TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

Algoritmos de estimación de distribución para el problema de la mochila

Curso 2018/2019

Alumno/a: Sierra Ibáñez, José Antonio

Director/es:
José del Sagrado Martínez

Índice General

Índice de figuras

Índice de algoritmos

1. Motivación

El problema que vamos a abordar es un problema clásico en ciencias de la computación. Este problema consiste en el problema de la mochila, con un cierto peso, una serie de objetos, con un beneficio y un peso, que serán los que se introduzcan en la mochila, pero con una condición básica, que no se supere el peso de la mochila. Dado este escenario, necesitaremos que nuestro algoritmo escoja los objetos para que el conjunto de éstos sea el óptimo, es decir, la mejor combinación de objetos con el mayor beneficio. Esta mochila también puede tener restricciones entre los objetos con opción a entrar. Podemos extrapolar este problema clásico a escenarios a gran escala como puede ser la distribución de los contenedores en un barco carguero, cual sería la combinación de estos para que el viaje obtenga el máximo beneficio.

Los Algoritmos de Estimación de Distribución (EDAs) pertenecen a la familia de los algoritmos evolutivos, fueron introducidos en el campo de la computación evolutiva. Este tipo de algoritmos metaheurísticos se aplican en diferentes campos y consisten es la aplicación de serie de pasos para resolver un problema concreto. Dada una función que evalúa la calidad/beneficio de las soluciones para el problema planteado, el algoritmo hace evolucionar una población de soluciones candidatas. Esta nueva descendencia se reproduce y se evalúa. Para crear las nuevas poblaciones se utilizan modelos de estimación de distribución, que son estimados de una base de datos que contiene los individuos seleccionados en la anterior generación. De forma general cada iteración de un EDA consiste en dos pasos:

- Estimar un modelo de distribución de probabilidad basándose en el conjunto de soluciones candidatas
- Muestrear el modelo de distribución de probabilidad anterior.

Objetivos

En este Trabajo Fin de Grado (TFG) aplicaremos distintos EDAs para la resolución de un problema clásico, el problema de la mochila (a su versión más habitual la mochila 0-1) con diferentes enfoques dependiendo de los datos suministrados.

El objetivo principal de nuestro proyecto es la creación de un paquete en R para la resolución del problema de la mochila aplicando algoritmos de estimación de distribución. Para ello, tendremos que:

- Establecer una representación del problema, de sus soluciones y la forma de evaluar estás últimas.
- Estudiar cómo adaptar los algoritmos de estimación de distribución a este problema.

Para afrontar estos objetivos realizaremos un paquete con el lenguaje de programación R [1, 24]. Es una implementación de software libre del lenguaje S pero con soporte de alcance estático. Se trata de uno de los lenguajes más utilizados en investigación por la comunidad estadística, siendo además muy popular en el campo de la minería de datos, la investigación biomédica, la bioinformática y las matemáticas financieras. Este paquete contendrá distintas funciones de inicialización y una función de evolución, siguiendo el esquema de los algoritmos de estimación de distribución, R nos proporciona un entorno donde este paquete puede ser compartido entre toda la comunidad.

Fases de desarrollo

Abordaremos la realización del TFG con las tareas que se describen a continuación y cuyo desarrollo temporal se muestra en la figura 1 1: ARREGLAR REFERENCIA

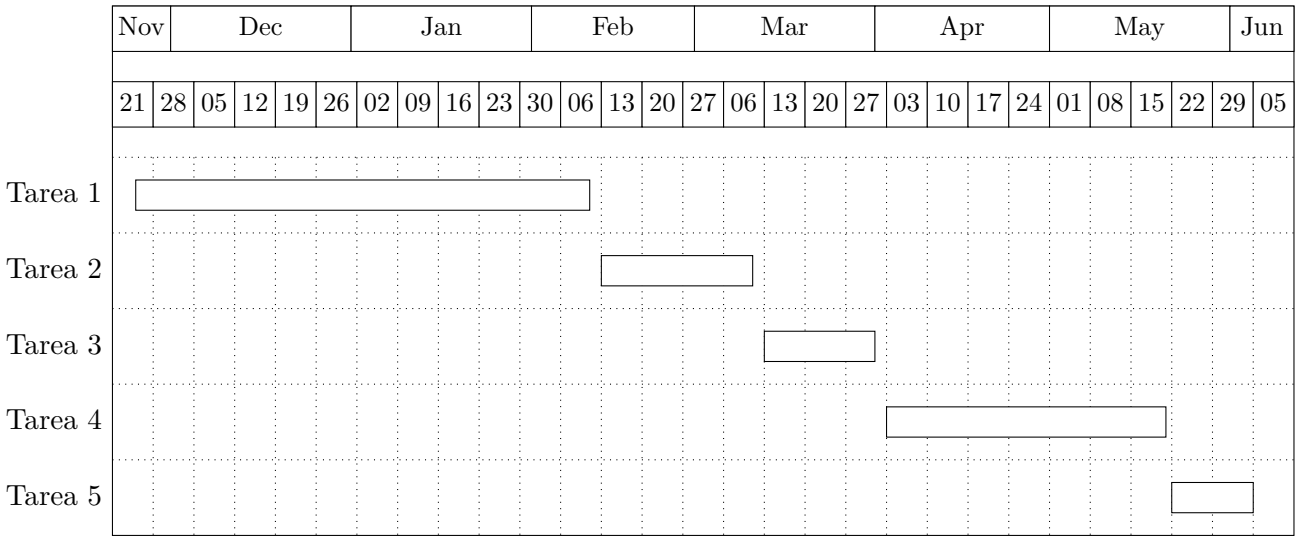


Figura 1: Diagrama de Gantt

- **Tarea 1**: Revisión bibliográfica.

  Sobre los EDAs
  Sobre el lenguaje R y su entorno de desarrollo RStudio

- **Tarea 2**: Representación del problema y de sus soluciones.

- **Tarea 3**: Definición de medidas de calidad de las soluciones.

- **Tarea 4**: Representación de modelos basados en estimación de distribución.

- **Tarea 5**: Creación de EDAs en R.

  Obtención de poblaciones (conjunto de soluciones candidatas)
  Aprendizaje y muestreo de modelos de distribución de probabilidad
  Creación de un paquete en R para el problema de la mochila con EDAs
  Prueba del paquete

2. Introducción a los algoritmos evolutivos

Los Algoritmos Evolutivos (EAs) se basan en el proceso evolutivo de la naturaleza, están muy relacionados con la biología, intentan imitar la evolución orgánica como estrategia para resolver problemas. Como en la naturaleza, parten de una población inicial que evoluciona a través de diferentes generaciones. Este es el proceso fundamental de estos algoritmos.

La población están compuesta por individuos donde cada uno de ellos representa una configuración en el espacio de soluciones. La calidad de los individuos se evalúa mediante una función de evaluación (objetivo o fitness) que se empleará en para el proceso de selección que comentaremos más adelante. La figura 2 muestra el esquema general de un EA.

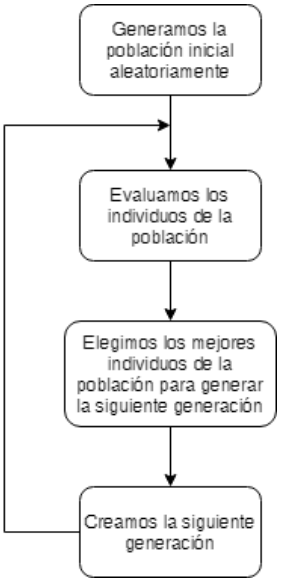


Figura 2: Esquema general algoritmo evolutivo

Los EA siguen una estrategia de búsqueda heurística (se basa en la función de evaluación) que forma parte de los enfoques de cálculo evolutivo, donde el número de soluciones, poblaciones o individuos se crean en cada generación, evolucionando una y otra vez hasta que se consigue una solución satisfactoria o se llega a una condición de parada.

Dentro de la computación evolutiva podemos encontrar los algoritmos genéticos (GAs), tienen una serie de operaciones genéticas, como pueden ser selección, cruzado y mutación, creación de nuevas generaciones de soluciones; también poseen con un criterio de parada, ej: el número de generaciones, el tiempo de ejecución o el alcance de un valor óptimo en la función objetivo.[15].

Los algoritmos de estimación de distribución conocidos como EDAs (Estimation of Distribution Algorithms) [4] constituyen un nuevo camino de los algoritmos evolutivos para “machine learning” y la minería de datos. Su secuencia de pasos es idéntica a los de los algoritmos genéticos[8], la característica los algoritmos de estimación de distribución respecto a otras estrategias de búsqueda evolutiva, como pueden ser los algoritmos genéticos, es que la evolución de una generación a la siguiente se realiza estimando la probabilidad distribución de los individuos más aptos y luego, muestreando el modelo inducido. Los EDAs se han convertido en una parte muy importante de los algoritmos evolutivos, debido a su fuerte fundamentación matemática.

El esquema genérico de los algoritmos evolutivos permite realizar adaptaciones y crear a partir de él otros tipos como hemos comentado anteriormente. El problema de este tipo de algoritmos es la determinación correcta de los parámetros que necesitan. Por ejemplo, en cualquier caso hay que determinar

el tamaño de la población, la representación de los individuos, la función de evaluación y en el caso de los GAs también hay que determinar los operadores genéticos, mientras que en el caso de los EDAs tendremos que establecer una distribución de probabilidad. Todo esto se convierte en un verdadero problema, ya que una mala elección de los mismos puede acarrear que el algoritmo obtenga soluciones alejadas del óptimo.

Un problema importante de los EDAs es analizar cómo la construcción del modelo de probabilidad puede afectar o influenciar en el espacio de búsqueda; hay que encontrar el equilibrio entre la exploración y la explotación. Hay una serie de artículos que muestra esta característica por medio de cambios en los parámetros que determinan el aprendizaje del modelo[15].

Otra cuestión es cómo caracterizar el espacio de búsqueda para que se refleje en el aprendizaje del modelo de probabilidad. Es esencial encontrar mecanismos que permitan a los EDAs muestrear el espacio de búsqueda eficientemente durante el proceso de optimización. Sin embargo, la cuestión de analizar la relación entre el espacio de búsqueda y la estructura de los modelos probabilísticos aprendidos se vuelve difícil debido a dos razones principales: la naturaleza estocástica de la búsqueda de EDAs y el hecho de que los métodos utilizados para aprender los modelos son, en general, capaces de encontrar sólo estructuras aproximadas, subóptimas. Este tipo de algoritmos se utilizan mucho en problemas combinatorios en el que un modelo matemático no se puede aplicar debido a su coste, por lo que encontramos en la computación evolutiva un camino en el que obtener solución a problemas complejos.

3. Algoritmos basados en Estimación de Distribuciones

En esta sección describimos en profundidad los EDAs, sus partes esenciales, funcionamiento y su esquema general.

Los EDAs fueron introducidos como una extensión de los algoritmos evolutivos, son algoritmos heurísticos de optimización que basan su búsqueda, al igual que los algoritmos genéticos, en la aleatoriedad. Se basan también, en el concepto de poblaciones evolutivas; pero la creación de las mismas no se lleva a cabo por medio de operadores tradicionales de cruce y mutación para la generación de nuevas soluciones como en los algoritmos genéticos. Las nuevas poblaciones de individuos se muestrea de una distribución de probabilidad, la cual es estimada a partir de la información generada por las soluciones obtenidas en poblaciones en iteraciones pasadas. Esto trata de predecir mejor los movimientos de las poblaciones en el espacio de búsqueda para evitar la necesidad de tantos parámetros.

En los EDAs, la nueva población es muestreada a partir de una distribución de probabilidad generada de la población anterior. Al mismo tiempo, mientras en otras heurísticas de la computación evolutiva las interrelaciones entre las diferentes variables que representan los individuos se tienen en cuenta implícitamente, en los EDAs, las interrelaciones entre variables están explícitamente expresadas a través de la distribución de probabilidad asociada a los individuos de cada población, seleccionados en cada generación a diferencia de los algoritmos genéticos, que están representadas de manera implícita. La dificultad será encontrar la distribución de probabilidad asociada a los individuos seleccionados en cada generación. [3]

Formalmente, por $X_i(i = 1, 2, ..., n)$ representaremos una variable aleatoria. Una posible instanciación de X_i se denota como x_i. $p(X_i = x_i)$ -o simplemente $p(x_i)$- se denotará la probabilidad de la variable X_i tomando el valor x_i. De la misma manera, $X = (X_1, ..., X_n)$ representará una n variable aleatoria y, $x = (x_1, x_2, ..., x_n)$ posibles valores.

El funcionamiento de estos algoritmos, por lo general, trabajan siguiendo estos pasos:

1) Primeramente, la población inicial D_0 se genera con N individuos. La generación de esos N

individuos se realiza por medio de una distribución uniforme para cada variable y cada variable es evaluada individualmente. La creación de la población inicial es un paso que puede variar dependiendo del problema y pueden influir en el rendimiento del EDA. Para cada problema se pueden utilizar diferentes técnicas, algunas reflejan las dependencias entre variables y otras las obvian, la tarea será buscar qué población inicial puede favorecer al funcionamiento del algoritmo EDA.

2) Seleccionar un número de individuos $Se(Se < N)$ de la población D_{l-1} siguiendo un criterio, que puede suele basarse en la función de evaluación. Denotamos por D_{l-1}^{Se} la población Se individuos seleccionados de la generación $l-1$. Los individuos seleccionados pasan a formar parte de la nueva población D_l.

En el caso de utilizar la función de evaluación como medida de la importancia de las soluciones, se eliminarán las de puntuación más baja y al mismo tiempo proporcionando un mecanismo de clasificación. En el caso de escoger las soluciones por medio de la función de evaluación se dice que la búsqueda es elitista.

Por otra parte, a veces se aplica el enfoque de minado de datos, que consiste en generar y evaluar varios subconjuntos de características; normalmente, métodos codificados (métodos iterativos de avance o retroceso) o métodos de búsqueda heurística (GA, EDA). Este enfoque se utiliza con un algoritmo de clasificación específico, ya que el resultado de la evaluación se utiliza durante la búsqueda.

3) Aprender un modelo probabilístico que refleja las interdependencias entre las n variables. Este paso también es conocido como el proceso de aprendizaje; es el momento más crucial ya que, a partir de ahí, si se representan adecuadamente las dependencias entre las variables para evolucionar hacia sujetos más favorables.

4) Finalmente, se obtiene la nueva población D_l, constituida por N nuevos individuos, a partir del muestreo de la distribución de probabilidad aprendida en el paso anterior. Si se sigue una búsqueda elitista, se asume que los mejores individuos de la población D_{l-1}^{Se} se quedan en D_l. A partir de esta nueva población creada, se vuelve a aprender una distribución de probabilidad y se repiten los pasos 3 y 4 hasta que se satisfaga una condición de parada.

El algoritmo 1 muestra el pseudocógido de los algoritmos de estimación de distribución. Como comentamos anteriormente la estructura básica de los algoritmos genéticos y los EDAs son muy similares. Los pasos (1), (2) y (3) son los mismos para muchos algoritmos evolutivos, mientras que los pasos (4) y (5) son específicos de los EDAs.

Algorithm 1 Código general EDA

1: $D_0 \leftarrow$ Se generan N individuos ▷ Población inicial
2: **repeat** for $l = 1, 2, ...$
3: $\quad D_{l-1}^{Se} \leftarrow$ Se seleccionan $Se \leq N$ individuos de D_{l-1} por medio de una función objetivo
4: $\quad p_l(x) = p(x \mid D_{l-1}^{Se}) \leftarrow$ Distribución de probabilidad estimada para la población l-ésima
5: $\quad D_l \leftarrow$ Se muestrea la nueva población a partir de $p_l(x)$
6: **until** Condición de parada

El paso crucial en los EDAs es encontrar las interdependencias entre las variables que representan en el espacio de búsqueda (paso 4). La idea básica consiste en inducir un modelo de probabilidad que refleje el comportamiento de los individuos de la población más favorables.

3.1. Ejemplo de aplicación de los EDAs

Para reflejar mejor la sección anterior, pasamos a exponer un ejemplo con la utilización del modelo EDA. Se trata del problema OneMax, que consiste en obtener la máxima suma de las variables, siendo:

$$OneMax(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} x_i$$

$$\text{Donde } x = x_1, x_2, ..., x_n \; x \in 0, 1$$

Donde n es el número de variables x_i en la posición i^{th}. Esta función tiene un óptimo global que es un string de todos unos. [16]

En este ejemplo nuestra el tamaño de nuestra población será de $N = 10$ y constará de $n = 6$ variables, generamos la primera generación, normalmente aleatoriamente. pero se puede obtener de diferentes maneras, dependiendo del problema, obtenemos diez soluciones:

	x_1	x_2	x_3	x_4	x_5	x_6	$h(x)$
1	1	0	1	1	1	0	4
2	1	1	0	1	1	0	4
3	0	0	1	0	1	0	2
4	0	0	1	0	1	1	3
5	1	0	1	0	0	0	2
6	0	1	1	1	0	0	3
7	1	0	1	1	0	0	3
8	1	0	1	0	1	0	3
9	1	1	1	1	1	0	5
10	0	1	1	0	1	1	4

Tabla 1: D_0.

Tratamos de maximizar la función OneMax (función objetivo), tenemos seis variables por lo que la función $h(x) = \sum_{i=1}^{6} x_i$. Como vemos en la tabla anterior, ya hemos obtenido $h(x)$.

Esta población inicial se ha obtenido al azar por medio del muestreo de la siguiente distribución de probabilidad: $p_0(x) = \prod_{i=1}^{6} p_0 x_i$, donde $p_0(X_i = 1) = 0.5$. Esto significa que la probabilidad de cada variable se encuentra factorizada como un producto de seis distribuciones marginales univariantes, independientes entre sí. El cuadro anterior se encuentra la generación inicial D_0.

El siguiente paso consiste en seleccionar algunos individuos de D_0. Como comentamos en el punto 2, esta selección puede ser realizada por medio de métodos de selección estandard en computación evolutiva, como puede ser el caso de la truncación, con un valor de 0.5 para este ejemplo, se seleccionan el subconjunto de soluciones más prometedoras (el 50% de las soluciones son seleccionadas). Denotamos por D_0^{Se} los individuos seleccionados, en caso de que existan empates en la función de evaluación se escogerá de manera aleatoria, entre el conjunto con el mismo resultado.

	x_1	x_2	x_3	x_4	x_5	x_6	$h(x)$
1	1	0	1	1	1	0	4
2	1	1	0	1	1	0	4
7	1	0	1	1	0	0	3
9	1	1	1	1	1	0	5
10	0	1	1	0	1	1	4

Tabla 2: D_0^{Se}

Una vez que tengamos los cinco individuos seleccionados, inducimos una distribución de probabilidad.

Aunque podemos y deberíamos utilizar una distribución conjunta que tuviera en cuenta las interdependencias de las variables, utilizaremos la representación más simple, supondremos que cada variable es independiente al resto. Por lo que matemáticamente:

$$p_1(x) = p_1(x_1, ..., x_6) = \prod_{i=1}^{6} p(x_i \mid D_0^{Se})$$

A partir de ello, podemos obtener el modelo a partir de los parámetros, $p(x_i \mid D_0^{Se})$ con $i = 1,..., 6$, se estima a partir de la frecuencia relativa, $\hat{p}(x_i \mid D_0^{Se})$. Por lo que, para este ejemplo obtendríamos los siguientes resultados:

$\hat{p}(x_1 \mid D_0^{Se})$	$\hat{p}(x_2 \mid D_0^{Se})$	$\hat{p}(x_3 \mid D_0^{Se})$	$\hat{p}(x_4 \mid D_0^{Se})$	$\hat{p}(x_5 \mid D_0^{Se})$	$\hat{p}(x_6 \mid D_0^{Se})$
0.8	0.6	0.8	0.8	0.8	0.2

Tabla 3: Probabilidad conjunta de D_0

Muestreando la distribución de probabilidad, $p_1(x)$, obtenemos una nueva población de individuos, D_1 de tamaño N y se sustituye D_0 por D_1 con la que volveremos a repetir el proceso de aprendizaje por medio de la distribución de probabilidad.

	x_1	x_2	x_3	x_4	x_5	x_6	$h(x)$
1	1	1	1	1	1	0	5
2	1	1	0	1	1	1	5
3	1	0	1	0	1	0	3
4	1	1	1	1	1	1	6
5	1	0	1	0	1	0	3
6	1	1	1	1	0	0	4
7	1	1	0	1	0	1	4
8	1	0	0	0	1	1	3
9	1	1	1	1	1	0	5
10	0	1	1	0	1	1	4

Tabla 4: Población D_1

Se vuelve a realizar el proceso de selección, obteniendo cinco individuos de D_1, obteniendo D_1^{Se}.

	x_1	x_2	x_3	x_4	x_5	x_6	$h(x)$
1	1	1	1	1	1	0	5
2	1	1	0	1	1	1	5
4	1	1	1	1	1	1	6
7	1	1	0	1	0	1	4
9	1	1	1	1	1	0	5

Tabla 5: Individuos seleccionados D_1^{Se}

Volvemos a obtener los parámetros por medio de la frecuencia relativa

$\hat{p}(x_1 \mid D_1^{Se})$	$\hat{p}(x_2 \mid D_1^{Se})$	$\hat{p}(x_3 \mid D_1^{Se})$	$\hat{p}(x_4 \mid D_1^{Se})$	$\hat{p}(x_5 \mid D_1^{Se})$	$\hat{p}(x_6 \mid D_1^{Se})$
1	1	0.6	1	0.8	0.6

Tabla 6: Probabilidad conjunta de D_1

Los pasos anteriores se repiten hasta que se verifique una determinada condición de parada, dependiendo de cómo esté programado el algoritmo pueden llegar a ser límite de generaciones o que, a partir de ciertas generaciones, el método devuelva siempre la misma población.

4. Taxonomía de los modelos EDA

En esta sección hablaremos de las distintas formas en las que los EDAs pueden incorporar o tener en cuenta la relación entre las variables, en vez de simplemente obviarlas.

4.1. Clasificación basada en la descomposición del problema

Todos los EDAs son clasificados dependiendo del número máximo de dependencias entre variables que pueden aceptar (número máximo de padres que la variable X_i puede tener en el modelo probabilístico)[22]. Así distinguimos entre EDAs:

1. **Sin dependencias**. Una de las maneras más simples de enfocar el problema es asumir que las variables son independientes. Bajo esta premisa, la distribución de probabilidad de cualquier variable individual no debe depender del valor de otras variables. Por lo que la distribución de probabilidad $P(x_1, x_2..., x_n)$ del vector $(x_1, x_2, \ldots, x_n)$ de n variables consiste en un producto de la distribución de cada variable individual:

   $$P(x_1, x_2..., x_n) = \prod_{i=1}^{n} p(x_i)$$

   Este tipo de EDAs suelen llamarse EDAs univariados (UMDA, Univariate Marginal Distribution Algorithm) y para n variables consta de n tablas de probabilidad y cada una de estas tablas define las probabilidades de diferentes valores de la variable correspondiente. Puesto que las probabilidades de diferentes valores de una variable deben sumar a uno, una de las probabilidades puede omitirse para cada variable. Este modelo de se utilizó en la sección 3.1.

   Existe otros modelos dentro de esta categoría como cGA (compact Genetic Algorithm) [10] o Population- Based Incremental Learning (PBIL)[2].

2. **Dependencias bi-variadas**. Estos EDAs son capaces de obtener la interacción entre cada par de variables construyendo un modelo basado en árbol. En estos modelos la probabilidad condicional de una variable puede solo depender, como máximo de otra variable, que será su padre en la estructura.

   Aquí encontramos algoritmos como MIMIC (Mutual Information Maximization for Input Clustering)[6] y BMDA (Bivariate Marginal Distribution Algorithm)[19].

3. **Dependencias múltiples**. Los modelos multivariables representan dependencias usando grafos dirigidos acíclicos o no dirigidos. Hay dos modelos representativos populares en EDAs: Redes Bayesianas y Redes de Markov.

   Una red bayesiana está representada por un grafo dirigido acíclico, donde cada nodo representa una variable y cada arista define una relación directa de dependencia. Asociado a cada variable X_i en el grafo existe una distribución de probabilidad condicionada a sus padres $p(X_i \mid \prod_i)$. Así, la distribución de probabilidad codificada por una red bayesiana puede escribirse como:

   $$p(X_1, X_2..., X_n) = \prod_{i=1}^{n} p(X_i \mid \prod_i)$$

   En caso de las redes de Markov se emplea un grafo no dirigido para representar las relaciones entre variables. Así, dos variables se consideran independientes entre sí, dado otro subconjunto de variables si todo camino entre esas variables está separado por una o más de las variables en el subconjunto.

   También podemos encontrar una subclase de modelos multivariables en los cuales, las variables son divididas en clusters, que son independientes entre sí; son llamados modelos de producto marginal.
   Los modelos multivariables más destacados se encuentran FDA (Factorized Distribution Algorithm)[18], EBNA (Estimation of Bayesian Networks Algorithm)[14] o EcGA (Extend compact Genetic Algorithm)[11].

5. EDAs en dominios discretos

Esta sección se ocupa de la aplicación de los EDAs en dominios discretos. Por dominio discreto se entiende aquel en el que todas las variables son discretas (pueden tomar un número finito de valores). Generalmente las variables son binarias. Vamos a describir los EDAs más significativos en cada una de las categorías de la taxonomía utilizada en el apartado anterior: UMDA (sin dependencias), MIMIC (dependencias bivariadas) y EBNA y ECGA (dependencias multivariadas).

5.1. Sin dependencias

UMDA - Univariate marginal distribution algorithm

UMDA fue introducido por (Mühlenbein & Paas, 1998)[20], estima el modelo de la distribución de la población, calculando la probabilidad conjunta de una selección de los mejores individuos en cada generación. Esta es una representación gráfica de las variables, vemos como no tienen ninguna conexión entre ellas ya que no se consideran las dependencias.

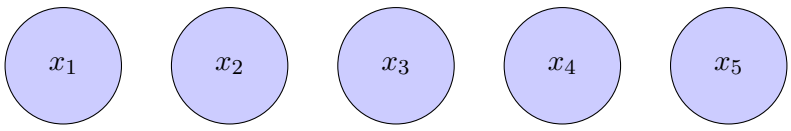


Figura 3: Estructura UMDA

UMDA utiliza la frecuencia de cada componente en una población de soluciones candidatas para crear una nueva población de soluciones candidatas. Se estima la probabilidad conjunta dada por:

$$p(x_i) = \frac{\text{individuos de la población con } X_i = x_i}{\text{todos los individuos de la población}}$$

Este algoritmo trabaja con variables binarias y usa un vector de probabilidad $P(x_1, x_2..., x_n)$ como modelo de probabilidad, donde p_i denota la probabilidad de que x_i sea igual a 1, es decir, que x_i esté incluida en la solución. Para aprender el vector de probabilidad, la probabilidad p_i se establece como la proporción $X_i = 1$ en la población de las soluciones seleccionadas.

Para crear nuevas soluciones, cada variable es generada independientemente basada en la posición del vector de probabilidad, en cada generación la distribución de probabilidad marginal de cada variable denota el comportamiento de los individuos seleccionados y la distribución conjunta se factoriza como un producto de distribuciones marginales univariantes independientes.

Algorithm 2 Código general UMDA

```
1:  function UMDA(Population_size, Selection_size)
2:      Población ← Inicializar(Population_size)
3:      EvaluarPoblación(Población)
4:      S_best ← ObtenerMejorSolución(Población)
5:      while (¬ CondiciónDeParada()) do
6:          Seleccionados ← ObtenerMejoresSoluciones(Población, Selection_size)
7:          V ← CalcularFrecuencia(Seleccionados)
8:          Descendientes ← ∅
9:          MuestrearDescendientes(V)
10:         EvaluarDescendientes(Descendientes)
11:         S_best ← ObtenerMejorSolución(Descendientes)
12:         Población ← Descendientes
13:     end while
14:     return S_best                                   ▷ La mejor solución
15: end function
```

Cada distribución marginal se estima a partir de las frecuencias marginales sobre los elementos seleccionados:

$$p(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i \mid D_{l-1}^{Se})}{N}$$

donde

$$\delta_j(X_i = x_i \mid D_{l-1}^{Se}) = \begin{cases} 1 & \text{si el } j-\acute{e}simo \text{ caso de } D_{l-1}^{Se}, X_i = x_i \\ 0 & \text{en otro caso} \end{cases}$$

Mientras que la mayoría de las EDAs trabajan manteniendo una población de soluciones candidatas, los EDAs incrementales reemplazan totalmente la población con el modelo probabilístico. En cada generación, algunas soluciones son creadas por medio de la función de optimización. Se necesita un método de selección para identificar un subconjunto de soluciones buenas de las que se calculan las probabilidades marginales univariadas.

UMDA trabaja muy bien en soluciones a problemas lineales sin dependencias entre las variables y también puede aplicarse en problemas de optimización combinatoria utilizando una heurística local.

5.2. Modelo Bi-variado

Los modelos bi-variados se utilizan para resolver el problema de interacción entre parejas de variables, se utiliza MIMIC. Cuando existe una dependencia de dos órdenes entre las variables, proporcionan un mejor resultado que los modelos univariados; pero están lejos del mundo real donde se producen múltiples interacciones.

MIMIC - Mutual information maximizing input clustering

El algoritmo MIMIC considera las interacciones por parejas entre las variables para computar la distribución de probabilidad. MIMIC usa una cadena de distribución (una permutación entre las variables) que maximiza la llamada información mutua de los vecinos para modelar la interacción entre variables.

MIMIC trabaja utilizando las soluciones más prometedoras para calcular la información mutua entre todos los pares de variables. Después, empieza con la variable con menos entropía condicional, una cadena de dependencia es añadida a las variables con el máximo de información con la última variable que fue añadida anteriormente. [6]

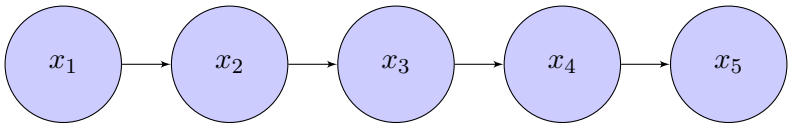


Figura 4: Estructura MIMIC

En cada iteración del EDA consiste en escoger las variables que tengan mayor resultado en su función objetivo, crear un modelo y generar la siguiente población de variables. El algoritmo trata de modelar la probabilidad conjunta real $p(x)$ de las variables seleccionadas, viene dada por:

$$p(X) = p(X_1 \mid X_2..X_n)p(X_2 \mid X_3...X_n)\ldots p(X_{n-1} \mid X_n)p(X_n)$$

Este proceso es repetido hasta que todas las variables han sido seleccionadas. El resultado es un modelo en árbol que consiste en una simple cadena de dependencias, en el que cada padre tiene exactamente un hijo. Una vez la estructura del modelo ha sido completada, la probabilidad de condición de cada variable basada en su padre es calculada de las soluciones prometedoras. Dando una permutación de n variables, $\pi = i_1, i_2, \ldots, i_n$, MIMIC descompone la distribución de probabilidad de $p'(X) = (X_{i_1}, X_{i_2}, \ldots, X_{i_n})$ como:

$$p'(X) = p(X_{i_1} \mid X_{i_2})p(X_{i_2} \mid X_{i_3})\ldots p(X_{i_{n-1}} \mid X_{i_n})p(X_{i_n})$$

Donde $p(X_{i_j}|X_{i_{j+1}})$ denota la probabilidad condicionada de X_{i_j}, dado por $X_{i_{j+1}}$. Las nuevas soluciones candidatas son generadas muestreando la distribución de probabilidad del modelo. El muestreo genera las variables en el orden contrario respecto a la permutación π, empezando por $p(X_{i_n})$ y terminando con $p(X_{i_1})$.

La tarea de optimización se lleva a cabo a partir de un algorimo greedy (voraz), un algoritmo muy eficiente para este tipo de casos aunque no garantiza el óptimo global de la distribución.

La entropía es un concepto introducido en la Teoría de la Información como medida de incertidumbre o de la información promedio que nos provee una variable aleatoria. Esta se calcula como:

$$H(X) = -\sum p(x) \log_2 p(x)$$

MIMIC utiliza un método greedy (voraz) para encontrar en el valor en i_n a i_1 que hace que la probabilidad $p(x)$ sea la más cercana a la distribución real $p'(x)$. La posición viene dada por i_n con la mejor entropía. i_{n-1} es la posición de la solución con la menor entropía, $H(X_{i_{n-1}} \mid X_{i_n}).i_{n-2}$ es la posición con el la menor entropía $H(X_{i_{n-2}} \mid X_{i_{n-1}})$. Y así, hasta que se seleccionan las soluciones del problema, estos valores junto con los individuos seleccionados son el modelo de las soluciones seleccionadas[6]. Por lo que:

$$H(X_1, ..., X_n) = \sum_{i=1}^{n} H(X_i \mid X_{i-1}, ..., X_1)$$

La estructura del algoritmo sería la siguiente:

Algorithm 3 Algoritmo MIMIC

1: $i_n \leftarrow \arg\min H(X_j)$ Obtener la variable con menor entropía
2: $i_k \leftarrow$ Obtener la variable con menor entropía condicionada $H(X_j \mid X_{i_{k+1}})$
3: ▷ Repetir este proceso con las variables no seleccionadas y escoger la de menor entropía con la anterior seleccionada

Después de la creación del modelo, se genera la siguiente generación. Primeramente, la posición i_n-ésima se genera basándose en la probabilidad $p(X_{i_n})$. Entonces, la posición i_{n-1} se genera en base al anterior valor de su probabilidad condicional $p(X_{i_{n-1}} \mid X_{i_n})$.

Para utilizar este método necesitamos una población a partir de la cual se extrae la entropía y probabilidades subyacentes para crear la nueva población.

El modelo MIMIC puede ampliarse para trabajar utilizando árboles de dependencias. Donde un hijo sólo puede tener un padre[6].

5.3. Múltiples dependencias

EBNA - Estimation of Bayesian Network Algorithm

EBNA es un algoritmo propuesto Etxeberria y Larrañaga[14]. Esta aproximación adopta una red Bayesiana como modelo de probabilidad el cual se aprende a partir de los elementos escogidos de cada generación. Hay diferentes tipos de métricas para evaluar la red para aprender de la estructura, la más común $EBNA_{BIC}$. En general, $EBNA_{BIC}$ busca la mejor estructura para la red Bayesiana usando el método de puntuación + búsqueda.

Formalmente, una red bayesiana (S, θ) representa una factorización gráfica de una distribución de probabilidad. La estructura S es un grafo dirigido acíclico que refleja el conjunto de condiciones entre las variables. Por el otro lado, θ es un conjunto de parámetros para la distribución de probabilidad asociada a cada variable. [9] [5]

La factorización de la distribución de probabilidad es descrita como

$$p(x) = \prod_{i=1}^{n} p(X_i \mid \prod_i)$$

donde $prod_i$ denota el valor de las variables $prod_i$, el conjunto de padres de X_i en el grafo S.

Gráficamente lo podemos ver cómo:

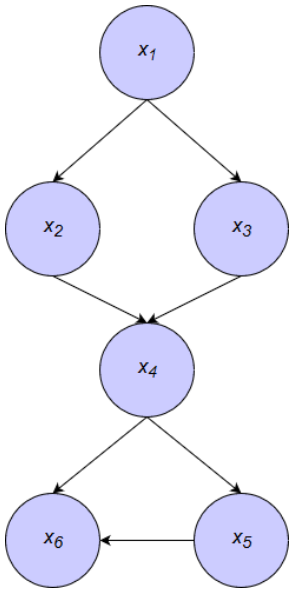


Figura 5: Estructura Red Bayesiana

Algorithm 4 Código general EBNA

1: $D_0 \leftarrow$ Se generan N individuos ▷ Población inicial
2: **repeat** for $l = 1, 2, ...$
3: $\quad D_{l-1}^{Se} \leftarrow$ Se seleccionan $Se \leq N$ individuos de D_{l-1} por medio de una función objetivo
4: $\quad p_l(x) = p(x \mid D_{l-1}^{Se}) \leftarrow$ Se estiman una búsqueda local para encontrar la estructura BN que optimiza la métrica
5: $\quad$ Se calculan los parámetros de la RB a partir de D_{l-1}^{Se}
6: $\quad D_l \leftarrow$ Se muestrea la nueva población a partir de la RB y se evalúa
7: **until** Condición de parada

Inicialmente se genera un grafo acíclico directo sin ninguna unión, con la misma probabilidad para todos los individuos. Una vez creada la población se selecionan los individuos con mejor resultado en

la función objetivo, normalmente es número suele ser la mitad de la población inicial. Si el número de seleccionados es cercano al número de individuos en la población inicial las poblaciones no evolucionarán mucho entre generación y generación.

EcGA - Extended compact Genetic Algorithm

ECGA propuesto por Harik [12], al contrario que los algoritmos anteriores, es una mezcla entre un EDA y GA, ya que utiliza la probabilidad marginal entre conjuntos pero la evolución se lleva a cabo por medio de operaciones genéticos. Identifica las conexiones entre conjuntos basado en la métrica Minimun Description Length (MDL), este criterio en "machine learning" dice que la mejor descripción de los datos viene dada por el modelo que los comprime mejor. Dicho de otra manera, aprender un modelo para los datos consiste en capturar las regularidades en los datos y utilizarlas para comprimirlo. Por lo tanto, cuando más podamos comprimir una información (cuanto más hayamos aprendido) mejor podremos predecirla.

El modelo probabilístico de este algoritmo se basa en la conexión entre conjuntos y la distribución marginal basada en estas conexiones. Una vez que se identifican estas conexiones, podemos realizar operaciones de cruzado. Este algoritmo se diferencia mucho de otros multivariables, en vez de utilizar un modelo probabilístico basado en probabilidades condicionales, ECGA genera su modelo basado en una distribución marginal especificado por medio de los conjuntos conectados. [21]

Usa un modelo que divide las variables en clusters independientes y cada uno de esos clusters es tratado como una única variable. La construcción del modelo empieza asumiendo que todas las variables del problema son independientes. En cada iteración de la construcción del modelo, se unen dos clusters para mejorar la calidad del modelo. La construcción del modelo termina cuando dos clusters no se pueden unir para mejorar la calidad del modelo. Una vez aprendida la estructura del modelo completado, una tabla de probabilidad se calcula para cada cluster basado en la población de soluciones seleccionadas y se generan nuevas soluciones muestreando la población obtenida anteriormente.

6. El problema de la mochila

6.1. Introducción

El problema de la mochila se puede describir como una selección de varios objetos que puede ser introducidos en una mochila; estos objetos o ítems serán los más útiles, ya que la mochila tiene un límite de capacidad. Este problema ha sido muy estudiado por su estructura simple y porque con él se puede modelar muchos de los problemas clásicos como la inversión de capital. Está clasificado como un problema NP-hard. [17]

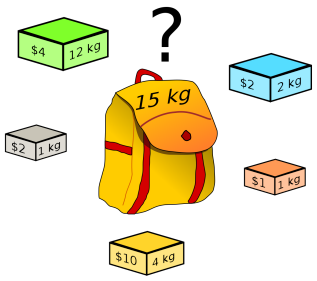


Figura 6: Representación gráfica del problema de la mochila

Hay diferentes adaptaciones de este problema y cada una difiere en su planteamiento; como puede ser en la representación que utilizan, la manera en que obtienen la población inicial y cómo trabajan con o sin las restricciones que impone el problema; entre ellas se encuentran el problema de la mochila 0-1, es la representación más común. Después, existen otras derivadas de la anteriormente mencionada como pueden ser el problema de la mochila multiobjetivo y por supuesto, múltiples mochilas.

6.2. Definición

Consiste en un problema de optimización combinacional, dado un conjunto de objetos, cada uno con un peso y un valor tenemos que determinar los ítems que serán introducidos en la mochila o colección.

Tenemos n objetos fraccionables y una mochila, el objeto i tiene un peso c_i y una fracción x_i de objeto i que produce un beneficio $b_i x_i$, el objetivo es llenar la mochila, de capacidad C, de manera que maximice el beneficio, estructurando el problema matemáticamente:

$$max \sum_{n=1}^{n} b_i x_i \quad \text{Función objetivo}$$

$$\text{Sujeto a} \quad \sum_{n=1}^{n} c_i x_i \leq C \quad \text{Restricción del problema clásico}$$

$$\text{con} \quad 0 \leq x_i \leq 1 \quad b_i > 0 \quad c_i > 0 \quad 1 \leq i \leq n$$

Como podemos comprobar, al ser un problema combinatorio el número de posibles combinaciones de objetos serían 2^n, por lo que el orden de un algoritmo por fuerza bruta $\mathcal{O}(2^n)$.

6.2.1. La mochila 0-1

Como comentamos anteriormente, el problema más común es la mochila 0-1, la cual restringe el número de x_i de ítems a cero o a una. Hay n elementos, x_1 a x_n donde x_i tiene un valor b_i y un peso c_i. El peso máximo de la mochila con los objetos que porta no puede superar C, queda presupuesto que todos los objetos no tienen valor o peso negativos. Para simplificar la representación se asume que los objetos están ordenados de mayor a menor peso. Matemáticamente podemos establecer el problema

de la siguiente manera:

$$max \sum_{n=1}^{n} b_i x_i \quad \text{Función objetivo}$$

$$\text{Sujeto a } \sum_{n=1}^{n} c_i x_i \leq C \quad \text{Restricción del problema clásico}$$

$$\text{Siendo } x_i \in \{0,1\}, (i = 1, 2, ..., n)$$

$$\text{Donde } x_i = \begin{cases} 1 \text{ si el objeto } i \text{ es seleccionado} \\ 0 \text{ en otro caso} \end{cases} \qquad \text{Valores de } x_i$$

La primera ecuación hace referencia a maximizar los resultados del problema a partir de la integración de múltiples variables que pertenecen al conjunto de objetos a escoger identificado por el subíndice i. La segunda indica que es necesario estimar el peso total de los artículos que serán guardados en una mochila cuya capacidad es determinada por la variable C. La última indica qué variables pertenecen a la solución identificadas por el valor i, es decir, forman parte de la mochila (valor 1) o no forman parte de la mochila (valor 0).

Aquí tenemos un ejemplo para planteamiento del problema:

Número de objetos $n = 4$ y máximo coste de la mochila $C = 15$
Beneficio de cada objeto $b_i = (38, 40, 24, 11)$
Coste de cada objeto $c_i = (9, 6, 5, 2)$

Una solución podría ser:

$$(x_1, x_2, x_3, x_4) = (0, 1, 1, 1) \rightarrow \text{beneficio } \mathbf{75} \text{ y coste } \mathbf{13}$$

7. El problema de la mochila 0-1 genérico con dependencias

El problema clásico de la mochila plantea una serie de objetos, con su valor y peso, y una mochila con una capacidad máxima. Este planteamiento no se ajusta muy bien la realidad. Pueden surgir una serie de dependencias; por ejemplo, puede que un objeto no puedan estar con otro, ej. gasolina y un mechero, en esta ocasión puede introducirse uno, pero no el otro (exclusión). También se plantea que para la introducción de un objeto en la mochila tenga que estar anteriormente otro (implicación). O que, obligatoriamente, dos objetos deban de estar en la mochila conjuntamente (combinación).

Pasemos a formular este tipo de interacciones entre las variables, a modo de grafo G=(**R,I,J,X**) [7].

- **X** el conjunto de objetos.
- $\mathbf{I} = (x_i, x_j) \mid (x_i \Rightarrow x_j$ cada par $(x_i, x_j) \in I$ es una interacción entre variables que se representa con un enlace dirigido $x_i \rightarrow x_j$.
- $\mathbf{J} = (x_i, x_j) \mid (x_i \odot x_j$ cada par $(x_i, x_j) \in J$ es una interacción entre variables que se representa con un enlace bidireccional $x_i \leftrightarrow x_j$.
- $\mathbf{X} = (x_i, x_j) \mid (x_i \oplus x_j$ cada par $(x_i, x_j) \in X$ es una interacción entre variables que se representa con un enlace tachado.

Para estas dependencias, siguiendo con el ejemplo de la sección anterior, podemos aplicar las restricciones en los objetos: siendo $R = \{x_1, x_2, x_3, x_4\}$ y el conjunto de interacciones $I = \{(x_1, x_3), (x_1, x_4)\}$, $J = \{(x_4, x_2)\}$, $X = \{(x_3, x_4)\}$. Se representa a modo de grafo en la siguiente figura:

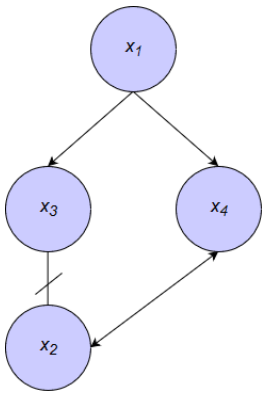


Figura 7: Grafo de dependencias

La solución anterior propuesta, $(x_1, x_2, x_3, x_4) = (0, 1, 1, 1)$, no sería una solución válida ya que viola las restricciones de implicación, debe de estar en la mochila x_1 para poder seleccionar x_3 o x_4 y las de exclusión, x_2 y x_3 no pueden estar en la mochila conjuntamente.

La única solución posible sería $(x_1, x_2, x_3, x_4) = (1, 0, 1, 0)$, teniendo en cuenta las restricciones del problema en concreto y las dependencias entre los objetos.

8. EDAs y el problema de la mochila

Como hemos estado comentando anteriormente el problema de la mochila 0-1, tiene una representación binaria. Por lo que cada posibles solución será un array binario con n dimensiones, escritas como:

$$(x_1, ..., x_i, ..., x_n)$$

El valor de 1 en la posición i indica que el objeto x_i se selecciona para su inclusión en la mochila. Cada objeto x_i tiene asociada un peso y un beneficio. Como ejemplo, si se seleccionan los objetos $1, 5$ y 6 de 6 objetos posibles, obtendremos el siguiente array binario:

$$(1, 0, 0, 0, 1, 1)$$

Para la creación de la población inicial de los EDAs afrontando el problema de la mochila 0-1 se han adoptados las estrategias de inicialización por medio de las siguientes distribuciones de probabilidad (hay que destacar que las técnicas de inicialización que utilizaremos contienen valores proporcionales, sus probabilidades no tienen por qué sumar 1):

- Con una distribución uniforme, para cada valor de x_i, $p(x_1, x_2, ..., x_n) = (\frac{1}{2})$
- Mediante la fracción del peso máximo de la mochila entre la sumatoria de los pesos de los objetos $p(x_i) = \frac{C}{\sum_{i=1}^{n} c_i}$
- Con la relación beneficio/coste para cada objeto $p(x_i) = \frac{b_i}{c_i}$
- Eligiendo una semilla probabilística, se realiza una población inicial con alguno de los demás métodos para así establecer una probabilidad a objeto si se encuentra en la población anteriormente creada
- Con una dependencia entre pares de objetos, mediante la menor entropía

Para continuar, seleccionaríamos las mejores conforme a la función de evaluación para así poder extraer la probabilidad derivada de la frecuencia de aparición de esas variables. Muestrearíamos dichas variables para obtener la primera generación, se repetiría el proceso hasta que la condición de parada se satisfaga.

8.1. Probabilistic logic sampling algorithm

Para muestrear una población a partir de la probabilidad inducida de nuestra población inicial se utiliza el "*Probabilistic logic sampling algorithm*" (PLS) para la creación de nuevos individuos. Este método consiste en: dado un conjunto de variables $X = X_1, ..., X_n$, una instanciación de todas las variables.

- Divide el intervalo [0,1] en k intervalos de su anchura proporcional a $p(x_i)$.
- Generamos un número aleatorio $r \in [0, 1]$.
- Determinamos la región en la que cae ese número aleatorio.

A modo ejemplo representaremos las probabilidades $p(x_i) = (0.3, 0.25, 0.27, 0.18)$

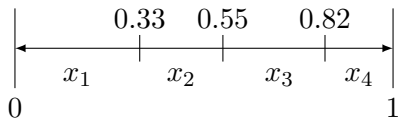


Figura 8: Linea real con la proporciones

Si $r = 0.2929$, entonces:

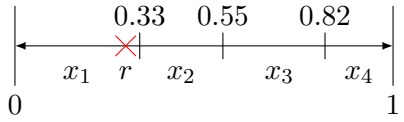


Figura 9: Objeto x_1 escogido

Vemos como se encuentra en el rango [0-0.33], en el que se encuentra el "peso", en términos de probabilidad, de x_1, el objeto que se escogería el mismo.

8.2. Convergencia de los algoritmos a partir la corrección de Laplace

Los métodos de inicialización pueden no permiten soluciones no factibles, por lo que para algunos x su valor puede ser 0. Por el contrario, si algunas soluciones creadas no fuesen factibles habría individuos que casualmente podrían estar en la solución final pero no es la solución inicial creada. Por ejemplo, cuando los individuos seleccionados en los pasos previos como $\exists j, \sum_{j=1}^{n} \delta_j(X_i = x_i \mid D_{l-1}^{Se}) = 0$, como vimos en la subsección de EDAs sin dependencias. Hay que tener en cuenta que la probabilidad conjunta puede ser cero debido a algunos $p_l(x) = 0$ y pueden converger a óptimo local. [23]

Para resolver este problema se modifica la forma de calcular las probabilidades. Se aplica la correlación de Lapacle:

$$p(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i \mid D_{l-1}^{Se}) + 1}{N + r_i}$$

Siendo r_i el número de diferentes valores que puede tomar x_i. Con este cambio estamos dando una pequeña oportunidad a los elementos rechazados inicialmente por el algoritmo.

9. El lenguaje de programación R

Antes de describir el nacimiento y las principales características pasaremos a hablar sobre la filosofía S, sobre la que deriva R.

La filosofía de S

El lenguaje R es un dialecto de S. El lenguaje S, sus bibliotecas y herramientas relacionadas son el resultado de la investigación en computación estadística en Bell Laboratories, que comenzó a finales de los años setenta. En particular, es importante indicar que el lenguaje S tiene sus raíces en el análisis de datos y no viene del escenario de la programación tradicional. Los investigadores enfocaron en cómo hacer el análisis de datos más fácil.

El lenguaje R es un sistema de código abierto distribuido bajo la licencia GPL, que a veces se describe como un "clon libre" de S. Más exactamente, es un proyecto separado, basado en el lenguaje S, pero con un número de instrucciones de software adicionales.

R

R es un sistema para el análisis estadístico y gráfico creado por Ross Ihaka y Robert Gentleman. Es a la vez un programa y un lenguaje considerado como un dialecto del lenguaje S.

Este lenguaje es desarrollado y distribuido por un grupo de estadísticos conocidos como *R Development Core Team*.

Figura 10: Logo lenguaje R

R está disponible en algunas formas: las fuentes (escrito principalmente en C y con algunas rutinas en Fortran), esencialmente para máquinas Unix y Linux, o algunos binarios precompilados para Windows, Linux y Macintosh. Una vez instalado, el programa se ejecuta una consola con el símbolo '>', que indica que R está esperando un comando[24].

Los archivos necesarios para instalar R, además de las fuentes o de binarios precompilados, son distribuidos en la website *Comprehensive R Archive Network* (CRAN) donde también están las instrucciones para instalarlo.

Cómo funciona R

R es un lenguaje interpretado, no compilado, significa que todos los comando escritos con el teclado son directamente ejecutados sin necesidad de compilar un programa completo como en la mayoría de los lenguajes computacionales (C, Fortran, Pascal,...).

```
> x <- 2+2
> x
[1] 4
> 
```

Figura 11: Ejemplo línea de comandos

La sintaxis de R es muy simple e intuitiva, una función siempre necesita escribir con paréntesis, incluso sin tener ningún parámetro. Si se escribe el nombre de la función sin paréntesis, R mostrará el contenido de la función.

Cuando se ejecuta R, las variables, datos, funciones, resultados son guardados en una memoria activa del ordenador en forma de objetos con un nombre. El usuario puede realizar acciones sobre esos objetos con operadores (aritméticos, lógicos, de comparación,...) y funciones (que son también objetos)[13].

Los argumentos de la función puede ser objetos, algunos están definidos por defecto; estos valores pueden ser modificados especificando las opciones. Una función en R puede no requerir argumentos: por lo que todos los argumentos están definidos por defecto (y esos valores son los que el usuario puede modificar), o directamente no se han definido argumentos.

```
> pow <- function(x, y) {
+     result <- x^y
+     print(paste(x,"elevado a", y, "es", result))
+ }
> pow(y=2, x=3)
[1] "3 elevado a 2 es 9"
>
```

Figura 12: Ejemplo función

9.1. Paquetes en R

Unos de los aspectos más importantes y potente del lenguaje R es la opción de crear código reproducible y analítico. Hay muchas colecciones de funciones R, datos y código compilado que está muy estandarizados. Esto hace que sea mucho más fácil compartir entre los usuarios de R. Estos paquetes pueden ayudar a resolver muchas tareas sin la necesidad de escribir ni una sola línea de código por él mismo. Los paquetes mayormente son desarrollados en R, pero hay otros lenguajes como Java o C++ que también tienen un buen potencial para este tipo de práctica.

R trae preinstalados unos paquetes como el paquete *base*, que hace de R un lenguaje real. Hay muchos paquetes en repositorios de internet, el más importante es el anteriormente comentado `CRAN`, tiene sobre unos 7000 paquetes.

Con RStudio el desarrollo de paquetes es más fácil, ofrece una serie de funciones que hacen posible la creación de un paquete con simples clicks.

9.1.1. Creando un paquete en R

En esta sección pasaremos a una pequeña guía de cómo se puede crear un paquete en R con la ayuda de RStudio.

Estructura del paquete

La estructura de los paquetes está muy bien definida. Puede ser un poco confusa al principio, pero después de comprender lo básico del desarrollo de los paquetes, es muy sencillo. Esta estructura hace que programar en R sea mucho más sencillo.

Basicamente, está dividido en siete partes:

- **Descripción**: Información básica del paquete.
- **R**: Es una carpeta que contendrá el código R, como scripts o funciones.

- **Test**: Carpeta que contiene los scripts para el correcto funcionamiento del paquete.
- **Man**: Un subdirectorio para la documentación.
- **Vignettes**: Documentación más detallada.
- **Data**: Directorio que puede contener las estructuras para la ejecución del paquete.
- **Namespace**: Definición de las funciones habilitadas una vez cargas el paquete.

Instalando devtools

Devtools es un paquete escrito por Hadley Wickham y Winston Chang, parte del equipo de desarrollo de RStudio. Este paquete llama a una serie de paquetes que ayudan al desarrollo de paquetes en R.

Figura 13: RStudio IDE

Instalamos devtools desde CRAN con:

```
install.packages("devtools")
```

También, además de ayudarnos, nos obliga a seguir una serie de convenciones para que nuestro paquete esté de forma estandarizada.

Creando un proyecto de paquete en RStudio

Para crear un proyecto de paquete en RStudio, clickamos sobre **File** | **New Project...** y seleccionamos **New Directory**.

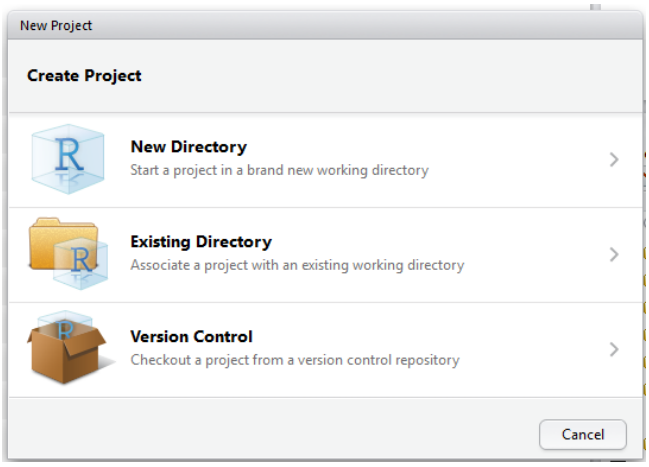


Figura 14: Menú de creación de RStudio

En la siguiente pantalla seleccionamos **R package**

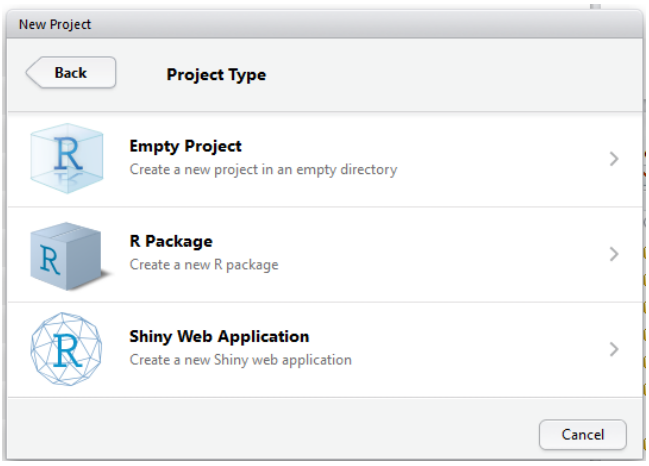


Figura 15: Opción paquete en R

Ahora tenemos que configurar los parámetros principales del paquete en R, como pueden ser el tipo de paquete, el nombre, la opción de crear el paquete a partir de archivos fuente y crear un proyecto de paquete en algún subdirectorio.

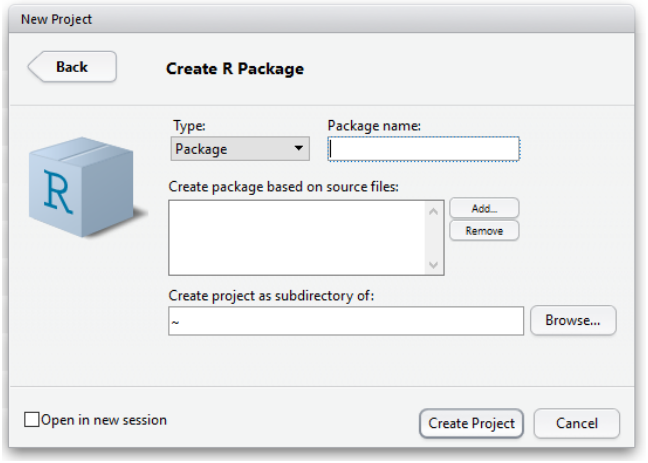


Figura 16: Pantalla de creación de paquete

Si sólo se quiere crear un paquete que contenga código R, selecciona el tipo Package. Hay otra opción con la que se puede utilizar código C++, que automáticamente instala todas las dependencias paquete `Rcpp` para extensión de los objetos R a nivel C++.

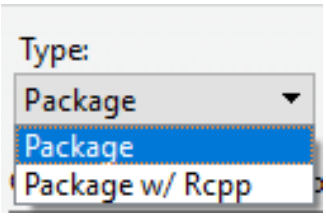


Figura 17: Selección de paquete

Una vez que se guarde el proyecto, RStudio creará automáticamente los archivos necesarios, como pueden ser las carpetas y los archivos *DESCRIPTION* y *NAMESPACE*.

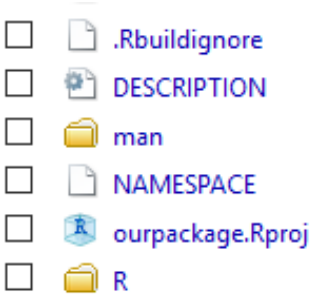


Figura 18: Estructura del paquete

Creando un script sencillo

El script que utilizaremos para nuestro ejemplo será una sencilla línea de código, una función. Se llamará `script.R` y será movido a la carpeta **R**.

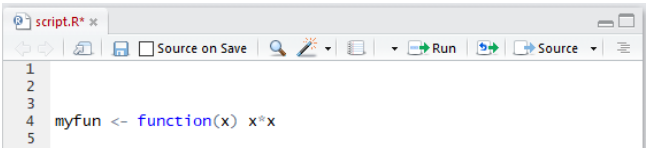


Figura 19: Función de nuestro paquete

Documentando nuestro código

Esta es una de las tareas más importantes; ya que, si nuestra intención es crear un paquete para que pueda utilizarlo los usuarios, necesitamos crear una documentación que sirva de guía o ayuda en su utilización.

Hay dos formas de hacerlo. La primera consiste en crear los archivos .Rd (R Documentation), que se encontrarán en la carpeta **man**. Para ello deberemos ir a **File** | **New File** | **R Documentation**, escribir el nombre de la función y escoger la plantilla correspondient. Como nuestro programa es una simple función, escogeríamos function. Una vez hecho esto deberíamos completar la documentación manualmente en el archivo .Rd. De esta manera puede ser un poco dificultoso, ya que al ser manual, puede ocasionar algún tipo de error. [13]

Otra manera, la más utilizada, es usar el paquete `roxygen2`. Nos proporciona una manera más sencilla de documentar el código, desde el propio código del programa. Su objetivo es hacer que documentar el código sea lo más fácil posible. También proporciona una forma estándar de documentación de paquetes. [24]

Instalamos roxygen2:

```
install.packages("roxygen2")
```

Figura 20: Paquete roxygen2

Recordar que para utilizar este paquete al construir y modificar la documentación de nuestro paquete debemos de modificar los parámetros de RStudio para que lo utilice. En **Tools** | **Project options** | **Builds Tools** | **Configure**

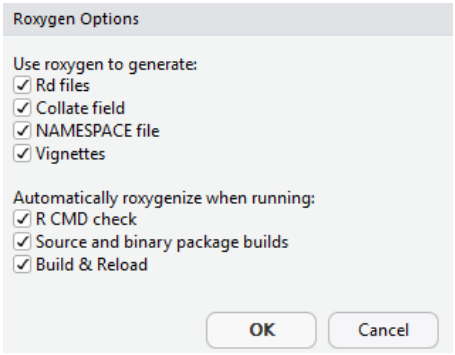


Figura 21: Configurando RStudio

Pasamos a documentar nuestro código:

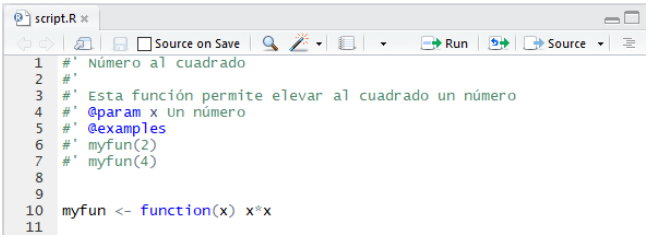


Figura 22: Comentando el código

Escribimos el nombre de nuestro paquete, una descripción de nuestra función; describimos los parámetros de la misma y algunos ejemplos. Para más información sobre este paquete, podemos acceder a la documentación oficial de CRAN y repositorio en GitHub .

Para generar la nueva documentación, debemos ejecutar:

```
devtools::document()
```

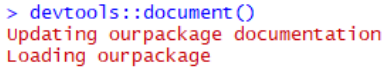


Figura 23: Actualizando documentación

Una vez realizado esto, en la carpeta man se habrá creado un archivo `myfun.Rd`.

Compilar el paquete

Cuando tenemos el programa y la documentación realizada, es hora de compilar nuestro paquete, para ello RStudio dispone de una serie de opciones en su pestaña **Build**

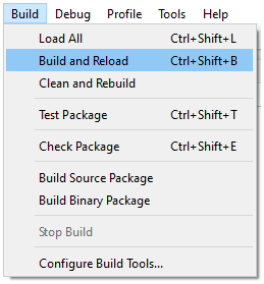


Figura 24: Compilando nuestro paquete

Podemos comprobar si la creación del paquete va a ser correcta por medio de test o check, para compilar utilizamos **Build and Reload**. Compilará el paquete además de reiniciar la sesión.

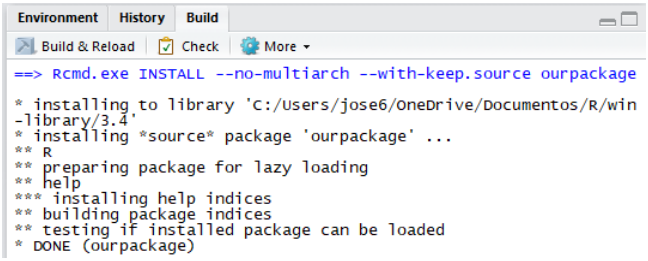


Figura 25: Consola de creación del paquete

Para comprobar que la documentación se ha creado correctamente podemos utilizar el comando ? con el nombre de la función que hemos descrito, es decir:

```
?myfun
```

Nos mostrará la documentación creada

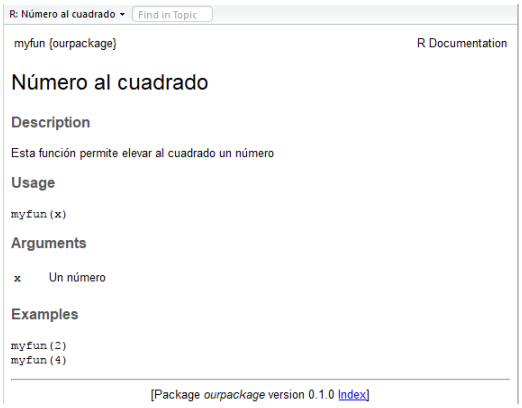


Figura 26: Visualización de la documentación

Ahora lo importante: el buen funcionamiento del paquete, haciendo uso de él

```
> myfun(2)
   [1] 4
> myfun(8)
  [1] 64
```

Figura 27: Comprobando resultados

10. El paquete EDAAlgorithms

El paquete que hemos creado implementa las diferentes inicializaciones comentadas en la sección 8, se permite crear una población inicial con una serie de parámetros que el usuario indique. Además, incluye un método que hará evolucionar la población inicial.

10.1. Entrada de datos

Para nuestro paquete la entrada de datos se hace por medio de archivos .csv, cada uno con una estructura diferente dependiendo de su contenido

10.1.1. Archivos con los que trabajamos

Las características de los objetos como pueden ser el nombre, peso y valor serán proporcionadas por archivo con la siguiente estructura:

```
req_ID;effort;satisfaction
r01;1;62
r02;4;55
r03;2;29
r04;3;41
```

Figura 28: Estructura de los objetos

La estructura del archivo debe ser la misma, el contenido del mismo puede ser cambiado al gusto del usuario.

También se dispone de la posibilidad de añadir dependencias entre los objetos por medio de archivos, para ello tendrá la siguiente estructura, tanto las dependencias de implicación, exclusión y combinación se componen de la misma:

```
head_req_ID;tail_req_ID
r03;r01
r19;r05
r02;r08
r15;r06
```

Figura 29: Estructura de las dependencias

Donde la primera columna, para el casos de las dependencias de implicación, corresponderá al antecesor del objeto de la segunda columna, es decir, por ejemplo para la primera columna, para que pueda estar el objeto $r01$ en la mochila, debe de estar anteriormente $r03$. Para las dependencias de exclusión significará que si está el objeto $r03$ el objeto $r01$ no puede estar en la mochila y viceversa. Por otra parte, para las dependencias de combinación, los objetos $r03$ y $r01$ deben de estar conjuntamente en la mochila. Hay que destacar que nuestro paquete sólo admite dependencias de combinación por

pares, no debe de estar la misma variables en dos filas distintas en el archivo de dependencias de combinación.

10.2. Utilización del paquete

Hay diferentes formas para inicializar una población, este paquete está implementado de tal manera que podemos crear una población inicial por medio de distintos enfoques y, por otra parte, evolucionar esta por medio de generaciones. La inicialización de la evolución están separadas para que el usuario pueda comprobar la población surgida de las distintas inicializaciones y su evolución en las distintas generaciones, además de poder cambiar los parámetros de ambos métodos.

10.3. Inicialización

10.3.1. Sin dependencias

Inicialización por medio de distribución uniforme

Cada objeto tiene la misma probabilidad, independientemente de los otros objetos e independiente de su ratio de beneficio y peso. Disponemos de tres métodos con una inialización mediante una distribución uniforme para cada objeto.

Este método hace una evaluación secuencial de los objetos, con un porcentaje del 50% para objeto. Por lo que la distribución tendría la siguiente forma:

$$p(x_1), \ldots, p(x_i), \ldots, p(x_n) = (0.5, \ldots, 0.5, \ldots, 0.5)$$

Método

```
initialize_PopulationSS(data, total_Weight, solNum, exc, imp, comb)
```

Ejemplo

```
> initialize_PopulationSS(data = "nrp20_data.csv",
              total_Weight = 46,
              solNum = 20,
              exc= "nrp20_dep_exc.csv",
              imp= "nrp20_dep_imp.csv",
              comb="nrp20_dep_comb.csv")
```

Parámetros de la función

- **data**: Nombre del archivo .csv que contiene los valores nombre, peso y satisfacción de nuestros objetos.
- **total_Weight**: Peso máximo de la mochila.
- **solNum**: Número de soluciones factibles en la población inicial.
- **exc, imp y comb**: Nombre de los archivo .csv que contienen las dependencias de exclusión, implicación y combinación respectivamente de los objetos.

Resultado

Un objeto dataframe con tantas soluciones como indicamos.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
r01+r02	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
r03	0	0	0	1	0	1	0	1	1	0	1	1	1	1	1	0	0	0	1	1
r04	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r05	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
r07	0	1	1	1	0	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1
r08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r09	1	1	0	1	1	1	1	0	0	0	1	0	1	1	1	0	0	1	1	0
r10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r11	0	1	1	1	0	1	1	0	0	0	0	0	0	1	0	0	0	1	1	1
r12	1	1	1	0	0	1	1	1	1	0	0	1	0	0	1	1	0	0	0	0
r13	1	1	1	1	1	0	1	1	1	0	1	0	1	1	1	0	1	1	0	1
r14	0	1	0	1	1	1	1	1	1	0	0	1	1	0	0	0	1	0	1	1
r15+r16	1	0	1	1	1	0	0	1	0	1	0	1	1	1	0	0	0	1	1	1
r17+r18	1	0	0	1	1	0	1	1	0	1	0	1	1	1	1	1	1	1	0	1
r19+r20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 30: Población inicial por medio de distribución uniforme

Por otra parte, podemos crear esta misma población evaluando aleatoriamente cada objeto. Para ello tenemos el siguiente método:

```
> initialize_PopulationRS(data, total_Weight, solNum, exc, imp, comb)
```

También disponemos de otro método de inicialización uniforme, con una probabilidad de cada objeto dada por:

$$p(x_1), \ldots, p(x_i), \ldots, p(x_n) = (\frac{C}{\sum_{i=1}^{n} c_i}, \ldots, \frac{C}{\sum_{i=1}^{n} c_i}, \ldots, \frac{C}{\sum_{i=1}^{n} c_i})$$

Método

```
> initialize_PopulationUR(data, total_Weight, solNum, exc, imp, comb)
```

Inicialización por medio otras distribuciones

Para esta sección tenemos dos métodos, uno que utiliza la probabilidad proporcional de su ratio entre la satisfacción y peso. Se calcula por medio de:

$$p(x_1), \ldots, p(x_i), \ldots, p(x_n) \propto (\frac{b_i}{c_1}, \ldots, \frac{b_i}{c_i}, \ldots, \frac{b_n}{c_n})$$

Método

```
> initialize_PopulationPR(data, total_Weight, solNum, exc, imp, comb)
```

El siguiente método utiliza una semilla probabilística. Dada una solución inicial (población) se obtiene una población inicial a partir de la simulación de la siguiente distribución de probabilidad:

$$(p(x_1), \ldots, p(x_i), \ldots, p(x_n))$$

donde para todos los $i = 1, \ldots, n$.

$$p(x_i) \begin{cases} \text{seed} & \text{si el objeto } x_i \text{ es seleccionado en la primera población.} \\ 1 - \text{seed} & \text{si el objeto } x_i \text{ no es seleccionado en la primera población.} \end{cases}$$

El valor de α puede ser elegido por medio del usuario, siendo un parámetro para el método llamado seed.

Método

```
> initialize_PopulationPS(data, total_Weight, solNum, imp, exc, comb, seed)
```

10.3.2. Dependencias bi-variada

También disponemos de inicializar una población a partir de otra por medio de dependencias entre pares de objetos, esto viene a ser:

$$p_\pi(x) = \{p_\pi(x) \mid p_\pi(x) = p(x_1 \mid x_2)p(x_2 \mid x_3) \dots p(x_{i-1} \mid x_i)p(x_i)\}$$

Método

```
> mimic (data, initialPoblation, total_Weight, solNum, imp, exc, comb)
```

Una vez creada la población inicial a partir de los anteriores métodos de inicialización podemos obtener la distribución de probabilidad condicionada, a partir del orden de evaluación de los objetos que se consigue por medio de la entropía de los objetos.

10.4. Evolución

Una de las partes más importantes de los EDAs es el método de evolución, dependiendo del número de generaciones, población inicial y el factor de selección pueden determinar mucho el resultado.

Método

```
> evolutionPoblation(data, initialPoblation, selectionFactor,
                numberGenerations, total_Weight, imp, exc, comb)
```

Ejemplo

```
evolutionPoblation(data = "data/nrp20_data.csv",
                        initialPoblation = initial,
                        selectionFactor = 0.5,
                        numberGenerations = 15,
                        total_Weight = 46,
                        exc = "nrp20_dep_exc.csv",
                        imp = "nrp20_dep_imp.csv",
                        comb = "nrp20_dep_comb.csv")
```

Parámetros de la función

- **data**: Nombre del archivo .csv que contiene los valores nombre, peso y satisfacción de nuestros objetos.
- **initialPoblation**: Población inicial creada por los métodos anteriormente comentados.
- **selectionFactor**: Factor de selección para la obtención de las mejores soluciones de cada generación. Si el valor es de 0.5, se escogerán la mitad de las mejores soluciones.
- **numberGenerations**: Numero de generaciones por la que va a evolucionar nuestra población inicial.
- **total_Weight**: Peso máximo de la mochila.
- **exc, imp y comb**: Nombre de los archivo .csv que contienen las dependencias de exclusión, implicación y combinación respectivamente de los objetos.

El funcionamiento de este método es muy simple, dada una población inicial, obtiene una cantidad de las mejores soluciones por medio del factor de selección, de dicho subconjunto se extrae la probabilidad subyacente para cada objeto. Una vez muestreada la nueva población, realizamos los mismos pasos hasta que se cumplan el número de generaciones que se le han indicado en el método.

Resultado

Un dataframe de una columna con el mejor resultado del conjunto de soluciones de cada generación, el nombre de la columna es la solución de la generación indicada, con la configuración de los objetos y su evaluación.

	11
r01+r02	1
r03	1
r04	1
r05	0
r06	0
r07	1
r08	1
r09	1
r10	0
r11	1
r12	1
r13	0
r14	1
r15+r16	0
r17+r18	1
r19+r20	0
Evaluation	493

Figura 31: Mejor solución, obtenida en la generación 11

11. Resultados y discusión

En esta sección presentamos una serie de resultado experimentales con las diferentes configuraciones de los métodos, aunque la propia selección de estas puede ser un problema a estudiar. Por ejemplo, determinando el número de generaciones que hacen que el algoritmo no converga y se quede estancado.

Para la prueba del paquete desarrollado hemos utilizado el conjunto de 20 elementos publicado en [7] y, además, hemos generado un conjunto con 50 elementos. Las tablas 7 y 8 reflejan las características de estos conjuntos de datos.

Conjunto de datos de 20 requsitos

Requisitos	Dependencias de implicación	Dependencias de exclusión	Dependecias de combinación	Satisfacción total	Peso total
20	8	0	2	893	85

Tabla 7: Características conjunto de datos de 20 requisitos

Conjunto de datos de 50 requsitos

Requisitos	Dependencias de implicación	Dependencias de exclusión	Dependecias de combinación	Satisfacción total	Peso total
50	66	0	8	2110	291

Tabla 8: Características conjunto de datos de 50 requisitos

11.1. Creación de las pruebas

Para la realización de las pruebas se dictan una serie de parámetros para que los experimentos entre los distintos conjuntos sea similares, aunque cada uno dispone de diferentes conjuntos de dependencias, ha sido ejecutadas **10 veces**.

Límite de coste	Tamaño de la población	Generaciones	Factor de selección
$PesoTotal$*0.2 y $PesoTotal$*0.3	Mayor o igual al nº de requisitos sin implicación	Mayor o igual al nº de requisitos sin implicación	0.5

Tabla 9: Configuración de pruebas

Como hemos observado anteriormente el paquete EDAAlgorithms contiene 4 métodos de inicialización además de MIMIC que debe de tener como parámetro de entrada cualquier método de inicialización anterior, por lo que encontramos 10 formas distintas de inicializar una población. El tamaño de la población o las generaciones se justifica con la premisa de que al menos, por puro azar, puedan haber objetos que no tengan dependecias de implicación. Para el factor de selección se escoge 0.5 por defecto, ya que es lo habitual. Hay que destacar que el método por semilla probabilística (initialize_PopulationPS) se ha optado por un valor de semilla de 0.95, como expone el libro de Larrañaga. [15]

Los experimentos para el conjunto de datos de **20 requisitos** se hará con la siguiente configuración:

Límite de coste	Tamaño de la población	Generaciones
17 / 26	20	20

Tabla 10: Configuración de pruebas para el conjunto de 20 objetos

Debido al tamaño de este conjunto los resultados son los mismos independientemente del método de inicialización. Para este experimento los tiempos rondan los **2,7 segundos** para cada inicialización y obtención de la mejor solución de todas las generaciones creadas, con un beneficio de **516** y **426** y un coste de **25** y **17** para los casos de peso máximo 26 y 17 respectivamente.

Experimentación con 50 requsitos

Para el conjunto de 50 requisitos, se han realizado con los siguientes parámetros:

Límite de coste	Tamaño de la población	Generaciones
88 / 59	20	20

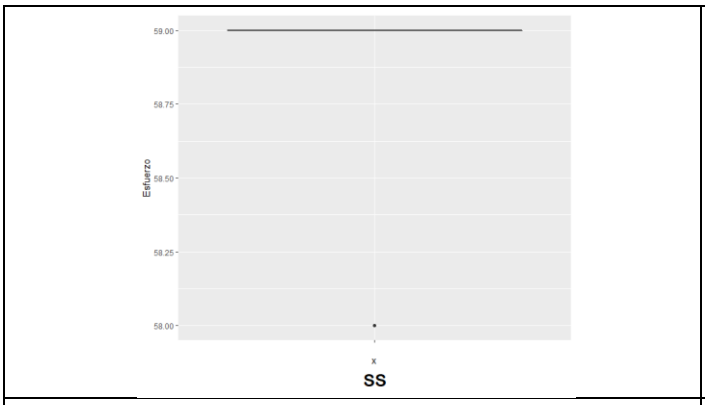
Tabla 11: Configuración de pruebas para el conjunto de 50 objetos

Aquí encontramos soluciones muchos más variadas que en el anterior conjunto, debido a que tiene muchas más opciones y el espacio de soluciones es más amplio. Si observamos en la siguientes páginas se muestran los resultados obtenidos de las distintas inicializaciones, dando resultados muy variados. Comenzamos comentándolos para un límite de coste máximo de 59.

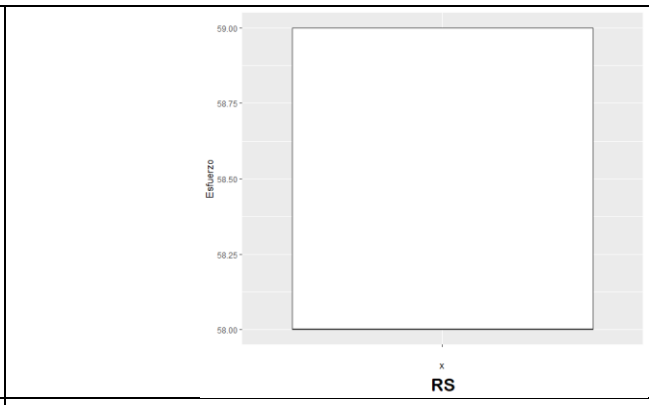
Para el coste observamos que todos los resultados son muy cercanos al coste máximo, en la búsqueda secuencial vemos que casi todas las soluciones se ajustan a ese valor. En otros métodos no es tanta la diferencia con un valor entre 57-59 y hasta en un caso 55.

Donde podemos percatar la diferencia entre las distintas soluciones es en la calidad, en el beneficio que ellas nos aportan. Para este caso, el método de la semilla probabilística (PS) nos proporciona

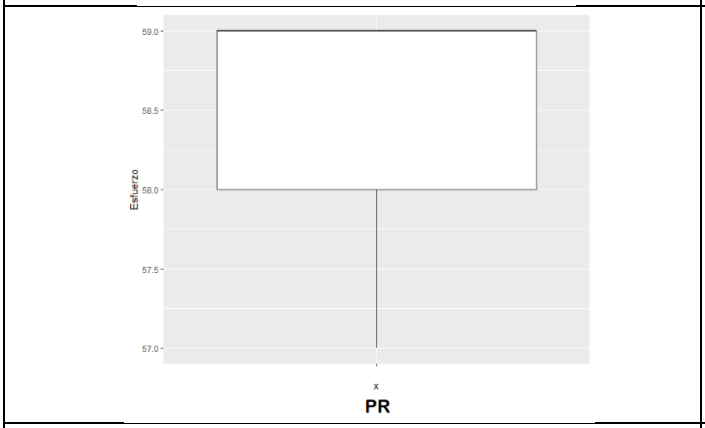
muy buenos, en la que destaca unos resultados muy compactos, no son tan amplios como podemos ver en SS (búsqueda lineal), RS (búsqueda aleatoria) o PS+MIMIC (semilla probabilística + MIMIC). En las siguientes páginas expondremos los resultados de esfuerzo y satisfacción para los casos de peso máximo de 59 y 88.



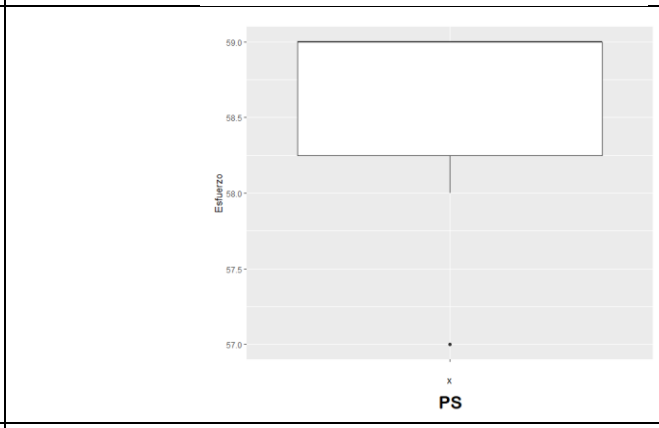
SS



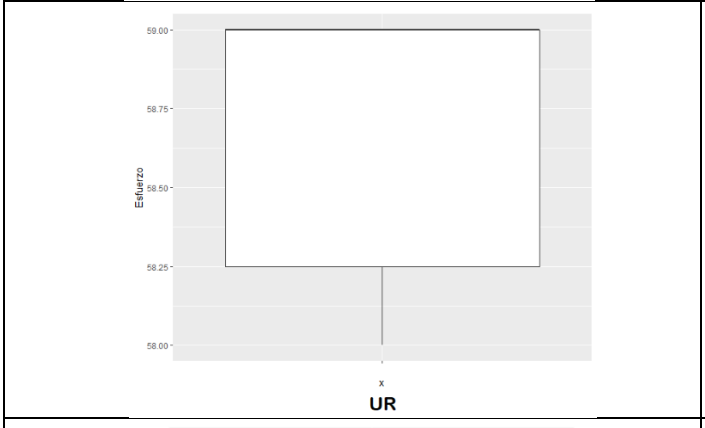
RS



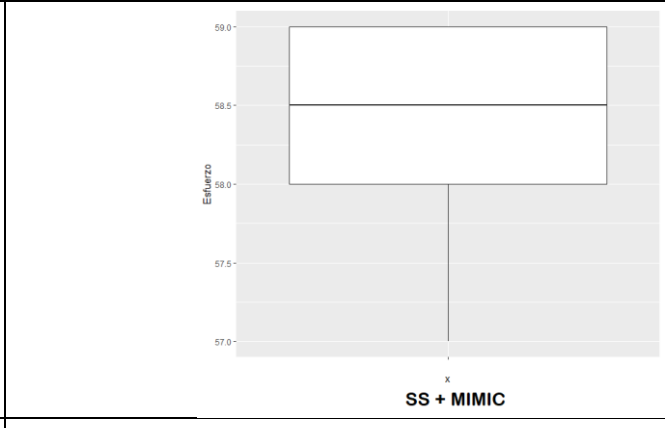
PR



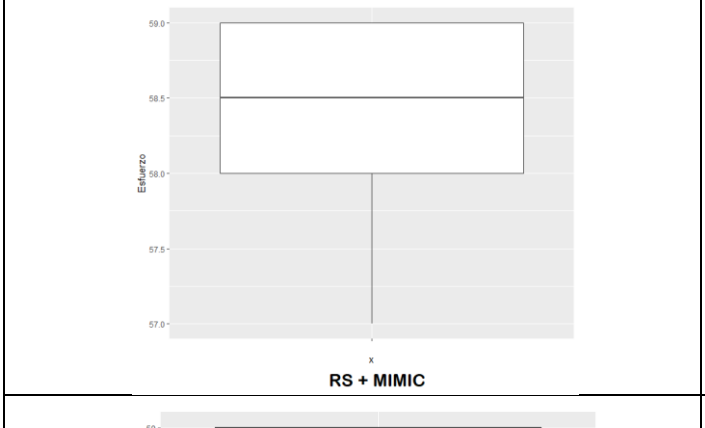
PS



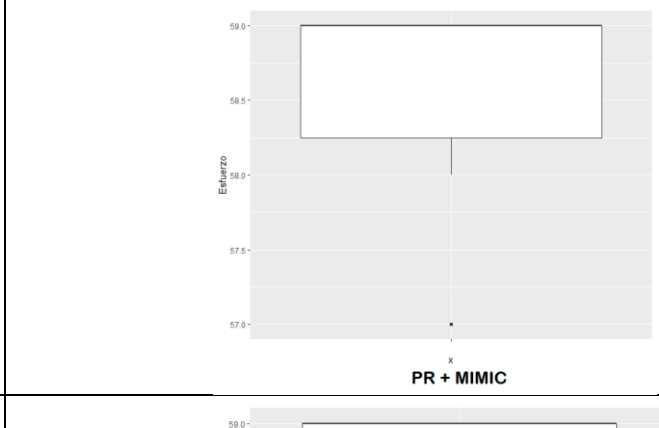
UR



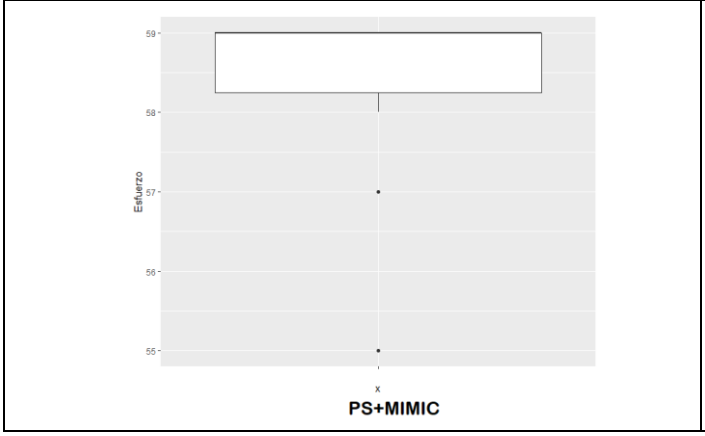
SS + MIMIC



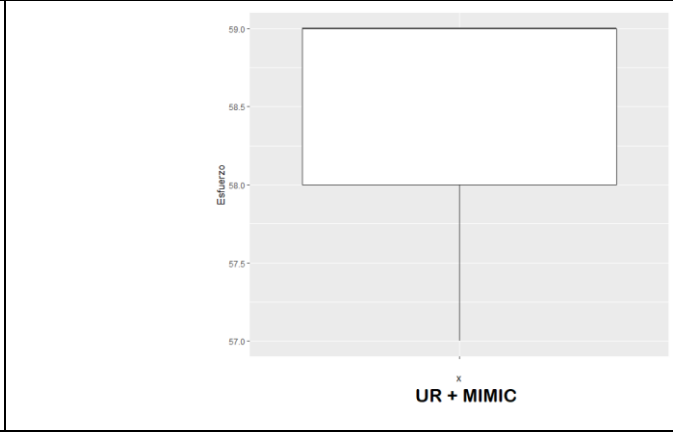
RS + MIMIC



PR + MIMIC

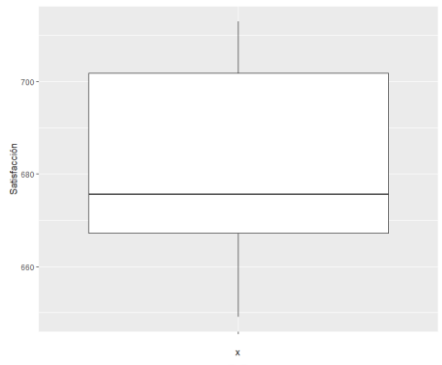


PS+MIMIC

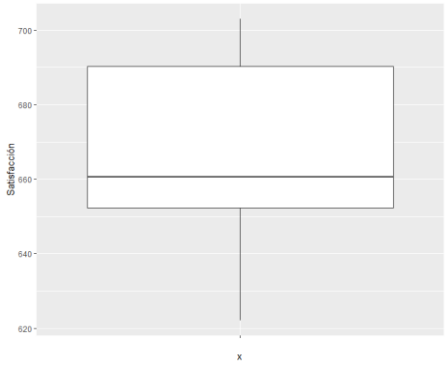


UR + MIMIC

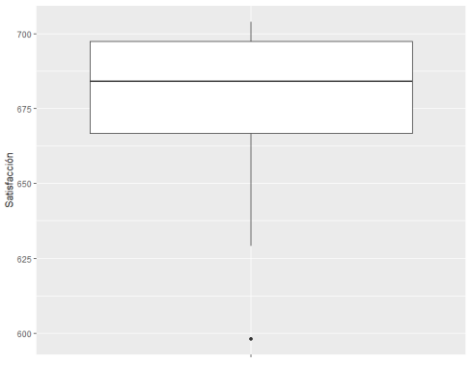
ESFUERZO PESO_MÁXIMO=59



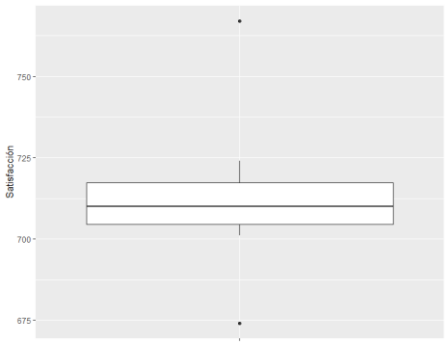
SS



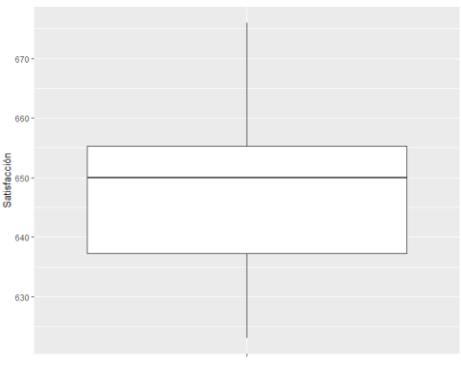
RS



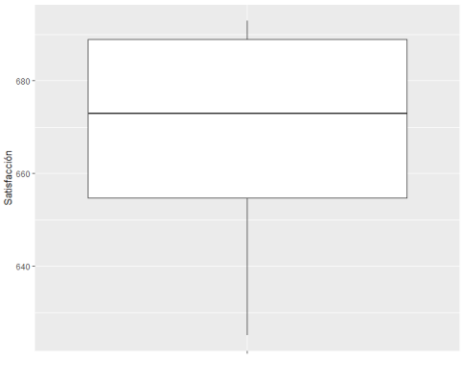
PR



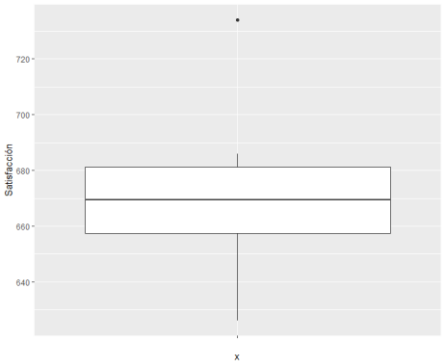
PS



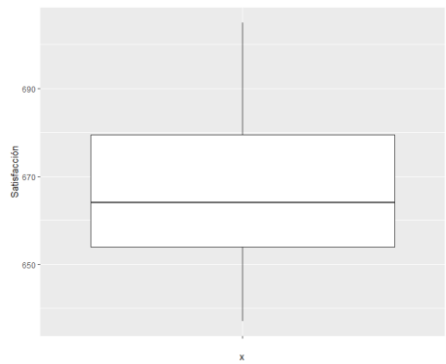
UR



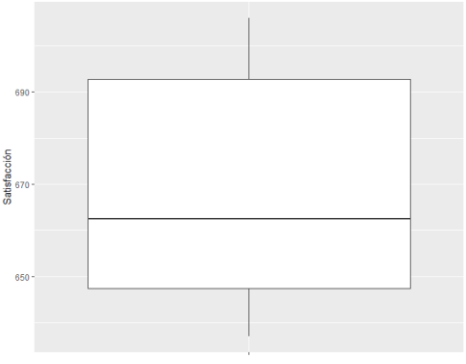
SS+MIMIC



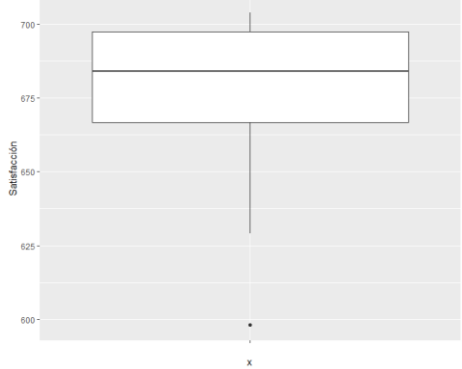
RS+MIMIC



PR+MIMIC



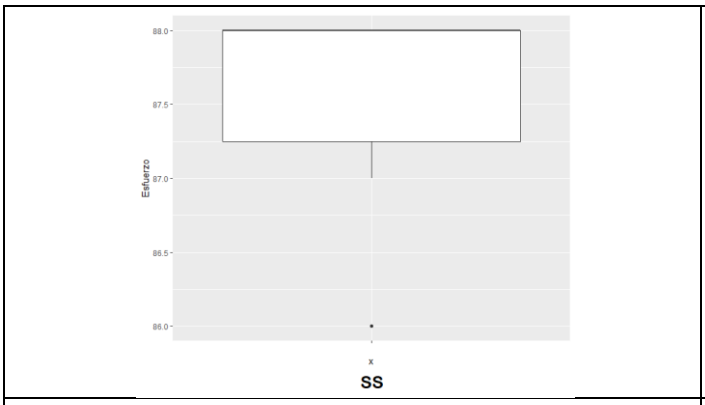
PS+MIMIC



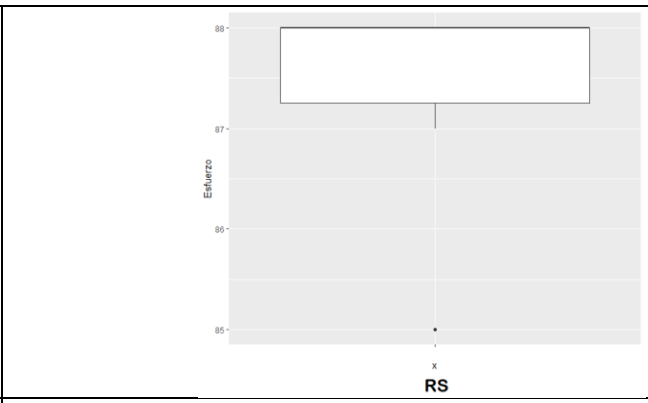
UR+MIMIC

SATISFACCIÓN PESO_MÁXIMO=59

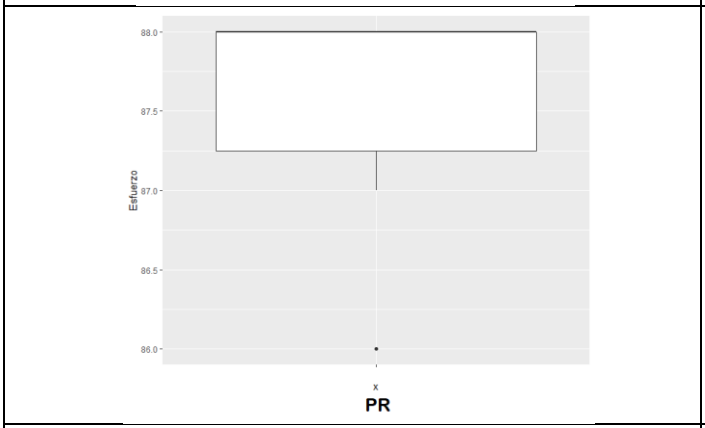
Para el caso de un peso máximo de 88 los resultados no son similares al anterior experimento, para el peso hay algunos métodos que no suelen llegar al peso máximo como puede ser el caso de PS+MIMIC o UR+MIMIC, pero otros como PS que si intenta obtener el peso máximo. Como comentamos, el valor de las soluciones está en su beneficio. Asombrosamente, PS+MIMIC es el que más beneficio suele tener, por lo tanto podemos concluir que la mejor solución tiene que intentar tener el máximo coste. Casi todas las soluciones están obteniendo soluciones con un beneficio entre 850 y 900.



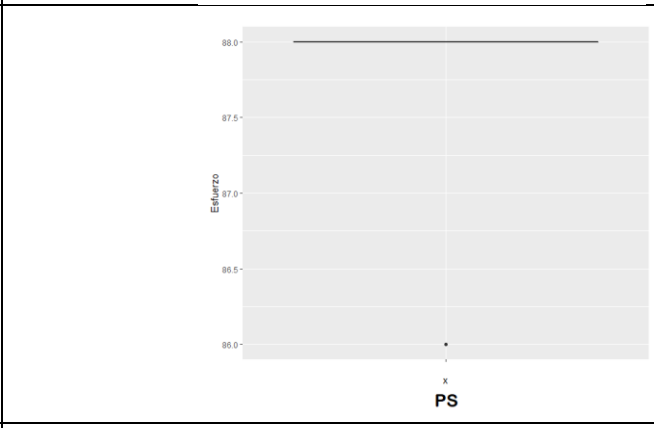
SS



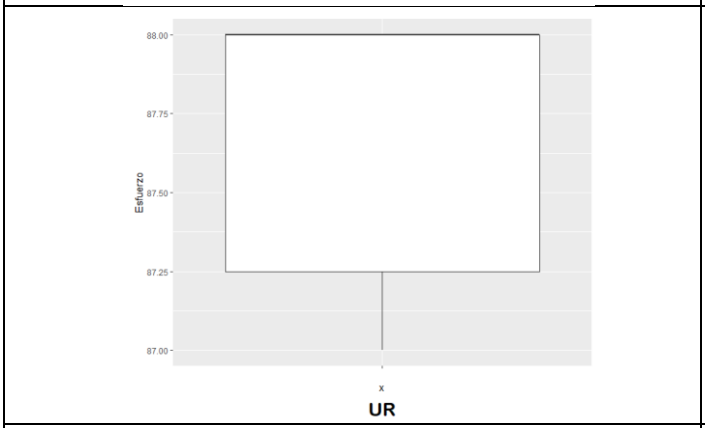
RS



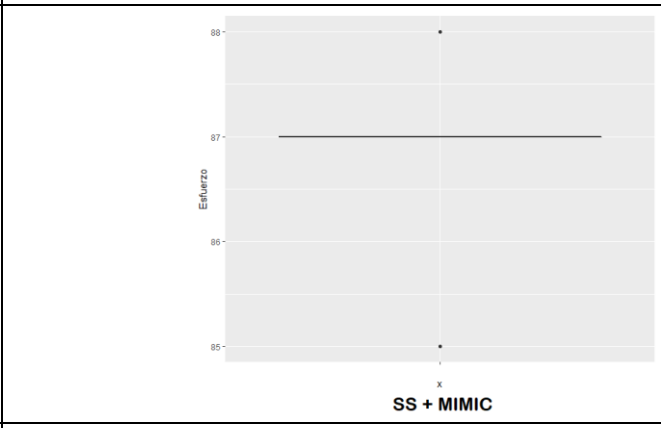
PR



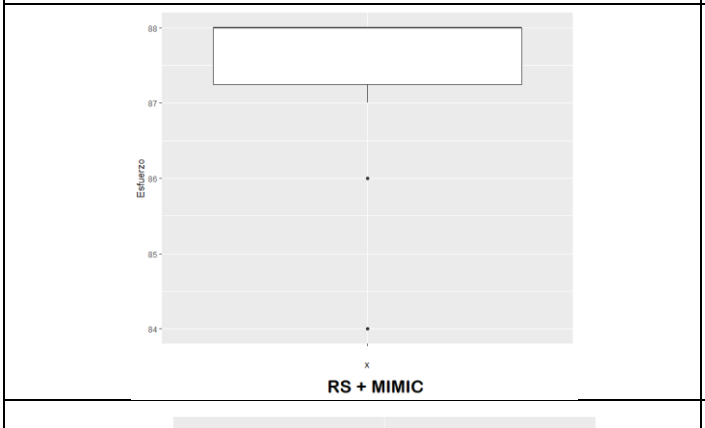
PS



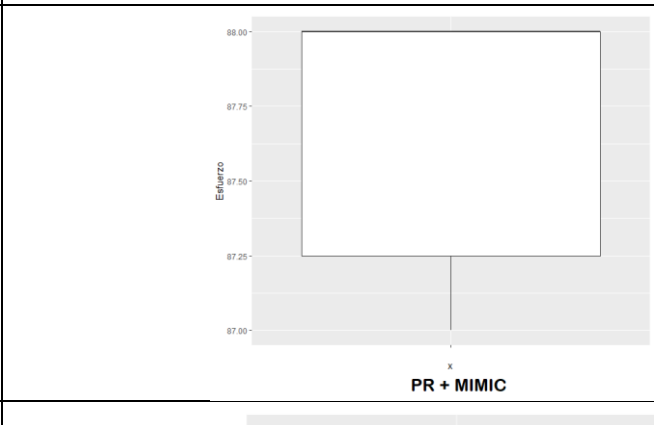
UR



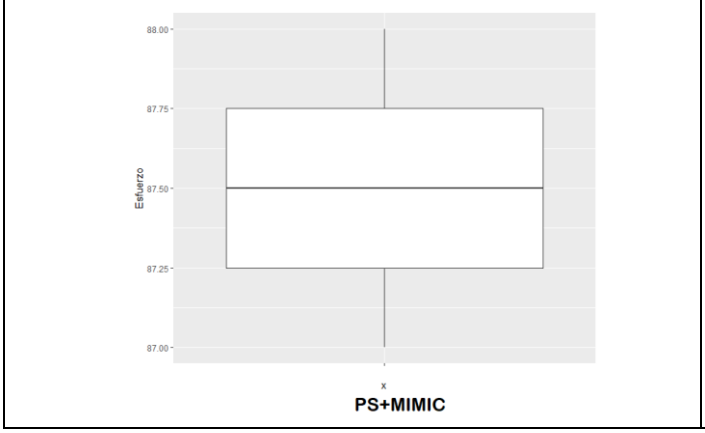
SS + MIMIC



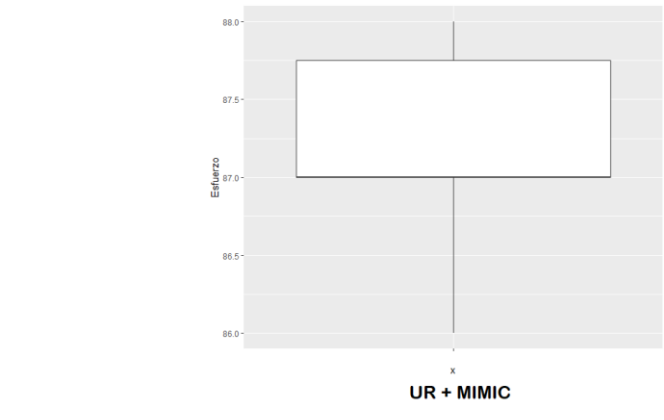
RS + MIMIC



PR + MIMIC

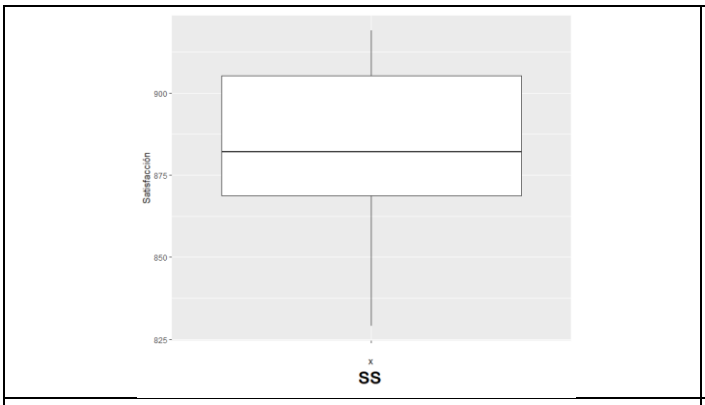


PS+MIMIC

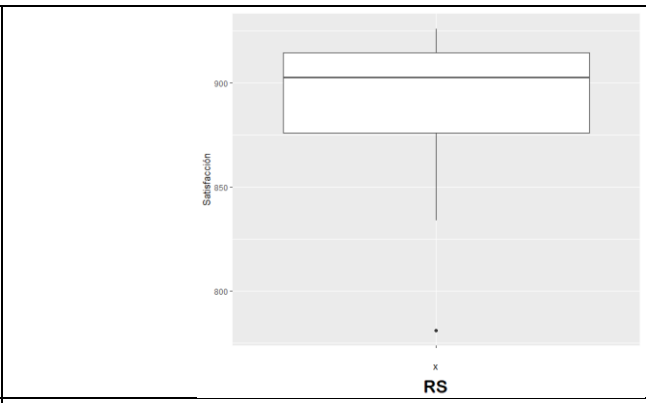


UR + MIMIC

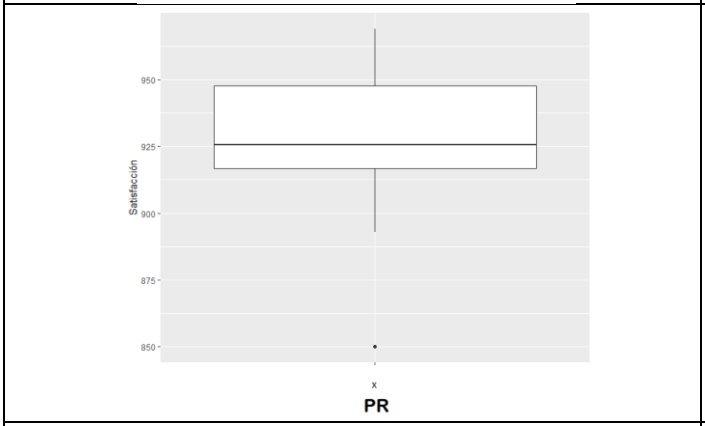
ESFUERZO PESO_MÁXIMO=88



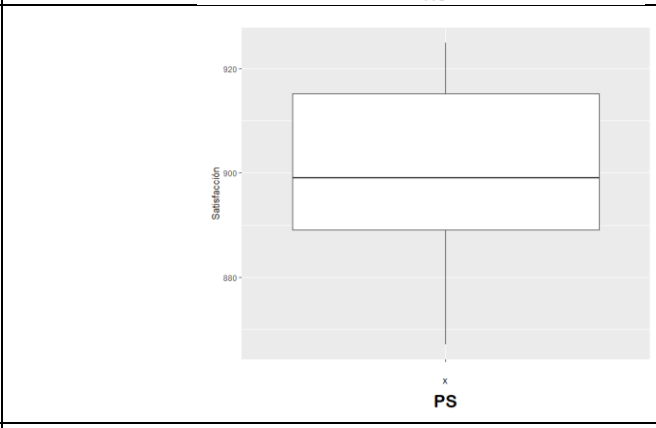
SS



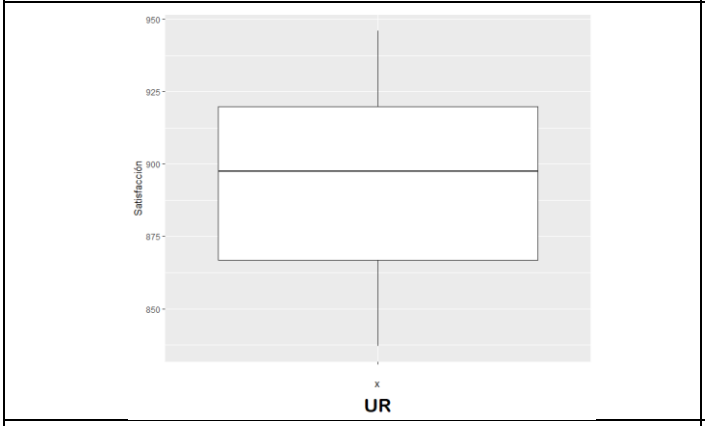
RS



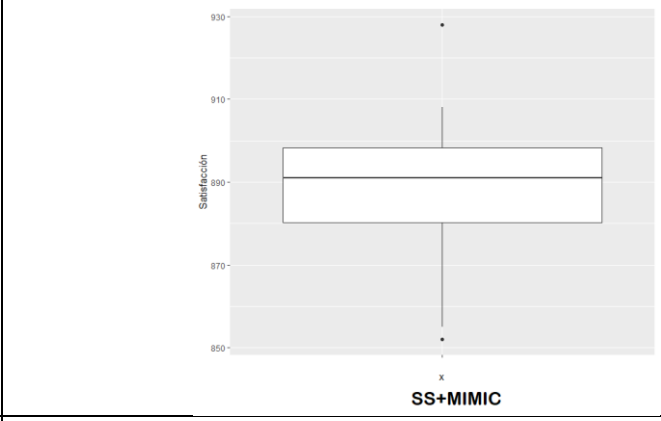
PR



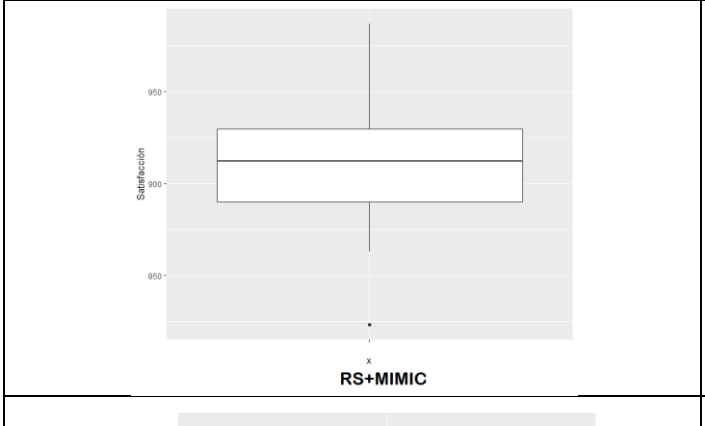
PS



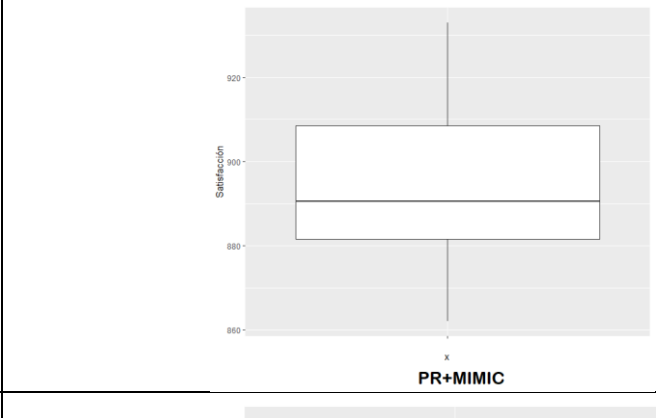
UR



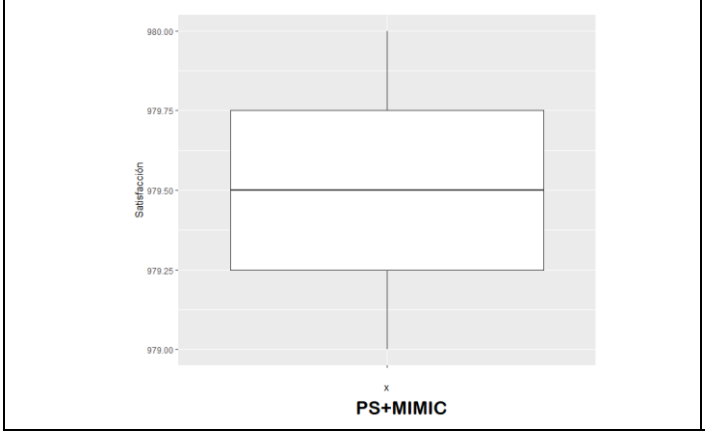
SS+MIMIC



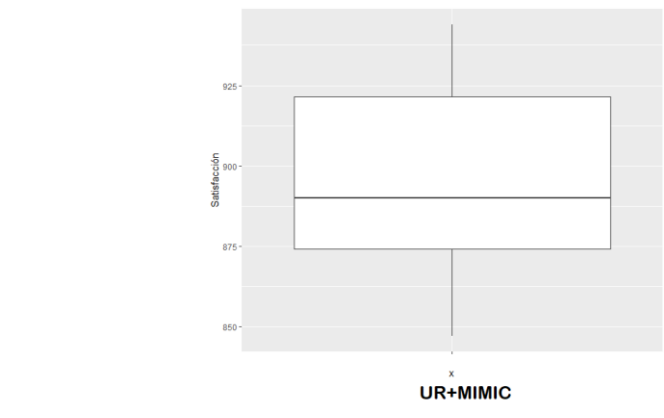
RS+MIMIC



PR+MIMIC



PS+MIMIC



UR+MIMIC

SATISFACCIÓN PESO_MÁXIMO=88

Aquí exponemos los tiempos medios y las carácterísticas de las soluciones obtenidas de la experimentación realizada, la mayoría de los métodos que realizan operaciones adicionales, como puede ser de PS o MIMIC que necesitan de una población anterior para poder realizar una nueva población. Como comentábamos anteriormente los resultados han sido calculado 10 veces, por lo que un número mayor supondría una mejora en la exactitud de los tiempos y la calidad de las soluciones.

También hay que destacar que además de la inicialización está el método de evolución, que necesita muestrear la población anterior para generar una nueva, en algunos casos puede darse la casualidad de que por medio del azar sea escogido un objeto, debido a que el método PLS puede estar realizando muestreos no válidos debido a las dependencias proporcionadas o al peso total de la mochila, el hecho de que un método se muestre aquí más rápido no significa que lo sea siempre.

50 Variables	Límite de coste	SS	RS	PR	PS	UR	SS+MIMIC	RS+MIMIC	PR+MIMIC	PS+MIMIC	UR+MIMIC
	59	54.359	57.027	54.437	57.727	54.03	57.86	58.601	55.409	62.549	59.973
Satisfacción		Mín: 649.0 1Q: 667.2 Med: 975.5 3Q: 701.8 Máx: 713.0	Mín: 622.0 1Q: 652.2 Med: 660.5 3Q: 690.2 Máx: 703.0	Mín: 634.0 1Q: 660.5 Med: 684.5 3Q: 691.5 Máx: 706.0	Mín: 674.0 1Q: 704.5 Med: 710.0 3Q: 717.2 Máx: 767.0	Mín: 611.0 1Q: 645.8 Med: 657.5 3Q: 672.8 Máx: 703.0	Mín: 625.0 1Q: 654.8 Med: 673.0 3Q: 689.0 Máx: 693.0	Mín: 626.0 1Q: 657.5 Med: 669.5 3Q: 681.2 Máx: 734.0	Mín: 637.0 1Q: 654.0 Med: 664.0 3Q: 679.5 Máx: 705.0	Mín: 635.0 1Q: 648.0 Med: 670.0 3Q: 693.2 Máx: 715.0	Mín: 850.0 1Q: 875.0 Med: 890.5 3Q: 920.0 Máx: 944.5
Esfuerzo		Mín: 58.0 1Q: 59.0 Med: 58.8 3Q: 59.0 Máx: 59.0	Mín: 58.0 1Q: 58.0 Med: 58.0 3Q: 59.0 Máx: 59.0	Mín: 58.0 1Q: 58.0 Med: 59.0 3Q: 59.0 Máx: 59.00	Mín: 57.0 1Q: 58.25 Med: 59.0 3Q: 59.0 Máx: 59.0	Mín: 57.0 1Q: 58.0 Med: 59.0 3Q: 59.0 Máx: 59.0	Mín: 57.0 1Q: 58.0 Med: 58.5 3Q: 59.0 Máx: 59.0	Mín: 57.0 1Q: 58.0 Med: 58.5 3Q: 59.0 Máx: 59.0	Mín: 57.0 1Q: 58.25 Med: 59.0 3Q: 59.0 Máx: 59.0	Mín: 58.0 1Q: 58.25 Med: 58.75 3Q: 59.0 Máx: 59.0	Mín: 57.0 1Q: 58.0 Med: 58.5 3Q: 59.0 Máx: 59.0

Tabla 12: Tiempos de las pruebas conjunto de 50 variables con límite de coste 59

50 Variables	Límite de coste	SS	RS	PR	PS	UR	SS+MIMIC	RS+MIMIC	PR+MIMIC	PS+MIMIC	UR+MIMIC
	88	57.589	57.066	57.888	54.437	61.056	62.708	62.266	59.155	63.545	59.973
Satisfacción		Mín: 829.0 1Q: 868.8 Med: 882.0 3Q: 905.2 Máx: 919.0	Mín: 781.0 1Q: 876.0 Med: 902.5 3Q: 914.5 Máx: 926.0	Mín: 850.0 1Q: 916.8 Med: 925.5 3Q: 947.8 Máx: 969.0	Mín: 867.0 1Q: 889.0 Med: 899.0 3Q: 915.2 Máx: 925.0	Mín: 837.0 1Q: 866.8 Med: 897.5 3Q: 919.8 Máx: 946.0	Mín: 852.0 1Q: 880.2 Med: 891.0 3Q: 89828 Máx: 928.0	Mín: 823.0 1Q: 890.0 Med: 912.0 3Q: 929.8 Máx: 987.0	Mín: 862.0 1Q: 881.0 Med: 890.5 3Q: 908.5 Máx: 933.0	Mín: 979.0 1Q: 979.25 Med: 979.5 3Q: 979.75 Máx: 980.0	Mín: 850.0 1Q: 875.0 Med: 890.5 3Q: 920.0 Máx: 944.5
Esfuerzo		Mín: 86.00 1Q: 87.25 Med: 88.00 3Q: 88.00 Máx: 88.00	Mín: 85.00 1Q: 87.25 Med: 88.00 3Q: 88.00 Máx: 88.00	Mín: 86.00 1Q: 87.25 Med: 88.00 3Q: 88.00 Máx: 88.00	Mín: 86.0 1Q: 88.0 Med: 88.0 3Q: 87.8 Máx: 88.0	Mín: 87.0 1Q: 87.25 Med: 88.0 3Q: 88.8 Máx: 88.0	Mín: 85.0 1Q: 87.0 Med: 88.0 3Q: 87.0 Máx: 88.0	Mín: 84.0 1Q: 87.27 Med: 88.0 3Q: 88.0 Máx: 88.0	Mín: 87.0 1Q: 87.25 Med: 88.0 3Q: 88.0 Máx: 88.0	Mín: 87.0 1Q: 87.25 Med: 87.5 3Q: 87.75 Máx: 88.0	Mín: 86.0 1Q: 87.0 Med: 87.5 3Q: 87.7 Máx: 88.0

Tabla 13: Tiempos de las pruebas conjunto de 50 variables con límite de coste 88

12. Conclusión y trabajos futuros

El uso de este paquete está destinado a la obtención de soluciones ajustando los parámetros que desee el usuario. Tiene a disposición una serie de métodos para ver el comportamiento de los diferentes algoritmos.

Los EDAs están entre los algoritmos evolutivos más potentes actualmente disponibles y hay numerosos aplicaciones en las que se ha demostrado que los EDA resuelven problemas insolubles con otros técnicas. Además, la mayoría de los EDAs nos ofrecen ventajas adicionales sobre los algoritmos evolutivos y otras metaherísticas, como es la propiedad de poder representar la población utilizando un modelo de probabilidad o de incluir información importante de rigurosas maneras.

El problema de la mochila es un problema clásico, que se adapta a una infinidad de casuísticas en el mundo real y puede ser extrapolado a muchos problemas como pueden ser la elección de recursos y planificación. No se había utilizado los EDAs en los problemas de elección de requisitos en la ingeniería del software.

Por otra parte, el lenguaje R nos permite la utilización de un lenguaje de programación potente, que trabaja muy bien sobre estructura ya definidas con unos métodos muy simples para simplificar la escritura del código. También nos proporciona un soporte para realizar paquetes, que nos proporcionan nuevas funcionalidades. En este escenario el IDE (Integrated Development Environment) RStudio nos proporciona una serie de herramientas que ayudan a la construcción del paquete, con una serie de herramientas con fines como simplificar el proceso o estandarizar el maquetado y funcionamiento del paquete.

La combinación de todas estas herramientas hacen del problema un lugar donde experimentar y añadir nuevas funcionalidades en el futuro.

Como continuación de este trabajo, se pueden añadir los siguientes aspectos:

- Añadir los algoritmos EBNA y EcGA al paquete.
- Optimizar los tiempos en la ejecución de las funciones del paquete.
- Publicar el paquete en el sitio oficial CRAN.
- Estudiar sobre los algoritmos híbridos.

Bibliografía

[1] Joseph Adler. *R in a nutshell*. O'Reilly, Sebastopol, CA, 2010.

[2] Shumeet Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, January 1994.

[3] Endika Bengoetxea. *Estimation of distribution algorithms*. PhD thesis, 2002.

[4] Endika Bengoetxea, Pedro Larrañaga, Isabelle Bloch, and Aymeric Perchant. Estimation of distribution algorithms: A new evolutionary computation approach for graph matching problems. In Mário Figueiredo, Josiane Zerubia, and Anil K. Jain, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 454–469, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[5] Roberto Santana Pedro Larrañaga Carlos Echegoyen, Jose A. Lozano. Exact bayesian network learning in estimation of distribution algorithms. *2007 IEEE Congress on Evolutionary Computation*, 2007.

[6] Jeremy S. De Bonet, Charles L. Isbell, Jr., and Paul Viola. Mimic: Finding optima by estimating probability densities. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, pages 424–430, Cambridge, MA, USA, 1996. MIT Press.

[7] José del Sagrado, Isabel M. del Águila, and Francisco J. Orellana. Multi-objective ant colony optimization for requirements selection. *Empirical Software Engineering*, 20(3):577–610, Jun 2015.

[8] B.Y. Kang D.W. Kim, S. Ko. Structure learning of bayesian networks by estimation of distribution algorithms with transpose mutation. 11(11), 2013.

[9] Isabelle Bloch Aymeric Perchant Claudia Boeres Endika Bengoetxea, Pedro Larrañaga. Inexact graphmatching by means of estimation of distribution algorithms. *ELSERVIER*, (14), 2001.

[10] Georges Harik, Fernando G. Lobo, and David E. Goldberg. The compact genetic algorithm. In *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, pages 523–528, 1998.

[11] Georges R. Harik, Fernando G. Lobo, and Kumara Sastry. Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In Martin Pelikan, Kumara Sastry, and Erick Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling*, volume 33 of *Studies in Computational Intelligence*, pages 39–61. Springer, 2006.

[12] Georges R. Harik, Fernando G. Lobo, and Kumara Sastry. Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In *Scalable Optimization via Probabilistic Modeling*, pages 39–61. 2006.

[13] Julian Hillebrand and Maximilian H. Nierhoff. *Mastering RStudio – Develop, Communicate, and Collaborate with R*. Packt Publishing, 2015.

[14] P. Larrañaga, R. Etxeberria, J. A. Lozano, J.M. Peña, and J. M. Pe na. Optimization by learning and simulation of bayesian and gaussian networks, 1999.

[15] Pedro Larraanaga and Jose A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[16] Helong Li, Yi Hong, and Sam Kwong. Subspace estimation of distribution algorithms: To perturb part of all variables in estimation of distribution algorithms. *Applied Soft Computing*, 11(3):2974 – 2989, 2011.

[17] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[18] Heinz Mühlenbein and Robin Höns. The factorized distribution algorithm and the minimum relative entropy principle. In Dirk V. Arnold, Thomas Jansen, Michael D. Vose, and Jonathan E. Rowe, editors, *Theory of Evolutionary Algorithms*, number 06061 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[19] Martin Pelikan and Heinz Muehlenbein. The bivariate marginal distribution algorithm. In Rajkumar Roy, Takeshi Furuhashi, and Pravir K. Chawdhry, editors, *Advances in Soft Computing*, pages 521–535, London, 1999. Springer London.

[20] Martin Pelikan and Heinz Mühlenbein. Marginal distributions in evolutionary algorithms. In *Proceedings of the International Conference on Genetic Algorithms Mendel*, volume 98, pages 90–95. Citeseer, 1999.

[21] Martin Pelikan, Kumara Sastry, and Erick Cantú-Paz. *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications (Studies in Computational Intelligence)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[22] Roberto Santana Yvan Saeys Jose Luis Flores Jose Antonio Lozano Yves Van de Peer Rosa Blanco Víctor Robles Concha Bielza Pedro Larrañaga Rubén Armañanzas, Iñaki Inza. A review of estimation of distribution algorithms in bioinformatics. *BioData Mining 2008*.

[23] Hitoshi Iba Topon Kumar Paul. Linear and combinatorial optimizations by estimation of distribution algorithms. *9th MPS Symposium on Evolutionary Computation*, 2002.

[24] Hadley Wickham. *R Packages*. O'Reilly Media, Inc., 1st edition, 2015.

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

Los algoritmos de estimación de distribución suponen un nuevo campo en la computación evolutiva, siguiendo la misma estructura que los algoritmos más utilizados en este entorno como pueden ser los algoritmos genéticos proponen una solución más sencilla y más adaptable al tipo de problema, ya que no necesitan tantos parámetros como pueden ser de cruce, mutación, etc, que pueden ser difíciles de calcular y optimizar.

Para ello, los EDAs nos muestran un opción en el que se esos parámetros son sustituidos por un modelo de probabilidad que se adapta a las condiciones y dependencia de las variables del problema.

Cómo problema a representar utilizaremos un problema clásico, el problema de la mochila, concretamente el problema de la mochila 0-1.

Utilizaremos el lenguaje R con el IDE RStudio para la creación de un paquete que pueda ser distribuido por toda la comunidad de este lenguaje y tener un entorno donde mejorar y buscar opiniones entre todos los desarrolladores.