶ Modelos sen custos de almacenamento.
- ▶ Modelos con déficits e sen custos de almacenamento.
- ▶ Modelos sen custos de almacenamento e con custos de transporte.
- ▶ ...

Máis información en

<https://cran.r-project.org/web/packages/InventorymodelPackage/index.html>

## O paquete ProjectManagement de



Gonçalves-Dosantos, J. C., García-Jurado, I., & Costa, J. (2020).  
ProjectManagement: an R Package for Managing Projects. *The R Journal*, 12(1),  
419-436.

## A xestión de proxectos

- ▶ A xestión de proxectos é unha parte importante no coñecemento dos sistemas de planificación, organización e control dos recursos para acadar uns obxectivos marcados.
- ▶ Os métodos más coñecidos son os PERT (*Program Evaluation and Review Technique model*) e CPM (*Critical Path Method*).
- ▶ Os modelos PERT/CPM analizan as cuestiós que caracterizan ó proxecto (tempo requerido por tarefa) e computan o tempo mínimo requerido para o proxecto.
- ▶ Identifican as actividades críticas, que son claves na modificación do tempo mínimo requerido para a execución do proxecto.
- ▶ Tamén é fundamental identificar aquelas actividades non-criticas (*slack/folguras*).

## A xestión de proxectos

---

- ▶ A xestión de proxectos trata habitualmente o problema de redistribuir os recursos.
- ▶ En ocasións, é conveniente reducir o tempo dunha actividade ó incrementar os seus costes.
- ▶ Noutras, cando a disponibilidade dos recursos é limitada nun periodo de tempo, cómpre regular o uso de tales recursos.
- ▶ Todas estas situacóns requieren a replanificación do proxecto.

## Algunhas ferramentas para a planificación de proxectos

---

- ▶ Un proxecto de software no eido da xestión de proxectos é o ben coñecido Microsoft Project.
- ▶ Esta ferramenta foi deseñada para crear e controlar un proxecto, a través da asignación de recursos a tarefas, xestionando presupostos e cargas de traballo e permitindo o seguimento da súa evolución.
- ▶ Esta non é a única ferramenta, xa que existen alternativas en código aberto, como OpenProj, PpcProject ou ProMes.

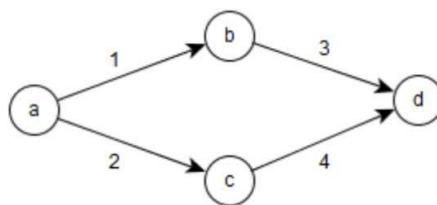
## A planificación de proxectos con ProjectManagement

- ▶ Permite a determinación das actividades críticas, as folguras de cada variable, a duración mínima do proxecto e os tempos *early* e *last* de cada actividad.
- ▶ Ofrece a representación gráfica do proxecto e da súa planificación, así coma o manexo dos costes para reducir o tempo mínimo do proxecto e dos recursos.
- ▶ Coñecida a duración das actividades, distribué o posible retraso do proxecto entre as diferentes actividades.
- ▶ Tamén inclúe a posibilidade de considerar que as duracións das actividades sexan variables aleatorias.

## A planificación de proxectos: notación previa

Sexa  $X$  un conxunto finito e  $N$  é un conxunto ordenado de  $n$  pares  $(x_1, x_2)$ , con  $x_1, x_2 \in X$ .

- ▶  $G$  é un grafo dirixido, con  $X$  o conxunto de nodos e  $N$  o conxunto de arcos.



- ▶ O arco  $i$  defínese por  $(x_{i1}, x_{i2})$ , sendo  $x_{i1} \in X$  o nodo inicial e  $x_{i2} \in X$  o nodo no que remata.
- ▶  $x_s \in X$ : nodo fonte, non existe ningún arco  $i$  tal que  $x_{i2} = x_s$ .
- ▶  $x_e \in X$ : nodo sumidoiro, non existe ningún arco  $i$  tal que  $x_{i1} = x_e$ .
- ▶ Dado  $x \in X$ , os predecesores de  $x$  son  $Pred(x) = \{i \in N | x_{i2} = x\}$  e os sucesores de  $x$ ,  $Suc(x) = \{i \in N | x_{i1} = x\}$ .

## A planificación de proxectos: notación previa

---

- ▶ Un proxecto  $P$  é a tupla  $P = (G, x^0)$ :
- ▶  $G = (X, N)$  é un grafo dirixido sen ciclos, cun nodo fonte e un nodo sumidoiro.
- ▶  $x^0 \in \mathbb{R}_+^n$  é un vector de duracións previstas non negativas.
- ▶  $N$  representa o conxunto de actividades no proxecto.
- ▶ O obxecto fundamental consiste en determinar a duración mínima de  $P$  para completar todas as actividades,  $D(G, x^0)$ .

## A planificación de proxectos: conceptos

---

**Earliest start time**, o instante máis temprá no que unha actividade poida ser comezada.

$$D_i^E(G, x^0) = \max_{j \in \text{Pred}(x_{i1})} \{D_j^E(G, x^0) + x_j^0\}$$

**Latest completion time**, o instante máis tardío no que unha actividade poida ser comezada.

$$D_i^L(G, x^0) = \begin{cases} \max_{j \in N; \text{Suc}(x_{j2})=\emptyset} \{D_j^E(G, x^0) + x_j^0\}, & \text{se } \text{Suc}(x_{i2}) = \emptyset \\ \min_{j \in \text{Suc}(x_{i2})=\emptyset} \{D_j^L(G, x^0) - x_j^0\}, & \text{noutro caso.} \end{cases}$$

**Slack da actividade  $i$** , o tempo que pode retrasarse o seu inicio sen afectar á duración do proxecto.

$$S_i(G, x^0) = D_i^L(G, x^0) - D_i^E(G, x^0) - x_i^0.$$

## Usando ProjectManagement

---

### Caracterizando un proxecto con ProjectManagement

- ▶ Tipo 1: Finish to start (FS). Se  $i \in N$  precede tipo 1 a  $j \in N$ ,  $j$  non pode comezar ata que  $j$  se finalice.
- ▶ Tipo 2: Start to start (SS). Se  $i \in N$  precede tipo 2 a  $j \in N$ ,  $j$  non pode comezar ata comezar  $i$ .
- ▶ Tipo 3: Finish to finish (FF). Se  $i \in N$  precede tipo 3 a  $j \in N$ ,  $j$  non pode acabar ata acabar  $i$ .
- ▶ Tipo 4: Start to finish (SF). Se  $i \in N$  precede tipo 3 a  $j \in N$ ,  $j$  non pode acabar ata comezar  $i$ .

## Usando ProjectManagement

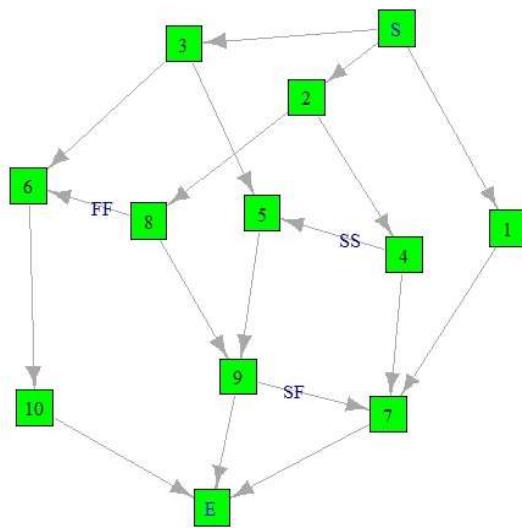
---

No que segue, analizaremos o proxecto con 10 actividades seguinte, coas duracións e relacións de precedencia dadas a continuación.

$N$	1	2	3	4	5	6	7	8	9	10
Immediate precedence type 1	-	-	-	2	3	3	1,4	2	5,8	6
Immediate precedence type 2	-	-	-	-	4	-	-	-	-	-
Immediate precedence type 3	-	-	-	-	-	8	-	-	-	-
Immediate precedence type 4	-	-	-	-	-	-	-	9	-	-
Durations	2	1.5	1	4.5	2	2.5	3	4	2	5

```
prec1and2<-matrix(0,nrow=10,ncol=10)
prec1and2[1,7]<-1; prec1and2[2,4]<-1;
prec1and2[2,8]<-1; prec1and2[3,5]<-1;
prec1and2[3,6]<-1; prec1and2[4,7]<-1;
prec1and2[5,9]<-1; prec1and2[6,10]<-1;
prec1and2[8,9]<-1; prec1and2[4,5]<-2
prec3and4<-matrix(0,nrow=10,ncol=10)
prec3and4[8,6]<-3; prec3and4[9,7]<-4
```

## Usando ProjectManagement



```
dag.plot(prec1and2,prec3and4)
```

## Usando ProjectManagement

Queda por introducir o vector coas duracións das actividades:

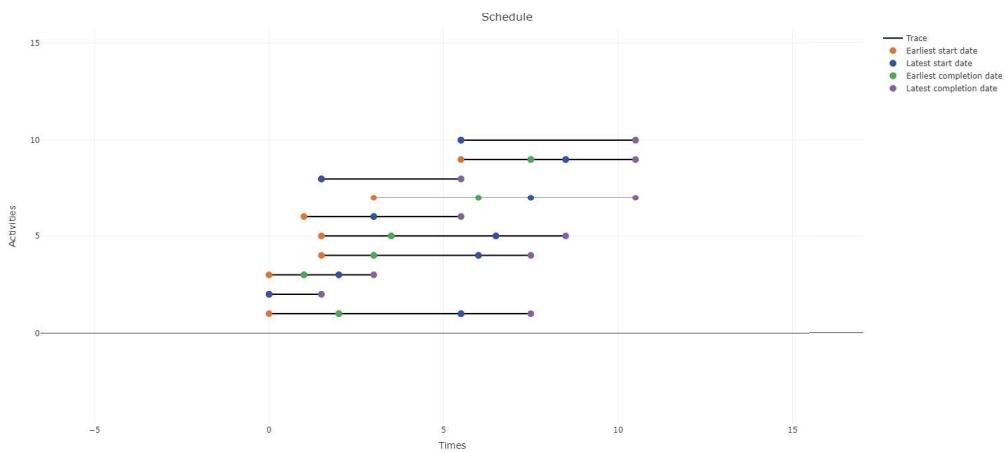
```
duration<-c(2,1.5,1,1.5,2,2.5,3,4,2,5)
```

A función `schedule.pert()` proporciona información sobre a planificación do proxecto, coma o tempo mínimo requerido para o completado de todas as actividades, así como os tempos *early* e *latest* asociados.

```
schedule.pert(duration,prec1and2,prec3and4)
```

Obsérvase unha duración total de 10.5 unidades de tempo, así coma outra información relevante para cada actividade. Tamén se representa gráficamente esta información.

## Usando ProjectManagement



## Máis funcionalidades de ProjectManagement

En moitas situacións específicas da xestión de proxectos, cómpre distribuir os diferentes retrasos entre as diferentes actividades (incentivos para os seus responsables ou como forma de distribuir penalizacións).

É posible definir xogos TU, onde a cada coalición se lle asigna o retraso das actividades do grupo.

### Regras de reparto

- ▶ Regras proporcionais.
- ▶ O valor de Shapley.

### Que pasa coas duracións descritas por variables aleatorias?

Máis información en

<https://cran.r-project.org/package=ProjectManagement>