EXTENSIÓN DE UN PAQUETE DE ALGORITMOS METAHEURÍSTICOS EN R PARA LA DOCENCIA

AUTOR
Ander Carreño López

TUTORES
Josu Ceberio Uribe
Borja Calvo Molinos

Escuela Universitaria de Ingenieros de Bilbao
Universidad del País Vasco /
Euskal Herriko Unibertsitatea
17/06/2016

Ander Carreño López: *Extensión de un Paquete de Algoritmos Metaheurísticos en R para la Docencia*, 17/06/2016.

Trabajo Fin de Grado presentado dentro del Grado en Ingeniería Informática de Gestión y Sistemas de Información en la Escuela Universitaria de Ingenieros de Bilbao (UPV-EHU).

A mi familia y amigos.

RESUMEN

En los últimos años se han realizado numerosos avances en el campo de la optimización combinatoria. Los algoritmos evolutivos han demostrado ser una herramienta muy valiosa para resolver problemas de optimización costosos. En concreto, los Algoritmos de Estimación de Distribuciones han atraído la atención de los investigadores y se han hecho numerosos avances al respecto. Sin embargo, no existe software que acerque estos avances a los alumnos. Este Trabajo de Fin de Grado trata de extender el paquete *metaheuR* para que los estudiantes puedan comparar, combinar y aprender estos algoritmos resolviendo los problemas de optimización clásicos junto con el libro *Bilaketa Heuristikoak* aun en desarrollo. Bajo el lenguaje de programación R, este proyecto recoge tres Algoritmos de Estimación de Distribuciones, el *Estimation of Bayesian Network Algorithm* que aprende una red Bayesiana como modelo probabilístico; el *Edge Histogram Based Sampling Algorithm* que aprende probabilidades marginales de orden dos y el *Plackett-Luce Estimation of Distribution Algorithm* que aprende un vector de pesos de la población. Además, se han realizado experimentos que verifican el correcto funcionamiento de los modelos implementados así como ejecuciones sobre problemas de optimización tales como el *Travelling Salesman Problem* o el *Maximum Independence Set*. Con estos resultados, también se han realizado comparativas y análisis de los resultados que dan una visión de la eficiencia de cada uno de los algoritmos.

AGRADECIMIENTOS

En primer lugar agradecer a los doctores Josu Ceberio y Borja Calvo por su excelente colaboración y dedicación. Sus sugerencias y correcciones han hecho posible este trabajo. Además, siempre han tenido una gran disponibilidad digna de agradecer.

Agradecer también a Nerea Martín por su constante apoyo especialmente en los momentos de mayor carga de trabajo.

Agradecer a mi hermano Asier por las recomendaciones y por el apoyo durante todo el trabajo. Así mismo, agradecer a mi padre Javier y mis abuelas Mª Luisa y Dolores por la ilusión y emoción que me han transmitido.

Por último agradecer a la Escuela Universitaria de Ingenieros de Bilbao por la posibilidad de realizar este proyecto.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

En este capítulo se introduce el trabajo de fin de grado realizado por Ander Carreño López titulado *"Extensión de un Paquete de Algoritmos Metaheurísticos en R para la Docencia"*.

1.1 ESTRUCTURA DEL DOCUMENTO

Este documento se compone de 8 capítulos y 2 apéndices. El Capítulo 1 es una introducción general. El Capítulo 2 se centra en los conceptos teóricos de los algoritmos mencionados a lo largo del documento. En el Capítulo 3 se recoge el objetivo principal de este proyecto, la arquitectura, el alcance, la especificación de tareas, el diagrama de *Gantt*, los riesgos posibles a lo largo del proyecto, y la evaluación económica. El Capítulo 4 se dedica a la captura de requisitos del proyecto. A continuación, en el Capítulo 5, se describe el análisis y diseño propuesto con los diagramas de clases y de secuencia que corresponden a los métodos implementados en el trabajo. En el Capítulo 6 se expone la cómo se ha desarrollado el trabajo. El Capítulo 7 plasma la experimentación realizada para comprobar los algoritmos desarrollados. Las conclusiones y el trabajo futuro forman el Capítulo 8, siendo este el último capítulo del documento.

1.2 PROBLEMA ABORDADO

En este proyecto se aborda el desarrollo de una extensión de un paquete de algoritmos metaheurísticos para la docencia en el lenguaje de programación R. El paquete tiene el nombre de *metaheuR* y está siendo desarrollada como material didáctico para la asignatura *Bilaketa Heurístikoak*[1] junto con el libro que se está escribiendo en paralelo.

Dado que es un paquete con algoritmos puramente científicos, el desarrollo de estos no es trivial. El fundamento matemático en el que se apoyan hace

1 La asignatura *Bilaketa Heuristikoak* se imparte en el grado de Ingeniería Informática en la Facultad de San Sebastián

que la adquisición de los conocimientos sea un trabajo añadido que se debe cumplimentar previamente. Además, el paquete consta con un único algoritmo de estimación de distribuciones implementado. Por otro lado, el marco de programación científica es diferente a lo desarrollado en el grado y requiere de técnicas de depuración e implementación diferentes.

Esto se aplica al lenguaje de programación R que es un lenguaje de programación vectorial de software libre bajo licencia GNU muy extendido en la comunidad científica por su capacidad en cálculo vectorial entre otros. Además, es uno de los lenguajes más utilizados para el desarrollo de aplicaciones de estadística computacional y análisis de datos [2]. Sin embargo, el paradigma y la filosofía de R difiere de un lenguaje de programación orientado a objetos y por tanto, aprender este lenguaje de programación es una tarea añadida.

1.3 MOTIVACIÓN

La optimización es uno de los campos más ambiciosos de la computación. Actualmente abarca problemas de distintos ámbitos como el industrial, el bioinformático, el financiero y la inteligencia artificial, entre otros. Hoy en día no existen muchos softwares que permitan implementar, diseñar y utilizar algoritmos de optimización combinatoria con enfoque didáctico. Por este motivo, el principal objetivo del trabajo es extender el paquete *metaheuR* para poder ofrecer así una herramienta que acerque estos algoritmos a los alumnos. Además, con los algoritmos desarrollados, los usuarios serán capaces de resolver problemas de optimización comunes, como el problema del agente viajero (*Travelling Salesman Problem*), el problema de la asignación cuadrática (*Quadratic assignment problem*), el problema de la mochila (*Knapsack problem*), entre otros.

1.4 PROPÓSITO

El propósito de este proyecto consiste en ampliar el apartado de algoritmos metaheurísticos de el paquete de R *metaheuR* implementando diferentes algoritmos de estimación de distribuciones (EDAs) [Mühlenbein y Paass, 1996]. Estos algoritmos se diferencian en la etapa de aprendizaje y muestreo del modelo probabilístico que los caracteriza, por ello, este TFG se centrará en estas tareas.

2 Análisis de uso de lenguajes de programación https://blog.uchceu.es/informatica/ranking-de-los-lenguajes-de-programacion-mas-usados-para-2015/

Los algoritmos desarrollados son los siguientes: *Bayesian Network Estimation Algorithm* (EBNA) [Bengoetxea y col., 2002; Etxeberria, Pedro Larrañaga y Picaza, 1997], el cual implica el aprendizaje y muestreo de una red Bayesiana; *Edge Histogram-Based Sampling Algorithm* (EHBSA) [Tsutsui, 2002; Tsutsui y col., 2003] en el que, en el dominio de permutaciones, se aprenden probabilidades marginales de orden dos; y *Plackett-Luce Estimation of Distribution Algorithm* [Josu. Ceberio y col., 2013], un algoritmo desarrollado específicamente para optimizar problemas de permutaciones.

Para este proyecto además se incluye el paquete externo *bnLearn* [Scutari, 2009] que implementa lo necesario para el aprendizaje y muestreo de redes Bayesianas.

Para ilustrar el correcto funcionamiento, se han realizado dos tipos de pruebas, en primer lugar, se ha verificado el funcionamiento de los métodos de aprendizaje, muestreo y cálculo de probabilidad desarrollados y, en segundo lugar, se han realizado ejecuciones de los algoritmos sobre los problemas clásicos de optimización como *Maximum Independence Set* (MIS) y *Travelling Salesman Problem* (TSP).

Por último, cabe decir que *metaheuR* es un proyecto de software libre que se puede obtener de *GitHub* en la siguiente dirección: `https://github.com/b0rxa/metaheuR`.

1.5 RAZONES DE LA ELECCIÓN DEL TFG

La elección de este proyecto ha sido motivada por una serie de razones que se describen a continuación.

- Estrecha relación con los proyectos personales. Dado que me gustaría continuar mi formación académica en el ámbito de la inteligencia artificial, creo que es conveniente introducirme en esta materia.

- Las herramientas y los lenguajes utilizados para el desarrollo de este proyecto son libres y multiplataforma. Una de las premisas más importantes a la hora de desarrollar algo es que sea lo más accesible posible para cualquier tipo de usuario. Por eso, utilizar herramientas que posibiliten esta capacidad ha sido decisivo.

- Trabajar en un proyecto real. Crear un paquete para la docencia es un proyecto que tiene una finalidad clara, ayudar a formar alumnos. Además, va a tener aplicación y va a ser utilizada tanto por alumnos como profesores por lo que el trabajo realizado va a ser útil.

- Servirá de referencia para otros TFG. Este proyecto tiene una estrecha relación con la rama de ciencias de la computación y hasta el momento, han sido pocos los alumnos que han desarrollado su TFG fuera del área de conocimiento de la Ingeniería del Software.
- La afinidad con los directores de proyecto ha sido clave para elegir el trabajo de fin de grado.

1.6 EL LENGUAJE DE PROGRAMACIÓN R

Dado que el software va a ser desarrollado en R y el carácter del mismo es eminentemente científico, su diseño se aleja del esquema habitual estudiado en ingeniería del software, por este motivo, en esta sección se tratan conceptos relacionados con la programación y la filosofía de R para el correcto entendimiento de los diferentes conceptos de la memoria.

R nació en 1993 por Ross Ihaka y Robert Gentleman en la universidad de *Auckland*, Nueva Zelanda; como una combinación del lenguaje de programación S creado por John Chambers en 1976. Actualmente es desarrollado y mantenido por *R Development Core Team* del cual Chambers es miembro.

R es un lenguaje de programación funcional e interpretado. Las tareas que se pretenden hacer se realizan mediante funciones. El estilo de programación deriva del concepto teórico programación funcional o *functional programming* que restringe a los métodos a obtener resultados consistentes o al menos, resultados que puedan ser ampliamente verificados. Sencillamente, este tipo de programación ayuda a que los resultados obtenidos sean los esperados antes de implementar la función.

Un complemento natural del lenguaje funcional son las clases y los métodos. Las clases manifiestan la naturaleza de los objetos transmitidos a través de las funciones. Cuando se crea una nueva clase, se busca la razón de la misma que se describe mediante los *slot* –comparable a los atributos en lenguaje orientado a objetos–. La relación entre clases, se representa a través de funciones genéricas propias del lenguaje R, un ejemplo de esta función es *plot* (ver Figura 1), con ella se pueden representar en forma de gráfico cualquier tipo de dato siempre y cuando este se ajuste a las especificaciones del método. Además, la clase debe implementar los métodos que instanciarán los objetos pertenecientes a esa clase y el método que validará los *slot*, asegurando la consistencia de los valores en todo momento.

Los métodos en R deben encapsular claramente una funcionalidad referente a la tarea que se quiere realizar. Los métodos bien definidos extienden de las funcionalidades genéricas del lenguaje, esto hace que los métodos enriquezcan

```r
plot(c(1,2,3,4,5))
```

a: R

```java
Dibujo d = new Dibujo();
d.plot();
```

b: Java

Figura 1: Ejemplo de funciones genéricas.

todas las funcionalidades, no solo las de la clase subyacente. Esto no ocurre en lenguajes de programación orientada objetos como Java o C++. En R, los métodos son parte de una amplia a la vez que compleja estructura funcional basada en objetos.

Como se muestra en la figura 1, en R el código para llamar a la función genérica *plot* es accesible para cualquier objeto, en este caso, un vector. Sin embargo, desde Java, imaginando que la función *plot* está definida en la clase Dibujo y siendo esta pública, es necesario instanciar un objeto de esta clase para después llamar a este método. Además, el método plot de Java únicamente está accesible desde la clase Dibujo mientras que en R cualquier objeto puede llamar a la función *plot*. En R las funciones tienen un alcance global mientras que en los lenguajes de programación orientados a objetos, las funciones tienen un ámbito de clase. No obstante, no todos los objetos de R pueden hacer uso de la función *plot* ya que esta tiene validaciones y solo funciona con ciertos tipos de datos. Por otro lado, se entiende como buena práctica sobrecargar este método en las clases definidas por el usuario en caso de tener un significado análogo; de esta forma se mantiene la misma estructura funcional de R.

R está diseñado para trabajar con datos que están relacionados entre sí independientemente del tipo. Por ello, R propone un tipo de dato muy característico que son los *data frame*. Este tipo de dato tiene una estrecha similitud con las tablas del lenguaje SQL por ejemplo. En lenguajes de programación como Java o C++, no es posible tener en una misma estructura de datos elementos de diferente tipo; por ejemplo en listas, pilas o colas. En el caso de las tablas SQL diferentes tipos de datos pueden coexistir. Esto es lo que ofrece R, una estructura donde poder almacenar estos datos de diferente tipo pero relacionados entre sí.

Una de las características más importantes que ofrece R es la vectorización de operaciones en el procesador, es decir, la posibilidad de aplicar operaciones simultáneas a vectores enteros y no limitarse a una ejecución secuencial de los datos a elementos simples. Esto hace que los cálculos con vectores o matrices sean mucho más eficientes que en los lenguajes de programación convencionales.

Por otro lado, al ser un software libre está ampliamente respaldado por la comunidad y permite crear paquetes o librerías adicionales como esta y distribuirla a terceros [Chambers, 2008].

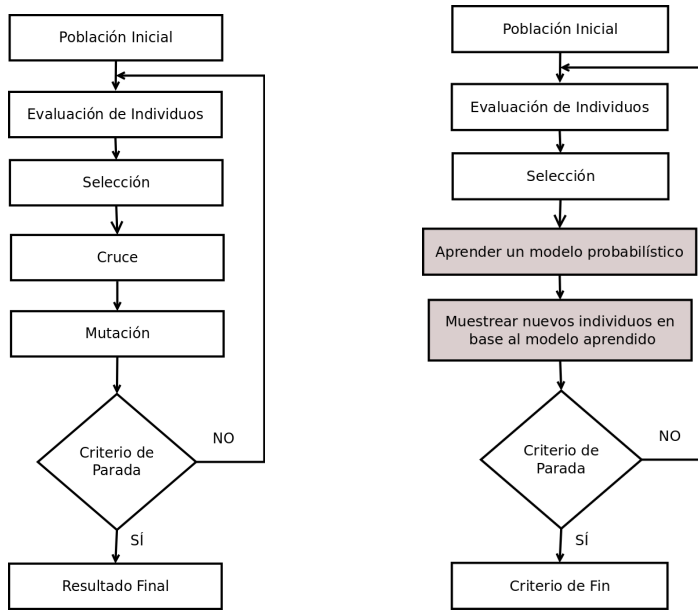
2 ALGORITMOS DE ESTIMACIÓN DE DISTRIBUCIONES

En este capítulo se presenta el fundamento teórico de los algoritmos de estimación de distribuciones que se van a incorporar al paquete *metaehuR* como fruto del trabajo realizado en este TFG.

Este TFG se centra en un conjunto de algoritmos metaheurísticos llamados Algoritmos de Estimación de Distribuciones cuyo acrónimo en inglés es *Estimation of Distribution Algorithms* (EDA) [Pedro Larrañaga y Lozano, 2002; Lozano, 2006; Pelikan, Goldberg y col., 2000; Pelikan, Sastry y col., 2007] que pertenecen a la familia de algoritmos evolutivos (EAs). La principal característica de los EAs es que utilizan técnicas inspiradas en la evolución natural de las especies. En la naturaleza, las especies cambian a lo largo del tiempo adaptándose a nuevas condiciones y entornos. Esta evolución hace que sobreviva el individuo que mejor se haya adaptado. Esta misma idea se traslada al campo de la optimización donde un individuo representa una solución para un problema; una población es un conjunto de soluciones (individuos) y, operaciones como cruce, mutación y selección son utilizadas para que los individuos (soluciones) evolucionen (mejoren). El paradigma más conocido de los EAs son los Algoritmos Genéticos (GAs) [Goldberg, 2000].

A continuación, en la Figura 2 se introduce un esquema comparativo de los GAs y los EDAs. Como mejora de los GAs, se introdujeron los EDAs en el campo de los algoritmos evolutivos. A cada generación, los EDAs aprenden un modelo probabilístico a partir de la población con el fin de representar la relación entre las características (*features*) más prometedoras de las soluciones. Muestreando el modelo probabilístico aprendido en la anterior generación, se generan nuevas soluciones. El algoritmo se detiene cuando cierto criterio de parada se cumple y devuelve la mejor solución encontrada hasta el momento.

Como se puede ver en la Figura 2 los pasos de un algoritmo genético son: evaluar los individuos para tener una medida de lo buenos que son *(fitness)*, ordenar los resultados de mayor a menor calidad *(fitness)*, seleccionar los mejores individuos. Después se procede al cruce entre individuos, habiendo varias técnicas de cruce en este aspecto. A continuación, se introduce, con cierta probabilidad, una mutación a cada nueva solución obtenida. Repitiendo estos pasos, se obtiene una nueva población.



(a) Esquema genérico de un algoritmo genético

(b) Esquema genérico de un EDA

Figura 2: Esquemas genéricos de las etapas de un GA y un EDA.

En un EDA la nueva población no es generada mediante el cruce y la mutación de individuos sino que son muestreados de la distribución de probabilidad que se aprende de la población anterior. De este modo, se consigue que la relación entre los individuos quede modelada mediante la probabilidad asociada a las soluciones de cada generación.

En realidad, la estimación del modelo probabilístico asociado a cada generación es la tarea más complicada y decisiva de un EDA y por tanto, es lo que determinará el funcionamiento y la calidad de los resultados obtenidos.

A continuación, se presenta el pseudocódigo genérico de un EDA. Pseudocódigo 1 [Bengoetxea y col., 2002].

Algorithm 1: Pseudocode for EDA approach.

```
1 D_0 ← Generate N individuals (the initial population) randomly.
2 Repeat for l = 1, 2, ... ,until a stopping criterion is met do
3     DS_{e l-1} ← Select S_e ⩽ N individuals from D_{l-1} according to a selection method.
4     p_l(x) = p(x|D^{S_e}_{l-1} ←) Estimate the probability distribution of an individual being
      among the selected individuals.
5     D_l ← Sample N individuals (the new population) from p_l(x)
6 end
```

En los siguientes apartados se introduce el fundamento teórico de los tres EDAs que se implementan en este proyecto. Además, se incluye una descripción detallada de las etapas de aprendizaje y muestreo de cada uno de ellos.

1. Estimation of Bayesian Network Algorithm (EBNA) [Etxeberria y Pedro Larrañaga, 1999]. Este modelo está sujeto al dominio de permutaciones y aprende una red Bayesiana como modelo probabilístico. Para muestrear este modelo, se sigue el método llamado muestreo lógico probabilístico (PLS). Por último, hay que destacar que el EBNA está sujeto al dominio binario con variables categóricas.

2. Edge Histogram Based Sampling Algorithm (EHBSA) [Tsutsui, 2002]. Este modelo se caracteriza por aprender una matriz de probabilidades marginales de orden 2. En la etapa de muestreo, se obtienen las nuevas soluciones mediante el método *roullete-wheel*.

3. Plackett-Luce Estimation of Ditribution Algorithm (PLEDA) [Josu. Ceberio y col., 2013]. Este algoritmo se caracteriza por implementar un modelo definido sobre el espacio de permutaciones. El aprendizaje consiste en extraer un vector de proporciones de la población dada mediante el algoritmo *Minorization-Maximization* (MM). En la etapa de muestreo se obtienen nuevas soluciones mediante el método *roullete-wheel*.

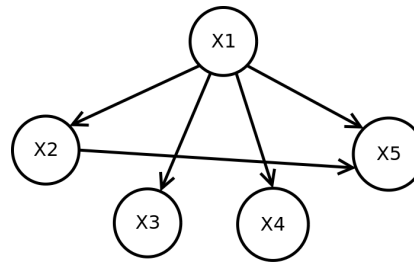


Figura 3: Ejemplo de red Bayesiana.

2.1 ESTIMATION OF BAYESIAN NETWORK ALGORITHM

El algoritmo de *Estimation of Bayesian Network Algorithm* (EBNA) es un EDA que a cada iteración, a partir de la población de individuos aprende una red Bayesiana. Después, se generan las nuevas soluciones muestreando la red. Una red Bayesiana es un modelo gráfico probabilístico en el cual los nodos representan las variables y los arcos las dependencias entre sí (ver Figura 3).

2.1.1 Aprendizaje

Dado un conjunto de individuos, aprender la red Bayesiana que mejor se ajusta a los datos es un problema de optimización *NP-hard*, es decir, la tarea no puede ser resuelta por un algoritmo en tiempo polinomial [Leeuwen, 1990 y Chickering y col., 1994]. Pese a que la comunidad científica ha desarrollado algoritmos que tratan de obtener la estructura que mejor se ajusta a los datos [Etxeberria, Pedro Larrañaga y Picaza, 1997 y P Larrañaga y col., 1996], estos requieren un coste computacional elevado. En consecuencia, en su lugar, se ha decidido utilizar un algoritmo de búsqueda local; en concreto, el *Hill Climbing*. Cabe decir que el éxito de la búsqueda local esta sujeto a la estructura inicial de la que partirá este algoritmo, sin embargo, se ha demostrado [Pedro Larrañaga, Poza y col., 1996] que funcionan bastante bien cuando la estructura inicial es adecuada.

El aprendizaje del modelo requiere aprender la estructura y los parámetros máximo verosímiles de la red Bayesiana. Las medidas más utilizadas que determinan la calidad de las estructuras son el *Akaike's Information Criterion* (AIC) [Akaike, 1974] y el *Bayesian Information Criterion* (BIC)[1] [Chickering y col., 1994] que implementa el algoritmo *Hill Climbing Algorithm*. En este proyecto,

1 También conocido como Jeffreys-Schwarz criterion.

se hará uso de la medida BIC. El *Hill Climbing Algorithm* trata de buscar una solución mejor a la actual a cada iteración. En caso de no encontrarla, propone la actual como mejor aproximación. Además, las estructuras son penalizadas en relación a su complejidad, de esta forma, no solo se obtiene una estructura buena sino que además, se consigue la red más simple de entre todas las de igual calidad.

Este algoritmo, dado un conjunto de soluciones D de tamaño N, donde $D = \{x_1, \ldots, x_N\}$, la medida de calidad de cada posible estructura de la red es obtenida calculando el estimador de máxima verosimilitud $\hat{\theta}$ para los parámetros θ y el logaritmo de máxima verosimilitud asociado, $\log P(D|S, \hat{\theta})$. La principal idea del EBNA es buscar la estructura gráfica probabilística que maximice $\log P(D|S, \theta)$. Esto se consigue midiendo la calidad de cada estructura fijándose en el logaritmo de máxima verosimilitud asociado a cada una de ellas [Bengoetxea y col., 2002].

2.1.2 Muestreo

El siguiente paso consiste en muestrear nuevas soluciones. En el caso del EBNA, es necesario muestrear una red Bayesiana. El muestreo de una red Bayesiana puede realizarse siguiendo el *Probabilistic Logic Sampling* (PLS). Este método garantiza que los individuos de la red son muestreados en orden ancestral, es decir, los padres, uno o varios, serán instanciados antes que los hijos. Esto supone ordenar los individuos. Se llamarán $\boldsymbol{\pi} = (\pi(1), \ldots, \pi(|V_D|))$ a todas los posibles ordenaciones que cumplan que los padres preceden a los hijos. Cuando los valores del padre pa_i de X_i hayan sido asignados, se simularán los valores de X_i siguiendo la distribución $p(x_i|\boldsymbol{pa}_i)$.

2.2 EDGE HISTOGRAM BASED SAMPLING ALGORITHM

El *Edge Histogram Based Sampling Algorithm* (EHBSA) es un EDA que se caracteriza por aprender una matriz de probabilidades de orden dos (adyacencias) de la población [Tsutsui, 2002; Tsutsui y col., 2003]. La población está formada por individuos que pertenecen al dominio de permutaciones. Después, se muestrean nuevas soluciones teniendo en cuenta dicha matriz para obtener nuevas soluciones.

Cabe decir que este tipo de EDA se propuso inicialmente para problemas definidos sobre el dominio de permutaciones, sin embargo, es perfectamente aplicable a cualquier otro problema de optimización combinatoria.

2.2.1 Aprendizaje

Dada una población de N individuos descritos como $D^t = \{\pi_0^t, \pi_1^t, \ldots, \pi_{N-1}^t\}$ a cada iteración t, el EHBSA aprende una matriz $E = \left[e_{i,j}\right]_{n \times n}$ de probabilidades marginales de orden dos en la que $e_{i,j}$ representa el ratio de veces que el elemento i precede de forma adyacente al elemento j en la solución π_k, donde $e_{i,j}$ se define con la siguiente expresión.

$$e_{i,j}^t = \begin{cases} \sum_{k=1}^{N} (\delta_{i,j}(\pi_k^t) + \delta_{j,i}(\pi_k^t)) + \epsilon = 1 & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Donde N es el tamaño de la población y L corresponde al tamaño de cada individuo, $\delta_{i,j}(\pi_k^t)$ es la función indicadora bajo la siguiente expresión.

$$\delta_{i,j}(\pi_k^t) = \begin{cases} 1 \text{ if } \exists h[h \in \{0, 1, \ldots, L-1\} \wedge \pi_k^t((h+1) \bmod L) = j)] \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

Por último, ϵ corresponde al parámetro de control definido como

$$\epsilon = \frac{2N}{L-1} B_{ratio} \tag{3}$$

Por tanto, el modelo probabilístico aprendido en un EHBSA corresponde a la matriz de adyacencias de la población.

2.2.2 Muestreo

[Tsutsui, 2002] propuso dos formas alternativas de muestrear, sin plantilla y con plantilla. Esto hace que el algoritmo tome nombres diferentes. En caso de muestreo sin plantilla el algoritmo toma el nombre de *Edge Histogram Based Sampling Algorithm Without Template* (EHBSA/WO). Este obtiene los nuevos individuos siguiendo una técnica de la ruleta sesgada (*roulette-wheel*). En cuanto al método de muestreo con plantilla *Edge Histogram Based Sampling Algorithm With Template* (EHBSA/WT), obtiene las nuevas soluciones utilizando ciertas restricciones para el Algoritmo *roulette-wheel* como se explican en las siguientes secciones.

2.2.3 EHBSA Without Template

En el EHBSA/WO se genera un nuevo individuo $\pi[]$ como se muestra en el Algoritmo 2, siendo N el tamaño de la población:

Algorithm 2: Pseudocódigo general para muestrear el modelo EHBSA.

$p \leftarrow 0$.
$\pi[0] \leftarrow random(0, N-1)$; // Obtener el primer elemento de forma aleatoria.
while $p < N-1$ **do**
  $rw[j] \leftarrow e^{t}_{s[p],j}\,(j = 0, 1, \ldots, N-1)$
  $rw[](rw[c[i]] \leftarrow 0$ for $(i = 0, \ldots, p)$
  $\pi[p+1] \leftarrow rw[x] / \sum_{j=0}^{N-1} rw[j]$
  $p \leftarrow p + 1$
end
return s

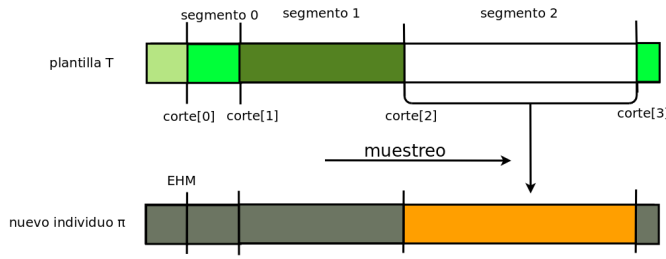


Figura 4: Ejemplo de plantilla del modelo EHBSA/WT.

Véase que en el EHBSA/WO solo es aplicable al dominio de permutaciones ya que una permutación se caracteriza por que un elemento únicamente aparece una vez en una secuencia. Por ello, el valor de un elemento solo tiene una referencia.

2.2.4 EHBSA With Template

En el EHBSA/WT se trata de estructurar el muestreo mediante segmentos para poder manipular la obtención de nuevas soluciones como se ilustra en la Figura 4. Para obtener un nuevo individuo, se escoge como plantilla una solución –normalmente de forma aleatoria–. Los n puntos de corte $(n > 1)$ son utilizados para dividir dicha plantilla dividiéndolo en n segmentos.

Para obtener un nuevo individuo, se obtienen los elementos acorde a los segmentos de la plantilla controlando así la etapa de muestreo.

2.3 PLACKETT-LUCE ESTIMATION OF DISTRIBUTION ALGORITHM

El modelo Plackett Luce es una generalización del propuesto por *Bradley-Terry* [Bradley y Terry, 1952]. El modelo *Bradley-Terry* se planteó para el dominio de permutaciones y se caracteriza por ser un modelo probabilístico gradual que descompone el proceso de obtener un nuevo individuo en n elementos en n etapas secuenciales. Este modelo esta sujeto a la expresión 4.

$$P(a \prec b) = \frac{w_a}{w_a + w_b} \tag{4}$$

donde w_a y w_b son pesos con valores positivos asociados a los elementos a y b. Nótese que cuando el valor w_a es más alto que w_b denota que el elemento a tiene mayor probabilidad que el elemento b.

El algoritmo *Plackett-Luce Estimation of Distribution Algorithm* (PLEDA) fue propuesto por [Josu. Ceberio y col., 2013] y fue diseñado para resolver problemas en el dominio de las permutaciones. Este algoritmo incorpora el modelo *Plackett-Luce*. Por tanto, los parámetros de cada elemento se especifican por un vector de parámetros $\mathbf{w} = (w_1, w_2, \ldots, w_n)$.

Formalmente, para cualquier elemento $i \in B$, siendo B el rango de todos los posibles conjuntos no nulos de $\{1, \ldots, n\}$, $P_B(i)$ es la probabilidad de que el elemento i sea elegido antes que los demás elementos posibles en B y se formaliza mediante la siguiente ecuación:

$$P_B(i) = \frac{w_i}{\sum_{j \in B} w_j} \tag{5}$$

Para explicar de forma sencilla el modelo *Plackett-Luce*, se pide imaginar una urna con infinitas bolas de colores. Considérese realizar un experimento que consiste en sacar bolas de la urna descrita anteriormente. El vector de parámetros $w = (w_1, w_2, \ldots, w_n)$ denota la proporción de bolas que hay de cada color. En la primera fase se extrae una bola de la urna, $\sigma(1)$. La probabilidad asociada a sacar esa bola es

$$\frac{w_{\sigma(1)}}{\sum_{j=1}^{n} w_{\sigma(j)}} \tag{6}$$

En la segunda fase, se obtiene otra bola de la urna $\sigma(2)$, sin embargo, se descartarán todas aquellas que sean del mismo color que la obtenida en la fase anterior. Por tanto, la probabilidad asociada en esta ocasión es

$$\frac{w_{\sigma(2)}}{\sum_{j=2}^{n} w_{\sigma(j)}} \tag{7}$$

posición	1	2	3	4	5
valor	0,3	0,1	0,02	0,5	0,08

Tabla 1: Vector de pesos estimado del modelo Plackett-Luce.

El proceso se repite hasta conseguir una permutación completa σ de todas las bolas de colores. La probabilidad de obtener la permutación σ es el producto de las probabilidades de las elecciones a cada paso:

$$P(\sigma) = \frac{w_1}{w_1 + w_2 + \ldots + w_n} \cdot \frac{w_2}{w_2 + \ldots + w_n} \cdot \ldots \cdot \frac{w_n}{w_n} \tag{8}$$

De forma más cerrada, la probabilidad de σ dado un vector de pesos se describe con la siguiente ecuación.

$$P(\sigma|\mathbf{w}) = \prod_{i=1}^{n-1} \frac{w_{\sigma(i)}}{\sum_{j=i}^{n} w_{\sigma(j)}} \tag{9}$$

2.3.1 Aprendizaje

El aprendizaje del modelo Plackett-Luce consiste en aprender los pesos w dado un conjunto de soluciones. En la literatura se han realizado varias técnicas como el *minorization Maximization* (MM) [Hunter, 2004] o el *Power-Expectation Propagation* (*Power-EP*) [Guiver y Snelson, 2009]. Pese a haber otros métodos para aprender el modelo probabilístico asociado al PLEDA, en este proyecto se ha optado por implementar el MM siguiendo las indicaciones de [Hunter, 2004] y [Josu. Ceberio y col., 2013]. Tras aplicar el algoritmo de aprendizaje, se obtiene como resultado un vector de pesos w que representa las proporciones de los valores posibles.

2.3.2 Muestreo

El muestreo es la etapa en la que se obtienen nuevas soluciones muestreando el modelo probabilístico aprendido como el de la Tabla 1. En este caso, se realiza mediante la técnica de *roulette-wheel*.

Teniendo un vector de pesos w como el de la Tabla 1, se obtiene un valor aleatorio α entre 0 y la suma de todos los valores del vector v. Se busca el menor índice j de la posición que cumpla que $\alpha >= \sum_{i=1}^{j} w(i)$. El índice se selecciona como elemento del nuevo individuo. Después se anula esa posición estableciendo su valor a 0. Se repite el proceso hasta obtener un individuo completo.