Ayudante interactivo para los algoritmos de Prim y Kruskal

Ouafae Debdi[1], Juan David Granada[2] y J. Ángel Velázquez Iturbide[1]
[1]Departamento de Lenguajes y Sistemas Informáticos I, [2] Departamento de Lenguajes y Sistemas Informáticos II
Universidad Rey Juan Carlos
C/ Tulipán s/n
28933 Móstoles, Madrid
{ouafae.debdi,david.granada,angel.velazquez}@urjc.es

Resumen

Uno de los elementos clave de los algoritmos voraces es una función de selección de candidatos que garantiza un resultado óptimo. Hemos desarrollado una colección de ayudantes interactivos diseñados para ayudar al alumno a identificar funciones de selección óptimas para problemas concretos. El proceso de identificación es un experimento (al estilo de las ciencias experimentales), en el que el alumno prueba de forma planificada posibles funciones de selección y decide cuáles son óptimas. En esta comunicación presentamos un ayudante interactivo desarrollado para soportar el método experimental aplicado al problema del árbol de recubrimiento de coste mínimo (resoluble por los conocidos algoritmos de Prim y Kruskal). La contribución de la comunicación es doble: un estudio bibliográfico sobre el tratamiento del problema en una selección de 12 libros de texto de reconocido prestigio, y el propio ayudante interactivo, llamado TuMiST. Esta aplicación educativa se ha utilizado en clase durante los cursos académicos 2008-2009 y 2009-2010.

1. Introducción

El trabajo aquí presentado es una contribución dentro de una línea de investigación sobre el desarrollo de aplicaciones educativas de animación de programas para técnicas de diseño de algoritmos. Dicha línea toma la taxonomía de Bloom [1] como marco para la especificación de aplicaciones educativas.

Hemos abordado el aprendizaje de los algoritmos voraces mediante un método experimental [15], que puede ser soportado por ayudantes interactivos (es decir, aplicaciones educativas interactivas). Hasta ahora, hemos desarrollado dos ayudantes interactivos [14][15]: AMO (concebido para ayudar en el estudio del problema de la mochila) y SEDA (para el problema de la selección de actividades [5]). En esta comunicación presentamos un tercer ayudante, llamado TuMiST. Como indica su nombre, el ayudante interactivo se ha concebido para ayudar en el problema del árbol de recubrimiento de coste mínimo, resoluble por los conocidos algoritmos de Prim y Kruskal.

La estructura de la comunicación es la siguiente. En la sección siguiente se presenta el método experimental diseñado. La sección tercera presenta los resultados del estudio bibliográfico realizado sobre el tratamiento del problema en una selección de 12 libros de texto de prestigio. La sección cuarta describe los elementos característicos de TuMiST frente a otros ayudantes interactivos. Por último, incluimos algunas líneas de trabajo futuro.

2. Experimentación interactiva con algoritmos voraces

Hemos comentado que usamos la taxonomía de Bloom [1] para especificar los objetivos educativos de nuestras aplicaciones. Se trata de un marco, muy conocido, para medir el grado de conocimiento de una materia por el alumno. En orden creciente de dificultad, pueden alcanzarse los niveles de: conocimiento, comprensión, aplicación, análisis, síntesis y evaluación.

Los objetivos de aprendizaje de los algoritmos voraces pueden alcanzarse de diversas formas. Hemos diseñado un método experimental para su aprendizaje activo. Lo presentamos con un ejemplo; puede encontrarse más información sobre el método en [15].

Sea el problema del árbol de recubrimiento de coste mínimo. Dado un grafo G=(N,A) no dirigido, conexo y valorado con costes no negativos, se dice que G'=(N,A') es un árbol de recubrimiento si enlaza todos los nodos de N mediante un número mínimo de arcos. Los arcos

deben tomarse de A; se necesitan exactamente |N|-1 arcos. El subgrafo resultante tiene estructura de árbol, sólo que carece de un orden que permita distinguir entre nodos antecesores y descendientes. El problema consiste en hallar un árbol de recubrimiento de G tal que la suma de los costes de los arcos seleccionados sea mínima.

Por ejemplo, sea el grafo G de seis nodos mostrado en Figura 1.

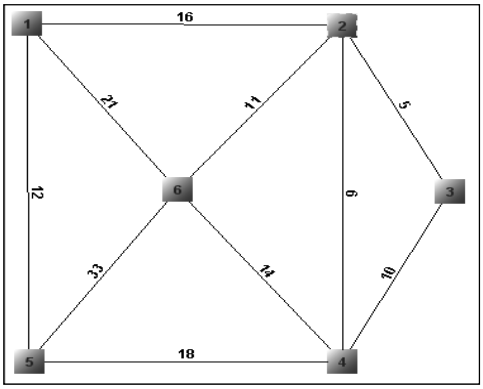

Figura 1. Grafo G de seis nodos

La Figura 2 presenta un subgrafo de G que es un árbol de recubrimiento de coste mínimo, con coste 50.

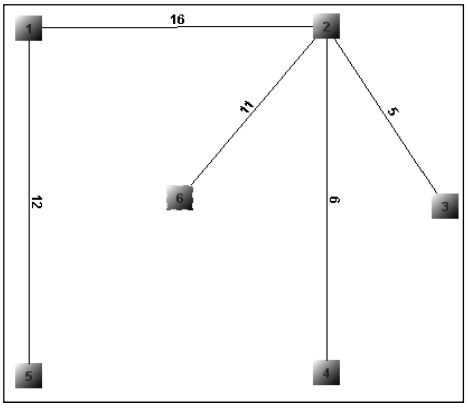

Figura 2. Grafo de recubrimiento de G con coste mínimo

Sin embargo, pueden formarse otros árboles de recubrimiento cuyo coste no es mínimo, como se ve en la Figura 3, con coste 60.

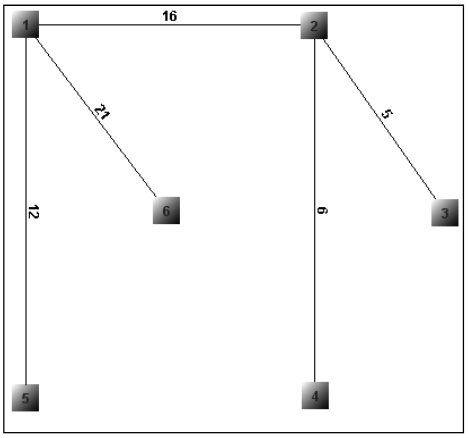

Figura 3. Grafo de recubrimiento de G cuyo coste no es mínimo

Ambas soluciones se han formado con un algoritmo voraz: se parte de un nodo y en cada paso se va añadiendo un arco de forma que el subgrafo va creciendo hasta formar un árbol de recubrimiento. El algoritmo voraz usado en los dos casos ha sido igual, con una diferencia importante: la función de selección utilizada. En el caso primero, se toma el arco de menor peso, mientras que en el segundo se toma un arco que une nodos de menor índice.

Lo presentado ilustra la esencia de nuestro método experimental para un aprendizaje activo de los algoritmos voraces. Proponemos ofrecer al alumno un conjunto de funciones de selección y dejarle experimentar de forma que deduzca qué estrategias parecen ser óptimas[1]. Podemos concebir varias funciones de selección de los arcos combinando tres factores:

- Índices de los pares de nodos que forman un arco o su coste.
- Orden creciente o decreciente de los valores.
- El subgrafo que se va construyendo debe ser conexo o se permitirán varios subgrafos inconexos (en todo caso, el subgrafo final siempre será conexo).

[1] Sólo pedimos evidencia experimental de que una función de selección *puede ser óptima*, ya que la certeza de que *es óptima* sólo se alcanza mediante una demostración formal.

Combinando estos tres factores, resulta un total de ocho funciones de selección distintas. La Tabla 1 muestra el resultado de aplicarlas.

Estrategia	Arcos seleccionados	Coste
Índice, crec., monografo	(1,2), (1,5), (1,6), (2,3), (2,4)	60
Índice, crec., multigrado	(1,2), (1,5), (1,6), (2,3), (2,4)	60
Índice, decr., monografo	(6,5), (6,4), (4,3), (6,2), (6,1)	89
Índice, decr., multigrado	(6,5), (6,4), (4,3), (6,2), (6,1)	89
Coste, crec., monografo	(1,5), (1,2), (2,3), (2,4), (2,6)	50
Coste, crec., multigrado	(2,3), (2,4), (2,6), (1,5), (1,2)	50
Coste, decr., monografo	(1,6), (5,6), (4,5), (1,2), (3,4)	98
Coste, decr., multigrafo	(5,6), (1,6), (4,5), (1,2), (3,4)	98

Tabla 1. Resultado de aplicar distintas funciones de selección

Las funciones de selección que producen costes mínimos para este ejemplo son la quinta y sexta. La función de selección quinta corresponde al algoritmo de Prim, mientras que la sexta corresponde al algoritmo de Kruskal.

En las dos secciones siguientes presentamos el trabajo realizado para desarrollar un ayudante interactivo que soporte el método experimental presentado para este problema. Comenzamos con la sección tercera, incluyendo los resultados de una búsqueda bibliográfica que nos permitió tomar elementos útiles en el diseño del ayudante interactivo y así "no reinventar la rueda". En la sesión cuarta se describe el ayudante interactivo TuMiST con detalle.

3. Búsqueda bibliográfica

Se realizó una búsqueda bibliográfica en 12 libros de texto de reconocido prestigio con el fin de extraer información útil para el desarrollo de un ayudante interactivo [2][3][4][5][6][7][8][9][10][11][12][13]. Los dos aspectos que más nos interesaban fueron:

- Ilustraciones de los algoritmos que resuelven el problema (sobre todo, Kruskal y Prim). El objetivo era adoptar el mejor diseño gráfico.
- Codificación de los algoritmos. El objetivo era utilizar el código más claro para la explicación de un algoritmo voraz. También era interesante conocer el lenguaje de programación en el que están codificados.

Ambos aspectos se recopilaron para cada libro, con indicación de la página donde se podían encontrar. Junto con esta información básica, recopilamos alguna información adicional para cada libro:

- Denominación del problema o de los algoritmos que lo resuelven.
- Sección y subsección donde se trata.
- Algoritmos voraces óptimos incluidos.
- Otros aspectos secundarios: ejemplos, demostración de optimidad de las funciones de selección, análisis de complejidad de los algoritmos.

En Tabla 2 y Tabla 3 presentamos de forma resumida los resultados de la búsqueda bibliográfica.

La información recogida fue analizada y nos permitió alcanzar los dos objetivos comentados antes sobre visualización y codificación.

Consideramos que la utilidad de esta información no acaba aquí, sino que conviene divulgarla porque es una documentación que puede ayudar a cualquier profesor universitario de algoritmia interesado en el problema.

4. Ayudante interactivo: TuMiST

TuMiST (*Tu*tor for the *Mi*nimum *S*panning *T*ree problem) es el nombre dado al ayudante interactivo que hemos desarrollado para dar apoyo a nuestro método de experimentación con este problema.

La Figura 4 presenta la interfaz de usuario de TuMiST. Se muestra un estado intermedio de la ejecución del algoritmo, para el grafo de la sección 2 y para la estrategia de orden creciente de coste de los arcos. Se han elegido dos arcos que parten del nodo 1.

Libro	Capítulo / sección	Algoritmos	Visualización & implementación	Denominación	Otros
M. H. Alsuwaiyel, *Algorithms Design Techniques and Analysis*	Capítulo 8 (8.3)	Prim, Kruskal	Prim: pseudocódigo (p. 245) Kruskal: pseudocódigo (p. 241)	*Minimum cost spanning trees*	– Ambos algoritmos: demostración de optimidad, análisis de complejidad, estructura de datos
S. Baase y A. Van Gelder, *Computer Algorithms: Introduction to Design and Analysis*	Capítulo 8 (8.2)	Prim, Kruskal	Prim: pseudocódigo y Java (p. 388) Kruskal: pseudocódigo (p. 412)	*Prim's minimum spanning tree algorithm* *Kruskal's minimum spanning tree algorithm*	– Prim: propiedades, demostración de optimidad, estructura de datos, análisis de complejidad – Kruskal: análisis de complejidad
G. Brassard y P. Bratley, *Fundamentos de algoritmia*	Capítulo 6 (6.3)	Prim, Kruskal	Prim: pseudocódigo (p. 221) Kruskal: pseudocódigo (p. 220)	Grafos: árboles de recubrimiento mínimo	– Ambos algoritmos: demostración de optimidad, análisis de complejidad
T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, *Introduction to Algorithms*	Capítulo 23 (23.2)	Prim, Kruskal	Prim: pseudocódigo (p. 572) Kruskal: pseudocódigo (p. 569)	*Minimum spanning trees*	– Algoritmo genérico: demostración de optimidad – Ambos algoritmos: análisis de complejidad – Prim: sugerencia de estructuras de datos
M. T. Goodrich y R. Tamassia, *Data Structures and Algorithms in Java*	Capítulo 10 (10.2)	Prim, Kruskal, Barůvka	Prim: pseudocódigo (p. 417) Kruskal: pseudocódigo (p. 413) Barůvka: pseudocódigo (p. 420)	*Minimum spanning trees*	– Ambos algoritmos: análisis de complejidad
E. Horowitz y S. Sahni, *Computer Algorithms*	Capítulo 4 (4.5)	Prim, Kruskal, Barůvka	Prim: pseudocódigo (p. 221) Kruskal: pseudocódigo (p. 224) Barůvka: sin código (p. 225)	*Minimum-cost spanning trees*	– Tres algoritmos: análisis de la complejidad

Tabla 2. Información bibliográfica extraída sobre el problema del árbol de recubrimiento de coste mínimo

Libro	Capítulo / sección	Algoritmos	Visualización & implementación	Denominación	Otros
N. Martí Oliet, Y. Ortega y J. A. Verdejo, *Estructuras de datos y métodos algorítmicos: ejercicios resueltos*	Capítulo 12 (12.13, 12.14, 12.15 y 12.16)	Prim, Kruskal	Prim: pseudocódigo (pp. 384-385) Kruskal: pseudocódigo (p. 387)	Árbol de recubrimiento de coste mínimo	– Prim: dos ejemplos con diferente implementación – Kruskal: un ejemplo – Ejemplo: pavimentación de calles en una ciudad
R. Neapolitan y K. Naimipour, *Foundations of Algorithms*	Capítulo 4 (4.1)	Prim, Kruskal	Prim: pseudocódigo y C++ (pp. 143-144) Kruskal: pseudocódigo y C++ (pp. 147 y 149)	*Minimum spanning trees*	– Ambos algoritmos: demostración y análisis de complejidad con comparación final
I. Parberry, *Problems on Algorithms*	Capítulo 9 (9.3)	Prim, Kruskal	Prim: pseudocódigo (p. 104) Kruskal: pseudocódigo (p. 104)	*Min-cost spanning trees*	
S. Sahni, *Data Structures, Algorithms, and Applications in Java*	Capítulo 18 (18.3.6)	Prim, Kruskal, Sollin	Prim: pseudocódigo (p. 731) Kruskal: pseudocódigo y Java (pp. 729-732) Sollin: descripción informal (p. 731)	*Minimum-cost spanning trees*	– Ambos algoritmos: demostración de optimidad, elección de estructura de datos, análisis de complejidad, comparación entre algoritmos
R. Sedgewick, *Algorithms in Java*	Capítulo 20	Prim, Kruskal, Barůvka, Euclidean	Prim: Java (p. 250) Kruskal: Java (p. 261) Barůvka: Java (p. 266)	*Minimum spanning trees*	– Ambos algoritmos: análisis de complejidad, estructura de datos, comparación de eficiencia
S. Skiena, *The Algorithm Design Manual*	Capítulo 4 (4.7)	Prim, Kruskal	Prim: pseudocódigo simplificado (p. 98) Kruskal: pseudocódigo (p. 99)	*Minimum spanning trees*	– Ambos algoritmos: análisis de complejidad

Tabla 3. Información bibliográfica extraída sobre el problema del árbol de recubrimiento de coste mínimo (cont.)

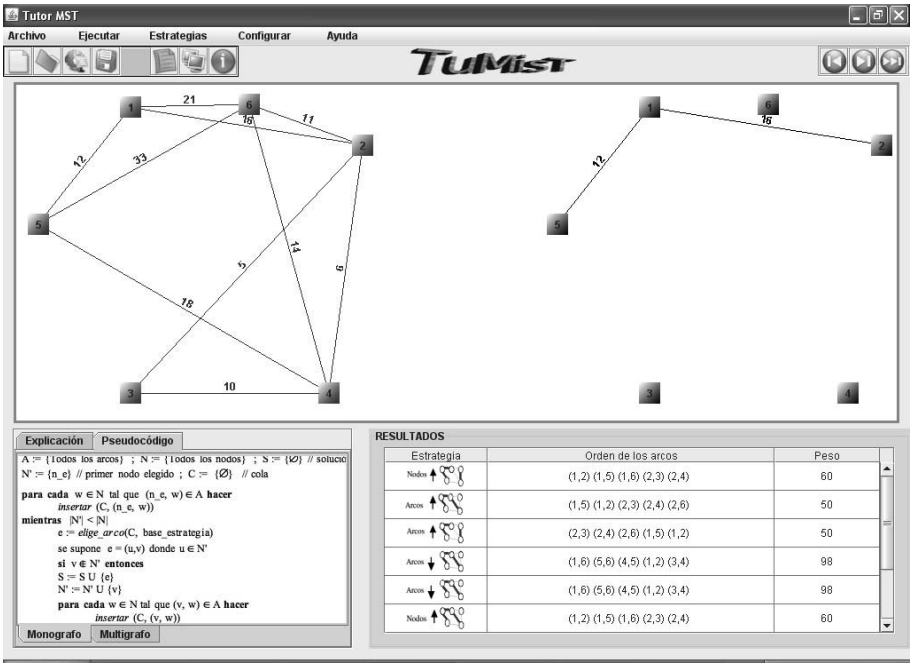

Figura 4. Una captura de pantalla del ayudante interactivo TuMiST

Se distinguen claramente tres zonas en la interfaz de usuario, aparte del menú principal y la barra de iconos. En la parte superior se encuentra el panel de visualización. En su izquierda muestra el grafo de entrada y en su parte derecha, el árbol de recubrimiento que se está construyendo.

En la parte inferior izquierda se encuentra el panel de teoría. Consta de dos pestañas: la pestaña de explicación contiene el enunciado del problema, y la de pseudocódigo (visible en la figura), el algoritmo. Esta pestaña permite acceder a otras dos, situadas en la parte inferior: una con el pseudocódigo para las funciones de selección que generan un solo subgrafo (visible) y otra para las funciones de selección que permiten crear varios subgrafos en estados intermedios. Se ha intentado utilizar un pseudocódigo "genérico", es decir, un algoritmo voraz que fuera independiente de la función de selección. Sin embargo, no ha sido posible y finalmente hemos tenido que incluir estos dos pseudocódigos.

Finalmente, la parte derecha de la pantalla contiene el panel de resultados, que muestra una tabla igual a la Tabla 1.

Al arrancar la aplicación, el usuario sólo encuentra contenido en los paneles de explicación y de pseudocódigo. El usuario debe leer, al menos, el panel de explicación para comprender el problema planteado.

Después, el usuario debe crear un grafo para ejecutar el algoritmo (con las distintas estrategias). La aplicación permite al usuario crear el grafo a partir de tres fuentes distintas:

- Teclado. Se introducen mediante un diálogo, como se muestra en la Figura 5. Obsérvese que está diseñado para editar grafos no dirigidos, así como para evitar arcos de un nodo a sí mismo. Esta forma de introducir datos es útil para crear un ejemplo concreto.
- Generador de números aleatorios. Es útil para producir rápidamente un grafo cualquiera.
- Fichero. Es útil para reproducir unos datos de entrada ya usados (y almacenados) anteriormente, normalmente de tamaño grande. Su formato de almacenamiento es sencillo, como muestra la Figura 6 para el grafo usado de ejemplo en la comunicación.

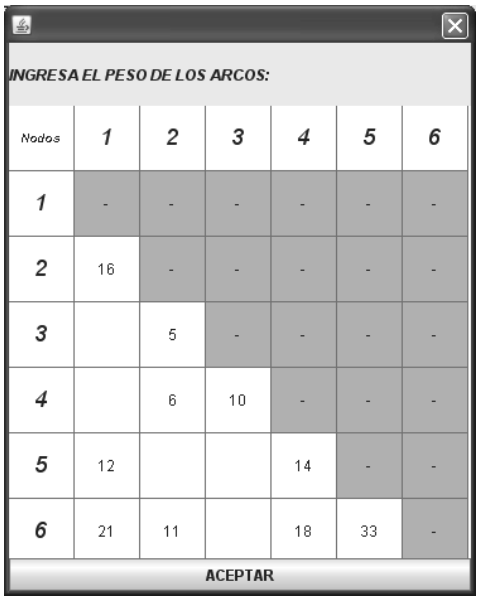

Figura 5. Diálogo para entrada de datos

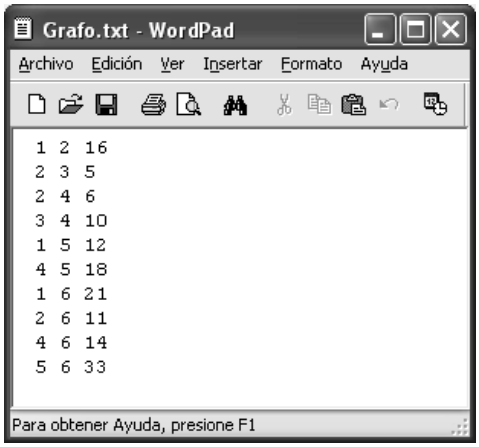

Figura 6. Formato en fichero de un grafo

La aplicación limita el número de nodos a 10, para que los ejemplos (y su visualización) sean manejables. El diálogo de entrada por teclado pide al usuario que concrete el número de nodos. Sin embargo, el generador genera aleatoriamente dicho número.

TuMiST dibuja automáticamente los grafos situando los nodos en formato circular para evitar solapamientos. El usuario puede reconfigurar manualmente los grafos visualizados, tanto el grafo de entrada como el árbol de recubrimiento. Los grafos de las Figuras 1-3 se han obtenido de esta forma.

Una vez introducido el grafo de entrada, el usuario puede elegir y ejecutar las funciones de selección y comparar sus resultados. Las funciones de selección se representan de forma concisa en la tabla de resultados mediante una combinación de los tres factores citados en la Sección 2 (véase la columna izquierda de la tabla de resultados en la Figura 4).

El usuario puede ejecutar cada función de selección paso a paso (adelante o atrás) o de forma atómica. También puede ejecutar todas las funciones de selección en un solo paso.

El usuario también puede, en todo momento, modificar los datos del grafo actual (mediante un diálogo igual al de entrada de datos) o crear un grafo nuevo. En estos casos, la tabla de resultados se vacía y se dibuja el nuevo grafo.

Finalmente, TuMiST proporciona algunas otras funciones para mejorar su usabilidad:

- Almacenar en un fichero de texto bien el grafo de entrada bien los resultados de las distintas funciones de selección. El formato de almacenamiento de un grafo ya se ha mostrado en la Figura 6. El fichero de resultados contiene los resultados de las funciones de selección aplicadas. La Figura 7 presenta el resultado de una función de selección sobre nuestro grafo de ejemplo.
- Configuración de colores de nodos y arcos.
- Ayuda interactiva.

```
Arcos elegidos con la estrategia:
Peso de los arcos en modo creciente
generando un monografo.

Arco: (1, 5) Con el coste: 12
Arco: (1, 2) Con el coste: 16
Arco: (2, 3) Con el coste: 5
Arco: (2, 4) Con el coste: 6
Arco: (2, 6) Con el coste: 11

El peso del MST es: 50.
```

Figura 7. Formato en fichero del resultado de aplicar una función de selección

Desde un punto vista técnico, TuMiST se ha desarrollado utilizando el lenguaje de programación Java (en particular la librería

Swing), la librería gráfica JGraph, el IDE NetBeans, y la herramienta de edición de imágenes Gimp.

5. Experiencia y disponibilidad

Hemos desarrollado los ayudantes interactivos para uso docente. En concreto, TuMiST se ha utilizado durante los cursos académicos 2008-2009 y 2009-2010 en la asignatura "Diseño y Análisis de Algoritmos", de tercer curso de Ingeniería Informática en la Universidad Rey Juan Carlos. El profesor usó AMO y TuMiST en las clases de teoría en el aula y los alumnos usaron SEDA para realizar la práctica de este tema. La experiencia ha sido muy positiva, tanto por la aceptación de los alumnos [14] como por las calificaciones obtenidas en las prácticas.

TuMiST está disponible, junto con los otros ayudantes interactivos AMO y SEDA, en la URL http://www.lite.etsii.urjc.es/greedex/. Se trata de un sitio web específico para estos ayudantes interactivos. Consta de seis secciones con diversa información: página principal, descripción, uso educativo, documentación técnica, documentación académica y descarga.

6. Conclusiones y trabajo futuro

Hemos presentado un método experimental para fomentar el aprendizaje activo de los algoritmos voraces y su aplicación al problema del árbol de recubrimiento de coste mínimo. Las dos aportaciones de esta comunicación son: un estudio bibliográfico sobre el tratamiento del problema en 12 libros de texto de prestigio, y un ayudante interactivo, llamado TuMiST, desarrollado para dar apoyo a la aplicación del método experimental a dicho problema.

Actualmente se están refundiendo los ayudantes interactivos desarrollados en uno solo. Su nombre es GreedEx (de "GREEDy algorithms" y "EXperimentation"). Asimismo, se están añadiendo nuevas facilidades que mejoren la usabilidad de la aplicación o que amplíen la cobertura del método experimental.

Agradecimientos

Este trabajo se ha financiado parcialmente con el proyecto TIN2008-04103/TSI del MICINN.

Referencias

[1] Bloom, B., Furst, E., Hill, W., y Krathwohl, D.R. *Taxonomy of Educational Objectives: Handbook I, The Cognitive Domain*. Addison-Wesley, 1956.

[2] Alsuwaiyel, M.H. *Algorithm Design Techniques and Analysis*.

[3] Baase, S., y van Gelder, A. *Computer Algorithms: Introduction to Design and Analysis*.

[4] Brassard, G., y Bratley, P. *Fundamentos de algoritmia*. Prentice-Hall, 1997.

[5] Cormen, T.H., Leiserson, C.E., y Rivest. R.L. *Introduction to Algorithms*. The MIT Press, 2ª ed., 2003.

[6] Goodrich, M.T., y Tamassia, R. *Data Structures and Algorithms in Java*. John Wiley & Sons, 2ª ed., 2001.

[7] Horowitz, E., y Sahni, S. *Fundamentals of Computer Algorithms*. Pitman, 1978.

[8] Martí Oliet, N., Ortega, Y., y Verdejo, J.A. *Estructuras de datos y métodos algorítmicos: ejercicios resueltos*. Pearson, 2004.

[9] Neapolitan, R., y Naimipour, K. *Foundations of Algorithms*. Jones and Bartlett, 1997.

[10] Parberry, I. *Problems on Algorithms*.

[11] Sahni, S. *Data Structures, Algorithms, and Applications in Java*, McGraw-Hill, 2000.

[12] Sedgewick, R. *Algorithms in Java*. Addison-Wesley, 2002.

[13] Skiena, S. *The Algorithm Design Manual*. Springer-Verlag, 1998.

[14] Velázquez Iturbide, J.Á., Lázaro Carrascosa, C.A., y Hernán Losada, I. "Asistentes interactivos para el aprendizaje de algoritmos voraces". En *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje, IEEE-RITA*, 4(3):213-220, agosto 2009.

[15] Velázquez-Iturbide, J.Á., y Pérez-Carrasco, A. "Active learning of greedy algorithms by means of interactive experimentation". En *Proc. 14th Annual Conf. on Innovation and Technology in Computer Science Education, ITiCSE 2009*. ACM Press, pp. 119-123.