Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Nuevos algoritmos para recuperación de "ítems empaquetados"

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Amit Stein, Juan Andrés Knebel

Director: Esteban Feuerstein

Codirectora: Isabel Méndez Diaz
Codirectora: Paula Zabala

Buenos Aires, 2017

NUEVOS ALGORITMOS PARA RECUPERACIÓN DE "ÍTEMS EMPAQUETADOS"

Los motores de búsqueda son la herramienta utilizada por los usuarios de Internet para obtener información acerca de diferentes elementos, como pueden ser sitios web, productos o mensajes.

El crecimiento de Internet provocó el incremento y protagonismo de las búsquedas. Como consecuencia, los motores comenzaron a ser cada vez más específicos y complejos.

Esta tesis se focaliza en el caso del usuario que requiere más de un elemento como respuesta a su búsqueda. Con las herramientas actuales el usuario debe repetir las búsquedas reiteradas veces combinando palabras, frases o atributos de los elementos. La calidad de las soluciones estarán sujetas a la imaginación y criterios del usuario.

Como solución a esta inquietud y a los problemas que presenta, se propone que el resultado de la búsqueda esté formado por conjuntos de elementos. Estos conjuntos, llamados *paquetes*, tendrán la particularidad de estar integrados por elementos relacionados bajo algún criterio de similitud y, al mismo tiempo, complementarios. El objetivo es que el usuario pueda lograr una mejor experiencia de búsqueda interviniendo en el inicio de la consulta y luego evaluando qué paquetes satisfacen sus intereses iniciales.

En este trabajo se presentan algoritmos heurísticos para devolver este tipo de resultados. Los mismos son evaluados en dos bases de datos (una corresponde a artículos académicos relacionados con Ingeniería de Software y la otra a artículos sobre atracciones turísticas).

Se llegará a la conclusión de que los algoritmos propuestos se comportan de acuerdo a las expectativas y mejorando los existentes en la literatura.

Palabras claves: Recuperación de la información, Agrupamiento, Heurísticas.

Índice general

1. INTRODUCCIÓN

1.1. Los motores de búsqueda

An algorithm must be seen to be believed.
Donald Knuth.

Un motor de búsqueda es un sistema de recuperación de la información diseñado para buscar información almacenada en un sistema informático. El propósito es reducir el tiempo requerido para brindarle al usuario resultados relevantes a sus búsquedas. Normalmente los resultados son presentados en forma de una lista ordenada.

Los motores de búsqueda cuentan con una interfaz que le permite al usuario especificar un criterio de búsqueda para que el motor encuentre los elementos que coincidan con tal criterio. En el caso de un motor de búsqueda de texto generalmente el criterio esta definido como un conjunto de palabras que se desea que contenga uno o más documentos.

La lista de elementos que coinciden con el criterio de búsqueda especificado en la consulta es generalmente clasificada, ésta clasificación ayuda al usuario a reducir el tiempo requerido en encontrar la información pretendida. Al igual que en los criterios de búsqueda, existen distintas alternativas de clasificación y se siguen desarrollando nuevas. De los tipos de clasificación se destacan: de similitud entre los elementos, por popularidad y de autoridad (la autoridad de un elemento es determinado por la cantidad de elementos que hacen referencia a él).

Con el desarrollo de Internet la cantidad de información que es almacenada en los distintos servidores alrededor del planeta se incrementa día a día [17]. El aumento de la información guardada requiere que los motores de búsqueda evolucionen constantemente y entren en escena nuevos buscadores especializados en temáticas puntuales. Como consecuencia, los motores adquieren cada vez más importancia para los usuarios de Internet, ya que son la única herramienta que poseen para poder acceder a la información.

Los primeros motores de búsqueda solo admitían una palabra o frase que el usuario ingresaba, entonces la tarea de los llamados "buscadores" consistía en encontrar una coincidencia exacta dentro de una colección de documentos. A medida que la cantidad de información comenzó a incrementarse, las estrategias iniciales dejaron de ser útiles. Por lo tanto, las formas de buscar fueron evolucionando y adaptándose a las necesidades del usuario con el objetivo final de entregar resultados completos y relevantes.

La **Recuperación de la Información** (IR por Information Retrieval en inglés) [2, 15, 20] es el área de investigación que estudia la búsqueda y recuperación de información en cualquier tipo de colección de recursos digitales. Los criterios elegidos para realizar las búsquedas son de lo más variados, por ejemplo, los precios de los pasajes aéreos entre dos ciudades, los libros de un determinado autor o el correo electrónico de confirmación de una compra. Según lo publicado en el artículo [2] se identifican dos etapas en la resolución de los problemas puntuales del área. La primera es la elección de un modelo que permita calcular la relevancia de un documento frente a una consulta. La segunda son las estructuras de datos y algoritmos que implementen el modelo del problema y lo resuelvan de manera eficiente.

Los motores de búsquedas que se utilizan en la web, como Google y Yahoo!, son clásicos ejemplos de una aplicación de IR. El proceso de búsqueda comienza cuando el usuario ingresa una consulta esperando obtener una colección de elementos que coincidan con el criterio de búsqueda elegido. Normalmente ocurre que son varios los elementos del universo que concuerdan la búsqueda realizada, pero se diferencian en los grados de relevancia que mantienen con el criterio elegido. Esta diferencia es utilizada para ordenar los elementos encontrados en lo que se conoce como ranking de resultados.

Los motores de búsqueda tienen sus propios algoritmos de ranking de resultados como por ejemplo, PageRank (actualmente Google utiliza una variante del original PageRank) [3] o HITS [14]. Dichos algoritmos son los encargados de valorizar los elementos pertenecientes a su dominio y asignarles un valor de relevancia entre ellos. La valorización de los elementos se realiza de acuerdo a la estructura de la web.

En los casos de las búsquedas por Internet la idea de PageRank [3] es determinar qué tan importante es una página en Internet. El mecanismo consiste en generar un voto hacia un página, por cada uno de los restantes que contenga un enlace hacia él. No solo importa la cantidad de votos sino también quién emite el voto. Generalmente, por cuestiones de mercado, estos algoritmos no son divulgados por ser una componente importante y clave en el modelo de negocio de los desarrolladores de motores de búsquedas.

A continuación se describe la motivación de este trabajo. Cuando el resultado basado en un ranking ordenado no satisface la expectativa del usuario, porque la respuesta requerida por el usuario esta conformada por un grupo de elementos, entonces deberá realizar varias consultas en el motor cambiando o combinando su criterio de búsqueda. De esta forma, el usuario podrá explorar todas las colecciones de elementos resultantes hasta lograr encontrar el resultado deseado.

1.2. Motivación

Esta tesis está motivada por el usuario de motores de búsquedas que espera un resultado diferente al de una lista ordenada de elementos. Esta preferencia se puede deber por la naturaleza de su interés o porque el dominio de su problema se podría expresar mejor como un conjunto de elementos, y no como listas de resultados de tamaños inmanejables para una persona.

En la publicación [1] se propone un enfoque en el cual los resultados de las búsquedas no se presenten como una simple lista "vertical". La idea de los autores es que la información generada se visualice agrupada bajo algún criterio de similitud que satisfaga las expectativas del usuario. De esa manera, se evita la necesidad de realizar una nueva intervención humana logrando una mejor experiencia de uso y adecuándose a las expectativas del usuario.

Para ilustrar un posible escenario se toma el ejemplo de un usuario con la idea de organizar su viaje utilizando los motores de búsqueda tradicionales de Internet. Planear un viaje típicamente requiere realizar múltiples búsquedas con distintos motores para recolectar la información de los diferentes destinos que se quiere visitar. Los datos que en general se necesitan son: la distancia geográfica, los precios de las atracciones, las actividades que se pueden realizar o las opiniones acerca de los destinos.

Otro ejemplo es el escenario de un coleccionista de discos que desea hacer una compra on-line de música de diferentes países. Al comprador no le interesa ninguna época en particular, pero sí quiere que los discos pertenezcan al mismo período. Se supone que se

cuenta con un presupuesto máximo para la compra, por ejemplo de $100. El coleccionista puede realizar una búsqueda en la tienda de discos on-line ingresando como palabra clave nombres de diferentes países.

En los siguientes ejemplos se presentarán casos de búsquedas y sus posibles resultados.

Búsqueda: Inglaterra

- Physical Graffiti - Led Zeppelin (1975)
- Perfect Strangers - Deep Purple (1984)
- It's Hard - The Who (1982)
- Wheels of Fire - Cream (1968)
- The White Album - The Beatles (1968)
- Innuendo - Queen (1991)
- Sticky Fingers - The Rolling Stones (1971)
- Please Please Me - The Beatles (1963)
- 461 Ocean Boulevard - Eric Clapton (1964)
- Physical Graffiti - Led Zeppelin (1975)

Búsqueda: Argentina

- El Cielo Puede Esperar - Attaque 77 (1990)
- Confesiones de Invierno - Sui Generis (1973)
- Kamikaze - Luis Alberto Spinetta (1982)
- Acariciando lo Aspero - Divididos (1991)
- La Biblia - Vox Dei (1971)
- Rock de la Mujer Perdida - Los Gatos (1969)
- Lo Mejor de Violeta Rivas - Violeta Rivas (1966)
- Se Dice de Mí - Tita Merello (1954)
- Películas - La Maquina de Hacer Pajaros (1977)

Búsqueda: Años 1960 - 1970

- Let it Bleed - The Rolling Stones (1969)
- Please Please Me - The Beatles (1963)
- My Favorite Things - John Coltrante (1960)
- Tommy - The Who (1969)
- Rock de la Mujer Perdida - Los Gatos (1969)
- Lo Mejor de Violeta Rivas - Violeta Rivas (1966)

Búsqueda: Años 1965 - 1975

- Tommy - The Who (1969)
- Fuente y Caudal - Paco de Lucía (1973)
- Fragile - Yes (1971)
- Rock de la Mujer Perdida - Los Gatos (1969)
- Let It Bleed - The Rolling Stones (1969)
- Confesiones de Invierno - Sui Generis(1973)
- 461 Ocean Boulevard - Eric Clapton (1974)
- Natty Dread - Bob Marley (1974)
- Un Muchacho Como Yo - Palito Ortega (1967)

A partir de los discos que obtuvo como resultado de su búsqueda, el comprador debe realizar la tediosa tarea de seleccionar aquellos de su interés analizando cada uno de los resultados obtenidos. Aún si el usuario es un excelente conocedor de música, sigue existiendo la posibilidad de que, por error y contra su intención inicial, seleccione discos de épocas diferentes o más de un disco del mismo origen. El análisis previamente mencionado adiciona el riesgo de no aprovechar de manera eficiente el presupuesto. Además las listas generadas en una situación de la vida real contendrán mucha más información que la mostrada en el ejemplo, por lo tanto las combinaciones que debería analizar manualmente el coleccionista escapan a la capacidad humana.

En este tipo de búsquedas o consultas, el usuario espera un conjunto de elementos relacionados entre sí. En la actualidad existen motores de búsqueda, como Carrot [18], que devuelven los resultados en colecciones de documentos relacionados. Este proceso, conocido como "agrupamiento" (*clustering* en inglés) [13], es el proceso de agrupar un conjunto de objetos de forma tal que los objetos similares pertenezcan al mismo grupo. Retomando el escenario de los discos de música, sí se elije como criterio de similitud el año de lanzamiento del disco, utilizando las técnicas de *clustering* tradicionales, se generaran tantos *clusters* como períodos emblemáticos hubiera. Al mismo tiempo en cada *cluster* se encontrarán todos los discos de ese período repitiendo el origen. Una vez obtenido el resultado se debe explorar todos los *clusters* para elegir los discos, considerando el

presupuesto inicial y la diversidad de procedencias.

Una respuesta más conveniente para el coleccionista es un conjunto de paquetes donde los discos dentro de cada paquete sean cercanos en el tiempo (*criterio de similitud*), correspondan a artistas de diferentes países (*criterio de compatibilidad*), estén representados varios países (*diversidad*) y se ajusten al presupuesto (*validez*). Además puede ser deseable que los paquetes de la lista cubrieran todos los períodos (*completitud*). Esta última condición no fue explorada en este trabajo pero si mencionada como posible extensión.

Si la tienda on-line aplicase un motor de búsqueda sofisticado bajo este enfoque, permitiendo ingresar un presupuesto máximo ($100 en el ejemplo), un criterio de similitud (en este caso año de lanzamiento) y uno de complementariedad (atributo origen del intérprete), un posible resultado es:

Paquete 1

- Physical Graffiti - Led Zeppelin (1975 - Inglaterra) $20
- Confesiones de Invierno - Sui Generis (1973 - Argentina) $20
- Natty Dread - Bob Marley (1974 - Jamaica) $30
- Saturday Night Fiver - Bee Gees (1977 - Australia) $30

Paquete 2

- Please Please Me - The Beatles (1963 - Inglaterra) $20
- My favorite things - John Coltrane (1960 - EEUU) $30
- Rock de la mujer perdida - Los Gatos (1969 - Argentina) $20
- Digan lo que digan - Rafhael (1967 - España) $20

Paquete 3

- Physical Graffiti - Led Zeppelin (1975 - Inglaterra) $20
- Saturday Night Fiver - Bee Gees (1977 - Australia) $30
- Un muchacho como yo - Palito Ortega (1967 - Argentina) $35

El grado de similitud de los elementos que forman un paquete define su calidad. En el caso del coleccionista, cuanto más contemporáneos sean los discos dentro de un paquete, mayor es su calidad. Además, el conjunto de paquetes que se obtiene como respuesta puede cubrir diferentes épocas, dándole diversidad a las opciones brindadas.

En concreto, lo que se quiere lograr en el escenario descripto, y en cualquier otro problema similar de búsquedas, es otorgarle al usuario un conjunto de paquetes que cumplan siempre con las siguientes propiedades:

- **Compatibilidad**: Los elementos dentro de un paquete deben ser similares según un criterio dado.
- **Validez**: El costo total de los elementos del paquete no puede superar el presupuesto.

- **Diversidad**: Los paquetes entre sí deben ser, en lo posible, diversos según el criterio utilizado para definir la compatibilidad.
- **Complementariedad**: Todos los paquetes cumplen que, para un mismo atributo especificado, los elementos dentro del paquete contienen un valor diferente para tal propiedad.

Es importante destacar que para este trabajo se asume que la calidad de los elementos individuales está garantizada por el sistema y es igual para todos ellos. Asumiendo esta premisa se puede asegurar la calidad de cada paquete obtenido dentro del resultado final. Al mismo tiempo los paquetes pueden ser comparados con cualquier otro proveniente del mismo algoritmo u otro diferente.

1.3. Definición formal del problema

En el artículo [1] se define formalmente el problema de recuperar un conjunto de paquetes $S = \{S_1, \ldots, S_k\}$ compuestos por elementos de $I = \{i_1, \ldots, i_n\}$. Siendo $P(I)$ el conjunto de partes de I entonces, $S_i \in P(I)$ un conjunto de objetos que satisfaga las reglas de **complementariedad** y **presupuesto**. La regla de **complementariedad** no permite que existan dos objetos con una misma característica (de acá en adelante la característica de un objeto se la referencia como atributo) en el mismo paquete. Mientras que la regla de **presupuesto** está definida para que la suma de los costos de los objetos no supere el presupuesto asignado.

Dado el conjunto de objetos $I = \{i_1, \ldots, i_n\}$, una función de similitud $s(u, v) : I \times I \to [0; 1]$, una de costo $f(u) : I \to (0; +\infty)$, un presupuesto $\beta \in \Re^+$ (servirá como cota máxima para formar un paquete), una función α para obtener el atributo complementario, un valor γ tal que $0 < \gamma < 1$ y $k \in N$ que indica la cantidad de paquetes que debe tener la solución, se desea hallar un conjunto válido de paquetes $S = \{S_1, \ldots, S_k\}$ que maximiza la función:

$$\sum_{1 \leq i \leq k} \sum_{u,v \in S_i} \gamma s(u, v) + \sum_{1 \leq i < j \leq k} (1 - \gamma)(1 - \max_{u \in S_i, v \in S_j} s(u, v)) \tag{1.1}$$

Cada elemento $S_i \in S$ es válido si y sólo si satisface las siguientes reglas:

- **Complementariedad:** se define α como la función que se aplica a un objeto del dominio para obtener el atributo utilizado en la evaluación de la complementariedad. Entonces debe cumplir que $\forall\ u, v \in S_i\ \alpha(u) \neq \alpha(v)$
- **Presupuesto:** dada la función de costo f y el presupuesto β, entonces $\forall\ S_i \in S\ f(S_i) \leq \beta$, donde $f(S_i)$ es la suma de costos de los elementos pertenecientes al paquete.

La Ecuación 1.1 es una típica función objetivo de un problema de agrupamiento (*clustering*), donde la calidad del *clustering* es una combinación entre la calidad de cada *cluster* (*intra-cluster*) y la separación entre *clusters* (*inter-cluster*). A través del parámetro γ el usuario puede definir el balance deseado entre las componentes `intra` e `inter` de una solución. En el caso de querer priorizar la cohesión de los paquetes sobre la diversidad, el valor de γ debe ser cercano a uno. En cambio, si se quiere priorizar la diversidad, el valor de γ debe ser cercano a cero.

1.4. Objetivos del trabajo

En [1] se demostró que el problema planteado en la sección anterior pertenece a la familia de problemas NP-Difícil, reduciéndolo al conocido problema de *Maximum Edge Subgraph problem* [10]. En el mismo artículo se presentan familias de algoritmos para resolver la misma problemática aquí planteada. La mayoría de las propuestas se basan el algoritmos heurísticos, junto con una propuesta de solución exacta utilizando programación lineal entera. Además se compara el rendimiento en cuanto a la eficiencia y calidad de las soluciones provistas.

En esta tesis se analizan, experimentan y proponen nuevas alternativas a los algoritmos ya conocidos para el problema *Recuperación de la Información Combinada*, con el fin de incrementar la eficiencia de los algoritmos y mejorar la calidad de las soluciones. En particular el trabajo se concentrará en los algoritmos *Produce and Choose* de [1], ya que demostraron ser los más efectivos. En los próximos capítulos se explica más detalladamente esta familia de algoritmos que utiliza un esquema de dos fases para obtener una solución. En la primera fase se produce un conjunto de paquetes válidos y en la segunda se selecciona un subconjunto de ellos.

Los cambios propuestos en esta tesis se focalizan en mejorar ciertos criterios de selección y *clustering* de los algoritmos. Se decidió concentrarse en estos elementos ya que resultan decisivos en la obtención de soluciones de mejor calidad. Por otro lado, se proponen dos metaheurísticas de búsqueda tabú, una como procedimiento de mejora en la etapa de producción de paquetes y otra para mejorar la solución obtenida por los distintos algoritmos constructivos.

Para evaluar los algoritmos propuestos se utilizan dos bases de datos. La primera corresponde a una base bibliográfica de artículos académicos [8]. Esta base contiene alrededor de 7800 artículos relacionados con la Ingeniería de Software, de 9800 autores, presentados en diferentes conferencias entre los años 1975 y 2011. Los artículos están catalogados por autores, tópicos que cubren (asignándole un porcentaje de relevancia a cada uno de los 38 tópicos que se consideran), conferencia en la que fue presentado y afiliaciones de los autores. La segunda base utilizada, contiene 200 atracciones turísticas de Europa con información del costo de la visita, tipo de atracción (parque, museo o edificio) y similitud existente entre ellas. En la Sección 4.1 se puede encontrar más detalle.

Cabe señalar que el desarrollo y los resultados obtenidos en esta tesis fueron presentados en Computing Conference (CLEI), 2016 XLII Latin American y publicados en IEEE Proceedings CLEI2016 [6]

A continuación se describe como se encuentran organizados los próximos capítulos de este trabajo. El Capítulo 2 exhibe las características de los algoritmos presentes en la literatura, diferencias y similitudes de los mismos, además una explicación en profundidad de los algoritmos utilizados en este trabajo. El Capítulo 3 propone nuevas alternativas para resolver el problema *Recuperación de la Información Compuesta*. El Capítulo 4 está dividido en dos secciones, la Sección 4.1 presenta las instancias que se utilizaron para experimentar y la Sección 4.2 compara experimentalmente los métodos propuestos y analiza la calidad de las soluciones encontradas. Finalmente el Capítulo 5 expone las conclusiones generales de la tesis y mejoras que surgen de la misma.

2. TRABAJOS PREVIOS

En la literatura pueden encontrarse diversos trabajos que abordan la noción de *Recuperación de la Información* con semánticas diferentes. La mayoría de las investigaciones tienen en común que ofrecen al usuario resultados en forma de conjuntos o paquetes de elementos de interés. Las diferencias radican en la construcción de los paquetes, las definiciones de similitud entre sus elementos y la diversidad de los resultados. A continuación se mencionan dos publicaciones relacionadas con la recuperación de la información que influyeron en este trabajo.

En el artículo [16] se plantea recuperar la información complementaria a un "ítem central" (*composite item*). La información obtenida es un conjunto de "paquetes satélites", que son conjuntos de ítems de diferente tipo al "ítem central" pero a su vez compatibles entre ellos. Un escenario de aplicación posible es un comprador de un teléfono celular (ítem central) que dispone de cierto presupuesto al que se le ofrecen paquetes que incluyen otros ítems relacionados como funda, tapa, parlantes, o cargador. El objetivo implica identificar todos los paquetes satélites válidos y maximales a partir de un ítem central. Un paquete maximal y válido es un conjunto de elementos que respetan un presupuesto, son compatibles con el ítem central y no son un subconjunto de otro paquete válido. Para la construcción de los paquetes satélites se utiliza un algoritmo que elige aleatoriamente ítems que sean compatibles con el ítem central y cumpla las restricciones de presupuesto y validez. Luego implementan algoritmos *greedy* basados en técnicas de clustering para lograr encontrar el mejor cubrimiento de los paquetes generados.

En la publicación [19] se sugiere que el resultado de un sistema de recomendación debería ser un conjunto de paquetes y no una clásica lista ordenada por los ítems más relevantes. Los autores consideran que muchas aplicaciones se beneficiarían al utilizar este criterio. Por ejemplo, puede aplicarse para un turista que está interesado en sugerencias de lugares o puntos de interés para su viaje. Suponiendo que dispone de un determinado tiempo y un presupuesto máximo, y que maneja una noción de compatibilidad entre ítems: visitar hasta tres museos, que el recorrido sea menor a cierta distancia, etc. Si cada atracción tiene asociado un ranking, la idea es ofrecer los mejores k paquetes de atracciones de tal manera que en cada paquete se respete el presupuesto y la compatibilidad entre los ítems. El algoritmo para generar el resultado que propone el artículo se obtiene a partir de una variación del algoritmo utilizado para resolver el problema de la mochila [12] y [11] para valores enteros.

Si bien el objetivo de ambos trabajos es entregar conjuntos de ítems relacionados que generan un valor agregado al usuario de la aplicación, ninguno tiene en cuenta la diversidad entre los paquetes sugeridos, formalizando la recuperación de la información como un problema de *clustering* que sólo considera la compatibilidad entre ítems. Como alternativa, en [1] se propone conjugar, en el conjunto de paquetes sugeridos al usuario, las nociones de diversidad, similitud y compatibilidad dependiendo de la necesidad y elección del mismo.

Los algoritmos propuestos y utilizados en esta tesis están inspirados en las propuestas e ideas de [1]. A continuación se describen los algoritmos originales con un mayor nivel de detalle. Esencialmente, se dividen en tres grandes familias de acuerdo a su enfoque:

- *Produce and Choose* (PAC): Utiliza un esquema de dos fases, en la primera se pro-

ducen paquetes válidos y en la segunda se eligen k de ellos. Para la producción de paquetes se implementaron dos algoritmos de *clustering*. El primero basado en *clustering jerárquico aglomerativo* (*C-HAC*) y el segundo en *k-means clustering* denominado *BOBO* (Bundles One-By-One), debido a que en cada paso se elige un elemento y a partir de él se construye un paquete (o bundle). Para la segunda fase se adaptaron heurísticas de la literatura para el problema *Maximum Edge Subgraph* [10].

En C-HAC inicialmente cada ítem forma un *cluster* unitario y luego sucesivamente en cada iteración se elige un par de *clusters* para ser unidos y generar un nuevo *cluster* válido, es decir que cumpla las restricciones de complementariedad y no supere el presupuesto β. El algoritmo finaliza cuando no existe un par de *clusters* que pueda ser unido en uno nuevo válido o se alcance una condición de parada (por ejemplo, un cierto número de *clusters* generados). Como criterio de elección de los *clusters* a unir se busca maximizar la función *Score*, que es la suma de las similitudes de todos los ítems del par candidato. Este criterio de unión sólo presta atención a la similitud intra-paquete, lo que hace que, cuando se busca una alta diversidad, el conjunto de paquetes generados de esa forma no sea necesariamente bueno.

BOBO-c está inspirado en el conocido método k-means [7]. En términos simples, consiste en generar $c*k$ clusters de un conjunto I de n ítems. El algoritmo comienza con todos los ítems del conjunto I como posibles pivotes en un conjunto P. Se selecciona un pivote, $p \in P$ y con los elementos de I se genera un *cluster* válido "alrededor"de éste. En caso que el *cluster* generado sea suficientemente bueno (su valor `intra` supera un parámetro μ) se agrega al conjunto de paquetes candidatos y los ítems del *cluster* se eliminan de P. La generación de paquetes continúa hasta que se cumpla el criterio de parada elegido, en este caso, la generación de $c*k$ paquetes. La estrategia *BOBO-c* puede dejar ítems excluidos de los paquetes generados. A raíz de la limitación del algoritmo, los autores desarrollaron una variante a la que llamaron *BOBO-E* (por Exhaustivo), el cual logra que todos los ítems pertenezcan a algún paquete. La desventaja radica en que su complejidad aumenta en comparación a la implementación inicial, impidiendo debido a sus tiempos de ejecución, su aplicación para grandes bases de datos.

Anteriormente se mencionó que los autores dividieron el problema en dos etapas: la fase de producción de paquetes, descripta previamente, y la fase de selección de paquetes, la cual se detalla a continuación. Luego de la producción de paquetes se deben seleccionar los mejores k que formarán la solución (selección simple). El problema de seleccionar los paquetes que maximizan la función objetivo se traduce en encontrar en un grafo completo G, el subgrafo de k vértices de mayor peso (considerando los pesos de los vértices y aristas). Los vértices representan los paquetes (con pesos asociados dados por sus valores `intra`) y cuyas aristas tienen como peso los valores `inter`. Con el problema debidamente identificado, los autores del trabajo [1] implementaron un algoritmo goloso. La implementación para la resolución del problema consiste en seleccionar iterativamente un paquete del conjunto que maximice la función objetivo evaluada en la solución temporal que se tiene hasta el momento. Señalan que en las primeras iteraciones de la selección el valor `inter` (o inter-paquetes) es despreciable con respecto a la suma de los valores `intra` (o intra-paquete). Como consecuencia concluyen que cuando se intenta privilegiar la

diversidad de paquetes en la solución final (valores bajos de γ), esta selección no representa una buena estrategia.

Como resultado de la forma en que las soluciones son generadas en las heurísticas del tipo *Produce and Choose* (construir una cantidad suficiente de paquetes y luego seleccionar un subconjunto de éstos), es de esperar que las soluciones generadas se enfoquen más en valorar la parte intra-paquete que la inter-paquete.

- *Cluster and Pick* (CAP): Sigue un esquema de dos fases al igual que las propuestas anteriores. La diferencia radica en que la primera etapa es la encargada de generar *k clusters* con alto grado de cohesión interno y una buena separación externa. Esto puede ser logrado con cualquier algoritmo conocido de "clustering". Se debe tener en cuenta que los *clusters* aquí obtenidos no necesariamente cumplen con las condiciones impuestas en la definición del problema, por ende puede ocurrir, con alta probabilidad, que no sean paquetes válidos. La segunda fase consiste en seleccionar el mejor paquetes válido de cada cluster. Para obtener el mejor paquete de un *cluster* se toman todos los elementos como pivotes y se construye un paquete válido al rededor de él, aquel que tenga el mayor valor de su función *Score* (la misma usada en los algoritmos *PAC*) será el elegido. Los paquetes obtenidos en la segunda etapa son los que formarán parte de la solución.

- Método IP: Se presenta un modelo de programación lineal entera que resuelve el problema de forma exacta. Los tiempos de ejecución de este enfoque lo hacen prohibitivo para grandes instancias.

Los autores de [1] evalúan los algoritmos propuestos sobre dos bases de datos reales. Una de ellas se refiere a atracciones turísticas en 10 ciudades europeas, donde por cada una de estas instancias se consideran 20 atracciones. El segundo conjunto de datos se compone de opiniones sobre restaurantes en 149 ciudades provenientes del sitio de Yahoo! Local. El número de restaurantes en cada ciudad varía de 300 a 2000. De la experimentación presentada, se concluye que las mejores soluciones se obtienen con los algoritmos que primero agrupan los ítems en paquetes válidos y luego seleccionan paquetes dentro de este agrupamiento (PAC).

En el próximo capítulo se presentan mejoras a los algoritmos y estrategias aquí detallados. Las modificaciones realizadas tienen como foco principal otorgarle mayor peso a los valores `inter` de los paquetes, en la producción y selección de los mismos. La introducción de los cambios es con el fin de obtener soluciones más diversas en comparación a las actuales. Para realizarlo se presenta una nueva función de similitud entre los paquetes al momento de su producción.

Entre las nuevas propuestas se incluye un nuevo algoritmo completamente diferente a todos los mencionados y una estrategia para mejorar las soluciones una vez construidas sin importar cuál fue el método que originó la solución.

3. NUEVAS PROPUESTAS

En este capítulo se presentan nuevas propuestas para resolver el problema de `Recuperación de Ítems Empaquetados`. El foco principal de las mismas es generar mejores soluciones a las obtenidas en [1], a partir de una gran cantidad de elementos pertenecientes al dominio del problema, utilizando como medida de comparación el valor de la Función Objetivo vista en el capítulo anterior en la Ecuación 1.1.

Las propuestas consisten en modificaciones de algoritmos conocidos, enunciados en el Capítulo 2 y en la presentación de otros nuevos. Con el fin de mejorar la función objetivo y obtener soluciones de mejor calidad, estas propuestas se enfocan en mejorar la parte `inter` de la función objetivo, por lo cual se intenta generar soluciones que contengan paquetes con una mayor diversidad entre si.

Una de las propuestas consiste en un conjunto de mejoras aplicadas al algoritmo `Produce-and-Choose`. Para la fase de producción, más específicamente para el algoritmo jerárquico, se plantean dos mejoras. La primera tiene como objetivo reducir el orden de complejidad del algoritmo. La segunda es para equilibrar los valores `inter` e `intra` de los paquetes producidos. En cuanto a la fase de selección, la mejora que se propone es para dar más preponderancia a la parte `inter`.

Por otro lado, se diseñó una heurística (golosa) que construya una solución considerando ambas partes de la función objetivo durante todo el proceso, manteniendo un tiempo de ejecución competitivo con los otros algoritmos.

Finalmente se propone aplicar una heurística de búsqueda tabú para las soluciones generadas por los otros algoritmos. La intención es realizar búsquedas entre las soluciones vecinas con el fin de mejorar la Función Objetivo mediante un proceso que evite caer en máximos locales, y de esta manera obtener una mejor solución.

3.1. Produce-and-Choose

Los algoritmos de la clase `Produce-and-Choose` generan una cierta cantidad de paquetes válidos para luego seleccionar aquellos que maximicen el valor de la función objetivo, los cuales formarán parte de la solución.

Como se mencionó en la Sección 1.3, los parámetros del algoritmo son: el conjunto de ítems I, el atributo complementario α, la función de presupuesto $f : 2^I \to \mathbb{R}$, el límite del presupuesto β, un valor $0 < \gamma < 1 \quad \in \mathbb{R}$ (para ponderar si se quiere paquetes más cohesivos frente a mayor diversidad) y una cantidad k de paquetes a generar.

Algoritmo 1: Produce-and-Choose

Data: $I, \alpha, f, \beta, k, \gamma$
Result: Conjunto válido de k paquetes
1 $cand \leftarrow ProduceBundle(I, \alpha, f, \beta)$
2 $G \leftarrow BuildBundleGraph(cand)$
3 **return** $ChooseBundles(k, \gamma, G)$

La función `ProduceBundle` genera a partir del conjunto de elementos un conjunto de paquetes candidatos.

`BuildBundleGraph` recibe el conjunto de paquetes candidatos, transformándolo en un grafo completo con peso en las aristas y en los vértices. Cada vértice representa un paquete del conjunto de candidatos. El peso del vértice es el valor `intra`, mientras que el peso de las aristas representa el valor `inter` entre los paquetes que corresponden.

Por último la función `ChooseBundles` busca el *subgrafo-k* que maximice el peso de los vértices y aristas. Los vértices del *subgrafo-k* corresponderán a los paquetes pertenecientes a la solución.

Como se puede observar la estructura de esta familia de algoritmos facilita la introducción de mejoras en sus dos fases críticas, en la generación y en la selección de paquetes. En las siguientes secciones se presentarán y analizarán las estrategias propuestas que se aplicaron tanto en la etapa de generación como también en la etapa de selección. De este modo se podrán combinar de la mejor manera, teniendo en cuenta el objetivo de la búsqueda requerida.

3.1.1. Mejoras en la generación de paquetes

La generación de paquetes se puede realizar a través de un proceso de agrupación de un conjunto de objetos que son *parecidos*. La finalidad es que los objetos del grupo sean similares entre sí y diferentes de los restantes grupos. Cuanto mayor sea la similitud **dentro** del conjunto (valor `intra`) y mayor sea la diferencia **entre** conjuntos el agrupamiento será mejor.

La noción de qué es un grupo correctamente constituido no puede ser definida con precisión y es por tal motivo que existen una gran cantidad de algoritmos de agrupamiento [4]. Sin embargo, se puede definir como un conjunto de objetos relacionados entre sí. Luego, para cada problema se deberá definir de que manera se constituye correctamente un grupo.

Para ejemplificar lo anterior, en los veinte puntos de la Figura 3.1 existe más de una forma válida de agruparlos y de acuerdo al contexto en el que se encuentren dependerá cuál de todas ellas es la mejor.

Si se quiere agrupar por proximidad, entonces la estructura más razonable es aquella en la que se visualizan dos grupos. A su vez cada uno de los subgrupos puede generar una nueva partición de acuerdo a la especificidad planteada. Por lo tanto la mejor definición de cómo se debe realizar la agrupación depende del contexto del problema.

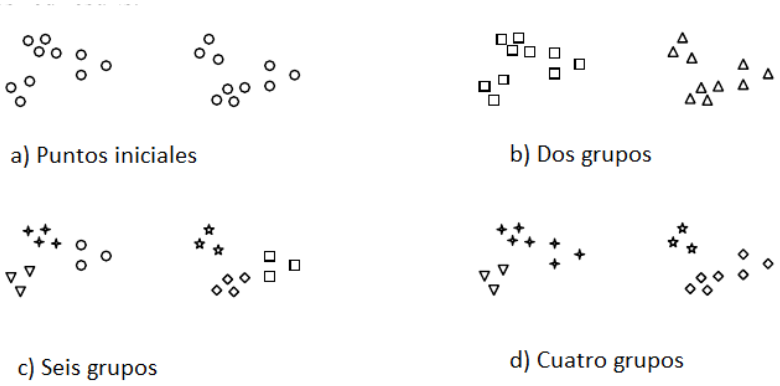


Fig. 3.1

A partir de las necesidades presentadas en este trabajo, se definió que cada grupo con-

tenga la máxima cantidad de ítems sin excederse en el presupuesto inicial. De esta manera se le ofrece al usuario paquetes con la mayor cantidad de ítems posible. La agrupación se realiza por la similitud de los objetos, que se obtiene a través de la **función de similitud**, para que los ítems dentro de cada grupo sean lo más parecidos posible y así obtener paquetes cohesivos.

En la literatura se pueden encontrar diferentes tipos de algoritmos de agrupamiento, la mayoría puede ser categorizado en particionales o jerárquicos [13].

En los métodos de agrupamiento particional se debe definir de antemano la cantidad de grupos finales que se quiere obtener. Luego cada uno de los objetos será asignado al grupo *más cercano* que le corresponda.

A diferencia de los métodos particionales, los jerárquicos construyen grupos mediante la partición recursiva de los grupos de objetos y no requieren predefinir una cantidad de grupos a generar. A su vez estas técnicas se suelen subdividir en dos estrategias: aglomerativas y divisivas, las cuales son detalladas más adelante.

Se plantean dos mejoras para el algoritmo jerárquico *Constrained hierarchical agglomerative clustering* presentado en el artículo [1], que se detallan en las próximas secciones:

1. La elección en cada iteración de la estrategia para combinar *clusters*.

2. La complejidad algorítmica.

El algoritmo *BOBO*, descripto en el Capítulo 2, se mantuvo sin cambios en pos de realizar comparaciones entre los resultados obtenidos a través de las mejoras propuestas y las obtenidas mediante la implementación original.

3.1.1.1. Bundles One-By-One

A continuación se detallan los pasos más importantes del algoritmo *BOBO-k* a partir de su pseudocódigo.

Algoritmo 2: BOBO-k

Data: $I, \alpha, f, \beta, \mu,$ cantidad de paquetes c
Result: Conjunto válido de paquetes

```
1  pivots ← I
2  cand ← ∅
3  while |C| < c and pivots ≠ ∅ do
4  |   pivot ← SelectPivot(pivots)
5  |   bundle ← BuildBundle(pivot, I, α, f, β)
6  |   if Score(bundle) ≥ μ then
7  |   |   cand ← cand ∪ {bundle}
8  |   |   I ← I \ {bundle}
9  |   |   pivots ← pivots \ {pivot}
10 |   else
11 |   |   pivots ← pivots \ {pivot}
12 |   end
13 end
14 return cand
```

La función `selectPivote` selecciona un pivote perteneciente al conjunto de pivotes. En [1, 15] se detalla que la selección se puede realizar de diferentes maneras. En este trabajo se

siguió con la misma implementación utilizada en la publicación recientemente mencionada, que es la selección aleatoria. La función `BuildBundle` es una implementación golosa que genera un paquete a partir de un elemento pivote. Internamente en cada iteración se incluye un ítem del conjunto I que maximiza la función intra y al mismo tiempo cumple con las restricciones de complementariedad y presupuesto. La función `Score` calcula el valor intra del paquete. Se dice que un paquete es suficientemente bueno si el valor intra supera el umbral establecido por μ.

3.1.1.2. Constrained hierarchical agglomerative clustering

Anteriormente se indicó que el agrupamiento jerárquico suele clasificarse en algoritmos *aglomerativo* y *divisivo*.

El aglomerativo comienza con cada objeto perteneciente a un grupo unitario y en cada iteración se unen dos grupos generando un grupo nuevo. Este algoritmo es conocido como *hierarchical agglomerative clustering* (HAC) [1, 15].

La estrategia divisiva, en cambio, comienza con todos los objetos en un mismo grupo y en cada paso se realiza la división de uno de los grupos.

Comúnmente a los algoritmos algomerativos (HAC desde ahora en adelante) se los puede visualizar como un dendrograma, como se ve en la Figura 3.2: en el eje X se encuentran los objetos que van a ser agrupados y en el eje Y se halla la distancia en la que los elementos se unirán. Las uniones de los objetos se representan mediante una línea vertical que comienza a la altura de la distancia en la que los grupos han sido unidos. Por ejemplo, en la figura mencionada, el grupo *4* se une al *5* en la distancia 2 formando el *cluster B*. En un paso posterior éste último se unirá al grupo *3* en una distancia cercana a 3 obteniendo el nuevo *cluster C*. En cada paso se genera una nueva unión hasta que solo quede un único grupo.

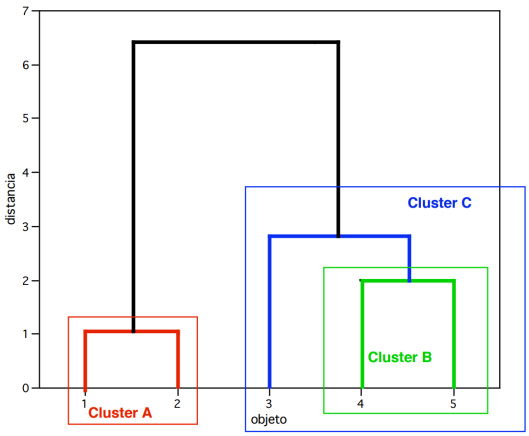


Fig. 3.2: Dendrograma

Cortando el dendrograma mediante líneas horizontales se puede determinar el número de paquetes o *clusters* en el que se divide el conjunto de objetos inicial. En la Figura 3.3 parte a) se puede ver que aplicando un corte en la distancia 5 se obtienen 2 paquetes, en cambio si se aplica a una distancia mayor a 2 se obtienen 3 paquetes, como puede observarse en la parte b) de la misma figura.

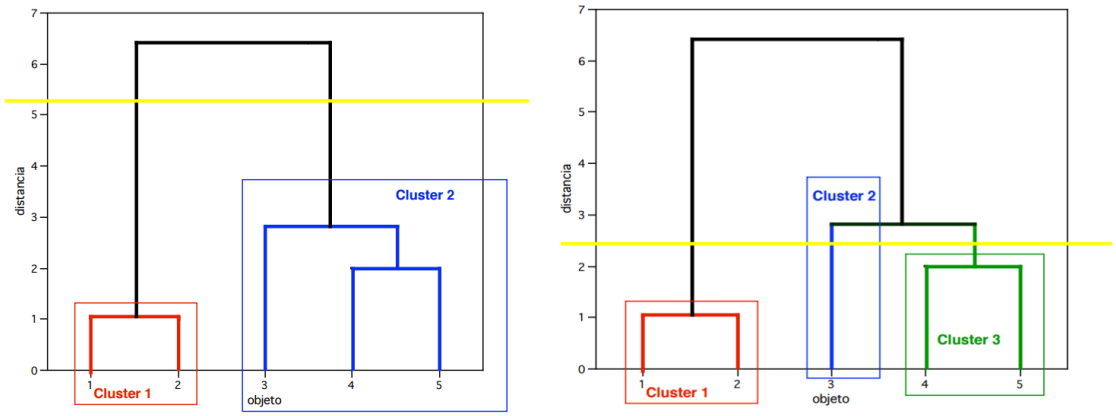


Fig. 3.3: Cortes en el dendrograma

El algoritmo *Constrained hierarchical agglomerative clustering* (C-HAC) introducido en [1], se basa en el algoritmo HAC y tiene la particularidad que cada nuevo grupo que se genera en cada iteración (a partir de la unión de dos grupos) cumple con las restricciones de complementariedad y presupuesto. Por ejemplo, el algoritmo evalúa la unión de los grupos S_1 y S_2. Si el grupo resultante $S_1 \cup S_2$ es inválido, o sea, no cumple con las reglas recientemente mencionadas, entonces el algoritmo no aplica dicho agrupamiento.

Algoritmo 3: C-HAC

Data: $I, \alpha, f, \beta, \gamma,$ cantidad de paquetes c
Result: Conjunto válido de paquetes

```
cand ← {{i} : i ∈ I}
while |cand| > c do
    bestScore ← −∞
    bestCandidate ← ∅
    foreach S_i ∈ cand do
        foreach S_j ∈ cand; S_i ≠ S_j do
            if ValidMerge(S_i, S_j, α, f, β) then
                if Score(S_i ∪ S_j) ≥ bestcore then
                    bestScore ← Score(S_i ∪ S_j)
                    bestCandidate ← {S_i, S_j}
                end
            end
        end
    end
    if bestCandidate = ∅ then
        break
    end
    cand ← cand \ {S} (∀S ∈ bestCandidate)
    cand ← cand ∪ bestCandidate
end
return cand
```

Analizando la complejidad del **algoritmo 3** vemos que el principal ciclo se ejecuta $N - c$ veces, donde N es la cantidad de ítems. En cada paso de la iteración principal se

unen los dos conjuntos con mayor similitud siempre y cuando el conjunto resultante de esa unión sea válido.

Cada uno de los ciclos interiores (*línea 5 y línea 6*) itera sobre los elementos contenidos en el conjunto de grupos candidatos a unir, realizando comparaciones entre todos los grupos existentes hasta el momento.

La función *Score* que se utiliza para decidir qué *clusters* unir, sólo toma en cuenta el valor `intra`, por lo que dicha operación puede ser tratada como constante, ya que se trata de un número fijo de comparaciones. Lo mismo pasa con la función que valida la unión de los conjuntos `ValidMerge`.

Tomando como peor caso que cada iteración tenga N pasos, el orden de complejidad de `C-HAC` es $\mathcal{O}(N^3)$.

Para que la dispersión de los paquetes que se generan en la fase de producción coincida con la esperada por el usuario se decidió en esta tesis, modificar el algoritmo `C-HAC` para incluir el valor `inter` en el cálculo de la selección del par de grupos que se une. De esta manera, al evaluar la unión entre dos grupos se considera tanto el valor `intra` como también así la distancia con el resto de los paquetes del posible nuevo paquete. A diferencia de la función *Score* del **algoritmo 3**, la que solo tenía en cuenta el valor `intra`, se definió la función *Intra-Inter* en la que ambas partes de la función objetivo son tenidas en consideración.

Entonces, para *Intra-Inter* dados dos *clusters* C_i y C_j se define $A(C_i, C_j)$ para representa la similitud entre C_i y C_j y $E(C_i, C_j)$ la inversa de la distancia.

$$A(C_i, C_j) = \sum_{u \in C_i, v \in C_j} s(u, v),$$

$$E(C_i, C_j) = 1 - \max_{u \in C_i, v \in C_j} s(u, v) \qquad \text{y}$$

$$\text{Intra-Inter}(C_i, C_j) = \gamma A(C_i, C_j) + t(1 - \gamma) E(C_i, C_j)$$

Lo que se quiere lograr con la función *Intra-Inter* es balancear entre la similitud y la distancia de los clusters que se analiza. El primer término de la función evalúa que tan similares son los clusters y el segundo término lo contrarresta con la distancia. De esta manera el *cluster* resultante incrementa el valor `inter` y uniéndose dos *clusters* con alta similitud, favoreciendo la dispersión en el *clustering* final. El factor t intenta equilibrar los dos términos de la sumatoria, de lo contrario, el segundo resulta siempre despreciable con respecto al primero. Por lo tanto se decidió que el valor que se le asigna a t corresponde a la cantidad de similitudes que se suman en A, finalmente

$$t = \frac{(\#C_i + \#C_j)(\#C_i + \#C_j - 1)}{2}$$

Si bien el objetivo principal de la modificación propuesta es aumentar la calidad de las soluciones, también se tuvo en cuenta la complejidad temporal del algoritmo. De lo contrario, las mejoras no hubiesen podido ser usadas en las pruebas, que se detallan en los próximos capítulos, por tratarse de instancias con miles de objetos.

Algoritmo 4: Intra-Inter C-HAC

Data: $I, \alpha, f, \beta, \gamma$
Result: Conjunto válido de paquetes

```
1  S ← {{i} : i ∈ I}
2  foreach s_i ∈ S do
3      foreach s_j ∈ S do
4          if validMerge(s_i, s_j, α, f, β) then
5              C[i][j] ← Score(s_i ∪ s_j)
6          else
7              C[i][j] ← −1
8          end
9      end
10     P[i] ← priority queue for C[i] sorted
11     P[i].Delete(C[i][i])            /* se elimina así mismo de la pila */
12 end
13 for n ← 1 to I.length do
14     k_1 ← arg max_k P[k].max()
15     k_2 ← máx P[k].max()
16     if validMerge(k_1, k_2, α, f, β) then
17         S ← S ∪ {k_1 ∪ k_2} \ {k_1} \ {k_2}
18         foreach s_i ∈ S do
19             P[i].Delete(C[i][k_1])
20             P[i].Delete(C[i][k_2])
21             if validMerge(s_i, {k_1 ∪ k_2}, α, f, β) then
22                 C[i][k_1] ← Inter − Intra(s_i, k_1 ∪ k_2, γ)
23                 C[k_1][i] ← Inter − Intra(s_i, k_1 ∪ k_2, γ)
24             else
25                 C[i][k_1] ← −1
26                 C[k_1][i] ← −1
27             end
28             P[i].Insert(C[i][k_1])
29             P[k_1].Insert(C[k_1][i])
30         end
31     end
32 end
33 return S
```

En el **algoritmo 4**, `Intra-Inter C-HAC`, se muestra el procedimiento propuesto en esta tesis para producir paquetes. Este algoritmo mejora la calidad y la complejidad de `C-HAC` incluyendo la información inter-paquete. A continuación se detallan los pasos relevantes del algoritmo y se analiza su complejidad.

`Intra-Inter C-HAC` cuenta principalmente con dos estructuras. La matriz de similitud C y un arreglo de colas de prioridad. Las filas $C[i]$ de la matriz C están ordenadas de forma decreciente en las colas de prioridad $P[i]$.

El algoritmo comienza con la inicialización de la matriz C de dimensión $n\ x\ n$ y con el arreglo de colas de prioridad $P[n]$, donde n es la cantidad de elementos. Como ya se había anticipado `Intra-Inter C-HAC` es un algoritmo jerárquico divisivo, por lo que al inicio cada elemento pertenece a su propio grupo. En la posición i, j, de la matriz C, se almacena el valor de la similitud de los grupos ω_i y ω_j

Luego de la inicialización de las estructuras, `Intra-Inter C-HAC` continúa con un ciclo en el que en cada iteración se obtiene entre todas las colas con $P[k].max()$ el grupo con

mayor grado de similitud. Siendo ω_{k_1} y ω_{k_2} los grupos con mayor similitud, se realiza la unión entre estos y se utiliza ω_{k_1} como la representación de la unión. Para el nuevo ω_{k_1} se calcula la similitud con el resto de los grupos y se actualiza en las colas de similitud.

Retomando las fórmulas presentadas anteriormente, para evitar recalcular en cada iteración los valores ya calculados, se redefinen las fórmulas A y E de forma recursiva. De esta manera para la implementación se mantienen estructuras con esos valores, por lo que la complejidad de la función *Intra-Inter* es constante. A partir de las relaciones entre la iteración r y $r-1$ se tiene que:

$$A^r(C_i \cup C_j, C_s) = A^{r-1}(C_i, C_s) + A^{r-1}(C_j, C_s) \quad \text{y}$$

$$E^r(C_i \cup C_j, C_s) = \text{máx}(E^{r-1}(C_i, C_s), E^{r-1}(C_j, C_s))$$

El ciclo de la inicialización de las estructuras es $\mathcal{O}(n^2)$. Mientras que el ciclo de unión de conjuntos es $\mathcal{O}(n^2 \log n)$ para una implementación de las colas de prioridad en que la inserción y borrado es $\mathcal{O}(\log n)$, resultando en una complejidad final del algoritmo de $\mathcal{O}(n^2 \log n)$.

3.1.2. Mejoras en la selección de paquetes

En *Produce and Choose* luego de la producción de paquetes se realiza la selección del subconjunto de paquetes que formarán parte de la solución. A continuación se detalla el comportamiento de la implementación original de [1] y los cambios propuestos.

El problema puede representarse como un grafo completo con pesos en las aristas y vértices. Los nodos corresponden a los paquetes generados en la primera parte del algoritmo y su peso equivale a su calidad o valor `intra`. Las aristas representan la distancia existente entre los nodos y su peso está definido en el valor `inter` de los paquetes involucrados. Teniendo en cuenta la transformación mencionada se puede interpretar que la solución de encontrar aquellos paquetes que maximicen la función objetivo es equivalente a encontrar un k-subgrafo completo de mayor peso.

La definición formal de encontrar el subgrafo completo de tamaño k de peso máximo entre nodos y vértices es la siguiente: dado el grafo $G = (V, E)$, las funciones de peso $\psi : E \to \Re$ y $\omega : V \to \Re$, el entero $k \leq |V|$ y el real $\gamma \in [0, 1]$, el resultado es un conjunto $V' \subseteq V$ tal que $|V'| = k$ y que maximiza el peso de los nodos y vértices del subgrafo $G' = (V', E')$ ponderado por el parámetro γ. La función a maximizar es la siguiente:

$$\gamma \sum_{v \in V'} \omega(v) + (1 - \gamma) \sum_{(u,v) \in E'} \psi(u, v) \tag{3.1}$$

El problema previamente mencionado puede reducirse a la conocida situación de hallar el k-subgrafo más denso [5] transformando el grafo del problema original a un grafo ponderado, en el que el peso de los vértices está dado por la siguiente función:

$$\omega(u, v) = \frac{\gamma}{2(k-1)}(\omega(u) + \omega(v)) + (1 - \gamma)\psi(u, v) \tag{3.2}$$

En [1] proponen utilizar como heurística el algoritmo 5 para hallar el k-subgrafo de mayor peso de un grafo ponderado. En la línea 1 se definen los pesos de las aristas, entonces en cada iteración se elimina el vértice que contenga la menor suma de pesos de sus aristas (líneas 4-5), hasta que quedarse con un grafo de k nodos.

Algoritmo 5: Selección de paquetes

Data: k, γ, el grafo con peso en los vértices y aristas $G = (V, E)$ donde $\forall S \in V / \omega(S) = \sum_{u,v \in S} s(u, v)$ y $\forall (S_i, S_j) \in E / \psi(S_i, S_j) = 1 - \text{máx}_{u \in S_i, v \in s_j}\, s(u, v)$

Result: Conjunto de k bundles

1 $\omega(u, v) = \frac{\gamma}{2(k-1)}(\omega(u) + \omega(v)) + (1 - \gamma)\psi(u, v)$
2 $S \leftarrow V$
3 **while** $|S| > k$ **do**
4 $\quad u \leftarrow \text{mín}_{u \in S} \sum_{v \in S} \omega(u, v)$
5 $\quad S \leftarrow S \setminus \{u\}$
6 **end**
7 **return** C

La debilidad de este algoritmo reside en que para seleccionar el próximo nodo a remover (paquete) no se considera el *valor* del nodo (valor intra). Entonces se puede dar el caso de que el nodo que se remueve pertenezca una solución de mejor calidad.

Presentamos un ejemplo para la etapa de producción en el cual se puede visualizar la debilidad de la heurística. Se tienen tres paquetes A, B y C, con sus respectivos valores `Intra`: $\omega(A) = 5$, $\omega(B) = 5$ y $\omega(C) = 5$ y el valor `Inter` entre los paquetes: $\psi(A, B) = 0$, $\psi(A, C) = 1$ y $\psi(B, C) = 0$. Bajo estos valores iniciales, se busca un solución que contenga dos paquetes con un $\gamma = 0,1$. Utilizando la definición del algoritmo 5 para obtener los valores de las aristas en el grafo ponderado, se calcula la función $\omega(u, v)$ para cada par de nodos y se obtienen los siguientes valores:

- $\omega(A, B) = 0,45$
- $\omega(A, C) = 1,15$
- $\omega(B, C) = 0,2$

El par de paquetes B, C es el que tiene el menor valor en la función ω, por lo tanto el paquete elegido para ser eliminado es C y la solución resultante contendrá a los paquetes A y B.

El valor de la función objetivo de esta solución generada es $0,9$, mientras que el valor de la función objetivo de la solución que contiene a los paquetes A y C es $1,4$.

A partir de los casos analizados en los cuales el algoritmo 5 no obtenía la mejor solución, se presenta una nueva forma de seleccionar paquetes enfocada en los puntos débiles observados. En la propuesta de esta tesis, a diferencia de algoritmo 5, la solución es generada iterativamente, agregando en cada paso el paquete que maximiza la función objetivo hasta alcanzar la cantidad de paquetes requeridos en la solución.

Utilizando el mismo ejemplo antes mencionado se puede observar que las soluciones posibles en la primera iteración son $S_1^1 = \{A\}$, $S_2^1 = \{B\}$ y $S_3^1 = \{B\}$. Dado que S_1^1 es el candidato con mayor valor de función objetivo, se selecciona este y para la segunda iteración las posibles elecciones son: $S_1^2 = \{A, B\}$, $S_2^2 = \{A, C\}$. Eligiendo finalmente con la solución S_2^2 que, como se vio anteriormente, es la de mayor función objetivo.

Previo a la experimentación del nuevo algoritmo de selección observamos en los casos estudiados que la relación entre la parte `intra` y la `inter` de la función objetivo es considerablemente mayor. Esto se debe a que en un paquete el valor `intra` está dado por la suma de todas las similitudes de los elementos (pertenecientes al paquete) entre sí, mientras que en el valor de la parte `inter` interviene solo una relación de elementos (entre

diferentes paquetes). Además, entre una solución S' que contiene n paquetes y la solución S'' que contiene $n+1$ paquetes, S'' tiene n relaciones inter-paquete más que S'.

Finalmente se decidió introducir un coeficiente **coef** a cada componente de la función objetivo al momento de seleccionar un paquete. El propósito es equilibrar los valores `intra` e `inter` en el instante de elección del paquete que formará parte de la solución. El coeficiente propuesto depende de una constante k, representando la cantidad de paquetes que debe contener la solución final y del tamaño de la solución parcial obtenida en cada paso de la iteración. En la *Selección de Paquetes Proporcional* (algoritmo 6) se puede ver que la única diferencia con el algoritmo 5 se encuentra en la línea 5 con la utilización de la función de objetivo modificada.

Formalmente la función de selección se define de la siguiente manera. Sea R el conjunto de paquetes producidos y $S_i \subseteq R$ el conjunto de paquetes seleccionados en la iteración i, se agrega a la solución el paquete que cumple con:

$$\max_{b \in (R/S_i)} \frac{k}{|S_i|}\gamma \sum_{v \in \{b\} \cup S_i} \omega(v) + \frac{k*(k-1)}{|S_i|*(|S_i|-1)}(1-\gamma) \sum_{v,w \in \{b\} \cup S_i} \psi(v,w) \tag{3.3}$$

Algoritmo 6: Selección de paquetes proporcional

Data: k, γ, el grafo con peso en los vértices y aristas $G = (V, E)$ donde $\forall S \in V / \omega(S) = \sum_{u,v \in S} s(u,v) y \forall (S_i, S_j) \in E / \psi(S_i, S_j) = 1 - \max_{u \in S_i, v \in s_j} s(u,v)$

Result: Conjunto de k bundles

1 $\omega(u,v) = \frac{\gamma}{2(k-1)}(\omega(u) + \omega(v)) + (1-\gamma)\psi(u,v)$
2 $S \leftarrow \emptyset$
3 $R \leftarrow V$
4 **while** $|S| < k$ **do**
5 $\quad c \leftarrow \max_{b \in (R/S)} \frac{k}{|S|}\gamma \sum_{v \in \{b\} \cup S} \omega(v) + \frac{k*(k-1)}{|S|*(|S|-1)}(1-\gamma) \sum_{v,w \in \{b\} \cup S} \psi(v,w)$
6 $\quad S \leftarrow S \cup \{c\}$
7 $\quad R \leftarrow R \setminus \{c\}$
8 **end**
9 **return** S

3.2. Algoritmo goloso

Para el problema de `Recuperación de Ítems Empaquetados` se propone una solución basada en heurísticas golosas, la cual abordará dos temas puntuales:

- Que la solución se construya teniendo la misma consideración sobre las dos partes de la función objetivo -la parte `inter` y la `intra`- durante todo el proceso.
- Que sea aceptable el tiempo de ejecución para grandes cantidades de ítems.

El objetivo del primer punto es proponer una variante a `Produce-and-Choose`. Las soluciones que se generan con `Produce-and-Choose` están más enfocadas en la parte `intra`, porque primero produce paquetes sin considerar la parte `inter` de la función objetivo.

Un algoritmo goloso es un tipo de heurística que construye la solución iterativamente seleccionando en cada paso la mejor opción local, esperando así lograr una solución de buena calidad. En la mayoría de los problemas el algoritmo goloso no encuentra la solución de buena calidad, pero son muy usados por su sencillez y velocidad de ejecución.

Por los motivos mencionados y por las características del algoritmo goloso, se decidió implementar esta heurística para encontrar una solución.

El algoritmo goloso que se propone, comienza con los k paquetes de la solución vacíos, y en cada iteración se agrega un ítem que no pertenece a la solución, ubicándolo en el paquete que maximiza la función objetivo, sin violar las restricciones del problema. El algoritmo finaliza cuando por alguna restricción no es posible agregar más objetos a la solución. Sea I el conjunto de ítems del problema, ω la función objetivo y en el inicio la solución $S_0 = \emptyset$ y $t = 1$ entonces se define $S_t = S_{t-1} \cup \{i\}$ dónde $i \in I$, $i \notin S_{t-1}$ y S_t es una solución válida.

Como se puede ver, esta implementación prioriza la cantidad de elementos de la solución, ante el valor objetivo de la misma. Con esto se quiere decir que en cada paso, si es posible, se agrega un ítem a la solución, pese a que disminuya el valor de la función objetivo. Por lo que un escenario posible es que $\omega(S_{t-1}) > \omega(S_t)$.

Algoritmo 7: Algoritmo heurística golosa

Data: $I, \alpha, f, \beta, k, \gamma$
Result: Conjunto válido de paquetes

```
1  cand ← {S_i : 1 ≤ i ≤ k, S_i = ∅}
2  isComplete ← False
3  while isComplete = False do
4      bestScore ← −∞
5      bestCandidate ← ∅
6      bestBundle ← ∅
7      foreach elem ∈ I do
8          foreach bundle ∈ cand do
9              if validMerge(bundle, {elem}, α, f, β) then
10                 score ← Score((Cand \ {bundle}) ∪ {bundle ∪ {elem}})
11                 if score > bestScore then
12                     bestScore ← score
13                     bestBundle ← bundle
14                     bestCandidate ← elem
15                 end
16             end
17         end
18     end
19     if bestCandidate ≠ ∅ then
20         cand ← (cand \ {bestBundle}) ∪ {bundle ∪ {bestCandidate}}
21         I ← I \ {bestCandidate}
22     else
23         isComplete ← True
24     end
25 end
26 return cand
```

3.3. Búsqueda Tabú

La solución hallada por cualquiera de los algoritmos presentados en las secciones anteriores es un máximo local, entonces se pueden aplicar técnicas para explorar sus soluciones vecinas. Para este trabajo se implementaron distintas variantes de la metaheurística **Tabú Search**

Tabú search [9] es una metaheurística para problemas de optimización que guía a un procedimiento de búsqueda local en la exploración del espacio de soluciones más allá del óptimo local. Permite moverse a una solución aunque no sea tan buena como la actual para poder escapar de óptimos locales y continuar la búsqueda de mejores soluciones. La búsqueda tabú comienza a partir de una solución inicial que puede ser generada aleatoriamente. Para este trabajo la solución inicial siempre es generada por alguno de los algoritmos vistos en las secciones anteriores.

La búsqueda local tradicional parte de una solución inicial s_0, calcula sus soluciones vecinas $N(s_0)$ y escoge una nueva solución s_1 perteneciente a ese vecindario ($N(s_0)$) de acuerdo a un criterio de selección. Es decir, se realiza un movimiento que aplicado a s_0 da como resultado s_1 y este proceso se realiza iterativamente. Es posible definir varios criterios de selección, pero uno de los más simples es tomar la evaluación de la función objetivo.

Al aplicar éstos métodos de búsqueda se pueden presentar dos problemas:

- Iteración indefinida. Para solucionarlo se establecen criterios de parada.
- Generación de un ciclo entre soluciones ya visitadas. Se debe introducir mecanismos para impedir esto.

La búsqueda tabú tiene como característica el uso de memoria adaptativa y exploración sensible según [9]. La noción de memoria adaptativa permite a la búsqueda tabú buscar soluciones vecinas de forma económica y efectiva, ya que las elecciones locales están guiadas por la información recolectada durante la búsqueda. La exploración sensible supone que una mala elección seleccionada estratégicamente produce más información que una buena elección aleatoria.

Para que la búsqueda tabú logre sus objetivos utiliza una serie de estructuras de memoria flexibles, condiciones para elegir estratégicamente un movimiento o no que dependen de una lista tabú (elementos o movimientos prohibidos) y función de aspiración. También cuenta con la facilidad de ser utilizada en casi cualquier tipo de problemas de optimización y puede comenzar con una implementación muy sencilla que con el tiempo evoluciona para incorporar mejoras específicas del problema en cuestión.

En la lista tabú se guardan las soluciones o movimientos que no pueden ser elegidos al momento de seleccionar la solución vecina. Un forma sencilla de construir la lista puede ser guardando el paso inverso al introducido, por ejemplo, si cambié el elemento x por y, entonces guardó como tabú la acción de cambiar y por x durante n iteraciones. En el caso de que un movimiento (o elemento) que pertenece a la lista tabú mejora la solución global o la mejor encontrada hasta el momento, se puede permitir el movimiento (o elemento) aunque no haya expirado su tiempo, estos casos son los que deben ser definidos en la o las funciones de aspiración.

Se implementaron las búsquedas tabú Inter-Paquete e Intra-Paquete. Una busca encontrar una mejor solución entre la actual y los paquetes ya generados. La otra consiste en mejorar los paquetes con los ítems que quedaron fuera de la solución.

3.3.1. Inter-Paquete

La búsqueda `Inter-Paquete` se diseñó especialmente para el algoritmo `Produce and Choose`. La idea surge de aprovechar los paquetes generados que no forman parte de la solución. Es por eso que se plantea un búsqueda tabú que comienza con una solución inicial

obtenida a través de `Produce and Choose` y luego se intenta buscar una mejor con los paquetes que no pertenecen a la instancia inicial.

Para esta implementación se define el conjunto de paquetes B que fueron construidos en la etapa de generación pero no fueron escogidos para formar parte de la solución S. Las instancias vecinas de la solución S, son aquellas que no contienen al paquete con menor valor Inter de S e incluye a un paquete que no pertenece a S. El paquete que se saca de S se agrega a la lista tabú durante una cantidad establecida de iteraciones. Esta lista contiene los paquetes que no pueden ser parte de la próxima solución a visitar. Se elige entonces como criterio de aspiración, la admisión de una solución vecina que contiene un paquete que está en la lista tabú, si mejora la función objetivo con respecto a la mejor solución encontrada hasta el momento.

El esquema del proceso para construir una nueva solución es el siguiente:

1. Se define para todo paquete b, c la función $inter(b, c) = (1 - \max_{\substack{u \in c \\ v \in b}} s(u, v))$
2. $S^* = S$, $\bar{S} = S$, $LT = \emptyset$
3. Mientras no se cumpla criterio de parada hacer:
   - *a*) Identificar el paquete b_r en $\bar{S}$ con menor inter:
   $$b_r = \arg\min_{b_i \in \bar{S}} \sum_{\substack{b_j \in \bar{S} \\ b_j \neq b_i}} (inter(b_i, b_j)) \tag{3.4}$$
   - *b*) Identificar el paquete, b_c, con menor inter en $\bar{S} \setminus \{b_r\}$ utilizando la misma Ecuación 3.4
   - *c*) Se define a δ como un umbral que la función *inter* debe sobrepasar.
   - *d*) Determinar el conjunto de paquetes candidatos C como los paquetes de $B \setminus \bar{S}$ que su valor inter supera a δ respecto a b_c:
   $$C = \bigcup_{b_i \in B \setminus \bar{S}} (inter(b_i, b_c) > \delta)$$
   - *e*) Determinar el mejor paquete de C no prohibido en la lista tabú LT según el criterio:
   $$b = \arg\max_{b_s \in C \setminus LT} \sum_{b_i, b_j \in \bar{S} \setminus \{b_r\} \cup \{b_s\}} (inter(b_i, b_j))$$
   - *f*) Evaluar el criterio de aspiración para los paquetes en $C \cap LT$
   $$b_{tabu} = \arg\max_{b_s \in C \cap LT} w(\bar{S} \setminus \{b_r\} \cup \{b_s\})$$
   - *g*) Si $\max(w(S^*), w(\bar{S} \setminus \{b_r\} \cup \{b\})) < w(\bar{S} \setminus \{b_r\} \cup \{b_{tabu}\})$
     entonces $\bar{S} = \bar{S} \setminus \{b_r\} \cup \{b_{tabu}\}$
     Si no $\bar{S} = \bar{S} \setminus \{b_r\} \cup \{b\}$
   - *h*) Actualizar LT
   - *i*) Si $w(\bar{S}) > w(S^*)$ entonces $S^* = \bar{S}$
4. Retornar S^*

3.3.2. Intra-Paquete

El objetivo de la búsqueda Intra-Paquete es buscar una mejor solución con paquetes más cohesivos. En esta búsqueda los vecinos de la solución S son aquellos a los cuales se les modifica un paquete con alguna de las siguientes acciones: agregar un elemento, quitar un elemento o intercambiar un elemento del paquete con otro elemento que no pertenece a la solución.

Para esta implementación existen tres listas tabú:

1. Una lista de elementos.
2. Una de movimientos.
3. Una de paquetes.

Los elementos que se encuentran en la lista tabú, no pueden ser seleccionados para incluirse en una solución. Los paquetes de la lista son los que no pueden ser seleccionados para generar una nueva solución. La lista de movimientos contiene el par de elementos que se intercambiaron para ir de la solución S a S', a fin de que esos elementos no se vuelvan a intercambiar.

La lista de los movimientos se utiliza para evitar ciclar entre las soluciones. Por ejemplo si las soluciones visitadas son $S, S_1, \ldots, S_n, S$, la lista prohíbe que se vuelvan a intercambiar el mismo par de elementos y así asegura que la próxima no sea S_1.

El criterio de aspiración que se utiliza, es permitir elegir un elemento de la lista tabú para ser intercambiado por un elemento de la instancia actual, sólo si el valor de la función objetivo es mayor que el valor de la mejor solución encontrada hasta al momento.

Desde la solución actual se realiza el movimiento a una nueva solución con los siguientes pasos:

Sea S el conjunto de paquete de la solución e I el conjunto de ítems, el paquete de (1) es $b = \text{mín}_{b_1 \in S} \sum_{v,w \in b_1} s(v,w)$. De b se define el centroide c del paso (2) con $c = \text{máx}_{v \in b} \sum_{w \in b} s(v,w)$. El ítem de (3) se obtiene de $i = \text{mín}_{v \in b}\, s(v,c)$. El ítem para reemplazar a i es $j = \text{máx}_{v \in I \setminus items(S)}\, s(v,c)$.

1. Obtener el paquete, p, menos cohesivo de la solución S.
2. Crear el paquete p' válido agregando a p un elemento que no pertenezca a la solución y no esté en la lista tabú.
3. Crear el paquete p'' válido quitando de p el elemento más alejado del centroide.
4. Crear el paquete p''' válido quitando de p el elemento más alejado del centroide y agregando el elemento que no pertenece a la solución S y no esté en la lista tabú.
5. Crear el paquete p'''' válido quitando de p el elemento más alejado del centroide y agregando el elemento que no pertenece a la solución S
6. Sea $S_i = S \setminus \{p\}$, entonces la solución vecina que se visita es S'
   $S' = \arg\max FO(S_i \cup \{p'\}), FO(S_i \cup \{p''\}), FO(S_i \cup \{p'''\}), FO(S_i \cup \{p''''\})$.

4. EXPERIMENTACIÓN COMPUTACIONAL

4.1. Instancias de pruebas

En esta sección se presentan las consultas utilizadas para evaluar las propuestas algorítmicas introducidas en este trabajo.

Las consultas se hicieron sobre dos bases de datos. Una de ellas, corresponde a la base de datos de artículos provista por [8]. La otra corresponde a atracciones turísticas de Europa obtenida de las pruebas realizadas en [1].

A continuación se describe para cada consulta, realizada sobre las bases de datos, la función de similitud, el atributo utilizado para definir la complementariedad de los elementos y el presupuesto elegidos para las instancias escogidas.

La tesis fue desarrollada en **C++** y para la experimentación se utilizó una máquina Desktop Intel(R) Core(TM) i5-4570T CPU @ 2.90GHz, 5.7G Ram, con DB: 5.5.46-MariaDB-1ubuntu0.14.04.2.

4.1.1. Base de datos de artículos

La base de datos utilizada es la proporcionada por [8]. La misma contiene unos 7800 artículos relacionados con la Ingeniería de Software presentados en diferentes conferencias entre los años 1975 y 2011. Los artículos se encuentran catalogados por autores, tópicos relevantes, conferencia donde fue presentado el trabajo y afiliaciones. Además los artículos están clasificados mediante `topicProfile`. La clasificación es asignada a cada artículo en base a las citas provenientes de otros trabajos publicados en conferencias específicas sobre alguno de los 37 tópicos que fueron tratados en [8]. Como resultado, cada `topicProfile` expresa el porcentaje de relevancia de cada uno de los tópicos encontrados dentro del artículo.

De los 9800 autores contenidos en la base de datos, se tiene la información de la universidad a la que cada uno pertenece y la región donde se encuentra dicha universidad.

El `topicProfile` es lo que permitirá definir la similitud, no sólo entre los artículos, sino también entre los autores y las universidades de la base de datos, de una manera prácticamente directa.

Los criterios de las búsquedas realizadas sobre la base de datos se concibieron a partir de lo que se considera que es de interés general. Por ejemplo, una posible consulta sobre esta base de datos puede ser la búsqueda de artículos sobre núcleos temáticos característicos en las distintas conferencias, de manera que observando un paquete se conozca qué se dice sobre este tema en cada una de las conferencias. En otro escenario, con el propósito de armar paneles de expertos, puede resultar de interés la búsqueda de investigadores que trabajan en tópicos similares con afiliación en distintas universidades. También se puede querer conocer universidades de distintas regiones con grupos de investigación trabajando en tópicos similares.

Para las búsquedas se deben realizar las siguientes definiciones como ya se mencionó en [1]:

- **Similitud**: Función que dado dos ítems devuelve la similitud entre estos.
- **Costo**: Función que dado un ítem devuelve el costo del mismo.

- **Presupuesto**: El presupuesto que se tiene, el cual no puede ser excedido por ningún paquete.
- **Complementariedad**: Propiedad del ítem que es único en cada paquete.

Para todas las búsquedas, sin importar el ítem que sea (artículo, autor o universidad), se definió que el costo de cada ítem sea de una unidad y que el presupuesto para cada búsqueda sea de cinco unidades. La elección del costo de los ítems en una unidad se debe a que el foco principal del trabajo esta centrado en mejorar las relaciones entre los paquetes y no generar funciones de costo sobre los elementos. En consecuencia, todos los paquetes de todos los resultados contienen como máximo cinco ítems. Se tomó esta decisión para que cada paquete contenga como máximo un número fijo de elementos. Además se estableció que sean diez los paquetes devueltos en cada búsqueda. El motivo para tomar esta decisión es que un humano pueda valorizar el resultado propuesto fácilmente. Entonces, de aquí en adelante, para cada criterio de búsqueda se deben definir únicamente la función de similitud y la propiedad de complementariedad. Estos parámetros se fijaron durante todo el experimento con el fin de acotar los casos de estudio.

El `Topic Profile` define el perfil de un artículo asignándole un porcentaje a cada tópico que se hace referencia. Por ejemplo, en el caso del artículo `A Cognitive-Based Mechanism for Constructing Software Inspection Teams` que se encuentra en la base de datos utilizada para las pruebas, el `Topic Profile` se compone por los tópicos REQUIREMENTS, RELIABILITY, TESTING y SOFTWARE QUALITY. El porcentaje de cada uno de estos es 71.43 %, 17.86 %, 7.14 % y 3.57 % respectivamente. Esto significa que el 71.43 % de los trabajos citados en este artículo fueron presentados en conferencias o publicados en revistas vinculadas al área REQUIREMENTS.

El modelo computacional del perfil de cada artículo es un vector cuya dimensión corresponde a la cantidad de tópicos y cada posición representa un tópico diferente. El valor de cada posición del vector es el porcentaje del tópico que le corresponde a ese artículo según el *Topic Profile* de la base de datos. Más adelante se explica cómo estos vectores se utilizan para comparar la similitud entre los artículos.

Para los autores no se cuenta con información más allá de los artículos que escribieron, pero sólo con eso alcanza para poder generar un perfil de autores. Para cada autor se hace la suma vectorial de cada uno de los `Topic Profile` de los artículos en los cuales participó y con eso se obtiene el `Topic Profile de Autores`. Para obtener el perfil de las universidades se aplicó el mismo criterio. Se realiza la suma vectorial de cada uno de los `Topic Profile de Autores` pertenecientes a la misma universidad y así se genera el `Topic Profile de Universidades`. En ambos casos se aplica la normalización sobre los vectores resultantes.

Para clarificar se presenta un ejemplo de los perfiles de los elementos:

Artículo	Topic Profile	Autores
Artículo 1	[0,20; 0,40; 0,40; 0,00]	Autor 1, Autor 2, Autor 3
Artículo 2	[0,30; 0,70; 0,00; 0,00]	Autor 2, Autor 3
Artículo 3	[0,00; 0,10; 0,00; 0,90]	Autor 2
Artículo 4	[0,00; 0,00; 1,00; 0,00]	Autor 1, Autor 3

Autor	Topic Profile	Universidad
Autor 1	[0,14; 0,27; 0,95; 0,00]	Universidad 1
Autor 2	[0,30; 0,74; 0,25; 0,55]	Universidad 2
Autor 3	[0,27; 0,60; 0,76; 0,0]	Universidad 2

Universidad	Topic Profile
Universidad 1	[0,14; 0,27; 0,95; 0,00]
Universidad 2	[0,31; 0,72; 0,54; 0,30]

Para la evaluación de los algoritmos propuestos en esta tesis, se realizaron las siguientes consultas:

1. Artículos con tópicos similares presentados en distintas conferencias.
   - **Similitud**: Función que compara el perfil de cada artículo.
   - **Complementariedad**: Lugar dónde fue presentado.
2. Autores que escribieron artículos con tópicos similares afiliados a universidades distintas.
   - **Similitud**: Función que compara el perfil de los autores.
   - **Complementariedad**: Universidad de pertenencia del autor.
3. Universidades en donde se escribieron artículos de tópicos similares que se encuentran en distintas regiones.
   - **Similitud**: Función que compara el perfil de las universidades.
   - **Complementariedad**: Región de la institución.

Para obtener resultados de mayor calidad, se eliminó de la base de datos aquellos artículos que no contengan la información del autor, de los tópicos (**topic profile**) o del lugar de publicación (**venue**). Quedando, luego de la depuración, 5500 artículos.

4.1.1.1. Función de similitud

La similitud se emplea para comparar dos objetos y determinar qué tan parecidos son entre sí. En este trabajo se definió la similitud entre los objetos de la base de datos de artículos mediante la **similitud coseno**. Esta es una medida de similitud entre dos vectores en un espacio vectorial provisto de un producto escalar que mide el coseno del ángulo comprendido entre ellos.

Entonces se define la función de similitud $S(U_i, V_j)$ para los vectores U_i y V_j a partir del producto escalar

$$\cos(\theta) = \frac{\overrightarrow{U_i}\overrightarrow{V_j}}{\overrightarrow{\|U_i\|}\overrightarrow{\|V_j\|}} \tag{4.1}$$

Para esta instancia los objetos (ahora artículos, autores o universidades) están representados por vectores, donde cada dimensión corresponde a un tópico cuyo valor se corresponde con el valor del tópico del objeto según la base de datos [8]. Por lo tanto

el objeto a se representa con el vector $V_a = [v_1, v_2, ..., v_3]$ que cumple con las siguientes propiedades:

1. $v_i \geq 0$
2. $\sum v_i = 1$

Como los componentes de todos los vectores son mayores o iguales a cero se obtiene que $0 \leq \cos(\theta) \leq 1$, que implica que $S(V_i, V_j) \in [0, 1]$.

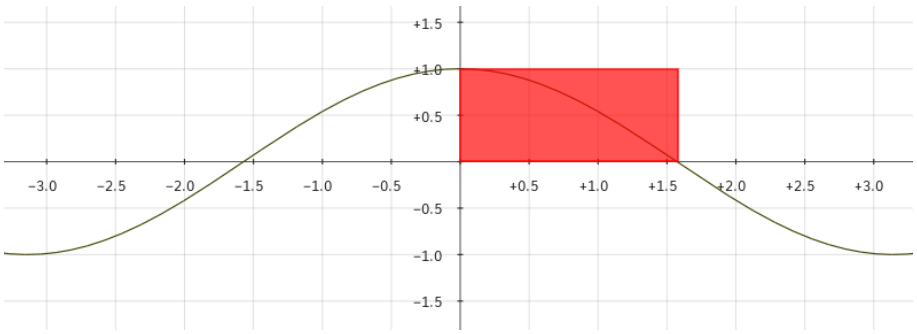


Fig. 4.1: Comportamiento de la función cos. En rojo la región que pertenece a la función de similitud

Con el objetivo de simplificar la ejecución de los algoritmos, considerando que el costo de calcular *cos*() de los vectores es alto, se decidió realizar el cálculo de la similitud de los artículos, autores y universidades previamente a la ejecución de los algoritmos de búsquedas.

4.1.2. Base de dato de atracciones turísticas

Se utilizó una instancia de datos correspondiente a 200 atracciones turísticas de Europa, con datos relevados del sitio *TripAdvisor*. De cada atracción se tiene la información del precio, del tipo (parque, museo, edificio) y de la distancia geográfica con el resto de las atracciones.

El propósito de la búsqueda es darle al usuario distintas opciones de circuitos turísticos que contienen atracciones, con los siguientes requerimientos: evitar realizar largos traslados, variedad en el tipo de atracción y que el costo del circuito no supere el presupuesto del turista. Por lo tanto el modelo de la búsqueda quedó diseñado de la siguiente manera:

- **Similitud**: La inversa de la distancia entre las atracciones.
- **Costo**: Precio de la atracción.
- **Presupuesto**: Presupuesto del turista.
- **Complementariedad**: Tipo de atracción.

Para las pruebas realizadas el presupuesto del turista se estableció en 50 unidades.

4.2. Resultados

En esta sección se analizarán los resultados computacionales comparando la calidad de las soluciones obtenidas por los algoritmos discutidos en Capítulo 3, sobre las bases de datos de Sección 4.1. Con el objetivo de evaluar las propuestas algorítmicas, se consideraron los siguientes métodos.

- $HACS$ PAC(C-HAC / selección simple)
- $BOBS$ PAC(BOBO-10 / selección simple)
- $BOBP$ PAC(BOBO-10 / selección proporcional)
- $BOBP + T$ PAC(BOBO-10 / selección proporcional) + tabú
- $HACP$ PAC(Intra-Inter C-HAC / selección proporcional)
- $HACP + T$ PAC(Intra-Inter C-HAC / selección proporcional) + tabú
- GOL Construcción golosa
- $GOL + T$ Construcción golosa + tabú

Tanto $HACS$ como $BOBS$ son los algoritmos propuestos en [1]. Los demás algoritmos son los propuestos en este trabajo. En `PAC`, la búsqueda tabú `Inter-Paquete` se realiza al finalizar la etapa de producción y la `Intra-Paquete` luego de la selección. En la `Búsqueda Golosa` se intenta mejorar la solución obtenida mediante la búsqueda tabú `Intra-Paquete`. Cabe señalar que no se tienen en consideración `BOBO-Ex` y `CAP`, ya que para el tamaño de la instancia los tiempos de ejecución de esos algoritmos resultaron prohibitivos. Para las búsquedas tabú se definió que la cantidad de iteraciones de permanencia de un elemento en la lista de prohibidos sea la relación entre la cantidad máxima de iteraciones y el tamaño de la solución inicial.

Para realizar una comparación entre la calidad de las soluciones obtenidas por los diferentes algoritmos, se ha evaluado para los $\gamma \in \{0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9\}$ el gap de deterioro de cada solución respecto de la mejor solución obtenida por alguno de los ocho algoritmos.

En el caso de la búsqueda de artículos, que es el escenario que contiene la mayor cantidad de objetos, los tiempos de ejecución para los algoritmos C-HAC ($HACS$, $HACP$ y $HACP + T$), BOBO ($BOBS$, $BOBP$ y $BOBP + T$) y los golosos (GOL y $GOL + T$) son del orden de los 5, 2 y 6 minutos respectivamente. Los incrementos de tiempo debido a la ejecución de las metaheurísticas de mejora son despreciables, están entre los 5 y 7 segundos. Por lo cual no se considera que el tiempo sea un factor que valga la pena analizar.

4.2.1. Análisis de los resultados obtenidos de la base de datos de artículos

Para comprender el comportamiento de los resultados de las búsquedas se diseñaron dos tipos de gráficos que permiten visualizar la cohesión de los paquetes y la dispersión entre ellos. De esta forma se podrá analizar la calidad del resultado obtenido, más allá del valor de la función objetivo.

Los gráficos del estilo de la Figura 4.2 permiten analizar la distribución de los tópicos de una solución a nivel de paquete y de la relación con otros. Las filas corresponden a los 10 paquetes obtenidos y las columnas a los 37 tópicos considerados. El tamaño del círculo representa la proporción del tópico en el perfil del artículo y el color hace referencia al paquete al cual el artículo pertenece. Por lo tanto, dos artículos tendrán gran similitud cuando los patrones de sus círculos coincidan, tanto en tamaño como en distribución. Si para un paquete la distribución entre los tópicos y el tamaño de los círculos es similar entre sus artículos se puede deducir que este paquete es cohesivo (tiene buen valor intra). Por otro lado, si los patrones de los círculos de los dos artículos más similares entre distintos paquetes no coinciden, esto indica que el resultado es diverso.

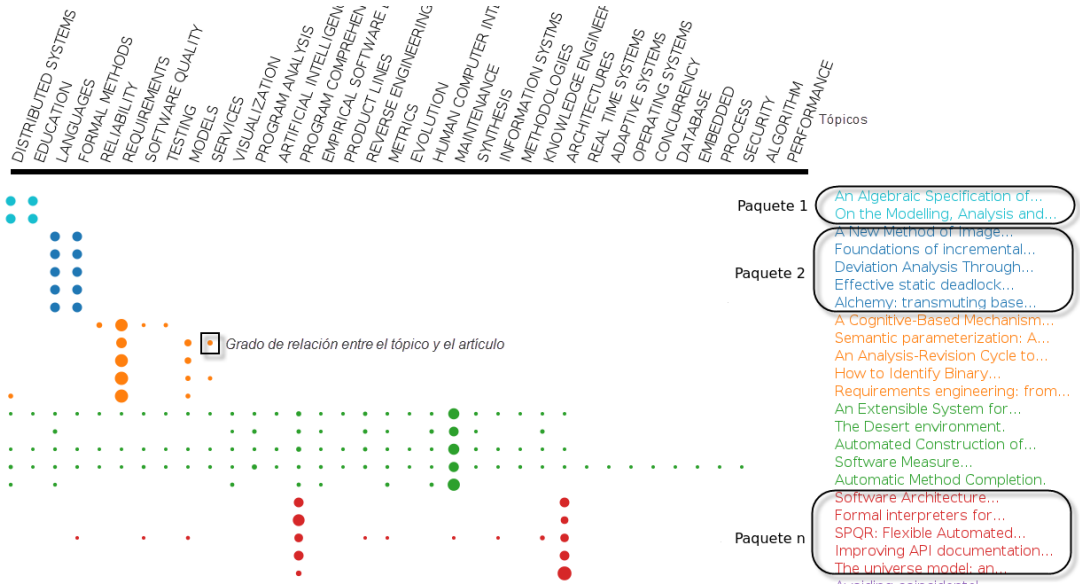


Fig. 4.2

Los gráficos de tipo burbuja de la Figura 4.3 son útiles para concluir el nivel de acoplamiento entre los paquetes de una solución, observando la relación entre los tópicos y los paquetes. Cada burbuja representa un tópico y cada círculo dentro de esa burbuja es un artículo. El tamaño del círculo es la proporción del artículo con el tópico y el color es el paquete al que pertenece. Entonces, si las burbujas contienen círculos de tamaños parecidos de más de un color se puede decir que ese resultado no es muy diverso. Por otro lado, mientras que el color de los círculos de las burbujas sea más homogéneo el resultado será más diverso. En cuanto a la cohesión de los paquetes, es más cohesivo cuando el tamaño de cada círculo dentro de las burbujas es similar (para el mismo color) y cada una de ellas contiene la misma cantidad, o ninguno, de círculos del mismo color.

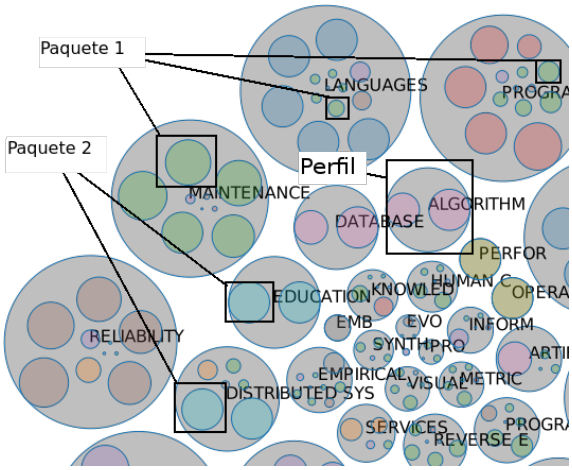


Fig. 4.3

4.2.1.1. Búsqueda de artículos

Para la búsqueda de artículos con tópicos similares en la Tabla 4.1 se comparan los resultados obtenidos de los ocho algoritmos. La solución con mejor resultado para cada γ se representa con un cero y en cada fila se tiene el gap de deterioro de cada solución respecto de la mejor solución obtenida con el resto de los algoritmos. Una primera observación es que los algoritmos `Intra-Inter C-HAC` reflejan el efecto buscado: a menores valores de γ donde el valor inter tiene mayor peso, se obtienen mejores soluciones. Es decir, haber considerado en el proceso de generación de paquetes la función Intra-Inter benefició a la calidad de las soluciones obtenidas. Los algoritmos `BOBO`, no obtuvieron soluciones de la calidad de los algoritmos `C-HAC` y el proceso de selección proporcional no logró una mejora consistente para todos los valores de γ. Las soluciones obtenidas con el algoritmo de construcción golosa, no alcanzaron a mejorar las soluciones de `C-HAC` pero fueron ampliamente mejores que las de `BOBO`. En promedio el gap de deterioro de `BOBO` fue de un 50 % mientras que el goloso fue de un 12 %.

Cabe resaltar el muy buen rendimiento de la búsqueda tabú, tanto en escenarios donde la solución inicial no es de buena calidad (algoritmo `BOBO`) así como también considerando soluciones de mejor calidad (algoritmo Intra-Inter `C-HAC`). En el primer caso, se obtienen porcentajes de mejora por encima del 70 %. En el segundo caso, para varios valores de γ la solución obtenida por la búsqueda tabú resultó ser la mejor opción y en otros con deterioros inferiores al 0,5 %.

Si bien el algoritmo goloso no alcanzó los valores obtenidos por las soluciones generadas por `C-HAC`, tiene como ventaja su fácil y rápida implementación. Sus tiempos de ejecución fueron levemente mayores a los que se obtuvieron con `BOBO` y menores a las ejecutadas por `C-HAC`. Por la forma en la que fue construido siempre genera paquetes completos. Si bien la definición formal del problema no obliga a esto último, se implementó de esta manera ya que a efectos de un usuario final es más interesante obtener paquetes completos, aunque esto signifique sacrificar la búsqueda de la solución óptima.

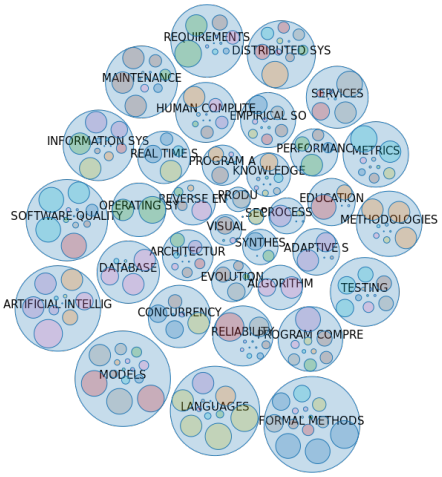
Tab. 4.1: Comparación de calidad de soluciones entre algoritmos para la búsqueda de artículos

γ	$HACS$	$BOBS$	$BOBP$	$BOBP+T$	$HACP$	$HACP+T$	GOL	$GOL+T$
0,1	2,05	32,70	36,18	7,45	0,42	**0,00**	4,53	3,53
0,2	2,11	38,06	41,19	8,23	**0,00**	**0,00**	4,92	3,85
0,3	2,31	45,21	47,35	8,01	**0,00**	**0,00**	9,17	7,66
0,4	0,14	49,08	51,22	15,51	**0,00**	**0,00**	10,40	9,35
0,5	**0,00**	52,35	54,23	17,87	0,31	0,31	12,97	10,38
0,6	**0,00**	55,16	56,04	14,43	0,05	0,05	14,78	13,69
0,7	**0,00**	56,88	56,57	17,02	0,41	0,41	16,21	15,08
0,8	**0,00**	57,86	57,86	16,11	0,56	0,30	18,10	17,60
0,9	**0,00**	58,92	58,92	15,91	0,48	0,35	20,47	17,61

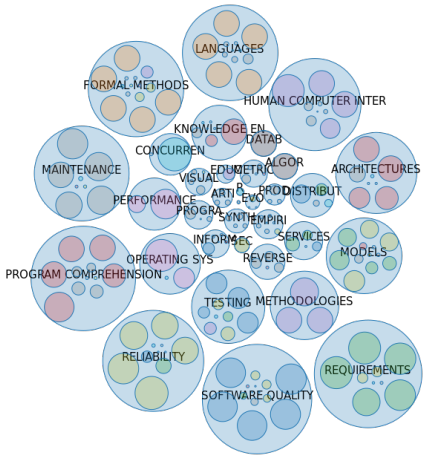
Para comprender la semántica de las soluciones se compararon las soluciones con $\gamma = 0,1$ y $\gamma = 0,9$. En la Figura 4.4 se observa que para $\gamma = 0,1$ los tópicos (representados por burbujas) para la solución de $BOBP$ están presentes en varios paquetes (representados por círculos de colores). En cambio en el $HACP+T$ la mayoría de las burbujas contiene círculos de un solo color. De esta manera se visualiza que la solución de $HACP+T$ es más diversa. En $\gamma = 0,9$ el resultado obtenido con $BOBP$ no se visualiza que cada burbuja contenga cinco círculos del mismo color, en cambio si sucede para la solución de $HACS$. Eso significa que los paquetes de la solución de $HACS$ son más cohesivos, ya que los cinco elementos de cada paquete tienen los mismos tópicos.

$\gamma = 0{,}1$

BOBP ver Figura 6.1

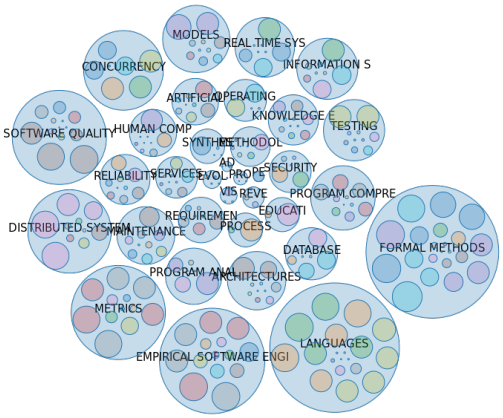


HACP + T ver Figura 6.2



$\gamma = 0{,}9$

BOBP ver Figura 6.3



HACS ver Figura 6.4

Fig. 4.4: Comparación entre las soluciones con menor(izq) y mayor(der) valor de la función objetivos para $\gamma = 0,1$ *y* $\gamma = 0,9$.
En la Sección 6.2 se pueden ver las imágenes con mayor claridad

La búsqueda tabú tiene su mayor impacto cuando es aplicada a la solución brindada por el algoritmo *BOBP*, en la Tabla 4.1 puede apreciarse que en promedio que la combinación de los algoritmos $BOBP + T$ mejora en un 37 % a *BOBP*, resultando en este contexto una buena alternativa por su bajo costo computacional.

En la Figura 4.5 se observa que para $\gamma = 0{,}1$, la solución dada por el *BOBP* tiene artículos de distintos paquetes con patrones muy similares indicando un bajo valor inter-

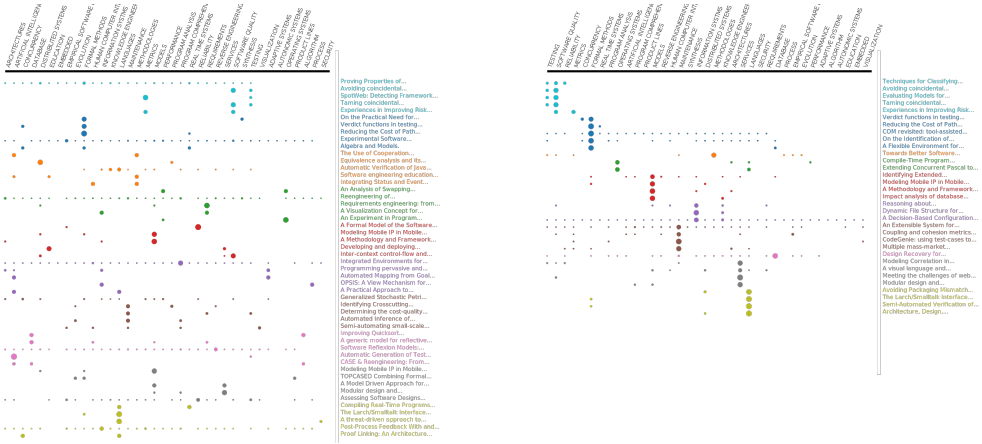
paquete. Por el contrario, luego de aplicar la búsqueda tabú los patrones de los artículos más similares entre distintos paquetes se volvieron más dispares, demostrando el aumento de la diversidad entre paquetes.

Para $\gamma = 0{,}9$ (en Figura 4.5) todos los paquetes de la solución brindada por el algoritmo $BOBP+T$ tienen al menos un artículo cuyo patrón consiste en muchos círculos pequeños distribuidos en la mayoría de los tópicos y el resto de los artículos del mismo paquete están representados con pocos círculos de gran tamaño, demostrando un bajo valor intra-paquete.

Puede observarse que los paquetes obtenidos luego de aplicar la búsqueda tabú son más cohesivos ya que los círculos de los artículos dentro de un mismo paquete siguen patrones más parecidos, con lo cuál se puede afirmar que la búsqueda tabú es capaz de mejorar las características de la solución en función del parámetro γ.

$BOBP$ ver Figura 6.5 | $BOBP+T$ ver Figura 6.6

$\gamma = 0{,}1$



$BOBP$ ver Figura 6.7 | $BOBP+T$ ver Figura 6.8

$\gamma = 0{,}9$

Fig. 4.5: Comparación de soluciones para BOBO-10 con y sin heurística de mejoramiento. En la Sección 6.2 se pueden ver las imágenes con mayor claridad

El usuario debe considerar el contexto de la búsqueda y decidir que tipo de resultados son necesarios o requeridos al momento de iniciar la búsqueda, de ésta manera podrá decidir acerca del valor de γ en pos de priorizar un objetivo sobre el otro. Una curva de Pareto (o de frontera eficiente) sobre diversidad en función de γ puede ayudar a examinar el balance (trade-off) entre los dos objetivos. La intención es que el usuario pueda analizar si una mejora significativa en el valor intra-paquete implica una degradación sustancial en el valor inter-paquetes y viceversa. Para ilustrar este análisis en las figuras 4.6 y 4.7 se comparan las soluciones obtenidas por los algoritmos $HACS$ y $HACP$ variando en pasos de $0,1$. Una primera evaluación muestra que la mayoría de las soluciones provistas por ambos algoritmos son no dominadas, es decir ninguna solución es mejor en ambos objetivos que cualquier otra solución.

El algoritmo $C-HAC$ de [1] ($HACS$) utiliza la función *Score* para decidir el par de *clusters* a unir en la etapa de producción de paquetes y en la selección simple en la segunda fase. Estos dos criterios omiten la diversidad de los paquetes. De acuerdo a los resultados de la Tabla 4.1, se pudo concluir que haber considerado la función Intra-Inter y la selección proporcional benefició la calidad de las soluciones cuando la misma es medida a partir de la función objetivo $w(S)$. Con el fin de evaluar que el $HACP$ es capaz de captar efectivamente la diversidad en la solución, se analizaron las soluciones para $\gamma = 0{,}5$ (todos los valores en la Tabla 6.1 del Apéndice). En este caso $HACS$ obtuvo un valor de $intra = 93,82$ y de $inter = 35,49$, mientras que el $HACP$ logró valores de $intra = 92,94$ y de $inter = 35,96$. A pesar de que la solución de $HACS$ es levemente superior, la solución del $HACP$ aumentó el valor $inter$ el $1,32\,\%$ con un deterioro del valor $intra$ del $0,93\,\%$. Observando las figuras 4.6 y 4.7 se puede ver que la cohesión intra-paquete de ambas soluciones es equivalente. Sin embargo, la diversidad en la segunda solución es mayor, ya que los patrones entre los artículos más similares de distintos paquetes son más heterogéneos.

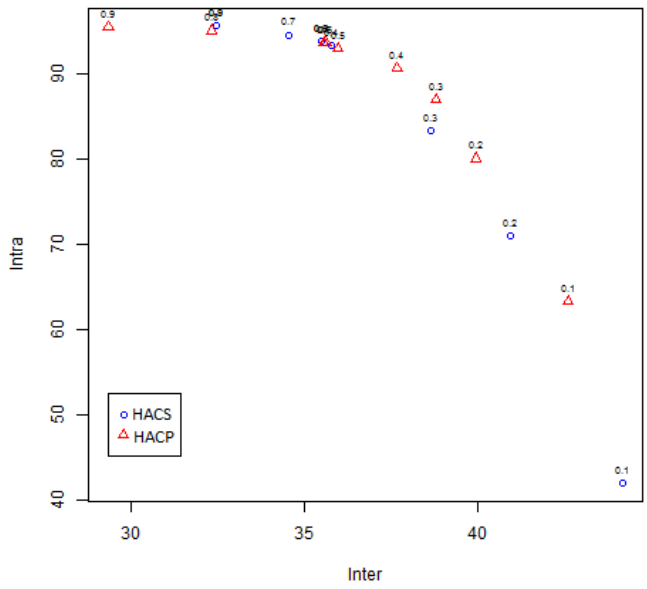


Fig. 4.6: Trade-off entre objetivos para los distintos valores de γ en $HACS$ y $HACP$

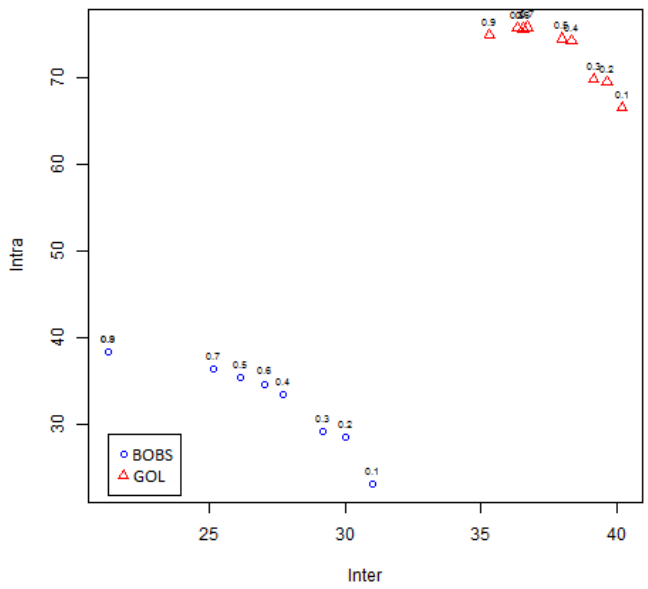


Fig. 4.7: Trade-off entre objetivos para los distintos valores de γ en $BOBS$ y GOL

4.2.1.2. Búsqueda de autores

En la Tabla 4.2 se muestran los gaps de deterioro de cada solución con respecto a la mejor solución obtenida para la búsqueda de "autores que escribieron artículos de tópicos similares que están afiliados a distintas universidades". Puede apreciarse en este caso que $HACP$ con la combinación de la búsqueda tabú halló la mejor solución en todos los casos. Hay que destacar el buen rendimiento que tuvo el algoritmo goloso en combinación con la búsqueda tabú, ya que el deterioro con respecto a la mejor solución fue muy bajo y en algunos casos superando a las soluciones de $HACS$.

Los algoritmos jerárquicos $C-HAC$ e $Inter-IntraC-HAC$ siguen demostrando que obtienen las mejores soluciones en comparación a $BOBO$, sobre todo $HACP+T$ que realiza una búsqueda tabú. Éstas búsquedas vuelven a indicar que las soluciones pueden ser mejoradas en porcentajes muy significativos. Como ocurrió en el caso del algoritmo $BOBP$, mejorando las soluciones en un promedio cercano al 30 % y también para $HACP$ y GOL que en todos los casos mejoraron sus valores iniciales. Con respecto a la estrategia de selección proporcional aplicada al algoritmo $BOBO$ no se observaron mejoras, aunque las soluciones obtenidas fueron cercanas en términos de función objetivo.

Tab. 4.2: Comparación de calidad de soluciones entre algoritmos para la búsqueda de autores

γ	$HACS$	$BOBS$	$BOBP$	$BOBP+T$	$HACP$	$HACP+T$	GOL	$GOL+T$
0,1	0,33	21,59	26,05	1,74	0,13	**0,00**	0,61	**0,00**
0,2	0,63	27,46	29,71	0,52	0,36	**0,00**	1,10	0,25
0,3	0,44	30,57	32,47	0,20	0,53	**0,00**	1,50	0,34
0,4	0,25	32,63	34,29	0,32	0,25	**0,00**	1,88	0,73
0,5	0,22	34,42	35,84	0,04	**0,00**	**0,00**	2,15	0,79
0,6	0,18	35,86	37,05	2,01	**0,00**	**0,00**	2,45	1,39
0,7	**0,00**	37,10	37,93	1,71	0,12	**0,00**	2,59	0,96
0,8	**0,00**	38,19	38,70	1,44	0,08	**0,00**	2,72	1,20
0,9	0,03	39,15	39,38	1,21	0,10	**0,00**	3,35	1,72

Para analizar el trade-off entre la parte inter y la intra en la Figura 4.8 se muestra los valores inter e intra de las soluciones obtenidas por los algoritmos $HACS$, $BOBS$, $HACP$ y GOL para todos los γ. Puede apreciarse, como es de esperar, que los valores del $BOBS$ son significativamente inferiores al resto de los algoritmos. En cambio los valores de $HACS$, $HACP$ y GOL se concentran en una región reducida, ya que sus soluciones fueron muy similares respecto al valor de la función objetivo.

En la Figura 4.9 se encuentran los valores inter e intra de los algoritmos $HACS$ y $HACP$. Puede verse como los valores de las soluciones obtenidas con $HACP$ no están dominadas por la parte inter y si más concentradas en la parte intra, a diferencia de los que ocurre con $HACS$.

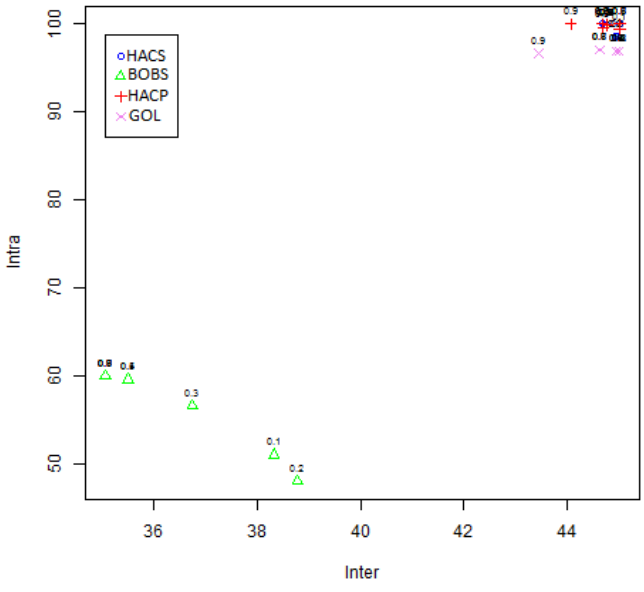


Fig. 4.8

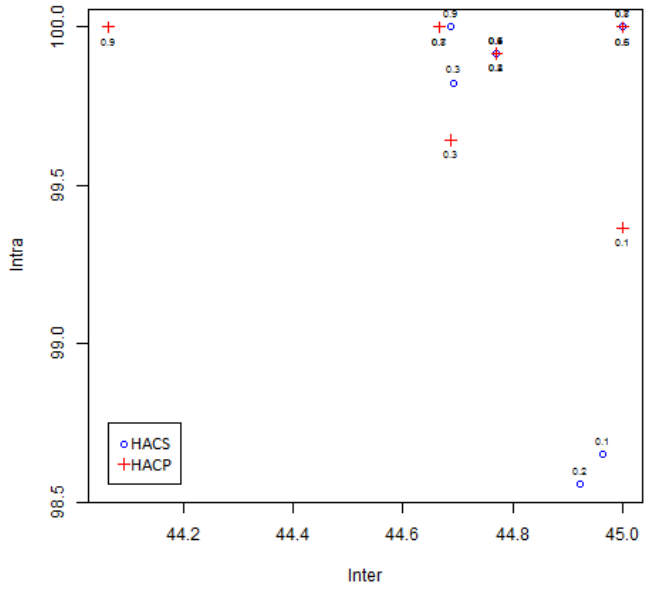


Fig. 4.9

En la Figura 4.10 de la solución generada por el algoritmo $HACP + T$ para $\gamma = 0{,}1$, se tienen las siguientes observaciones:

1. Todos los paquetes contienen autores que pertenecen a los mismos tópicos.
2. No existe un tópico que esté presente en más de un paquete.
3. Todos los paquetes utilizan el máximo del presupuesto.

Por (1) y (2) la calidad intra e inter paquete es máxima respectivamente. Con (3) se cumple que la solución obtenida es la de máxima calidad para todo γ. Para todas las búsquedas realizadas, las soluciones que se obtuvieron cumplían con las observaciones mencionadas.

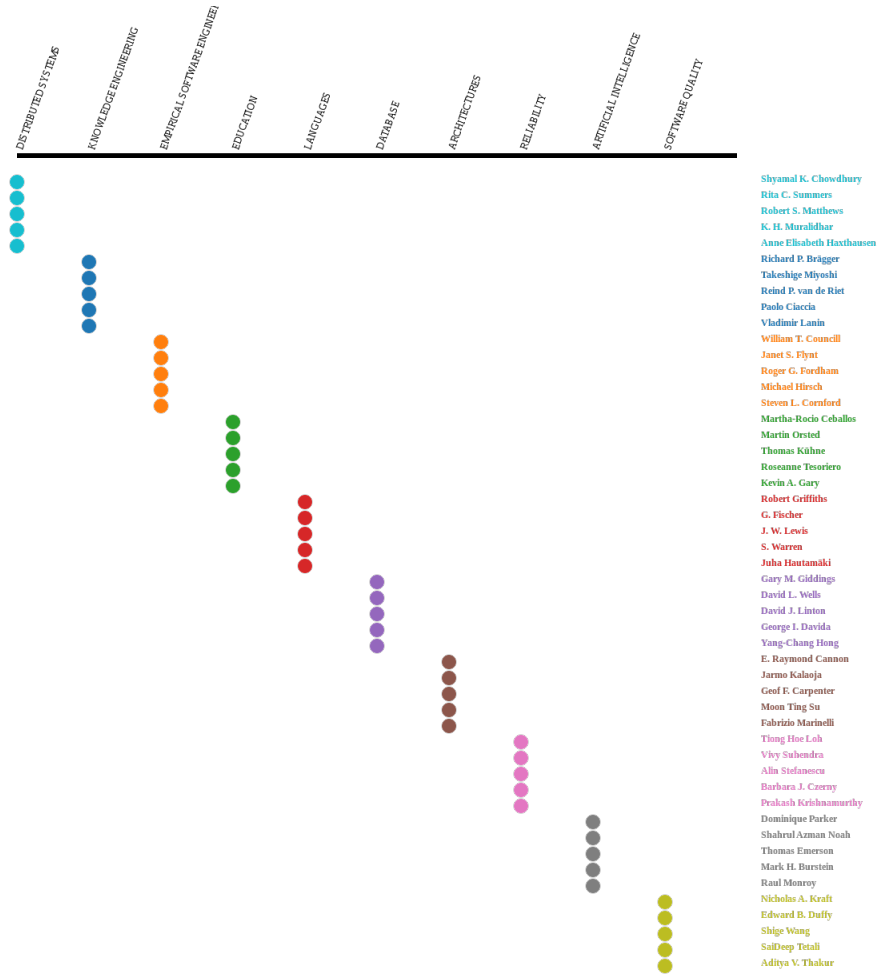


Fig. 4.10

4.2.1.3. Búsqueda de universidades

En este escenario lo que más se destaca de la Tabla 4.3 es el comportamiento del algoritmo $HACS$ que generó una mejor solución para los valores de $\gamma = 0{,}1$ y $\gamma = 0{,}2$ con respecto al algoritmo $HACP + T$. En el resto de las soluciones obtenidas (de $HACS$ y $HACP + T$) se aprecia un deterioro cada vez más significativo a medida que crece el valor del γ.

Por otro lado la función objetivo de las soluciones provista por los algoritmos $BOBS$, $BOBP$ y GOL se encuentran muy alejadas de las soluciones provistas por los jerárquicos. En el caso del algoritmo goloso GOL las soluciones mejoran con respecto a los algoritmos $BOBS$ y $BOBP$ cuando el valor de γ disminuye.

Al igual que en el resto, la búsqueda tabú mejoró las soluciones iniciales del `BOBO` y del algoritmo goloso considerablemente. Se destaca que para las soluciones jerárquicas la búsqueda tabú siempre mejora la solución inicial sin importar que tan buena sea.

Tab. 4.3: Comparación de calidad de soluciones entre algoritmos para la búsqueda de universidades

γ	$HACS$	$BOBS$	$BOBP$	$BOBP+T$	$HACP$	$HACP+T$	GOL	$GOL+T$
0,1	**0,00**	30,89	31,26	15,05	1,39	1,33	10,97	9,73
0,2	**0,00**	40,35	40,16	22,09	1,72	1,07	22,83	20,42
0,3	2,22	50,85	49,98	27,01	2,37	**0,00**	36,06	28,55
0,4	5,72	60,65	58,96	33,76	1,36	**0,00**	48,44	30,38
0,5	8,59	69,76	67,66	31,46	1,92	**0,00**	60,18	34,32
0,6	12,09	77,50	74,71	33,85	2,53	**0,00**	70,16	35,80
0,7	12,59	82,97	80,09	31,52	**0,00**	**0,00**	78,04	29,36
0,8	15,52	88,12	84,93	32,90	**0,00**	**0,00**	85,63	36,34
0,9	17,46	88,10	87,79	31,16	**0,00**	**0,00**	92,13	28,26

4.2.2. Análisis de los resultados obtenidos de la base de datos de atracciones turísticas

En las búsquedas realizadas sobre la base de datos de atracciones turísticas las soluciones obtenidas con las modificaciones propuestas en este trabajo sobre los algoritmos `PAC` mejoran a los algoritmos originales en todos los casos, en la Tabla 4.4 puede verse la comparación. Es para destacar que en este escenario el algoritmo goloso supera a $HACS$ no sólo para algunos valores de γ sino para todos. La combinación de los algoritmos `PAC` con la búsqueda tabú consiguió mejores resultados. Por otro lado, en el algoritmo goloso (GOL) el uso de la búsqueda tabú no mejoró las soluciones en contraste a lo sucedido en las demás consultas.

Tab. 4.4: Comparación de calidad de soluciones entre algoritmos para la búsqueda de atracciones turísticas

γ	$HACS$	$BOBS$	$BOBP$	$BOBP+T$	$HACP$	$HACP+T$	GOL	$GOL+T$
0,1	6,49	22,89	22,30	9,99	0,05	**0,00**	0,52	0,52
0,2	13,50	27,60	26,57	16,51	0,09	**0,00**	1,09	1,09
0,3	21,10	32,76	29,73	20,58	0,15	**0,00**	1,70	1,70
0,4	29,38	36,68	33,51	25,10	0,20	**0,00**	2,37	2,37
0,5	38,43	42,05	38,44	30,94	0,27	**0,00**	3,10	3,10
0,6	48,35	47,73	43,86	34,75	0,34	**0,00**	3,90	3,90
0,7	59,29	51,47	50,54	43,46	0,41	**0,00**	4,78	4,78
0,8	71,36	57,07	56,14	45,61	**0,00**	**0,00**	5,62	5,62
0,9	84,97	60,59	59,81	44,04	**0,00**	**0,00**	7,33	7,33

4.2.3. Análisis general de los resultados obtenidos

Se obtuvieron mejores resultados en términos de los valores de la función objetivo, así como en la sensibilidad al criterio de valorización de cada objetivo sin perjuicio en el tiempo de ejecución. En el caso de los algoritmos `PAC` usando generación de paquetes mediante las técnicas `BOBO` y aplicando la búsqueda tabú el resultado mejoró considerablemente, como puede verse en Tabla 4.1 y Tabla 4.2.

Se observó que las soluciones obtenidas utilizando el algoritmo goloso fueron mejores en comparación a las generadas por la metodología `PAC`, solo cuando en la etapa de producción se utilizó la estrategia `BOBO`. Por el contrario, la producción jerárquica derivó en

la obtención final de mejores paquetes. Si bien la técnica de las implementaciones de `BOBO` son golosas, éstas solo hacen hincapié en los valores intra de la función objetivo. Mientras qué la nueva propuesta lo hace sobre la totalidad de la función objetivo.

Para la estrategia `PAC` las implementaciones de `BOBO` fueron en todos los casos las más rápidas, resultando ser las ideales para instancias significativamente grandes. La solución obtenida mejora considerablemente si se le aplica la búsqueda tabú. En las pruebas realizadas con `HAC` con un universo de elementos menor a los 10000 se cuadruplican o quintuplican los tiempos de ejecución debido a su complejidad ($\mathcal{O}(N^2 \log n)$). Sin embargo en instancias más chicas como las atracciones turísticas los tiempos fueron muy cercanos entre ellos.

Las soluciones obtenidas usando `Efficient-HAC` y búsquedas golosas para valores de γ menores a 0,5 fueron "similarmente buenas", dado que los resultados en términos de función objetivo fueron cercanos. En cambio para γ más cercanos a uno se observó que el valor de la función objetivo de las soluciones `HAC` resultaron significativamente mayores. En todos los casos se identificó que las relaciones inter-paquetes son mejores en las soluciones del algoritmo goloso, logrando soluciones más interdependientes.

A partir de una experimentación preliminar con `BOBO` para valores $c = 1, 5$ y 10, resultó $c = 10$ la opción más competitiva. Para los valores $c > 10$ se realizaron pruebas y los valores obtenidos no mejoraron significativamente las soluciones en relación al incremento del tiempo requerido para su ejecución. Igualmente se compararon los resultados entre $BOBO - 160$ y `HAC`. La experimentación se debe a que la cantidad máxima de paquetes que genera el `HAC` es cercano a 1600 y los que genera $BOBO - 160$ antes de la selección también. Se esperaba tener unos valores más cercanos entre sí, pero sucedió que a pesar de generar más paquetes los resultados fueron levemente mejor que la variante $BOBO-10$. Se observa que la diferencia entre los valores de la función objetivo de cada solución aumenta a medida que γ se acerca a 1. Se supone que esto se debe a la estrategia `Produce and Choose`.

Comparando los resultados obtenidos al realizar la selección de a un candidato contra la selección de a pares se obtuvo que para `HAC` los tiempos aumentaron a 40 minutos y para $BOBO - 10$ de 2 minutos. En cuanto al valor de la función objetivo el único beneficiado fue $BOBO - 10$ ya que para `HAC` no mejoró pese al incremento del tiempo de ejecución.

5. CONCLUSIONES

La búsqueda y recuperación de información es una de las aplicaciones más utilizadas en Internet. El estudio de esta problemática y la propuesta de nuevos modelos para satisfacer los distintos requerimientos de los usuarios es una necesidad y una importante motivación para la investigación. En este trabajo se propusieron significativas mejoras sobre los algoritmos propuestos en *Composite Retrieval of Diverse and Complementary Bundles* y se incluyó una nueva forma de hallar soluciones (búsqueda golosa) y dos estrategias (búsquedas tabú) que colaboran con las técnicas de búsquedas aquí presentadas.

A la luz de los resultados observados en el capítulo anterior, se puede concluir que los cambios propuestos en los algoritmos del tipo PAC fueron exitosos en términos de los valores de la función objetivo. En la mayoría de los escenarios el valor de la función objetivo de la solución fue superior al de las soluciones obtenidas con algoritmos ya existentes, sólo en algunos pocos casos esto no ocurrió.

En las etapas de producción y selección de paquetes, la elección de utilizar las dos medidas `inter` e `intra` resultó ser acertada sobre todo cuando los valores de γ están cercanos al cero. Como se ve en la Ecuación 1.1 en los casos en los cuales el valor `inter` tiene más peso es justamente con γ tendiendo a cero.

El uso de las estrategias tabú resultó ser una opción muy adecuada para todos los tipos de búsquedas realizadas en el desarrollo de esta tesis, sobre todo por la facilidad de poder usarse a partir de cualquier solución inicial sin importar la estrategia elegida al inicio de la pesquisa.

En cuanto a la nueva heurística de la familia de algoritmos golosos, tiene la ventaja de contar con implementaciones simples y fácilmente adaptables a los cambios. Es una alternativa muy interesante cuando se prefiere mejorar el tiempo de respuesta aceptando un deterioro en la calidad de la solución. Mejor aún, si se lo combina con una búsqueda tabú, que probó ser un complemento muy útil, mejorando en la mayoría de los casos la calidad de la solución final, sin pérdida de rendimiento.

5.1. Trabajo futuro

Como parte final del trabajo, se describen los temas que quedaron abiertos para trabajos futuros.

Uno de los principales temas es que el algoritmo pueda generar soluciones en el que un ítem esté presente en todos los paquetes. A pesar de que no es parte del problema, sería interesante que cada paquete haga un cubrimiento de un tema particular. Por ejemplo, el escenario en cual un cliente visita una tienda de discos virtual (presentado en la introducción), si un mismo disco se encuentra en todos los paquetes, entonces cada paquete será «similar» a este disco. De esta forma, se puede personalizar el resultado con el perfil del cliente.

Un punto en el que se deberá trabajar en futuros desarrollos es el de la eficiencia algorítmica. Si bien en este trabajo se mejoró la complejidad del algoritmo de partición jerárquico, aún esta pendiente encontrar una solución más escalable. Por lo que teniendo en cuenta las estructuras de los algoritmos PAC y de las búsquedas tabú descriptos en este trabajo, se supone que es posible adaptarlos para que se ejecuten en paralelo. De esta

forma se podría optimizar el tiempo de ejecución y realizar búsquedas en escenarios con más elementos.

6. APÉNDICE

6.1. Tablas de resultados

Tab. 6.1: Valores la parte Intra e Inter de las soluciones para los algoritmos HACS, HACSP, BOBS y GOL

γ	Algoritmo	Valores Intra paquete	Valores Inter paquetes
$0,1$	HACS	$41,93$	$44,17$
	HACSP	$63,27$	$42,62$
	BOBS	$23,08$	$30,99$
	GOL	$66,6$	$40,19$
$0,2$	HACS	$71,04$	$40,95$
	HACSP	$80,1$	$39,95$
	BOBS	$28,46$	$30,03$
	GOL	$69,58$	$39,63$
$0,3$	HACS	$83,27$	$38,63$
	HACSP	$86,99$	$38,79$
	BOBS	$29,19$	$29,18$
	GOL	$69,9$	$39,15$
$0,4$	HACS	$93,3$	$35,78$
	HACSP	$90,67$	$37,67$
	BOBS	$33,41$	$27,69$
	GOL	$74,39$	$38,32$
$0,5$	HACS	$93,82$	$35,49$
	HACSP	$92,94$	$35,97$
	BOBS	$35,45$	$26,17$
	GOL	$74,57$	$37,97$
$0,6$	HACS	$93,82$	$35,49$
	HACSP	$93,67$	$35,62$
	BOBS	$34,66$	$27,03$
	GOL	$75,76$	$36,55$
$0,7$	HACS	$94,58$	$34,56$
	HACSP	$93,7$	$35,56$
	BOBS	$36,37$	$25,18$
	GOL	$75,92$	$36,7$
$0,8$	HACS	$95,61$	$32,48$
	HACSP	$95,07$	$32,33$
	BOBS	$38,39$	$21,29$
	GOL	$75,87$	$36,34$
$0,9$	HACS	$95,61$	$32,48$
	HACSP	$95,48$	$29,35$
	BOBS	$38,39$	$21,29$
	GOL	$74,99$	$35,31$

6.2. Imágenes en tamaño original

$BOBP$ con $\gamma = 0{,}1$

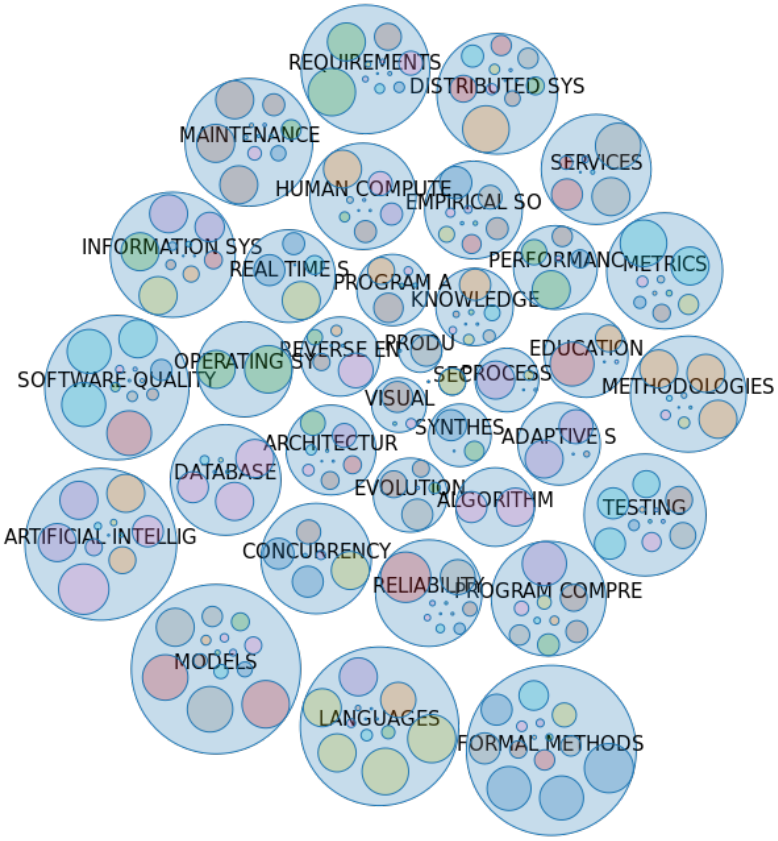


Fig. 6.1

$HACP + T$ con $\gamma = 0{,}1$

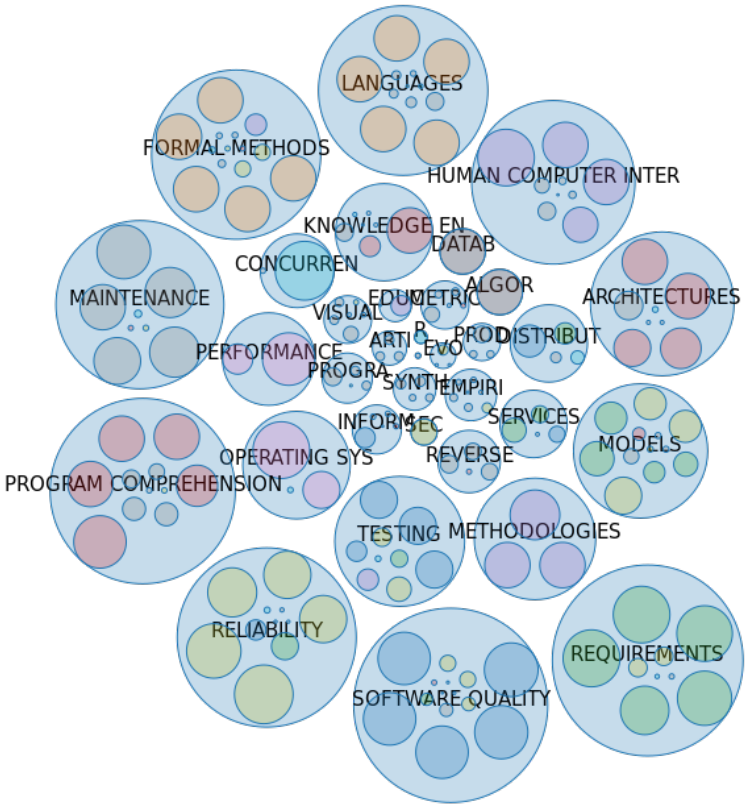


Fig. 6.2

$BOBP$ con $\gamma = 0{,}9$

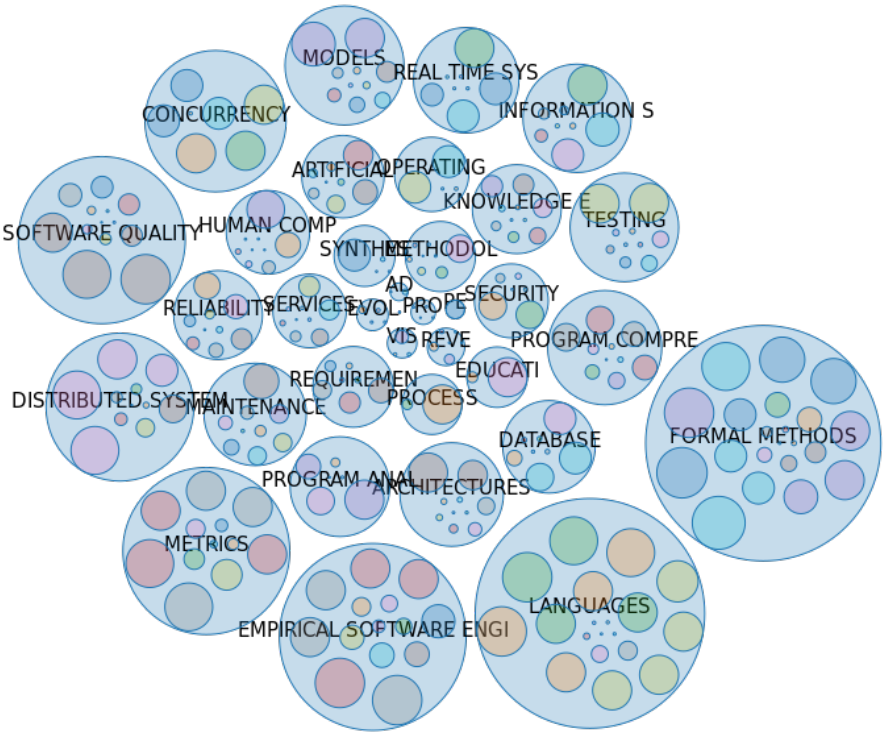


Fig. 6.3

$HACS$ con $\gamma = 0{,}9$

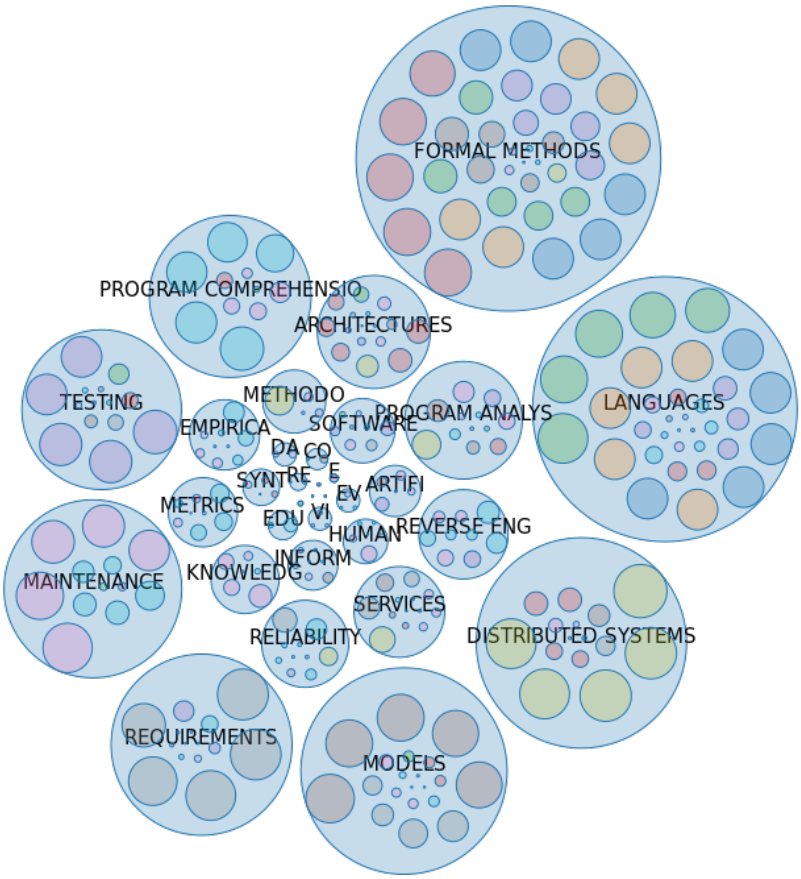


Fig. 6.4

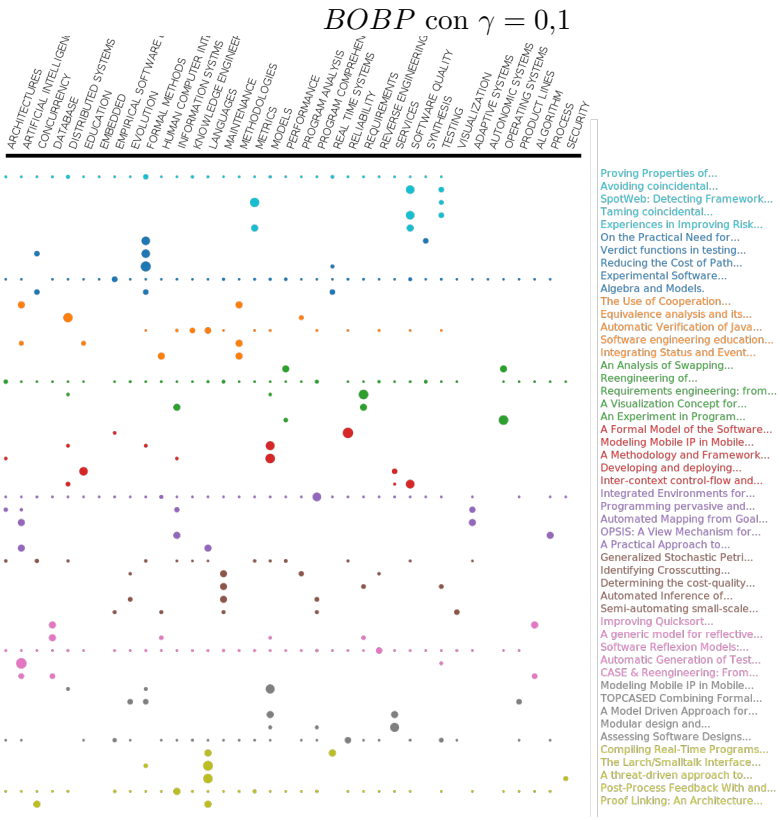

$BOBP$ con $\gamma = 0{,}1$
ARCHITECTURES
ARTIFICIAL INTELLIGEN
CONCURRENCY
DATABASE
DISTRIBUTED SYSTEMS
EDUCATION
EMBEDDED
EMPIRICAL SOFTWARE
EVOLUTION
FORMAL METHODS
HUMAN COMPUTER INT
INFORMATION SYSTEMS
KNOWLEDGE ENGINEE
LANGUAGES
MAINTENANCE
METHODOLOGIES
METRICS
MODELS
PERFORMANCE
PROGRAM ANALYSIS
PROGRAM COMPREHEN
REAL TIME SYSTEMS
RELIABILITY
REQUIREMENTS
REVERSE ENGINEERING
SERVICES
SOFTWARE QUALITY
SYNTHESIS
TESTING
VISUALIZATION
ADAPTIVE SYSTEMS
AUTONOMIC SYSTEMS
OPERATING SYSTEMS
PRODUCT LINES
ALGORITHM
PROCESS
SECURITY
Proving Properties of...
Avoiding coincidental...
SpotWeb: Detecting Framework...
Taming coincidental...
Experiences in Improving Risk...
On the Practical Need for...
Verdict functions in testing...
Reducing the Cost of Path...
Experimental Software...
Algebra and Models
The Use of Cooperation...
Equivalence analysis and its...
Automatic Verification of Java...
Software engineering education...
Integrating Status and Event...
An Analysis of Swapping...
Reengineering of...
Requirements engineering: from...
A Visualization Concept for...
An Experiment in Program...
A Formal Model of the Software...
Modeling Mobile IP in Mobile...
A Methodology and Framework...
Developing and deploying...
Inter-context control-flow and...
Integrated Environments for...
Programming pervasive and...
Automated Mapping from Goal...
OPSIS: A View Mechanism for...
A Practical Approach to...
Generalized Stochastic Petri...
Identifying Crosscutting...
Determining the cost-quality...
Automated Inference of...
Semi-automating small-scale...
Improving Quicksort...
A generic model for reflective...
Software Reflexion Models:...
Automatic Generation of Test...
CASE & Reengineering: From...
Modeling Mobile IP in Mobile...
TOPCASED Combining Formal...
A Model Driven Approach for...
Modular design and...
Assessing Software Designs...
Compiling Real-Time Programs...
The Larch/Smalltalk Interface...
A threat-driven approach to...
Post-Process Feedback With and...
Proof Linking: An Architecture...

Fig. 6.5

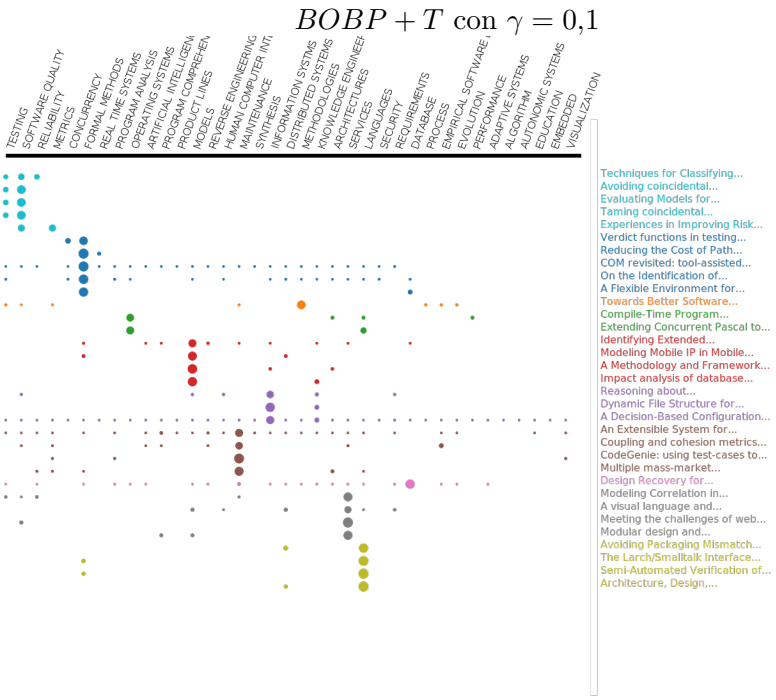

$BOBP + T$ con $\gamma = 0{,}1$
TESTING
SOFTWARE QUALITY
RELIABILITY
METRICS
CONCURRENCY
FORMAL METHODS
REAL TIME SYSTEMS
PROGRAM ANALYSIS
OPERATING SYSTEMS
ARTIFICIAL INTELLIGEN
PROGRAM COMPREHEN
PRODUCT LINES
MODELS
REVERSE ENGINEERING
HUMAN COMPUTER INT
MAINTENANCE
SYNTHESIS
INFORMATION SYSTMS
DISTRIBUTED SYSTEMS
METHODOLOGIES
KNOWLEDGE ENGINEE
ARCHITECTURES
SERVICES
LANGUAGES
SECURITY
REQUIREMENTS
DATABASE
PROCESS
EMPIRICAL SOFTWARE
EVOLUTION
PERFORMANCE
ADAPTIVE SYSTEMS
ALGORITHM
AUTONOMIC SYSTEMS
EDUCATION
EMBEDDED
VISUALIZATION
Techniques for Classifying...
Avoiding coincidental...
Evaluating Models for...
Taming coincidental...
Experiences in Improving Risk...
Verdict functions in testing...
Reducing the Cost of Path...
COM revisited: tool-assisted...
On the identification of...
A Flexible Environment for...
Towards Better Software...
Compile-Time Program...
Extending Concurrent Pascal to...
Identifying Extended...
Modeling Mobile IP in Mobile...
A Methodology and Framework...
Impact analysis of database...
Reasoning about...
Dynamic File Structure for...
A Decision-Based Configuration...
An Extensible System for...
Coupling and cohesion metrics...
CodeGenie: using test-cases to...
Multiple mass-market...
Design Recovery for...
Modeling Correlation in...
A visual language and...
Meeting the challenges of web...
Modular design and...
Avoiding Packaging Mismatch...
The Larch/Smalltalk Interface...
Semi-Automated Verification of...
Architecture, Design,...

Fig. 6.6

$BOBOP$ con $\gamma = 0{,}9$

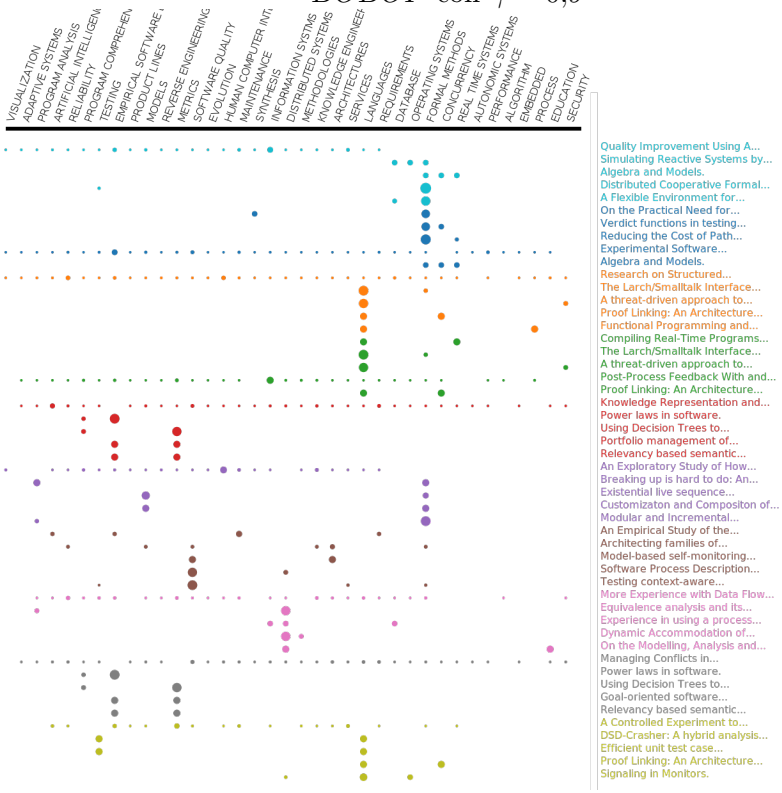


Fig. 6.7

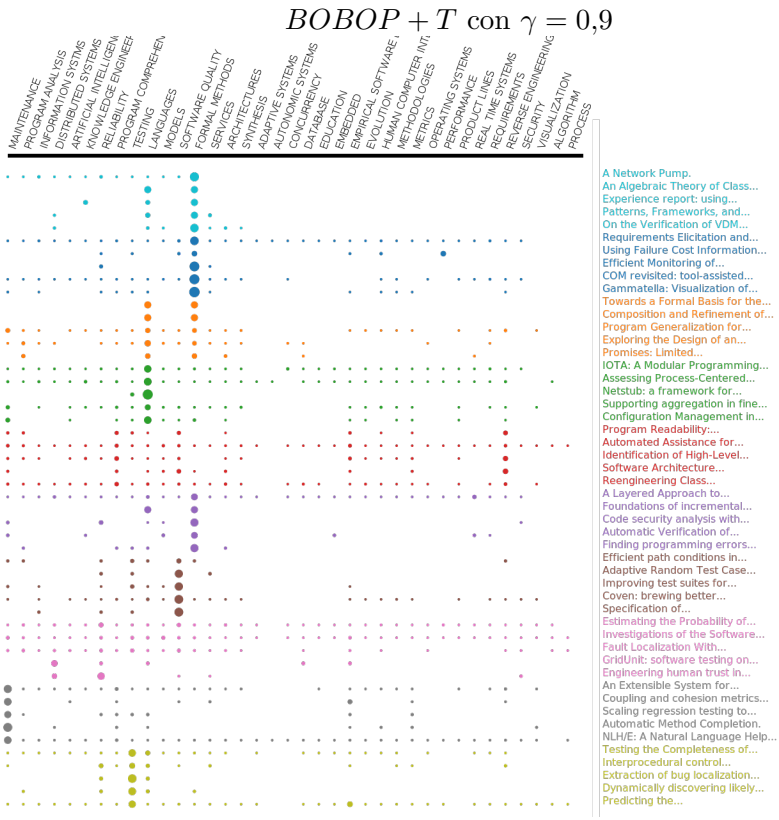

$BOBOP + T$ con $\gamma = 0{,}9$
MAINTENANCE
PROGRAM ANALYSIS
INFORMATION SYSTEMS
DISTRIBUTED SYSTEMS
ARTIFICIAL INTELLIGENCE
KNOWLEDGE ENGINEER
RELIABILITY
PROGRAM COMPREHEN
TESTING
LANGUAGES
MODELS
SOFTWARE QUALITY
FORMAL METHODS
SERVICES
ARCHITECTURES
SYNTHESIS
ADAPTIVE SYSTEMS
AUTONOMIC SYSTEMS
CONCURRENCY
DATABASE
EDUCATION
EMBEDDED
EMPIRICAL SOFTWARE
EVOLUTION
HUMAN COMPUTER INT
METHODOLOGIES
METRICS
OPERATING SYSTEMS
PERFORMANCE
PRODUCT LINES
REAL TIME SYSTEMS
REQUIREMENTS
REVERSE ENGINEERING
SECURITY
VISUALIZATION
ALGORITHM
PROCESS
A Network Pump.
An Algebraic Theory of Class...
Experience report: using...
Patterns, Frameworks, and...
On the Verification of VDM...
Requirements Elicitation and...
Using Failure Cost Information...
Efficient Monitoring of...
COM revisited: tool-assisted...
Gammatella: Visualization of...
Towards a Formal Basis for the...
Composition and Refinement of...
Program Generalization for...
Exploring the Design of an...
Promises: Limited...
IOTA: A Modular Programming...
Assessing Process-Centered...
Netstub: a framework for...
Supporting aggregation in fine...
Configuration Management in...
Program Readability:...
Automated Assistance for...
Identification of High-Level...
Software Architecture...
Reengineering Class...
A Layered Approach to...
Foundations of incremental...
Code security analysis with...
Automatic Verification of...
Finding programming errors...
Efficient path conditions in...
Adaptive Random Test Case...
Improving test suites for...
Coven: brewing better...
Specification of...
Estimating the Probability of...
Investigations of the Software...
Fault Localization With...
GridUnit: software testing on...
Engineering human trust in...
An Extensible System for...
Coupling and cohesion metrics...
Scaling regression testing to...
Automatic Method Completion.
NLH/E: A Natural Language Help...
Testing the Completeness of...
Interprocedural control...
Extraction of bug localization...
Dynamically discovering likely...
Predicting the...

Fig. 6.8

Bibliografía

[1] S. Amer-Yahia, F. Bonchi, C. Castillo, E. Feuerstein, I. Méndez-Díaz, and P. Zabala. Composite retrieval of diverse and complementary bundles. *IEEE Trans. Knowl. Data Eng*, pages 2662–2675, 2014.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst*, pages 107–117, 1998.

[4] V. Estivill-Castro. Why so many clustering algorithms: A position paper. *SIGKDD Explor. Newsl*, pages 65–75, 2002.

[5] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. *Algorithmica*, pages 410–421, 2001.

[6] E. Feuerstein, J. A. Knebel, I. Méndez-Díaz, A. Stein, and P. Zabala. New algorithms for composite retrieval. In *2016 XLII Latin American Computing Conference (CLEI)*, pages 1–9. IEEE, 2016.

[7] E. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classification. *Biometrics*, pages 768–769, 1956.

[8] C. Ghezzi, G. Ghezzi, P. Lanzi, M. Sangiorgio, and G. Tamburrelli. A data-driven journey through software engineering research. 2013.

[9] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[10] A. V. Goldberg. Finding a maximum density subgraph. 1984.

[11] E. Gossett. *Discrete Mathematics with Proof*. Wiley Publishing, 2009.

[12] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, Boston, MA, 1972.

[13] L. Kaufman and P. J. Rousseeuw. *Finding groups in data : an introduction to cluster analysis*. Wiley, 1990.

[14] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, pages 604–632, 1999.

[15] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[16] S. Basu Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu. Constructing and exploring composite items. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pages 843–854. ACM, 2010.

[17] Internet Live Stats. Internet users, 2016.

[18] D. Weiss and S. Osinski. Carrot, 2004.

[19] M. Xie, L. V. S. Lakshmanan, and P. T. Wood. Breaking out of the box of recommendations: From items to packages. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, pages 151–158. ACM, 2010.

[20] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, pages 1–56, 2006.