Búsqueda por Similaridad aplicada en la Recuperación de Factores que inciden en el Cálculo del Índice de Riesgo para la Salud de la Vivienda Urbana

Frittelli V, Steffolani FA, Teicher RG, Rojas M, Picco JE, Vázquez JC
Universidad Tecnológica Nacional – Facultad Córdoba, Argentina

Resumen.
El proyecto RNA-AC es el resultado de un convenio de cooperación entre la UTN-Córdoba y el Instituto de Investigaciones Geo-Históricas de la Provincia del Chaco (IIGHI-CONICET). Este último trabaja en un modelo conceptual mediante el cual busca determinar en qué forma influyen ciertos factores en el riesgo para la salud humana en contextos urbanos. No se conocen expresiones matemáticas que formalicen las relaciones entre los factores citados y el riesgo final, y los investigadores de la UTN-Córdoba abordaron el problema computacionalmente. Primero, se planteó un modelo de redes neuronales capaz de captar los datos de los factores incidentes, para luego afinar esta red para que pueda identificar las relaciones entre dichos factores y el índice final de riesgo. Segundo, para lograr una perspectiva más amplia y disponer de técnicas alternativas para identificar relaciones, se planteó otro modelo computacional mediante búsqueda por similaridad, como parte del subproyecto RNA-EH. Aquí exponemos el trabajo realizado en ese subproyecto, basado en los mismos datos y referencias usados en el proyecto marco RNA-AC, pero desarrollado "en paralelo" y sin cruce de resultados hasta su culminación. Se espera con esto, poder comprobar cada modelo computacional desarrollado y efectuar comparaciones de rendimiento, ventajas, desventajas y potencialidades.

Palabras Clave: Riesgo. Salud. Contexto Urbano. Factor de Riesgo. Índice Final de Riesgo. Modelo Computacional. Red Neuronal Artificial. Búsqueda por Similaridad.

Introducción.

El Instituto de Investigaciones Geo-Históricas (*IIGHI-CONICET*) de la Provincia del Chaco, Ciudad de Resistencia, ha propuesto el modelo de la Figura 1 como un esquema para el cálculo del índice de riesgo para la salud, en el contexto de la vivienda urbana. En este modelo se toma a la salud no como la ausencia de enfermedad, sino como un proceso en el que intervienen factores sociales, geográficos, políticos, culturales, etc.

En el modelo de la Figura 1 hay *Factores Medibles* (representados por las *burbujas de color gris claro* y que serán referidas como *FM* de aquí en adelante) que normalmente deberían estar disponibles a partir de datos obtenidos en censos en cada ciudad; y hay también *Conceptos Demográficos Abstractos* (representados por las *burbujas de color gris oscuro* y que serán referidas como *CDA* de aquí en adelante), que los investigadores del IIGHI-CONICET consideran que son determinados o influidos de alguna manera por los *FM*.

Cuando en el párrafo anterior se dice que los *FM* influyen "de alguna manera" en la determinación de los *CDA*, se quiere indicar que no se tiene aún una idea exacta de cómo influye cada *FM* en cada *CDA*. Esto es: no hay un modelo matemático (ecuaciones

diferenciales, algebraicas o de otro tipo) que lo justifique, aunque se presume que se trata de relaciones complejas no lineales. Por ejemplo, puede verse en el modelo que el *CDA* llamado *Exposición* se supone influenciado por cuatro *FM*, que son los siguientes: nivel de *Pobreza* de la familia en ese hogar, el hecho de haber *Mujeres Jefas* o a cargo del sostenimiento de ese hogar, la presencia de *Niños Menores* habitando en el hogar y la presencia de *Ancianos* habitando en el hogar. Y en forma similar pueden describirse los *CDA* designados como *Fragilidad*, *Resiliencia* y *Riesgo Físico* (este último también suele designarse como *Amenaza*, en forma indistinta): cada uno de estos tres se supone influenciado de alguna forma por los *FM* que se marcan en el modelo. Puede notarse además, que un *CDA* puede estar determinado por la influencia de otros *CDA* calculados en forma previa, como es el caso del *CDA* designado como *Vulnerabilidad del Contexto* (o simplemente, *Vulnerabilidad*), que se calcula a partir del impacto de los *CDA Exposición*, *Fragilidad* y *Resiliencia*. Con todo esto, se calcula un *Índice de Riesgo Total* (la salida del modelo), que esencialmente se calcula a partir de los *CDA Vulnerabilidad* y *Riesgo Físico* (o *Amenaza*).

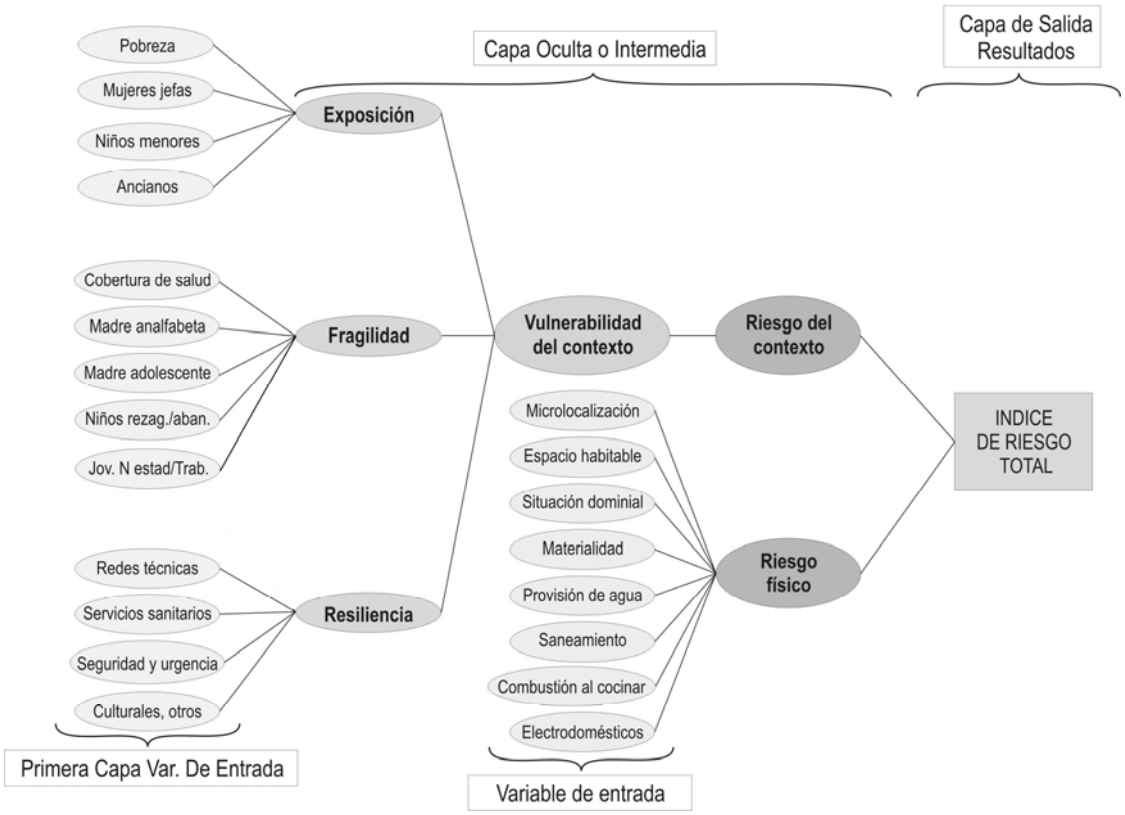

Figura 1: Modelo conceptual del IIGHI-CONICET para el cálculo del riesgo de la vivienda urbana para la salud

El modelo de la Figura 1 ha sido desarrollado por expertos demógrafos del IIGHI en base a sus conocimientos y experiencia de campo. Los procesos heurísticos aplicados por estos expertos les han permitido confeccionar tablas fiables en las que se determina el valor porcentual aproximado de cada *CDA* frente a valores dados de los distintos *FM*, tomados éstos últimos como variables independientes (o con dependencias ya incluidas en sus estimaciones porcentuales, como es el caso del *CDA Vulnerabilidad*).

En esas tablas aportadas por los investigadores en demografía, los *FM* se miden en *porcentajes de hogares* que tienen el carácter indicado (un número entre 0 y 100) y los *CDA* también se miden en porcentajes (números entre 0 y 100) o en valores lingüísticos (muy bajo, bajo, moderado, alto, muy alto) que se han traducido a números entre 0 y 100 usando elementos de lógica difusa o herramientas similares. Estos expertos han proporcionado numerosas planillas de cálculo de ejemplos de valores reales que ellos estiman adecuados para cada *CDA* según distintos valores de los *FM* involucrados. Un ejemplo (extraído de una de las planillas citadas) puede verse como sigue:

Exposición				
Pobreza	**Mujeres Jefas**	**Niños <**	**Ancianos**	**Ind. Expos.**
10 a 25	25 a 50	10 a 25	50 a 75	**0.20 a 0.50**
10 a 25	50 a 75	75 y +	25 a 50	**0.40 a 0.75**
10 a 25	75 y +	50 a 75	10 a 25	**0.40 a 0.75**
10 a 25	10 a 25	25 a 50	75 y +	**0.20 a 0.50**
10 a 25	0 a 10	0 a 10	0 a 10	**0.10 a 0.25**

Tabla 1: Ejemplo de Tabla de Cálculo de un CDA

La Tabla 1 muestra un pequeño conjunto de valores reales para el cálculo del índice del *CDA Exposición*. Los valores de los *FM Pobreza*, *Mujeres Jefas*, *Niños Menores* y *Ancianos* están dados como un rango de valores de porcentajes, obtenidos de censos, y en base a esos rangos los expertos en demografía asignan heurísticamente un valor final para el índice del *CDA Exposición* (en la última columna de la tabla). En este caso, si se observa la primera fila de la tabla, se ve que para el *FM Pobreza* se ha medido un valor estimado entre el 10% y el 25% para el conjunto de hogares censado. Para el *FM Mujeres Jefas* se midió un valor que está en el rango de entre el 25% y el 50% de los hogares censados. Y para esos mismos hogares se midió con rangos de porcentajes que están entre el 10% y el 25% el *FM Niños Menores*, y entre el 50% y el 75% para el *FM Ancianos*. Para el conjunto de esos rangos de porcentajes, finalmente los investigadores expertos del IIGHI asignaron un valor de probabilidad en el rango entre 0.20 y 0.50 al índice de medición del *CDA Exposición*. Y lo mismo vale para el resto de las filas de la Tabla 1.

Con estas tablas (y muchas otras similares para el resto de los *CDA*) los investigadores de la UTN Córdoba han entrenado *redes neuronales artificiales* que utilizan luego los expertos demógrafos como herramienta de cálculo (todo esto en el marco del Proyecto RNA-AC). Aún así, se ha pensado que es posible desarrollar una herramienta eventualmente más versátil que una red neuronal para realizar esta tarea. Es entonces que, como parte del

subproyecto RNA-EH, se planteó el objetivo de programar un *motor de búsqueda por similaridad* que sirva como alternativa a la red neuronal, y en este documento se presenta la forma en que esto se hizo.

Elementos del Trabajo y Metodología.

A partir de las tablas de valores presentadas por los demógrafos, en las que ya están calculados los índices o rangos de probabibilidad para cada *CDA* y el *Riesgo Total* para el conjunto de los *CDA*, el problema a abordar es el siguiente: Recibir como entrada un conjunto de valores *a* para diversos *FM* (valores que posiblemente no figuren estrictamente en ninguna de las tablas precalculadas), tal que para *a* no se conoce un índice o rango de probabilidad, y retornar el índice o rango de probabilidad *r* más adecuado para el *CDA* correspondiente, o para el *Riesgo Total Final*. Cuando aquí se dice *retornar el índice más adecuado*, se quiere expresar que la aplicación computacional debe buscar entre todas las tablas precalculadas un conjunto de valores ya existente *b* de rangos de *FM* o de *CDA* que sea el más similar o próximo (o posiblemente igual) al conjunto *a* con los valores de entrada, y retornar como índice sugerido para *a* el que corresponda al conjunto *b* encontrado.

En un contexto general, este problema se conoce como *búsqueda por similaridad* (o también, *búsqueda por proximidad* [1] [7] [14]), y la idea esencial es poder medir la *distancia* entre dos elementos u objetos *a* y *b* de un conjunto *C*. Intuitivamente, se supone que mientras más próximos (o más cercanos) estén *a* y *b*, más similares son. Lo esperable es contar con una función que tome los dos objetos que se desea comparar, y retorne un valor numérico (un número real, no necesariamente entero) que de algún modo mida la *distancia* (o *similaridad*) entre esos dos elementos. Si tal función cumple con determinadas condiciones formales, entonces la propia función se denomina *función de distancia métrica* y el conjunto para el cual esa función aplica se dice *espacio métrico*. Las propiedades que debe tener una función *d(a, b)*, tal que $d : C \times C \rightarrow R$ (siendo *R* el conjunto de los números reales) para ser considerada métrica en un conjunto *C* y convertir a *C* en un espacio métrico, son las siguientes:

i. **No negatividad**: $d(a, b) >= 0 \quad \forall a, b \in C$
ii. **Simetría**: $d(a, b) = d(b, a) \quad \forall a, b \in C$
iii. **Reflexividad**: $d(a, a) = 0 \quad \forall a \in C$
iv. **Positividad Estricta**: $d(a, b) > 0 \quad \forall a, b \in C, \ a \neq b$
v. **Desigualdad Triangular**: $d(a, b) <= d(a, c) + d(c, b) \quad \forall a, b, c \in C$

La primera propiedad establece que la distancia entre dos objetos del conjunto no debe ser negativa, tal como se esperaría en el mundo físico. La segunda implica la conmutatividad en la medición: la distancia debería ser el mismo valor, sin importar desde cual de los

objetos se comience a medir. La tercera propiedad es clara y obvia: estipula que la distancia de un objeto a si mismo debe ser cero. La cuarta propiedad supone que si la distancia medida es mayor a cero, entonces los objetos comparados son distintos (lo cual equivale a decir que si la distancia es cero, entonces los objetos son iguales, pudiéndose expresar de esta otra forma: $d(a, b) = 0 \Rightarrow a = b \quad \forall a, b \in C$) Nótese sin embargo, que en muchos casos esta propiedad podría no cumplirse o no ser exigible: podría darse para algunos contextos que la función mida una distancia cero entre dos objetos que no sean iguales. Si no se exige el cumplimiento de esta propiedad para la función distancia en el conjunto *C*, pero se cumplen todas las otras, entonces la función se denomina *distancia pseudométrica*, y el conjunto *C* en el cual se aplica se designa como *espacio pseudométrico*. Como se verá, éste será el caso en el planteo: los conjuntos de tuplas entre los cuales se deberá buscar una en particular, serán pseudométricos pues la función distancia que será definida no garantizará la cuarta propiedad. Sin embargo, esto no constituye un problema, pues los algoritmos que se usarán para la búsqueda se adaptarán sin inconvenientes a esta situación: simplemente, en el desarrollo se darán como *muy similares* (y no necesariamente como iguales) a dos objetos que tengan distancia cero entre ellos.

La quinta propiedad es la clave de todos los sistemas de búsqueda por similaridad que se han planteado para encontrar objetos similares a *x* en un espacio *C*. Se llama también *desigualdad del triángulo* o *desigualdad triangular*, y es la que facilita que dentro de un conjunto *C* de *n* elementos, puedan plantearse estructuras de datos y algoritmos de búsqueda que permitan ahorrar comparaciones y no caer en una simple búsqueda secuencial que llevaría indefectiblemente a un peor caso con tiempo de ejecución *O(n)* (orden lineal). Geométricamente, la desigualdad triangular especifica que medir la distancia directamente entre dos objetos *a* y *b*, debería dar un número menor o a lo sumo igual, que medir la distancia entre ambos pero pasando por un tercer punto *c* en el medio. Gráficamente:

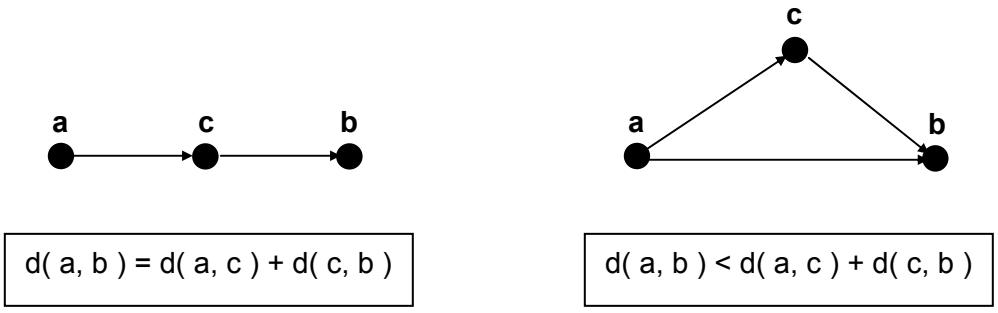

Figura 2: La Desigualdad Triangular

Puede verse que la si la función distancia cumple con la propiedad de la desigualdad triangular, entonces la forma más corta de pasar de un punto a otro será midiendo directamente la distancia entre ellos. Nunca se obtendrá una distancia menor si se pasa por un tercer punto entre ellos: a lo sumo la distancia será la misma, si los tres objetos son colineales en ese espacio.

Ahora bien: un espacio métrico es cualquier conjunto *C* dotado de una función *d*, tal que *d* es una forma de distancia métrica. En la práctica, *C* podría ser un conjunto de objetos que no necesariamente representen números o elementos matemáticos. El espacio métrico podría estar compuesto por cadenas de caracteres, o por imágenes, o por archivos de audio, o por cualquier tipo de elementos entre los cuales un usuario pudiera estar interesado en buscar objetos similares. Por caso, el conjunto de tuplas con valores demográficos podría ser tratado como un espacio métrico, si se dotase al mismo de una función de distancia métrica.

El hecho es que no siempre es fácil plantear la forma análitica o algorítmica de una función de distancia (métrica o no) entre los elementos de un conjunto cualquiera. Por ejemplo, si el conjunto contiene cadenas de caracteres, existe un conocido algoritmo que define una forma de distancia métrica entre dos cadenas, llamado *distancia Levenshtein*[8] en honor a Vladimir Levenshtein que propuso el mecanismo original. Este algoritmo se basa en contar la cantidad mínima de eliminaciones, agregados o sustituciones que deben hacerse en la primera cadena para convertirla en la otra, y el número así obtenido se retorna como la distancia entre esas cadenas. Sin embargo, plantear un algoritmo fiable que mida distancias entre otras clases de objetos no es tan claro ni tan directo: ¿cómo medir la distancia entre dos imágenes o dos melodías? ¿Cómo medir la similaridad entre dos huellas digitales o dos cadenas de ADN? O más cercano a nuestro problema: ¿cómo medir qué tan similares son dos tuplas de valores demográficos, y además garantizando que la función aplicada sea métrica o pseudométrica?

Para facilitar una respuesta, en la práctica los investigadores y desarrolladores intentan representar a cada objeto del conjunto o espacio, en forma *vectorial*[7]: esto es, se busca que los objetos considerados puedan manejarse como vectores formados por *k* valores de coordenadas reales. Por ejemplo, si se trata de un conjunto de imágenes, se intenta captar en forma numérica una cierta cantidad *k* de elementos de cada imagen, y luego representar a la imagen como un *vector* con *k* valores de coordenadas numéricas. Y lo mismo valdría para otros tipos de objetos en otros espacios. Si esta representación vectorial fuera válida, entonces se contaría con mayor libertad para plantear formas de medir distancias, puesto que sería posible usar información geométrica y de coordenadas que no estarían disponibles en un espacio métrico general.

En el caso en estudio, está claro que un conjunto *a* de entrada puede representarse como una tupla de *k* valores v_i (medidos si se trata de valores de *FM* o calculados si se trata de valores de *CDA*), tal que para esa tupla se desconoce el índice o rango de probabilidades *x* que le correspondería:

$$a = \{ v_1, v_2, \ldots, v_k \} \Rightarrow x$$

Llamemos *T* al conjunto de las *n* tuplas precalculadas (dadas en las planillas de cálculo provistas por los demógrafos). Entonces, cada conjunto b_j en *T* es también una tupla de *k* valores u_i pero de tal forma que para cada una de estas tuplas b_j existe un índice o rango de valores de probabilidad r_j ya conocido y asignado por los investigadores:

$$b_j = \{ u_1, u_2, ..., u_k \} \Rightarrow r_j$$

Es claro entonces que cada tupla en *T* puede representarse como un vector *k-dimensional*, perteneciente así a un espacio vectorial EV_i que será subconjunto de *T*. Y lo mismo vale para las tuplas o conjuntos de entrada *a*:

$$a = \vec{a} = \left(v_1, v_2, \cdots, v_k \right) \Rightarrow x$$

$$b_j = \vec{b}_j = \left(u_1, u_2, \cdots, u_k \right) \Rightarrow r_j$$

Es fácil ver que cada espacio vectorial EV_i así caracterizado en *T*, es también un *espacio euclídeo* pues puede aplicarse a cada vector *b* el cálculo de la *norma vectorial euclídea* (que coincide con el módulo o longitud de cada vector en ese espacio euclídeo):

$$|| b || = \sqrt{u_1^2 + u_2^2 + \cdots + u_k^2}$$

Y de esta forma, cada espacio vectorial EV_i en *T* es entonces un espacio vectorial normado.

En la práctica, ya se indicó que cada tupla b_j en *T* viene dada de forma que cada elemento es un rango de porcentajes, lo cual es un problema si se pretende que cada tupla se represente como un vector, pues se esperaría que cada elemento (o coordenada del vector) sea un valor real único. Por otra parte, la presencia de rangos en lugar de valores simples complica todos los procesos de identificación y comparación de tuplas incluso en el modelo de redes neuronales planteado en el proyecto marco *RNA-AC*. Por ese motivo, tanto en el modelo de redes neuronales como en este de búsqueda por similaridad, se decidió seguir el mismo enfoque práctico: cuando sea necesario comparar dos tuplas, se tomará como valor representativo de cada intervalo de valores al valor medio del intervalo dado. Así, si se retorna sobre la Tabla 1, entonces la primera fila de la tabla, con los siguientes valores:

Pobreza	Mujeres Jefas	Niños <	Ancianos	Ind. Expos.
10 a 25	25 a 50	10 a 25	50 a 75	**0.20 a 0.50**

Tabla 2: Extracto de Valores de Riesgo

sería representada con este vector, considerando sólo números enteros al calcular los valores medios, a los efectos de aplicar cálculos de norma, o de similiaridad (o proximidad):

$$b = \vec{b} = \left(17, 37, 17, 62 \right) \Rightarrow [0.20 .. 0.50]$$

Ahora bien: si en el modelo matemático cada tupla puede representarse como un vector en un espacio vectorial normado, el problema se reduce a: Dado el vector *k-dimensional* de entrada a para el cual se desconoce el índice o rango de probabilidades x, encontrar en el espacio vectorial EV_i subconjunto de T que corresponda, el vector b_j más similar (o igual) a a, y retornar y proponer para el vector a el índice o rango de probabilidades r de b_j.

Lo anterior supone entonces tener la capacidad de comparar dos tuplas representadas como vectores y decidir qué tan similares son entre ellas. Esto implica que se necesita una función para medir las distancias entre esas tuplas. Pero en el caso en estudio, cada espacio vectorial en T es un espacio vectorial normado, y todo espacio vectorial normado es a su vez un espacio métrico, pues la norma o módulo induce la función distancia: en todo espacio vectorial normado V, se puede definir la distancia d entre dos vectores a y b como la norma, módulo o longitud del vector diferencia entre a y b:

$$d(a, b) = || a – b ||$$

Es fácil comprobar que la función distancia así propuesta es (en principio) métrica, y por lo tanto, cada subconjunto de tuplas de k componentes en T, sería también un espacio vectorial métrico. Sin embargo, se recuerda que cada tupla b en algún subconjunto de T viene dada por una sucesión de intervalos de porcentajes, y se ha tomado el valor medio de cada intervalo para poder representar a cada tupla como un vector y medir distancias. Luego, dos tuplas a y b que presenten intervalos de porcentajes diferentes pero con los mismos valores medios, serían representadas con el mismo vector, y eso haría que la función distancia mida una distancia cero entre ellos, aunque las tuplas originales no sean estrictamente iguales. Por lo tanto, en las condiciones dadas, la función distancia inducida por la norma se comporta como pseudométrica, y los espacios vectoriales subconjunto de T serán espacios vectoriales pseudométricos.

El hecho de trabajar con una función de distancia pseudométrica (y por lo tanto con un espacio vectorial pseudométrico) no conlleva problema alguno desde el punto de vista de las estructuras de datos y algoritmos en que se basa el modelo de búsqueda: simplemente, encontrar un objeto a distancia cero de aquel que se está buscando significará que se ha dado con un objeto muy similar (que posiblemente será el más similar o incluso igual) al que se está buscando. Y a los efectos del objetivo de este estudio, ese resultado es perfectamente válido.

Contar con una función de distancia métrica (o pseudométrica) para el conjunto *T* de objetos es el primer paso para la solución al problema de la búsqueda de objetos similares o próximos. El segundo paso importante es organizar los objetos en el conjunto *T* para favorecer búsquedas tan rápidas como sea posible, sobre todo considerando que la cantidad de elementos *n* en *T* es o puede ser muy grande.

Es obvio que siempre se puede hacer un recorrido secuencial simple: dado el objeto *a* que se quiere buscar, se toma uno por uno a cada uno de los objetos en el subconjunto de *T* que corresponda y se miden sus distancias con *a*, hasta encontrar el más similar (o los *k* más similares). Pero como se dijo, esto llevaría a un proceso que en el peor caso implicaría un recorrido completo del subconjunto analizado, con un tiempo de ejecución *O(n)*.

El objetivo de todas las estructuras de datos que han sido propuestas[2] [3] [4] [10] por los investigadores para la búsqueda por similaridad es organizar los datos del espacio métrico o pseudométrico analizado, pero de forma de permitir búsquedas que en el caso promedio sean al menos de *orden sublineal* (o sea: $O(n^{\alpha})$ con $0 < \alpha < 1$) Esto es, una búsqueda en promedio no debería tener que comparar contra todos los objetos del espacio, sino que debería poder ahorrarse un número significativo de comparaciones.

Por otra parte, las necesidades de búsqueda por similaridad (o por proximidad) podrían ser diferentes de acuerdo a cada contexto. En algunos casos, los usuarios requerirán poder encontrar simplemente el objeto más similar al objeto *x* dado (y en este caso se habla de *búsqueda del vecino más próximo*). En otros casos, se querrá encontrar el conjunto de *k* objetos más similares a *x* (y se tiene entonces lo que se conoce como *búsqueda de los k vecinos más próximos*). Y finalmente, también podría darse la situación en que se desee encontrar el conjunto de objetos más cercanos a *x*, dentro de un radio *r* de distancias posibles (que se conoce como *búsqueda por rango*). Formalmente, cada caso se plantea de la siguiente manera:

- **Búsqueda por rango (o *Range Query*)**: recuperar todos los objetos de *T* que estén a distancia *r* (o menos) de *x*.
  $(x, r)_d \Rightarrow \{ u \in T \ / \ d(x, u) <= r \}$

- **Búsqueda del vecino más próximo (o *Nearest Neighbor Query*)**: recuperar el objeto en *T* que sea el más similar a *x*.
  $NN(x) \Rightarrow \{ u \in T \ / \ \forall y \in T, \ d(x, u) <= d(x, y) \}$

- **Búsqueda de los *k* vecinos más próximos (o *k-Nearest Neighbor Query*)**: Recuperar los *k* objetos más similares a *x* en *T*.
  $NN_k(x) \Rightarrow U \subseteq T \setminus |U| = k, \ \forall u \in U, \ y \in T - U, \ d(x, u) <= d(x, y)$

Se han propuesto numerosas estructuras de datos para búsquedas por similaridad con estos objetivos (permitir búsquedas de acuerdo a uno o más de los tres casos citados, con tiempo

de ejecución de orden al menos sublineal)[9] [10] [11] [12] [13]. Para el presente trabajo, se ha seleccionado la que quizás sea la más clásica de esas estructuras (y probablemente haya sido la primera en ser planteada), denominada *Arbol de Burkhard-Keller* (a la que simplemente se designará como *BKT* por sus iniciales en inglés: *Burkhard-Keller Tree*)[5].

En esencia, un *BKT* es un *árbol de caminos múltiples*, inicialmente dispuesto para búsquedas por rango cuando se tienen distancias discretas (o sea, las distancias entre elementos son números enteros). Sin embargo, puede adaptarse también para búsqueda del vecino más próximo o para los *k* vecinos más próximos mediante algoritmos generales de aproximación, que de hecho han sido implementados en el modelo en estudio.

La fase de construcción de un *BKT* puede resumirse como sigue: se toma un elemento cualquiera *p* del espacio métrico *T* de objetos disponible. En esta fase del proceso, se dice que *p* se comporta como un *pivot*, y se ubica a *p* como raiz del árbol. Luego se miden las distancias de *p* a cada uno de los objetos restantes en *T*, y para cada posible valor *i* de distancia obtenido se crea un subárbol de *p*, que contenga a todos los objetos de *T* cuya distancia a *p* sea igual a *i*. Esto es, cada subárbol de *p* será un conjunto S_i con la siguiente caracterización:

$$S_i = \{ s \in T \setminus d(s, p) = i \}$$

Luego, cada subárbol S_i se estructura recursivamente de la misma forma: en cada subárbol se toma un nuevo pivot (que será la raiz de ese subárbol), se calculan las distancias de todos los elementos de ese subárbol al nuevo pivot, y se asignan subárboles para la nueva raiz de acuerdo a las distancias así obtenidas. Este proceso continúa hasta que cada posible subárbol tenga un solo elemento, o bien hasta que en el posible subárbol quede un número pequeño *b* de elementos, que pueden ser directamente asignados como una lista simple en lugar de un subárbol, para ahorrar tiempo. Puede verse que el *BKT* así formado ocupará un espacio en memoria proporcional a la cantidad *n* de objetos que haya en *T*, por lo que el espacio ocupado es *O(n)*. Puede probarse que el tiempo promedio necesario para construir el árbol mediante este proceso es *O(n*log(n))*. Obviamente, este mecanismo de construcción se aplica por única vez. Una vez construido el *BKT* (que de aquí en adelante servirá como *índice* para realizar búsquedas rápidas) el mismo puede persistirse en disco y ser recuperado en forma completa cada vez que se lo requiera.

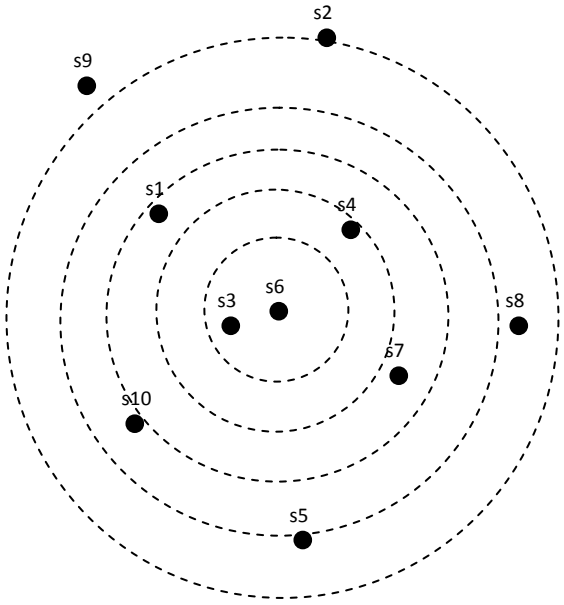

Figura 3: Un conjunto de objetos en un espacio métrico bidimensional tomando como pivot central al objeto s6

En la Figura 3 se puede ver una representación gráfica, a modo de aproximación conceptual, de un conjunto de objetos *{ s1, s2, ..., s10 }*. Se supone que el conjunto es métrico, y representable en un plano de dos dimensiones, para simplificar. Se puede ver como se subdivide el espacio métrico si se toma como punto central o pivot inicial para construir un *BKT* al objeto *s6*. Se puede suponer que cada círculo concéntrico alrededor de *s6* representa un rango de distancias. El primer círculo (o sea, el más interior y más cercano a *s6*) representa un rango de distancias de 1: esto quiere decir que cualquier objeto que esté dentro de ese círculo, o justo en su circunsferencia, estará a distancia 1 (o menos) de *s6*. Note que las distancias serían efectivamente valores entre 0 y 1 (para este primer círculo) si la función distancia retornase valores reales, pero se ha supuesto (y desarrollado) una función que retorna valores enteros para poder construir el árbol *BKT*. En la gráfica, el objeto *s3* se encuentra a una distancia de menos de 1 de *s6*, por lo que la función distancia retornará una medición de 0. Los valores decimales en la medición de distancia, simplemente son ignorados. Con el mismo criterio, se puede ver que *s4* estaría a distancia 1 de *s6* (pues está dentro del círculo de rango 2), y que los objetos *s1* y *s7* están a distancia 2 de *s6*. A los efectos de la construcción del *BKT*, no importa si *s1* está "un poco" más cerca o más lejos de *s6* que *s7*. Sólo importa que ambos están en un rango de distancia discreta de 2. Se puede ver que el objeto *s5* se encuentra justo en la circunsferencia del círculo de radio 4, por lo cual su distancia a *s6* es exactamente 4, mientras que *s8* está entre los círculos de radio 4 y radio 5. La función distancia tomaría para *s8* una distancia entera de 4, haciendo que se considere que *s5* y *s8* están a la "misma distancia" 4 de *s6*.

Con estas ideas, se puede mostrar (Figura 4) cómo quedaría un *BKT* cuya raiz contenga al pivot *s6*. En principio, cada pivot puede seleccionarse al azar, pero se han propuesto técnicas[6] para realizar esta selección de forma que aumente luego la eficiencia de búsqueda del árbol. Para simplificar, en la gráfica se muestran sólo la raiz y el nivel

siguiente del árbol. Recordar que si en un subárbol hay pocos elementos, estos pueden quedar dispuestos simplemente como una lista, en lugar de rearmar ese subárbol en forma recursiva tomando nuevamente un pivot.

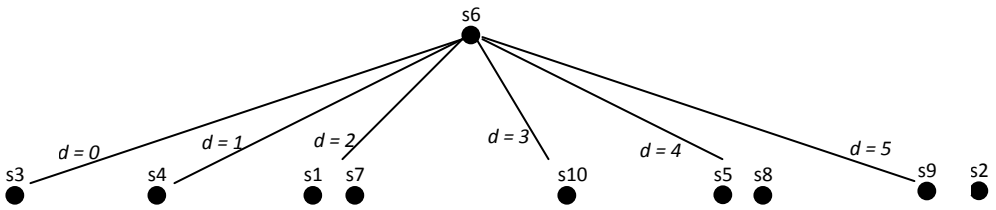

Figura 4: Un árbol BKT de dos niveles...

Una vez armado el *BKT*, por fin se podrán realizar consultas sobre él y buscar objetos similares a alguno dado *x*. Un *BKT* es esencialmente una estructura de datos para *consultas por rango*. El mecanismo o algoritmo de consulta por rango sería como se indica: Sea *x* el objeto para el cual se desea encontrar los más próximos, y sea *r* el radio de consulta. En una consulta por rango, interesa entonces recuperar todos los objetos del *BKT* que estén a distancia *r* o menos de *x*. Se comienza para ello midiendo la distancia de *x* al objeto *p* que está en la raiz del árbol (*p = s6* en este caso). Luego se entra en todos los subárboles S_i tales que:

$$d(p, x) - r <= i <= d(p, x) + r$$

y en cada uno de esos subárboles se procede recursivamente. Si se llega a un nodo del árbol en el que hay una lista de objetos, se la recorre secuencialmente. La idea es que cada vez que se tome la distancia de *x* a un elemento *s* del árbol (sea contra un pivot o contra un elemento de una lista) y se obtenga una distancia $d(x, s) <= r$ entonces se rescte al objeto *s* y lo incorpore al conjunto de salida que se debe retornar.

Es claro que procediendo de esta forma, se esperaría que en una búsqueda promedio sólo algunos de los subárboles del *BKT* sean recorridos (y estos a su vez podrían recorrerse sólo en forma parcial), por lo que la cantidad final de comparaciones a realizar será menor que *n* (el número de objetos en el árbol), con lo cual se tiene un tiempo de ejecución promedio de orden sublineal. La pregunta es: ¿qué es lo que garantiza que no se esté "olvidando" a ningún objeto próximo a *x* que pudiera estar en alguno de los subárboles no recorridos? Esa garantía es aportada por la *desigualdad del triángulo*. Se puede ver que todos los subárboles no recorridos, contienen elementos *s* que se encuentran a una distancia *i* respecto de alguno de los pivots o raices *p* del árbol, pero tal que esa distancia *i* deja al objeto *x* fuera del rango *r* de búsqueda. Formalmente, se trata de objetos *s* para los cuales se cumple:

$$d(p, s) = i \setminus |d(p, x) - i| > r \qquad ①$$

Pero la desigualdad del triángulo implica que medir la distancia de *x* a cualquier pivot *p* en forma directa, debería dar un valor menor o igual que medir la distancia de *x* a *p* pasando por un objeto intermedio *s*. Es decir, debería ser:

$$d(p, x) <= d(p, s) + d(s, x)$$

El término *d(s, x)* mide la distancia entre *x* (el objeto buscado) a un objeto cualquiera *s* que está en un subárbol no recorrido. Si se despeja ese término, queda:

$$d(s, x) >= d(p, x) - d(p, s)$$

y por la expresión ① se obtiene que:

$$d(s, x) >= d(p, x) - d(p, s) > r$$

Por lo tanto, los subárboles no explorados no pueden contener objetos próximos a *x*: todos esos objetos están a una distancia mayor a *r* respecto de *x*.

A modo de ejemplo, se muestra en la gráfica de la Figura 5 el mismo espacio de objetos de la Figura 3, pero ahora suponiendo en ella la inclusión de un objeto *x* que se desea buscar. Si el radio de búsqueda *r* es 2, entonces interesa recuperar todos los objetos que estén a una distancia de 2 o menos de *x*. El círculo negro centrado en *x* que se muestra en la Figura 5 es de radio 2 (tomando las mismas proporciones que se tomaron para formar la gráfica del espacio centrado en *s6*) y puede verse que los objetos que se encuentran dentro de un radio de distancia 2 alrededor de *x* son los objetos *s6*, *s3* y *s7*, que son que los que fectivamente serían recuperados desde el árbol de la Figura 4 aplicando el algoritmo antes descripto.

Hasta aquí se ha visto como puede usarse un *BKT* para *búsqueda por rango* (y se dijo que esa era la razón de ser esencial de un *BKT*). No obstante, un *BKT* (y muchas otras estructuras de búsqueda por proximidad diseñadas para búsqueda por rango) pueden adaptarse para permitir *búsquedas del vecino más próximo* o *búsquedas de los k vecinos más próximos*. Existen distintas variantes algorítmicas para hacer esto[7], y una de ellas (la que se ha aplicado en este trabajo), se designa como *incremento del radio*. La idea básica es la siguiente: se usa el algoritmo descripto para búsqueda por rango, pero se comienza con un radio de búsqueda muy pequeño, y se repite sistemáticamente el proceso aumentando progresivamente el valor del radio (siempre en valores muy pequeños de variación), hasta que se encuentre el número deseado de objetos (1 o *k*, dependiendo del tipo de búsqueda que se esté haciendo). Formalmente, si la estructura que se usa a modo de índice está planteada para distancias reales (no necesariamente enteras), entonces la búsqueda del objeto *x* deberá comenzar con un radio *r* tal que

$$r = a^i \varepsilon \quad (a > 1)$$

de forma tal que ε es un infinitésimo *(0 < ε < 1)*. Se comienza con *i = 0*, y se incrementa sucesivamente ese valor hasta llegar a encontrar al menos un primer objeto (lo cual sería suficiente si se busca el vecino más próximo) o hasta llegar al menos a encontrar *k* objetos (si se buscaban los *k* vecinos más próximos). Finalmente, se puede hacer un proceso ajustando el radio entre $[r = a^{i-1}\varepsilon]$ y $[r = a^i \varepsilon]$ hasta quedar exactamente con el número deseado de elementos. Nótese que aún si se está buscando el vecino más próximo, es posible que el conjunto de salida contenga más de un objeto (y no necesariamente un objeto único), ya que podría haber varios objetos a la misma distancia mínima de *x*. Por otra parte, si se están buscando los *k* vecinos más próximos, el conjunto de salida podría contener todos los *n* elementos de la estructura usada como indice, si fuera el caso que *k >= n*.

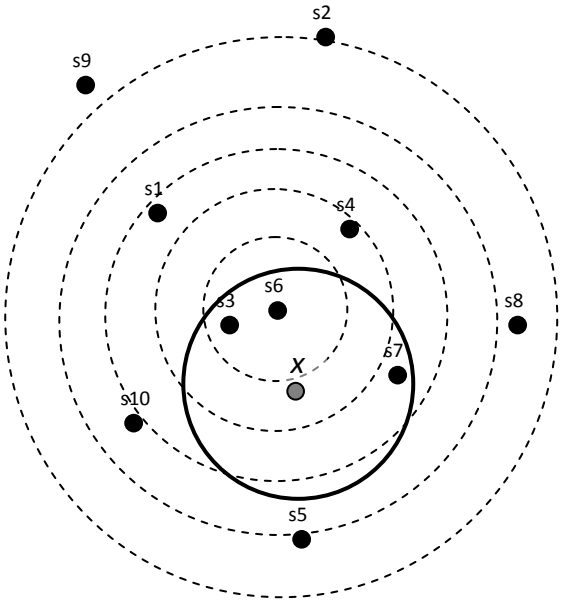

Figura 5: Búsqueda por rango (con r = 2) de un objeto x en un espacio métrico

No obstante lo anterior, queda claro que la estructura de soporte de búsquedas es un *BKT*, planteado para distancias discretas. Por lo tanto, el proceso es esencialmente más simple: basta con comenzar con un radio de búsqueda *r = 0* (el menor radio con el que se puede comenzar si las distancias son discretas), y aplicar en forma iterativa el algoritmo de búsqueda por rango aumentando en cada vuelta el valor del radio en una unidad. El proceso se detendrá cuando se llegue a un conjunto de salida no vacío (si se buscaba el vecino más próximo) o cuando se llegue a tener exactamente *k* elementos en el conjunto de salida (si se buscaban los *k* vecinos más próximos).

Puede verse rápidamente que tanto para la búsqueda del vecino más próximo como para los *k* más próximos, el tiempo de ejecución en promedio se va acercando cada vez más al tiempo requerido para una búsqueda por rango con el radio *r* exacto (el cual se desconoce al comenzar el proceso). Mientras mayor sea el radio, mayor será la cantidad de objetos que se esperaría encontrar, y por lo tanto la complejidad crece.

Si bien se ha aplicado esta estrategia del incremento del radio, existen otras técnicas algorítmicas[7] que han sido planteadas por muchos investigadores para encontrar el vecino más próximo o los *k* más próximos en estructuras para búsqueda por rango (como el *algoritmo de vuelta atrás con radio decreciente*, o la *vuelta atrás con prioridad*, o alguna variante de la estructura conocida como *GNAT* (*Geometric Near-Neighbor Access Tree*)[4] para favorecer estas búsquedas especiales).

La mayor parte de las estructuras de búsqueda que existen o se han planteado para búsquedas por proximidad han sido pensadas y diseñadas para ser construidas una vez, y luego permanecer "inmutables" como un conjunto estático. Si bien sería esperable poder agregar nuevos objetos al índice creado, la operación de inserción podría no ser posible en algunas de estas estructuras (salvo creando todo el índice nuevamente), debido a que el propio proceso de construcción incluye saber de antemano algunas características del conjunto completo de objetos que conforma el espacio métrico de búsqueda. Lo mismo ocurre con el proceso de eliminación de objetos del índice: el borrado de un objeto es un proceso normalmente más complicado (cuando es admisible), debido a que altera de manera profunda el contenido de la estructura.

En el contexto del proyecto *RNA-AC* (subproyecto *RNA-EH*) las condiciones de trabajo hacen suponer que al crear el índice o estructura de búsqueda debería estar previsto el poder insertar nuevos elementos a posteriori de la creación inicial (sin afectar de manera significativa el rendimiento de los procesos de búsqueda posteriores): se ha supuesto que los investigadores en demografía podrían aceptar como una muy buena funcionalidad, el poder agregar al índice una nueva tupla con un rango de porcentajes propuesto como resultado de una búsqueda previa. Y también se ha supuesto que estos investigadores podrían simplemente querer eliminar del índice una tupla que parecía adecuada, pero que finalmente haya resultado ser no representativa.

Todas estas consideraciones han incidido en la decisión final de utilizar un *BKT* como soporte para el índice de búsqueda: es cierto que el *BKT* es quizás la más simple de las estructuras de búsqueda por proximidad, pero esa simpleza la hace también flexible en cuanto a las necesidades planteadas[5][7]. En un *BKT* la inserción de nuevos objetos puede manejarse en forma razonablemente eficiente y directa, sin requerir una reorganización total del árbol (aunque eventualmente podría hacer falta "limpiar todo" y volver a construirlo para favorecer una selección más adecuada del pivot inicial y un mejor balance del árbol).

Y en cuanto a los borrados o eliminaciones de objetos, es cierto que esta operación no es físicamente aceptable (ya que la eliminación de un pivot provocaría que todos sus hijos queden huérfanos, cortando todos los caminos de búsqueda a partir del pivot faltante), pero puede manejarse en forma de *marcado lógico*: si se desea eliminar un objeto, simplemente

lo deja en el árbol pero se "marca" como eliminado para no destruir la estructura lógica del índice. Esta forma de trabajar las eliminaciones a la larga puede provocar pérdida de eficiencia (en consumo de memoria y tiempo de ejecución), por lo que también sería necesaria eventualmente la "limpieza total" y la reconstrucción completa.

Existen estructuras de búsqueda por proximidad que permiten plenamente las operaciones de inserción y borrado sin problemas de pérdida de eficiencia (tales como las estructuras *FQT*, *FQTH*, *FQA* y otras[7]) pero la sencillez en el planteo del *BKT* para este modelo demográfico terminó siendo el factor que inclinó la balanza en la decisión.

Resultados.

El objetivo inicial del proyecto *RNA-AC* era el de crear, alimentar y entrenar una *red neuronal artificial* para permitir el reconocimiento de tuplas conocidas y representativas de ciertos valores demográficos, y encontrar y proponer valores aceptables para nuevas tuplas no conocidas (por no haber sido censadas, por ejemplo). Los investigadores del proyecto asignados a este própósito lograron desarrollar una sofisticada aplicación basada en este modelo, que funciona y se adapta correctamente a la situación planteada.

Sin embargo, se pensó en la posibilidad de plantear un modelo alternativo basado en técnicas de *búsqueda por proximidad*, que pudiera desarrollarse en paralelo al proyecto marco de redes neuronales, para contar con una segunda visión, que pudiera ampliar horizontes y obtener conclusiones que validen, complementen o incluso refuten los caminos elegidos. Esa fue una de las razones de ser del subproyecto *RNA-EH*, que aquí se ha expuesto a nivel conceptual.

Esencialmente, el modelo de *búsqueda por proximidad* ha demostrado ser válido y fiable para el problema de la búsqueda de tuplas similares. Se ha mostrado que se puede construir una estructura de búsqueda a modo de índice (un *BKT* en el caso en estudio) en un tiempo de ejecución promedio relativamente corto ($O(n*log(n))$) y usando una cantidad de memoria linealmente proporcional al número de tuplas ($O(n)$). La estructura de búsqueda elegida es también razonablemente eficiente al momento de hacer *búsquedas por rango* (con tiempo de ejecución sublineal en promedio: $O(n^{\alpha})$ tal que $0 < \alpha < 1$), aunque el tiempo de ejecución crece cuando se trata de búsqueda del *vecino más próximo* o de los *k más próximos*, debido a la aplicación iterativa del algoritmo de búsqueda por rango con incremento gradual del radio (en otras palabras, el valor de α crece significativamente para estas variantes en el análisis del peor caso).

Discusión.

El paso final (en el cual se está trabajando) consiste en cruzar los resultados obtenidos por el funcionamiento de ambos modelos: tanto el modelo de red neuronal como el de búsqueda por similaridad *satisfacen los requerimientos de búsqueda planteados*, pero se está aún en la fase de comparación de ambos modelos para determinar si uno es mas rápido que el otro,

o usa mejor los recursos de memoria, o es mas fiable en cuanto a los resultados entregados (entre otros parámetros de comparación).

Alternativamente, el modelo de *búsqueda por similaridad* admitiría simplemente cambiar la estructura usada (el *BKT*) por cualquier otra que encaje en el planteo polimórfico con que fue diseñada la aplicación. Técnicamente, se ha desarrollado la primera versión del motor de búsqueda inicialmente en lenguaje *Java* (simplemente porque todos los investigadores que trabajaron en el planteo del modelo son buenos programadores *Java*), y se está trabajando en migrar ese desarrollo a *C#* para adaptarlo a los requerimientos del proyecto marco *RNA-AC*, cuya red neuronal se planteó en *C#*. Pero tanto en un lenguaje como en el otro, la idea es que cualquier clase que pretenda representar y soportar a un índice de búsqueda por proximidad, deberá implementar la interface *Indexador*, que en *Java* luce básicamente así:

```
public interface Indexador < E extends Medible >
{

  /**
   * Inserta un nuevo objeto x en la estructura. Cada estructura deberá indicar cómo
   * debe comportarse el método en situaciones de inserción repetida.
   * @param x el objeto a insertar en el árbol.
   */
  void add( E x );

  /**
   * Elimina un objeto x del árbol. La técnica de eliminación dependerá de la
   * naturaleza de la estructura: en algunas será factible la eliminación física, y en
   * otras sólo será posible la eliminación por marcado lógico.
   * @param x el objeto a buscar y eliminar del árbol.
  */
  void remove( E x );


  /**
   * Recupera todos los objetos de la estructura, que estén a una distancia de
   * r o menos del objeto x tomado como parámetro. En general, se espera que si
   * x es null o r es negativo, el método retorne null.
   * @param x el objeto del cual se buscan similares.
   * @param r el radio de búsqueda.
   * @return una lista con todos los objetos a distancia <= r de x.
   */
  ArrayList < E > rangeQuery( E x, int r );

  /**
   * Recupera el "vecino más próximo" (nearest neighbor) al objeto x
   * tomado como parámetro. Si hubiera varios objetos a la misma distancia
   * mínima, el método los retorna todos. Esto es: el método retorna una
```

```
     * lista con todos los objetos a menor distancia de x. Se espera que si x
     * es null el método retorne null.
     * @param x el objeto del cual se busca al vecino más próximo.
     * @return una lista con los objetos a menor distancia de x.
     */
    ArrayList < E > nNQuery( E x );

    /**
     * Recupera los "k vecinos más próximos" (k nearest neighbors) al objeto x
     * tomado como parámetro. Esto es: el método retorna una lista con los k objetos
     * a menores distancias de x. Se espera que si x es null o k es menor a 1, el
     * método retorne null. Cada estructura deberá determinar qué hacer si k mayor a
     * la cantidad
     * de objetos que ella contiene.
     * @param x el objeto del cual se buscan sus k vecinos más próximos.
     * @param k la cantidad de objetos a recuperar.
     * @return una lista con los k objetos a menores distancias de x.
     */
    ArrayList < E > nNKQuery( E x, int k );
}
```

Como se ve, la interface prevé cinco métodos obvios: los dos primeros (*add()* y *remove()*) deberán implementar las operaciones de inserción y borrado respectivamente dentro de la estructura (de acuerdo a las consideraciones de flexibilidad que se han enunciado en un apartado anterior). Y los otros tres métodos (*rangeQuery()*, *nNQuery()* y *nNKQuery()*) deberán implementar las tres operaciones básicas de la búsqueda por proximidad, que respectivamente son la *búsqueda por rango*, la *búsqueda del vecino más próximo*, y la *búsqueda de los k vecinos más próximos*. De hecho, en el modelo desarrollado se ha programado una clase *BKT* que implementa esta interface y aplica los algoritmos que se han descripto a lo largo de este desarrollo. Lo anterior supone que si en la práctica se deseara probar con otras estructuras de búsqueda, el modelo está diseñado con la flexibilidad suficiente como para “reemplazar una pieza por otra”, y el polimorfismo hará el resto.

Agradecimientos.

El equipo de trabajo que desarrolló el modelo basado en búsqueda por similaridad expresa su agradecimiento y reconocimiento a todos los miembros del proyecto *RNA-AC* que fueron los que comenzaron con este desarrollo, en conjunto con el *IIGHI* de la Provincia del Chaco.

Referencias.

[1]. Ricardo Baeza-Yates. Searching: an algorithmic tour. Encyclopedia of Computer Science and Technology, 37:331–359, 1997.

[2]. Ricardo Baeza-Yates, Walter Cunto, Udi Manber, and Sun Wu. Proximity matching using fixed-queries trees. In M. Crochemore and D. Gusfield, editors, Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, pages 198–212. Springer-Verlag, 1994.
[3]. Tolga Bozkaya and Meral Ozsoyoglu. Distance-based indexing for high dimensional metric spaces. In ACM Press, editor, Proceedings of the ACM International Conference on Management of Data (SIGMOD 1997), pages 357–368, May 1997.
[4]. Sergey Brin. Near neighbor search in large metric spaces. In 21st conference on Very Large Databases, 1995.
[5]. Walter A. Burkhard and Robert M. Keller. Some approaches to best-match file searching. Communications of the ACM, 16(4):230–236, April 1973.
[6]. Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. Pivot selection techniques for proximity search in metric spaces. In SCCC 2001, Proceedings of the XXI Conference of the Chilean Computer Science Society, pages 33–40. IEEE Computer Science Press, 2001.
[7]. Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and Jose Luis Marroquín. Searching in metric spaces. ACM Computing Surveys, 33(3):273–321, September 2001.
[8]. Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, 1966.
[9]. Luisa Micó, José Oncina, and R. Enrique Vidal. A new version of the nearestneighbor approximating and eliminating search (aesa) with linear pre-processing time and memory requirements. Pattern Recognition Letters, 15:9–17, 1994.
[10]. Gonzalo Navarro. Searching in metric spaces by spatial approximation. In Proceedings of String Processing and Information Retrieval (SPIRE'99), pages 141– 148. IEEE Computer Science Press, 1999.
[11]. Enrique Vidal. An algorithm for finding nearest neighbors in (aproximately) constant average time. Pattern Recognition Letters, 4:145–157, 1986.
[12]. Peter Yianilos. Data structures and algorithms for nearest-neighbor search in general metric spaces. In Proceedings of the fourth annual ACM-SIAM Symposium on Discrete Algorithms, pages 311– 321. ACM Press, 1993.
[13]. Peter Yianilos. Excluded middle vantage point forests for nearest neighbor search. In Proceedings of the 6th DIMACS Implementation Challenge: Near neighbor searches (ALENEX 1999), January 1999.
[14]. Pavel Zezula, Giuseppe Amato, Vlatislav Dohnal, and Michal Batko. Similarity search. The metric space approach, volume 32 of Advances in Database Systems. Springer, 2006.

Datos de Contacto:

- *Mgtr. Ing. Valerio Frittellli [UTN Córdoba – email: vfrittelli@gmail.com]*
- *Ing. Felipe Adrián Steffolani [UTN Córdoba – email: fsteffolani@gmail.com]*
- *Ing. Romina Gabriela Teicher [UTN Córdoba – email: rteicher@gmail.com]*
- *Dra. María del Carmen Rojas [IIGHI – CONICET – email: rojas_herrera@arnet.com.ar]*
- *Ing. Juan Eduardo Picco [UTN – Córdoba – email: jepicco@scdt.frc.utn.edu.ar]*
- *Ing. Juan Carlos Vázquez [UTN – Córdoba – email: jcvazquez@sistemas.frc.utn.edu.ar]*