J. Ángel Velázquez Iturbide

Un Análisis Cualitativo de Dificultades durante el Aprendizaje de Técnicas de Diseño de Algoritmos de Optimización

Número 2017-01

Serie de Informes Técnicos DLSI1-URJC
ISSN 1988-8074
Grupo Docente de Lenguajes y Sistemas Informáticos I
Universidad Rey Juan Carlos

Índice

Un Análisis Cualitativo de Dificultades durante el Aprendizaje de Técnicas de Diseño de Algoritmos de Optimización

J. Ángel Velázquez Iturbide

Grupo Docente de Lenguajes y Sistemas Informáticos I, Universidad Rey Juan Carlos,
C/ Tulipán s/n, 28933, Móstoles, Madrid
angel.velazquez@urjc.es

Resumen. Este informe presenta un análisis cualitativo de las dificultades que presentan los alumnos para el aprendizaje de algunas técnicas de diseño de algoritmos de optimización. Se analizan las dificultades relacionadas con: la determinación de las decisiones óptimas en algoritmos voraces con ordenación inicial de los candidatos, la implementación de algoritmos heurísticos, el razonamiento sobre la optimidad de algoritmos heurísticos, y el diseño de algoritmos de vuelta atrás o de ramificación y poda. Se realiza el análisis a partir de los informes presentados para la realización de prácticas de una asignatura de algoritmos. Cada práctica se realizó con sistemas distintos: OptimEx o de elección libre. Se incluyen los enunciados de prácticas usados, el método de análisis, y los resultados detallados y comentados.

Palabras clave: Algoritmos de optimización, técnicas de diseño de algoritmos, análisis cualitativo, análisis de documentos, malconcepciones.

1 Introducción

En trabajos anteriores hemos analizado diversas dificultades y malconcepciones de los alumnos sobre cuestiones relacionadas con la optimidad. Estos análisis se han realizado en el contexto de prácticas de comparación de la optimidad de algoritmos alternativos para un mismo problema, bien algoritmos voraces [1][2] bien algoritmos diseñados con distintas técnicas [3][4]. Estos trabajos se realizaron, respectivamente, con los sistemas GreedEx [5] y OptimEx [6][7].

En este informe analizamos las dificultades de los alumnos con otras tareas relacionadas con diversas técnicas de diseño de algoritmos de optimización. La estructura del informe es la siguiente. El apartado 2 describe el protocolo y método de análisis utilizados. Los apartados 3, 4 y 5 presentan, respectivamente, los resultados obtenidos con las prácticas 1, 2 (sólo un parte) y 3a. Los apartados 6 y 7 contienen, respectivamente, un resumen de resultados y unas conclusiones. Finalmente, tres apéndices contienen el enunciado de las prácticas analizadas.

2 Protocolo y Análisis

Esta evaluación se realiza en el contexto de la asignatura optativa "Algoritmos Avanzados", de cuarto curso del Grado en Ingeniería Informática. Se utilizan los informes de las prácticas entregadas por los alumnos de dos grupos (presencial y online) en septiembre-diciembre del curso académico 2015-16.

El temario de la asignatura incluía varias técnicas algorítmicas para la resolución de problemas de optimización: técnica voraz, algoritmos heurísticos y aproximados, vuelta atrás, ramificación y poda, y programación dinámica. Se proponía la realización de 5 prácticas:

1. Técnica voraz. Dado el problema de selección de actividades [8, cap. 16] [9, cap. 4] y pseudocódigo para su resolución voraz, se identifican dos funciones de selección óptima: orden creciente de instantes de fin de actividad (en adelante, representado F↑) y orden decreciente de instantes de inicio (en adelante, representado C↓). Se pide completar el algoritmo y ampliarlo para que no dependa de que las actividades estén ordenadas.
2. Algoritmos heurísticos. Se plantea el problema del plan de sedes de coste mínimo [9, cap. 6] y se esbozan dos algoritmos heurísticos. Se pide desarrollarlos, comprobar su optimidad con OptimEx y justificar su optimidad o suboptimidad (en el primer caso, con un razonamiento intuitivo y en el segundo, con un contraejemplo de su optimidad).
3. Técnicas de búsqueda. La práctica se realizó con dos entregas:
   a. Dado el mismo problema del plan de sedes de coste mínimo, se pide desarrollar un algoritmo de vuelta atrás y otro de ramificación y poda.
   b. Se pide comprobar la optimidad de estos dos algoritmos y los dos algoritmos heurísticos de la práctica 2.
4. Eliminación de recursividad múltiple redundante. Se da una función recursiva f y se pide analizar su redundancia y eliminarla mediante memorización y tabulación.
5. Técnica de programación dinámica. De nuevo, se plantea desglosada:
   a. Dado el mismo problema del plan de sedes de coste mínimo, se pide diseñar un algoritmo recursivo y convertirlo en un algoritmo de programación dinámica (tabulado).
   b. Se pide comprobar la optimidad de estos dos algoritmos y los cuatro algoritmos ya comparados en la práctica 3b.

En este informe, analizamos las prácticas 1, 2 (solamente la implementación de algoritmos heurísticos y el razonamiento sobre su optimidad) y 3a. El análisis de la experimentación con la optimidad realizado en las prácticas 2, 3b y 5b ha sido objeto de otro informe técnico [10], así como el análisis de la práctica 5a [11]. El objetivo de la práctica 4 (eliminación de la recursividad redundante) no parece plantear problemas destacables a los alumnos.

La pauta de realización de todas las prácticas fue común. A una hora y día convenidos, el enunciado de cada práctica estaba disponible en el campus virtual para que los alumnos se lo descargaran. También estaba disponible cualquier material complementario que fuera necesario para realizar la práctica. En las práctica 2 se

debía utilizar el sistema OptimEx [6][7], mientras que en las demás prácticas podía utilizarse el sistema de preferencia del alumno (normalmente BlueJ, Eclipse o NetBeans)

A la misma hora, se iniciaba una sesión de prácticas de 2 horas de duración para los alumnos del grupo presencial. Los alumnos disponían de una semana de plazo para entregar el informe de cada práctica por medio del campus virtual.

Tras la corrección de cada práctica y el envío de comentarios por parte del profesor, se les permitía corregir errores y reenviar la práctica en el plazo de varios días.

Incluimos los enunciados de las tres prácticas como apéndices de este informe.

Para el análisis cada práctica, se construyó una tabla donde aparecieran los elementos que se consideraron inicialmente importantes. Se realizaron dos rondas de análisis:

1. Ronda primera. Se construyó una tabla inicial y se fueron escribiendo etiquetas para sus campos.
2. Rondas posteriores. Hubo que revisar las columnas y las etiquetas de la tabla según se avanzaba en el análisis o se escribía el informe.

Conviene comentar que cada ronda no se realizó analizando secuencialmente los informes, sino que implicó numerosas consultas a los mismos y modificaciones de las tablas construidas.

3 Práctica 1: Algoritmos Voraces

Se recogieron un total de 45 informes de 38 grupos (27 parejas y 11 individuales). Estos informes se enviaron de la siguiente forma:

- Primer plazo de entrega: 37 envíos, 35 del grupo presencial (23 individuales y 11 parejas) y 3 del grupo on-line (todos individuales).
- Segundo plazo: 7 envíos, 5 del grupo presencial (3 individuales y 2 parejas) y 2 del grupo on-line (una entregada por primera vez, individual).
- Tercer plazo: 1 envío del grupo on-line (por tanto, ya fuera de plazo).

Presentamos los resultados del cuestionario en la Tabla 1. La tabla incluye una fila por cada informe entregado. Consta de las siguientes columnas:

- “Grupo”. Asigna un número a cada grupo, indicando también si es individual “I” o una pareja “P”.
- Tres columnas sobre el uso de estructuras de datos: si se respeta la signatura del método principal dada en el enunciado, si se utilizan vectores innecesarios y si se modifican los vectores originales.
- Cuatro columnas que caracterizan el algoritmo desarrollado: algoritmo de resolución del problema, algoritmo de ordenación usado (si es el caso), si el algoritmo calcula el resultado correcto y si se aportan ejemplos para ilustrarlo.

- Dos columnas con observaciones sobre la práctica y los comentarios abiertos realizados por los grupos en el apartado de conclusiones del informe.

Tabla 1. Análisis de los informes de la práctica 1

Grupo	Respeta tipos enunciado	Vectores auxiliares extra	Respeta vectores originales	Selección de candidatos	Algoritmo de ordenación	Resultado correcto	Aporta ejecuciones	Otras observaciones	Comentarios abiertos
1. I	✓	✓		Ordena índices pero asocia al final	Burbuja con centinela			– Entregada en 2º plazo – Aunque determina índices, aplica el algoritmo original a datos sin ordenar	– Parecía fácil – Quisiera usar una clase Actividades
2. I	✓		✓	Búsqueda					– Reconoce que no funciona
3. I	ArrayList		✓	Búsqueda		✓			– Parecía fácil
	✓	✓	✓	Búsqueda		✓	✓		– Igual – Quisiera usar una clase Actividades
	✓	✓	✓	Copia y ordena índices	Selección	✓	✓	– Entregada fuera de plazo	– Igual
4. I	ArrayList		✓	Búsqueda		✓	✓		
5. P	✓		✓	Búsqueda		✓	✓		– Quisieran usar clases
6. I	✓		✓	Ordena índices	Propio[1]	✓	✓		– Práctica fácil

[1] Señala que el algoritmo de ordenación proporcionado en la asignatura y el interno de GreedEx ordenan sin tener en cuenta la longitud de la actividad

7.	I	✓[2]	✓	✓	Copia y ordena índices, complicado	Sort (ArrayList)	✓	✓	– Problema de actividades con mismo final
8.	P	✓		✓	Ordena índices	Selección (oscuro)	✓	✓	
9.	P	✓	✓	✓	Búsqueda		✓		– Lo más difícil era controlar la primera iteración de la búsqueda
		✓	✓	✓	Copia y ordena índices	Ordenación por mezcla	✓		
10.	I	✓		✓	Ordena índices	Inserción	✓		
11.	I	✓	✓	✓	Copia, ordena y asocia	Inserción	✓		– Dificultad de la técnica de índices – Reclama más ejercicios
12.	I	✓		✓	Ordena índices	Inserción	✓		– Poco tiempo (por trabajo)
13.	P	✓		✓	Copia y ordena actividades	Inserción			

[2] Las respeta en la cabecera pero introduce ArrayLists dentro del algoritmo

	✓	✓	✓	Ordena índices	Inserción	✓	✓	– Dificultad en entender código proporcionado
14. P	✓	✓	✓	Copia, ordena y asocia	Inserción	✓	✓	– Dificultad en entender código proporcionado – Sin tiempo para tutoría
15. I	✓		✓	Ordena índices	Inserción	✓		– Dificultad en entender enunciado – Quizá no ha entendido el papel de la ordenación
16. I	✓		✓	Ordena índices	Inserción	✓		– Dificultad en entender la técnica (abstracta)
17. I	✓		✓	Ordena índices	Inserción	✓	✓	– Cierta dificultad
18. P	✓	✓	✓	Ordena índices pero asocia al final	Inserción	✓	✓	
19. P	✓	✓	✓	Búsqueda		✓	✓	– Dificultad con posiciones – Poco tiempo
	✓	✓	✓	Ordena índices	Selección	✓	✓	– Poco tiempo

20. I	✓		✓	Ordena índices	Inserción	✓			– Novedad técnica de índices
21. I	✓		✓	Ordena índices	Inserción	✓	✓		– Quisiera usar otras estructuras de datos – Cierta dificultad de la técnica
22. I	✓	✓	✓	Ordena índices pero asocia al final	Selección	✓	✓[3]	– La cabecera incluye el vector de índices	– Cierta dificultad de la técnica
23. I	✓	✓	✓	Copia y ordena índices	Burbuja	✓			– Cierta dificultad de la técnica
24. I	✓		✓	Ordena índices	Indeterminado	✓	✓		– Cierta dificultad de la técnica – Habla como si la hubiera inventado él
25. P	✓	✓	✓	Copia y ordena índices	Burbuja	✓			
26. I	✓			Ordena vectores	Burbuja	✓		– La cabecera incluye el vector de índices	
	✓	✓		Copia y ordena índices	Burbuja			– Se equivoca y altera los vectores originales	

[3] Ilustra la generación de índices con vectores, pero no aporta una ejecución del algoritmo completo.

27. I	✓		✓	Ordena índices	Inserción	✓		
28. I	✓		✓	Ordena índices	Inserción	✓	✓	– Utilidad de los apuntes
29. P	✓	✓	✓	Ordena índices pero no los usa	Inserción		✓	
30. P	✓		✓	Ordena índices	Inserción	✓		
31. I	✓	✓	✓	Búsqueda		✓		
	✓		✓	Ordena índices	Inserción	✓	✓	
32. I	✓	✓	✓	Ordena índices	Indeterminado	✓		– Elimina una rama del if del algoritmo voraz original
33. I	✓	✓	✓	Ordena índices	Selección	✓		– Elimina una rama del if del algoritmo voraz original
34. I	✓		✓	Ordena índices	Selección			– Faltan partes por modificar en el algoritmo voraz original
35. I	✓		✓	Ordena índices	Inserción	✓		
36. I	✓		✓	Ordena índices	Rápida	✓		– Sugiere usar otras estructuras de datos
37. P	✓		✓	Ordena índices	Inserción	✓		

38. I	✓	✓	Ordena índices	Inserción	✓	– El algoritmo de ordenación también pasa como parámetro el vector de comienzos

El aspecto más relevante es el tipo de modificación realizada sobre el algoritmo original para manejar datos desordenados. Encontramos:

- 31 grupos (81’6%) realizan alguna clase de ordenación de los datos de entrada frente a 7 grupos que realizan una búsqueda de cada candidato (18’4%). Ante la realimentación proporcionada por el profesor, 4 grupos sustituyeron la búsqueda por una ordenación, sumando 35 soluciones con ordenación (92’1%).
- Las 35 soluciones con ordenación pueden dividirse en tres clases:
  - Soluciones que utilizan un vector de índices sin alterar los datos de entrada (24 grupos, 68’6%). Todas las soluciones son correctas salvo una (G29), que realiza correctamente la ordenación basada en índices, pero luego no usa el vector de índices en la resolución del problema.
  - Soluciones que crean alguna copia de los vectores originales previa a la ordenación (6 grupos, 17’1%). La solución de un grupo es especialmente complicada (G07). Podemos considerar estos casos como una manifestación de la relativa dificultad de la técnica o un exceso de celo de los alumnos para no alterar los vectores originales.
  - Soluciones más complicadas (5 grupos, 14’3%) que crean el vector de índices pero solamente lo usan al final para asociar el resultado del algoritmo voraz con los datos de entrada. Para resolver el problema, copian los datos de entrada en vectores auxiliares y aplican el algoritmo voraz sin modificar. Todas las soluciones salvo una (G01) parecen estar bien.
- Solamente dos grupos (G01, G26) alteran el contenido de los vectores originales al ordenarlos. El grupo G26 lo intenta corregir en su segunda entrega creando copias de los vectores originales, pero produce un algoritmo incorrecto.

Diversos comentarios corroboran la relativa dificultad de la técnica:

- Comentarios en este sentido (14 grupos, 36’8%), de forma explícita o implícita. Algunos grupos indican la abstracción de la técnica (G16), su novedad (G20) o la utilidad de los apuntes para comprenderla (G28).
- Entre los comentarios, 4 grupos hubieran querido disponer de más tiempo para realizar la práctica y a 1 grupo le hubiera gustado realizar más ejercicios.

Sin embargo, la dificultad no es excesiva: solamente 6 grupos (15’8%) entregan un algoritmo que no resuelve el problema. Un grupo lo arregla en una segunda entrega (G13), mientras que otro corrige su alteración de los vectores originales produciendo un algoritmo erróneo (G26).

Por último, algunos hallazgos menores son:

- En proporción, se reciben más segundas entregas de la parejas que de los grupos individuales.
- La mayor parte de los alumnos han respetado la signatura del método principal, aunque hay alumnos que ha usado colecciones de Java (3 grupos) o les gustaría usarlas (5 grupos).

4 Práctica 2: Algoritmos Heurísticos

En esta práctica se planteaba el problema del plan de sedes de coste mínimo (ilustrado con un ejemplo) y se esbozaban dos algoritmos heurísticos para resolverlo (en una línea cada uno y sin ejemplo de aplicación). También se pedía comparar sus resultados y determinar su optimidad con ayuda de OptimEx. Esta parte de la práctica no se analiza aquí porque ya se ha analizado en otro informe técnico [10].

Se recogieron un total de 37 informes en el primer plazo de entrega: 34 del grupo presencial (23 individuales y 11 parejas) y 3 del grupo on-line (todos individuales). Solamente se presentaron 12 informes en el segundo plazo: 10 del grupo presencial (6 individuales y 4 parejas) y 2 del grupo on-line. Por último, se recibió una tercera entrega del grupo presencial. Por tanto, el número total de informes recogidos es 50.

En este apartado, por coherencia y sencillez, mantenemos la numeración de grupos realizada en el informe técnico [10, Tabla 1]. En dicha tabla pueden consultarse los detalles de las características de los algoritmos heurísticos desarrollados.

4.1 Implementación de Algoritmos Heurísticos

La mayor parte de los grupos (28 grupos, 75’7%) implementó correctamente los algoritmos. Sin embargo, varios grupos (9 grupos: G01, G04, G06, G08, G11, G18, G31, G36, G37; 24’3%) confundieron la función objetivo (del problema) y la función de selección (del algoritmo heurístico) en los algoritmos heurísticos. Interpretaron e implementaron correctamente la función de selección, que no dependía del coste del traslado. Sin embargo, no tuvieron en cuenta este coste para calcular el coste de la solución, es decir, repitieron el criterio de selección del algoritmo heurístico para el cálculo de costes.

En detalle, seis grupos diseñaron mal el algoritmo primero (G06, G08, G11, G18, G36, G37) y otros dos grupos, los dos algoritmos (G01, G04). Ya durante la sesión de laboratorio el profesor observó que algunos grupos tenían esta confusión y se aclaró, por lo potencialmente esta confusión la podían haber tenido más grupos.

Esta confusión entre problema y algoritmo se da cuando varios grupos expresan, a veces de forma confusa, que no resuelven el mismo problema:

- “Ya que el primer algoritmo tiene una restricción menos, este caso no existe el coste de traslación, y sería un problema relajado” (G06). Este grupo aporta una sugerencia potencialmente constructiva: en su primer informe “El primer algoritmo puede servir como cota mínima del segundo algoritmo”.
- “Los algoritmos con datos de precondición resultan más óptimos y efectivos que los que tienen más datos (coste f)” (G11).
- “Por ello, no nos parece del todo lógico comparar la optimalidad de estos 2 algoritmos entre sí puesto que, realmente, no resuelven exactamente el mismo problema pero aun así vamos a hacer esta comparativa ya que es lo que se nos ha pedido”. “Como ya hemos explicado anteriormente, la introducción del coste de traslado cambia completamente el problema ya que para obtener un resultado óptimo para cualquier caso es necesario el uso de algoritmos más

complejos. Es por eso que no encontramos el sentido de tener que comparar ambos algoritmos cuando son en gran parte distintos" (G18).

- "El primer algoritmo según OptimEx es óptimo en el 100% de los casos, pero como hemos dicho antes el resultado de este no es realista al no incluir el coste de traslado de sede" (G36).

Ocho de los nueve grupos que tenían la práctica mal, la repitieron (salvo G31). El resultado es que 7 grupos corrigieron los problemas señalados. Seis grupos revisaron el primer algoritmo (G06, G08, G11, G18, G36, G37), en otro caso los dos (G04).

Otro grupo que tenía mal los dos algoritmos heurísticos no corrigió los errores (G01). Además, utilizó terminología propia de otros problemas, como "peso" para el coste de traslado. Sin embargo, corrigió los algoritmos heurísticos en la práctica 5b, incluyendo el primero en la primera entrega y ambos en la segunda.

Una incidencia interesante es que había otro grupo que tenía un error menor en los algoritmos heurísticos (G27): en dos condiciones debía escribir '<' en lugar de '<='. Esto no le afectó en esta práctica pero sí en la práctica 3b, donde obtuvo resultados superóptimos. Tras el aviso del profesor, lo corrigió en la segunda entrega.

4.2 Razonamiento sobre Optimidad

El otro aspecto que analizamos es el razonamiento sobre la optimidad de los algoritmos propuestos. Se pedía justificar informalmente su optimidad, en caso de ser óptimos, o presentar contraejemplos sencillos, si eran subóptimos. Dado que los algoritmos eran subóptimos, debían aportare contraejemplos. La tarea era relativamente sencilla sin necesidad de pensarla "en el vacío". Al disponer de los resultados de OptimEx, podían analizar, para cada algoritmo, las características de los casos que habían dado resultados subóptimos y simplificarlos y diseñar contraejemplos más sencillos.

Recordemos que se recogieron 37 informes en el primer plazo de entrega, 12 en el segundo plazo y una tercera entrega fuera de plazo. Por tanto, el número total de informes recogidos es 50.

En la Tabla 2 presentamos las condiciones y resultados del experimento de cada grupo y las características de sus contraejemplos (en general, de su razonamiento sobre optimidad).

Tabla 2. Experimento y contraejemplos de la práctica 2

Grupo	Algoritmos	Condiciones experimento	Propuesto óptimo	Resultados	¿Bien?	Contraej.
1. I	Voraz 1 Voraz 2 Algoritmo "óptimo" (dos bucles anidados)	– Voraces mal – Marcado algoritmo 3 – 50	nº 3	Voraz 2, superóptimo		– Contraej. del 2 (long. 4) – Voraz 1 va a tener "un resultado menor, no el esperado"

	Voraz 2 Algoritmo "óptimo"	– 30 – Maximiza	nº. 3	nº. 3 (maximizar)		– ¡Igual! (no hay alg. 1)
2. I	Voraz 1 Voraz 2	– 100	Ninguno	Ninguno óptimo	✓	– 4 contraejs. con 4 situaciones (generados por OptimEx, tablas de resultados, long. 5)
3. I	Voraz 1 Voraz 2	– 3×20 – Muestra datos de entrada – 3 experi-mentos: sin marcar, marcando voraz 1, y voraz 2	Voraz 2	– Ninguno – Resultados superópts.		– Sin contraej.
	Voraz 1 Voraz 2	– 100	Ninguno	Ninguno	✓	– Contraejs. de los dos (generados por OptimEx, tablas de resultados, long. 12)
4. P	Voraz 1 Voraz 2	– Ambos mal (f. objetivo) – Muestra diálogo generación – 90	Voraz 1	– Resultado ej. del enunciado – Voraz 1		– Contraejs. del 2 (generados por OptimEx, long. 5; manual, long. 4)
	Voraz 1 Voraz 2	– Algoritmos arreglados – Muestra diálogo generación – 38	Ninguno	Ninguno	✓	– Contraej. de los dos (generados por OptimEx, long. 5)
5. I	Voraz 1 Voraz 2	– 5.000 – Muestra diálogo generación	Ninguno	Ninguno	✓	– Contraejs. de los dos – Generados por Optimex (tablas de resultados, long. 10) – Diseñados manualmente (long. 4)

6. I	Voraz 1 Voraz 2	– Voraz 1 mal – 1.000	Voraz 1	Voraz 1		– Justificación 1 – Contraej. del 2 (long. 2)
	Voraz 1 Voraz 2	– Voraz 1 revisado – 1.000	Ninguno	Ninguno	✓	– Contraej. de los dos (igual, long. 2)
7. P	Voraz 1 Voraz 2	– 100 – Muestra diálogo generación	Ninguno	Ninguno	✓	– Contraej. del 2 (long. 6, generado por OptimEx)
8. P	Voraz 1 Voraz 2	– Voraz 1 mal – 100	Voraz 1	Voraz 1		– Justificación 1 – Contraej. del 2 (long. 3)
	Voraz 1 Voraz 2	– Voraz 1 revisado – 100	Ninguno	Ninguno	✓	– Contraejs. del 1 y 2 (longs. 3 y 5)
9. I	Voraz 1 Voraz 2	– 5.000	Ninguno	Ninguno	✓	– Contraejs. para los dos (long. 4)
10. I	Voraz 1 Voraz 2	– 273	Ninguno	Ninguno	✓	– Contraej. del 2 (long. 5)
11. I	Voraz 1 Voraz 2	– Voraz 1 mal – 100 – Muestra diálogo generación	Voraz 1	Voraz 1		– Justificación 1
	Voraz 1 Voraz 2	– Voraz 1 revisado – 100 – Muestra diálogo generación	Ninguno	Ninguno	✓	– Muestra filas de la tabla de resultados pero sin mostrar los datos asociados
12. P	Voraz 1 Voraz 2	– 847 (parece que los ha generado por tiempo)	Ninguno	Ninguno	✓	– Contraej. del 1 (long. 2) – Contraej. del 2 (long. 3)
13. P	Voraz 1 Voraz 2	– 5.000	Ninguno	Ninguno	✓	– Sin contraejs.
	Igual	– Igual	Igual	Igual	Igual	– Contraej. del 2 (long. 5)
14. I	Voraz 1 Voraz 2	– 100, 500	Ninguno	Ninguno	✓	– Contraej. del 2 (long. 5)
15. I	Voraz 1 Voraz 2	– 168	Ninguno	Ninguno	✓	– Se remite a la tabla histórica para el 1 – Contraej. del 2 (long. 4)

16. I	Voraz 1 Voraz 2	– 65	Ninguno	Ninguno	✓	– Contraejs. del 1 y del 2 (generados por OptimEx, long. 4)
17. P	Voraz 1 Voraz 2	– 100	Ninguno	Ninguno	✓	– Contraejs. para los dos (long. 4)
18. P	Voraz 1 Voraz 2	– Voraz 1 mal – 1.000	Voraz 1	Voraz 1		– Contraej. del 2
	Voraz 1 Voraz 2	– Voraz 1 revisado – 4×500	Ninguno	Ninguno	✓	– Contraej. para los dos (long. 3)
19. P	Voraz 1 Voraz 2	– 2, 3	Voraz 2	Voraz 2		– Sin contraejs.
	Voraz 1 Voraz 2	– 400 – Voraz 2 marcado óptimo	Voraz 2	Voraz 1 superópt.		Igual
	Voraz 1 Voraz 2	– 2.077 – Muestra diálogo generación – Probado con datos más pequeños y variando el coste del traslado	Voraz 2	Ninguno		– Contraej. del 1 (long. 2)
20. I	Voraz 1 Voraz 2	– 2×500 – Muestra 2 diálogos generación	Ninguno	Ninguno	✓	– Contraej. 1 (long. 3) – Mejor 2 (generaliza casos)
21. I	Voraz 1 Voraz 2	– 10.000	Ninguno	Ninguno	✓	– Contraej. los dos (long. 2)
22. I	Voraz 1 Voraz 2	– 100	Ninguno	Ninguno	✓	– Contraej. 1 y 2 (long. 2)
23. I	Voraz 1 Voraz 2	– 70	Ninguno	Ninguno	✓	– Contraej. 1 y 2 (long. 4)
24. I	Voraz 1 Voraz 2	– 1.000, 2.500, 4.600 – Muestra 3 diálogos generación	Ninguno	Ninguno	✓	– Contraej. 1 (long. 2) – Mejor 2

25. I	Voraz 1 Voraz 2	– 81	Ninguno	Ninguno	✓	– Contraej. 1 y 2 (long. 4) – Entiende "contrej." al revés, al ser el problema de minimización: caso donde el algoritmo devuelve un valor menor (subóptimo) que el otro
26. I	Voraz 1 Voraz 2	– 108	Ninguno	Ninguno	✓	– Contraej. de los dos (long. 4)
27. I	Voraz 1 Voraz 2	– 917 – Muestra diálogo generación	Ninguno	Ninguno	✓	– Contraej. 1 y 2 (generado por OptimEx, long. 3)
28. P	Voraz 1 Voraz 2	– 22 (1 seg.) – Muestra diálogo de generación, resto de diálogos y datos generados	Ninguno	Ninguno	✓	– Contraej. 1 (long. 4)
	Igual	Igual	Igual	Igual	Igual	– Contraej. de 1 y de 2 (long.4)
29. P	Voraz 1 Voraz 2	– 1.000	Ninguno	Ninguno	✓	– Sin contraejs.
30. I	Voraz 1 Voraz 2	– 5.127 – Muestra diálogo generación	Ninguno	Ninguno	✓	– Contraej. de 1 y 2 (long. 3)
31. I	Voraz 1 Voraz 2	– Voraz 1 mal – 921	Ninguno	Ninguno	✓	– Sin contraejs.
32. I	Voraz 1 Voraz 2	– 100	Ninguno	Ninguno	✓	– Contraej. de 1 y 2 (long. 2)
33. I	Voraz 1 Voraz 2	– 100	Ninguno	Ninguno	✓	– Contraej. de 1 y 2 (long. 2)
34. I	Voraz 1 Voraz 2 Otros 6 voraces Vuelta atrás	– 10, 6×200, 1.200 – Especifica rangos de generación	Vuelta atrás	Vuelta atrás	✓	– Contraejs. de todos los voraces (longs. 2, 3 y 4)

35. P	Voraz 1 Voraz 2	– 237	Ninguno	Ninguno	✓	– Contraej. 1 no lo dan por obvio – Contraej. 2 (generado por OptimEx, long. 10)
36. I	Voraz 1 Voraz 2	– Voraz 1 mal – 1.000	Voraz 1	Voraz 1		– Sin contraejs.
	Voraz 1 Voraz 2	– Voraz 1 revisado – Voraz 2 = 1 (será error) – 1.000	Ninguno	Ninguno	✓	– Contraej. para los dos (long. 3)
37. I	Voraz 1 Voraz 2	– Voraz 1 mal – 1.198	Voraz 1	Voraz 1		– Justificación 1 – Contraej. 2 (long. 2)
	Voraz 1 Voraz 2	– Voraz 1 revisado – 1.007	Ninguno	Ninguno	✓	– Contraej.1 (long. 5) y 2 (long. 2)

La Tabla 3 muestra el razonamiento de los grupos en su primera entrega, teniendo en cuenta su propuesta de algoritmos óptimos y su correspondiente justificación o contraejemplo. Los subrayados corresponden a grupos que hicieron dos entregas.

Tabla 3. Resultados sobre razonamiento de corrección en la práctica 2 (N=37)

Propuesta	**# (%) grupos**	**Grupos**
Ambos algoritmos subóptimos, contraejs. de 2	18 (48’65%)	G02, G05, G09, G12, G15, G16, G17, G21, G22, G23, G25, G26, G27, G30, G32, G33, G34, G35
Primer algoritmo óptimo, justificación, contraej. del 2	3 (8’1%)	G06, G08, G37
Ambos algoritmos subóptimos, contraej. del 1	3 (8’1%)	G20, G24, G28
Ambos algoritmos subóptimos, contraej. del 2	4 (10’8%)	G01, G07, G10, G14
Primer algoritmo óptimo, justificación	1 (2’7%)	G11
Primer algoritmo óptimo, contraej. del 2	2 (5’4%)	G04, G18
Ambos algoritmos subóptimos, sin contraejs,	3 (8’1%)	G13, G29, G31
Primer algoritmo óptimo, sin justificación ni contraejs.	1 (2’7%)	G36
Segundo algoritmo óptimo, sin justificación ni contraejs.	2 (5’4%)	G03, G19

Hemos dividido la tabla en tres partes, distinguiendo tres casos:

- Respuesta completa: grupos que han justificado de forma correcta y completa su conclusión sobre los algoritmos óptimos. Obsérvese que no analizamos si su propuesta es correcta, sino si la justificación es coherente con ella. Asimismo, las valoraciones de optimidad suelen ser superficiales pero simplemente analizamos si han hecho un intento de justificación. Suman 21 grupos (56'8%).
- Respuesta semicompleta: grupos que solamente han razonado sobre uno de los algoritmos. Suman 10 grupos (27%).
- Respuesta incompleta: grupos que no han razonado sobre ningún algoritmo. Suman 6 grupos (16'2%).

Algunos casos de la primera subcategoría (5 de 18) han cuidado la selección de sus contraejemplos:

- Un grupo (G02) distingue 4 casos de optimidad relativa (ambos algoritmos con resultado óptimo, cada uno óptimo por separado y ninguno óptimo, siendo en este último caso el contraejemplo común a ambos algoritmos)
- Dos grupos (G05, G16) distinguen 3 casos (los anteriores menos el caso de ningún algoritmo óptimo).
- Dos grupos (G21, G26) presentan un contraejemplo común a ambos algoritmos heurísticos.

Otro grupo (G34) de la misma subcategoría presenta diversos contraejemplos para los dos algoritmos del enunciado ¡y para 6 más, de diseño propio!

Sin embargo, otros grupos de la misma subcategoría hicieron aportaciones de contraejemplos del primer algoritmo heurístico con peor calidad:

- Sólo presenta los resultados de los contraejemplos (pero no los datos en sí) en la tabla histórica (G15).
- Dados los peores resultados del algoritmo primero, da por obvio encontrar contraejemplos (G35).

Asimismo, un grupo interpretó el concepto de contraejemplo al revés (G25): ejemplo en el que se muestra que el algoritmo calcula un valor óptimo (en lugar de subóptimo). Sin embargo, mostró que era capaz de aportar ejemplos razonados.

En el subapartado anterior comentamos que algunos grupos implementaron mal el primer algoritmo heurístico pero afirmaban o intuían que el proceso no era coherente. Estos grupos quedan repartidos entre las 3 categorías (1, 2, 1) pero ha afectado a su respuesta en este apartado.

La entrega fue repetida por 13 grupos, aunque dos de ellos presentan el apartado de razonamiento sin variación alguna (G01, G19). El primero simplemente eliminó ("ocultó") el algoritmo primero, sin modificar el apartado de razonamiento. El otro grupo realizó dos entregas: en la segunda no modificó nada de este apartado, mientras que en la tercera aportó un contraejemplo del algoritmo subóptimo (pero no justificación del óptimo).

De los otros 11 grupos que repitieron la entrega, 10 la resolvieron de forma correcta y completa. En la Tabla 4 se muestra la evolución de todos los grupos. Los grupos que presentaron una entrega distinta aparecen subrayados.

De nuevo, podemos resaltar algunos informes recatalogados en la categoría correcta:

- Dos grupos (G06, G18) aportan un contraejemplo simultáneo de ambos algoritmos heurísticos.

Asimismo, encontramos algunos informes poco satisfactorios:

- Un grupo (G11) presenta los contraejemplos en la tabla histórica, con sus resultados, pero sin mostrar los datos.
- Un grupo (G13) proporciona un contraejemplo del algoritmo segundo pero considera innecesario darlo para el primero, dado que calcula una solución óptima en menos casos.

También es interesante conocer la naturaleza de los razonamientos realizados. Los resultados se muestran en la Tabla 4 para las dos entregas.

Tabla 4. Resultados sobre razonamiento de corrección en las dos entregas de la práctica 2 (N=37)

Propuesta	**# (%) grupos**	**Grupos**	**# (%) grupos**	
Ambos algoritmos subóptimos, contraejs. de 2	18 (48’65%)	G02, G05, G09, G12, G15, G16, G17, G21, G22, G23, G25, G26, G27, G30, G32, G33, G34, G35	28 (75’7%)	G02, G03, G0 G11, G12, G1 G21, G22, G2 G30, G32, G3
Primer algoritmo óptimo, justificación, contraej. del 2	3 (8’1%)	G06, G08, G37		
Ambos algoritmos subóptimos, contraej. del 1	3 (8’1%)	G20, G24, G28	2 (5’4%)	
Ambos algoritmos subóptimos, contraej. del 2	4 (10’8%)	G01, G07, G10, G14	4 (10’8%)	G01,
Primer algoritmo óptimo, justificación	1 (2’7%)	G11		
Primer algoritmo óptimo, contraej. del 2	2 (5’4%)	G04, G18		
Segundo algoritmo óptimo, contraej. del 1			1 (2’7%)	
Ambos algoritmos subóptimos, sin contraejs,	3 (8’1%)	G13, G29, G31	2 (5’4%)	
Primer algoritmo óptimo, sin justificación ni contraejs.	1 (2’7%)	G36		
Segundo algoritmo óptimo, sin justificación ni contraejs.	2 (5’4%)	G03, G19		

Cuatro grupos propusieron, en su primera entrega, el primer algoritmo heurístico como óptimo y lo justificaron (G06, G08, G11, G37). Su justificación es de las siguientes formas:

- Su resultado es igual a la suma del coste mínimo de cada mes (G06, G37).
- En cada etapa dan el menor coste posible (G08, G11)

En cuanto a los contraejemplos, vemos si los diseñaron manualmente o aprovecharon casos generados aleatoriamente por OptimEx. La Tabla 5 muestra las dos formas de generación. Utilizamos como referencia la última entrega de cada grupo, al ser la más completa. Hubo 2 grupos que no aportaron contraejemplos en dicha entrega (G29, G31).

Tabla 5. Forma de producir los contraejemplos de la última entrega de la práctica 2 (N=35)

Fuente	**# (%) grupos**	**Grupos**
Manual	19 (54’3%)	G01-2, G06-2, G09, G12, G15, G17, G18, G19-3, G20, G21, G22, G24, G25, G28-2, G30, G32, G33, G34, G36-2
OptimEx	10 (28’6%)	G02, G03-2, G04-2, G07, G08-2, G11-2, G16, G26, G27, G35
Ambos	2 (5’7%)	G05, G37-2
Indeterminado	4 (11’4%)	G10, G13-2, G14, G23

Es difícil determinar con seguridad el origen de numerosos contraejemplos. Podemos afirmarlo de aquellas prácticas con contraejemplos muy pequeños y con valores pequeños o regulares (origen manual) o con contraejemplos grandes o mostrados en las tablas de OptimEx (generados por OptimEx).

Veamos también el tamaño de los contraejemplos. En este caso, algunos grupos aportan contraejemplos de varias longitudes (véase Tabla 6). Obsérvese que el número total de contraejemplos (41) difiere del número de grupos (37) porque hay grupos que aportan contraejemplos de tamaños diversos (G12 y G37-2 aportan 2, G34 aporta 3).

Tabla 6. Tamaño de los contraejemplos en la última entrega de la práctica 2 (N=41)

Tamaño	**# (%) grupos**	**Grupos**
2	10 (24’4%)	G06-2, G12, G19-3, G21, G22, G24, G32, G33, G34, G37-2
3	7 (17’1%)	G08-2, G12, G18-2, G20, G30, G34, G36-2
4	12 (29’3%)	G01-2, G05, G09, G15, G16, G17, G23, G25, G26, G27, G28-2, G34
5	7 (17’1%)	G02, G04-2, G08-2, G10, G13-2, G14, G37-2
6	1 (2’4%)	G07
10	2 (4’9%)	G05, G35
12	2 (4’9%)	G03-2, G11-2

5 Práctica 3a: Algoritmos de Búsqueda

Se recogieron un total de 44 informes de 35 grupo (23 individuales y 12 parejas). Estos informes se entregaron de la siguiente forma:

- Primer plazo de entrega: 34 envíos, 31 del grupo presencial (19 individuales y 12 parejas) y 3 del grupo on-line (todos individuales).
- Segundo plazo: 10 envíos, 9 del grupo presencial (6 individuales –uno de ellos por primera vez– y 3 parejas) y 1 del grupo on-line (individual).

Presentamos los resultados del cuestionario en la Tabla 7. La tabla incluye una fila por cada informe entregado. Las columnas tienen el siguiente significado:

- "Grupo". Asigna un número a cada grupo, indicando también si es individual "I" o una pareja "P".
- "Árbol de búsqueda". Indica si el carbol de búsqueda diseñado para resolver este problema es correcto.
- "Comprobación de validez". Indica si se ha identificado correctamente esta comprobación.
- "Implementación incremental". Indica si el algoritmo de vuelta atrás realiza incrementalmente los cálculos necesarios para construir y comprobar soluciones. Se ha examinado directamente el código de cada grupo, ya que parecen no entender el significado de "implementación incremental" (el grupo G08 llega a decirlo expresamente), aunque lo aplican. Esta columna se repite posteriormente para el algoritmo de ramificación y poda.
- "Algoritmo de vuelta atrás". Identifica el tipo de algoritmo desarrollado en este apartado.
- "Definición de función de cota". Identifica la función de cota diseñada.
- "Algoritmo de ramificación y poda". Identifica el tipo de algoritmo desarrollado en este apartado.
- "Comentarios abiertos". Comentarios realizados por los grupos en el apartado de conclusiones del informe.

A efectos del análisis posterior, la práctica entregada por el grupo G10 en el segundo plazo es considerada igual que las entregadas en el primer plazo, ya que no es el resultado de una mejora. Por tanto, consideramos que hay 35 primeras entregas y 9 segundas entregas.

Tabla 7. Análisis de los informes de la práctica 3a

Grupo	Árbol de búsqueda	Comprobación de validez	Implementación incremental	Algoritmo de vuelta atrás	Definición de función de cota	Implementación incremental	Algoritmo de ramificación y poda	Comentarios abiertos
1. I	✓	Confunde con coste de cada solución	Calcula el coste en cada "hoja" Almacena todas las soluciones	Enumeración (a partir de representación binaria, no incrementalmente)	Confunde con mejor solución encontrada	–	Vuelta atrás	
	✓	✓	Copia vector y calcula el coste en cada nodo	Vuelta atrás (mezcla de estilo declarativo y con acumulador)	Confunde con soluciones construidas y mejor solución encontrada	×	Vuelta atrás	
2. I	✓	–	Calcula el coste en cada hoja	Enumeración	✓	×	Ramificación y poda (ineficiente en cada nodo)	– OptimEx ha ayudado a encontrar problemas con la definición de cota
3. I	✓	–		Algoritmo voraz	✓	×	Algoritmo voraz	– Sugiere uso de vídeos para teoría y ejs
4. P	✓	✓	Calcula el coste en cada hoja	Enumeración	Confunde con mejor solución encontrada	–	Vuelta atrás	
5. I	✓	✓	✓	Vuelta atrás	✓	✓	Ramificación y poda	
6. I	✓	–	✓ (solución en un vector de 1 elemento)	Vuelta atrás (mezcla de estilo declarativo y con acumulador)	Confunde con mejor solución encontrada	–	Vuelta atrás	– Ha usado OptimEx
7. P	✓	✓	✓	Vuelta atrás	Confunde con mejor solución encontrada	–	Vuelta atrás	– Han medido tiempos y comprobado que el segundo algoritmo tarda menos que el primero
	✓	✓	✓	Vuelta atrás	✓	✓	Ramificación y poda	– Ídem

8. P	✓	✓	✓	Vuelta atrás	✓ (confusa porque la parte conocida se computa aparte)	✓	Ramificación y poda	– No comprenden qué significa "implementación incremental"
9. I	✓	✓	✓	Vuelta atrás	✓ (no computa coste del resto de la solución)	✓	Ramificación y poda	– Práctica difícil
10. I	✓	Confunde con mejor solución encontrada	✓	Vuelta atrás	✓	✓	Ramificación y poda	(Entregada en el segundo plazo)
11. I	✓	–	✓	Vuelta atrás	Confunde con mejor solución encontrada (valor inicial igual al menor de los costes de estar siempre en una sede o en la otra)	–	Vuelta atrás	– No ha tenido ningún problema de minimización resuelto que le sirva de guía
	✓	–	✓	Vuelta atrás	✓ (no calcula una cota inicial inferior, sino el menor de los costes de estar siempre en una sede o en la otra)	✓	Ramificación y poda	– Ídem
12. P	✓	Confunde con mejor solución encontrada	✓	Vuelta atrás	Confunde con mejor solución encontrada	–	Vuelta atrás	
	✓	✓	✓	Vuelta atrás	✓ (no calcula una cota inicial inferior, sino el coste de estar siempre en la sede primera)	✓	Ramificación y poda	

13. P	✓	✓	✓	Vuelta atrás	✓ (no calcula una cota inicial inferior, sino el menor de los costes de estar siempre en una sede o en la otra)	No se actualiza	Ramificación y poda	– Dificultad de la técnica por: recursividad, retroceso – Utilidad de los ejs proporcionados
14. I	✓	Confunde con mejor solución encontrada		Algoritmo confuso (código propio)	✓ (no calcula una cota inicial inferior, sino el menor de los costes de estar siempre en una sede o en la otra)	No se actualiza	Algoritmo confuso (código propio)	– Práctica difícil – Se sobrevalora
15. I	✓	Confunde con precondición	✓	Vuelta atrás	Confunde con mejor solución encontrada	–	Vuelta atrás	– Le resultó difícil diseñar el árbol de búsqueda
	✓	Confunde con precondición	✓	Vuelta atrás	Confunde con el coste de cada etapa	–	Ramificación y poda	– Ídem
16. P	✓	Confunde con solución completa	✓ (solución en un vector de 1 elemento)	Vuelta atrás	Confunde con mejor solución encontrada	Valor inicial igual al el coste de estar siempre en la sede primera	Vuelta atrás	– Les ha costado bastante – Han usado SRec
17. P	✓	✓	✓ (solución en un vector de 1 elemento)	Vuelta atrás	Confunde con mejor solución encontrada	Valor inicial igual a la solución del segundo algoritmo heurístico	Vuelta atrás	– Consideran que búsqueda es la técnica a usar en problemas de este tipo
18. P	✓	–	✓	Vuelta atrás	✓ (no calcula una cota inicial inferior, sino el mayor de los costes de estar siempre en una sede o en la otra)	Va disminuyendo (inspirado en mochila 0/1)	Ramificación y poda	– Reconocen que la segunda parte no la han acabado bien

	✓	Confunde con mejor solución encontrada	✓	Vuelta atrás	✓	No se actualiza	Ramificación y poda	– Han modificado el algoritmo de vuelta atrás mirando otros ejemplos – Modificación de vuelta atrás para obtener ramificación y poda, fácil
19. I	✓	Confunde con: – solución completa, y – comparación de soluciones completas	✓	Vuelta atrás	✓ (no calcula una cota inicial inferior, sino el menor de los costes de estar siempre en una sede o en la otra)	×	Ramificación y poda	– Ha usado OptimEx
20. I	✓	✓	✓ (solución en un vector de 1 elemento)	Vuelta atrás	Confunde con mejor solución encontrada	Valor inicial igual al el coste de estar siempre en la sede primera	Vuelta atrás	
21. I	✓	–	✓	Vuelta atrás (toma un valor inicial máximo a partir de los datos de entrada)	✓	✓	Ramificación y poda	– Práctica difícil – No le gusta (y le aburre) tener que documentar
22. P	✓	Confunde con mejor solución encontrada	✓	Vuelta atrás (toma como valor inicial una cota inferior)	Confunde con mejor solución encontrada	–	Vuelta atrás (toma como valor inicial una cota inferior)	– Dificultad de determinar el valor inicial de una solución
23. I	✓	Confunde con: – solución completa, y – comparación de soluciones completas	✓	Vuelta atrás	✓	✓	Ramificación y poda	– Ha usado OptimEx ¡con cada algoritmo individual! – Concepto de cota difícil

24. I	✓	✓	✓	Vuelta atrás	Confunde con mejor solución encontrada	–	Vuelta atrás	– "Algoritmo aparentemente tedioso" – "Ha sido necesario el uso del algoritmo voraz para realizar algunas comprobaciones sobre las primeras soluciones que daban los nuevos algoritmos"
	✓	✓	✓	Vuelta atrás	✓	×	Ramificación y poda	– Ídem
25. I	✓	✓	✓	Vuelta atrás (valor inicial igual al mayor de los costes de estar siempre en una sede o en la otra)	Confunde con mejor solución encontrada	–	Vuelta atrás (valor inicial igual a suma de costes mayores)	– Le gusta la mezcla de recursividad e iteración – Dificultad de depurar el algoritmo de vuelta atrás – Ha medido tiempos mucho mejores con su algoritmo de "ramificación y poda"
26. P	✓	✓	✓	Vuelta atrás	✓ (no calcula una cota inicial inferior, sino el menor de los costes de estar siempre en una sede o en la otra)	✓ (como variable global)	Ramificación y poda (se compara con el coste de un mes)	
27. P	✓	✓	✓	Vuelta atrás	✓ (no calcula una cota inicial inferior, sino el coste de estar siempre en la segunda sede)	×	Ramificación y poda	– Dificultad de determinar la función de cota
28. I	✓	✓	✓	Vuelta atrás	Confunde con mejor solución encontrada (más otras confusiones)	✓	Vuelta atrás (valor inicial igual a suma de costes de la primera sede)	– Ha usado OptimEx

	✓	✓	✓	Vuelta atrás	✓	× (no se actualiza)	Ramificación y poda	– Compara ambos algoritmos contando el número de llamadas recursivas
29. I	✓							– Incompleto: necesita una tutoría
30. I	✓	✓	✓	Vuelta atrás	Confunde con mejor solución encontrada	–	Vuelta atrás	– Utilidad de esquema – Dificultad de manejo de variables
31. I	✓	✓	✓ (guarda vector de tantos costes como meses)	Vuelta atrás	Confunde con mejor solución encontrada	–	Vuelta atrás	
32. I	✓	✓	✓	Vuelta atrás	✓	✓	Ramificación y poda	– Diseña otro algoritmo de ramificación y poda mejorado – Compara rendimiento de algoritmos – Dificultad de diseñar funciones de cota
33. P	✓	− (no lo entiende)	Calcula el coste en cada hoja	Enumeración	Confunde con mejor solución encontrada	Al no calcular incrementalmente el coste de la solución, la solución parcial (“cota”) siempre vale cero	Vuelta atrás	
34. I	✓	✓	✓	Vuelta atrás (mezcla de estilo declarativo y con acumulador)	Confunde con mejor solución encontrada	–	Vuelta atrás	– Le resulta más sencillo no usar el esquema de la asignatura

35. I	✓	✓	✓	Vuelta atrás	Confunde con mejor solución encontrada	–	Vuelta atrás	– Ha usado OptimEx con el algoritmo de vuelta atrás – Dificultad de depuración del algoritmo de vuelta atrás
	✓	✓	✓	Vuelta atrás	✓	✓	Ramificación y poda	– Ídem

La primera observación a partir de los resultados es que todos los alumnos diseñaron un árbol de búsqueda correcto.

Como elemento de diseño del algoritmo de vuelta atrás, también se pedía identificar la condición de validez a comprobar sobre las soluciones parciales. La Tabla 8 muestra los resultados.

Tabla 8. Identificación de la condición de validez en los algoritmos de vuelta atrás de la práctica 3a (N=35)

Condición de validez	**# (%) grupos**	**Grupos**
Correcta	17 (48’6%)	G04, G05, G07, G09, G13, G17, G20, G24, G25, G26, G27, G28, G30, G31, G32, G34, G35
Comparación entre nueva solución y mejor solución actual	5 (14’3%)	G08, G10, G12, G14, G22
Comprobación de solución completa	4 (11’4%)	G01, G16, G19, G23
Precondición	1 (2’9%)	G15
Ninguna	8 (22’9%)	G02, G03, G06, G11,G18, G21, G29, G33

Puede comprobarse que aproximadamente la mitad de los grupos identificaron correctamente la comprobación de validez a satisfacer las soluciones parciales (en este caso, todas las soluciones parciales son válidas, es decir, no deben satisfacer ninguna comprobación de validez).

Las dos categorías siguientes corresponden, respectivamente a los grupos que identificaron que debía compararse cada nueva solución completa con la mejor solución encontrada hasta el momento y que debía determinarse cuándo una solución parcial ya era una solución completa.

El grupo G15 confundió la comprobación de validez con la precondición del problema (datos de entrada positivos). Por su respuesta a la implementación incremental, este grupo tenía una grave incomprensión de los elementos de la especificación de un problema de optimización.

La categoría de “ninguna” incluye grupos que no identifican ninguna comprobación. Con frecuencia, dan una explicación del proceso de búsqueda.

Nueve grupos realizaron una segunda entrega, de los que 4 tenían este apartado bien. De los 5 grupos restantes, 2 grupos que habían contestado algo, lo corrigieron satisfactoriamente (G01, G12), 2 no lo corrigieron (G11, G15) y otro grupo pasó de no identificar ninguna comprobación a la comparación de una nueva solución con la mejor hasta el momento (G18).

La clase de algoritmo desarrollado está relacionado parcialmente con la implementación incremental. El resultado se muestra en la Tabla 9.

Tabla 9. Clases de algoritmos desarrollados la práctica 3a (N=35)

Clase de algoritmo	# (%) grupos	Grupos
Vuelta atrás	28 (80%)	G05, G06, G07, G08, G09, G10, G11, G12, G13, G15, G16, G17, G18, G19, G20, G21, G22, G23, G24, G25, G26, G27, G28, G30, G31, G32, G34, G35
Enumeración	4 (11’4%)	G01, G02, G04, G33
Voraz	1 (2’9%)	G03
Confuso	1 (2’9%)	G14
Ninguno	1 (2’9%)	G29

Puede observarse que 28 de los 35 grupos (80%) desarrollan un algoritmo de vuelta atrás. Asimismo, 4 grupos (11’4%) desarrollaron algoritmos de enumeración. La diferencia entre ambas clases de algoritmos es que esos últimos no realizan operaciones incrementalmente, sino solamente en los nodos de hoja. En este problema, dichas operaciones se limitan al cálculo del coste de cada solución completa, no teniendo excesiva influencia sobre el rendimiento del algoritmo. Sin embargo, en problemas con comprobaciones de validez, pueden tener una gran influencia sobre el tamaño del árbol de búsqueda generado.

Un algoritmo de enumeración (G01) generó las soluciones contando con números binarios en lugar de basarse en un árbol de búsqueda.

Los 3 grupos restantes desarrollan algoritmos distintos: voraz (G03), confuso (G14, con una estructura de bucles anidados) y ningún algoritmo (G29).

A veces, las soluciones correctas contienen detalles que reducen su calidad:

- Utiliza un vector para guardar los sucesivos costes que se van generando en cada rama del árbol de búsqueda (G31).
- Utilizar un vector de un solo elemento (G06, G16, G17, G20, G31). Dado lo peculiar de este detalle, podría comprobarse si hay evidencias de plagio.

Un detalle curioso es el valor inicial tomado como el mejor valor conocido. La Tabla 10 muestra estos valores, salvo para cinco grupos: los 3 grupos antes identificados con soluciones que no son de vuelta atrás ni de enumeración (G03, G14, G29) y dos grupos que presentan algoritmos con estilos relativamente declarativos, no precisando un valor inicial (G01, G02).

Tabla 10. Mejor valor inicial usado por el algoritmo de vuelta atrás en la práctica 3a (N=30)

Mejor valor inicial	# (%) grupos	Grupos
Integer.MAX_VALUE	9 (30%)	G05, G06. G08, G13, G17, G19, G23, G26, G35
MAX	1 (3’3%)	G11
1.000.000	1 (3’3%)	G31
10.000	1 (3’3%)	G04
9.999	1 (3’3%)	G12
1.000	2 (6’7%)	G27, G30

0	2 (6’7%)	G09, G34
-1	6 (20%)	G10, G15, G18, G24, G28, G32
Valor guardado en celda especial	1 (3’3%)	G07
Máximo del coste de sede en cada mes más traslado	1 (3’3%)	G21
Máximo de la suma de los costes de una y otra sede	1 (3’3%)	G25
Suma de costes de la primera sede	3 (10%)	G16, G20, G33
Mínimo del coste de sede en cada mes	1 (3’3%)	G22

Puede observarse que la mitad de los grupos presentan valores muy grandes, aunque siendo estrictos solamente es correcto el valor inicial Integer.MAX_VALUE. El valor MAX debe ser una constante global declarada en la clase Java, pero no incluida en el informe.

Otros 9 grupos (30%) presentan valores que pueden recibir una interpretación de caso especial. Destaca el caso -1 (6 casos, 20%), probablemente influido por el uso de este valor en problemas de maximización.

Otros 6 casos (20%) calculan un valor a partir de los datos de entrada, bien un valor que no va a ser solución óptima (G21), un valor correspondiente a una solución válida (G25, G16, G20, G33) o incluso u valor inferior al óptimo (G22). Obviamente, este último caso es incorrecto.

Casi todos los grupos que hicieron una segunda entrega tenían bien el algoritmo de vuelta atrás en su primera entrega. La excepción fue G01, que corrigió su algoritmo de enumeración por otro de vuelta atrás, aunque con ineficiencias.

Veamos la definición de funciones de cota. Los resultados en muestran en la Tabla 11.

Tabla 11. Definición de funciones de cota en la práctica 3a (N=35)

Función de cota	**# (%) grupos**	**Grupos**
Correcta	9 (25’7%)	G02, G03. G05, G08, G09, G10, G21, G23, G32
No proporciona una cota inferior	6 (17’1%)	G13, G14, G18, G19, G26, G27
Comprobación extra con mejor solución encontrada	19 (54’3%)	G01, G04, G06, G07, G11, G12, G15, G16, G17, G20, G22, G24, G25, G28, G30, G31, G33, G34, G35
Ninguna	1 (2’9%)	G29

Una cuarta parte de los alumnos formularon una cota inferior válida (el menor de los costes de estar siempre en una sede u otra).

Otra categoría de alumnos diseñaron una cota aparentemente razonable pero que en realidad no era una cota inferior. Las cotas definidas por estos grupos son:

- Calcular el coste de estar siempre en una sede o en otra, quedándose con el valor menor de ambas posibilidades (G13, G14, G19, G26).
- Igual pero calculando el mayor de ambas situaciones (G18).
- Igual pero calculándolo sólo para la segunda sede (G27).

El conjunto de ambas categorías engloba a alumnos que han comprendido el concepto de función de cota. La dificultad de la segunda categoría podría estar relacionada con que el problema propuesto es de minimización, menos intuitivo y que se habían visto con menos detalle en clase.

Una tercera categoría agrupa a los alumnos que añadieron una comprobación extra, que es correcta para problemas de minimización pero no es una función de cota: continuar sólo mientras la solución en construcción sea menor que la mejor solución encontrada hasta el momento. Suponen más de la mitad de los alumnos (54’3%). Consideramos a estas soluciones correspondientes a la técnica de vuelta atrás.

Veamos algunas peculiaridades menores de las soluciones. La solución de algunos grupos de la primera categoría es deficiente:

- Dos grupos (G02, G19) calculan el valor completo de la cota en cada nodo del árbol de búsqueda.
- G08 la presenta de forma confusa porque la parte conocida y la previsión de la cota se calculan en partes distintas del algoritmo.
- G09 omite en su cota la parte posterior a la etapa actual, resultando en una cota muy inferior y, previsiblemente, de escasa utilidad.
- G26 compara el valor de la cota con el coste del alquiler del mes actual (en lugar de la mejor solución encontrada hasta el momento).

Algunas incidencias de los grupos de la segunda categoría son:

- No actualizar el valor inicial de la cota (G13, G14).
- La cota se va actualizando de forma que su valor disminuye (G18).

Algunas soluciones de vuelta atrás también presentan la peculiaridad de que no parten de un valor de una solución inicial ficticia (infinito), sino de:

- El coste de estar siempre en la primera sede (G16, G20, G28).
- El coste menor de estar siempre en una de las dos sedes (G11).
- La cota inferior (G22).
- Un valor calculado de forma similar a la cota pero tomando costes mayores (G25).
- El valor calculado por el segundo algoritmo heurístico de la práctica 2 (G17).

La mayor parte de las segundas entregas (7 de 9) correspondían a grupos de la tercera categoría, es decir, que creían diseñar según ramificación y poda (por realizar una comprobación extra). En general, estas segundas entregas suponían una mejora pero seguían teniendo deficiencias:

- Un grupo mejoró su algoritmo de ramificación y poda (G18), pero sin actualizar su cota.
- Dos grupos corrigieron satisfactoriamente sus algoritmos de vuelta atrás, desarrollando algoritmos de ramificación y poda (G07, G35).
- Dos grupos corrigieron satisfactoriamente sus algoritmos de vuelta atrás, desarrollando algoritmos de ramificación y poda, pero sin actualizar la cota de forma incremental (G24, G28).
- Tres grupos corrigieron sus algoritmos de vuelta atrás, desarrollando algoritmos de ramificación y poda, pero con deficiencias más graves: calcular una cota inferior poco útil (G15: igual al coste en la etapa actual) o calcular cotas superiores (G11: mínimo del coste de estar siempre en una sede o en la otra, G12: coste de estar siempre en la primera sede),
- Un grupo cambió su algoritmo de enumeración por un algoritmo de vuelta atrás (G01).

Por último, los alumnos han incluido una gran diversidad de comentarios abiertos. Los comentarios relevantes sobre las técnicas de búsqueda o sobre la práctica son:

- Dificultades de la técnica (9 comentarios):
  - Definición de las funciones de cota (G22, G23, G27, G32).
  - Elementos básicos de las técnicas (G08: “implementación incremental”, G13: recursividad, G15: árbol de búsqueda, G34: esquema de programación).
  - Falta de ejemplos (G11).
- Práctica difícil, o más difícil que otras anteriores (7 comentarios: G09, G14, G16, G21, G29, G30, G35).
- Han usado programas auxiliares con distintos fines, sobre todo OptimEx pero también SRec (7 comentarios: G02, G06, G16, G19, G23, G28-1, G35).
- Han comparado los dos algoritmos de búsqueda para comprobar que el algoritmo de ramificación y poda es más eficiente que el de vuelta atrás (4 comentarios: G07, G25, G28-1, G32).
- Sugieren diversa mejoras o reconocen virtudes (4 comentarios) (G03: uso de vídeos, G13 y G18-2: más ejemplos, G30: utilidad del esquema).

6 Resumen de Resultados

Resumimos en primer lugar los hallazgos de la práctica 1, sobre la implementación eficiente de algoritmos voraces:

- Los alumnos perciben la conveniencia de ordenar los candidatos siempre que sea posible.
- Dos terceras partes de los alumnos utilizan la técnica de ordenación mediante índices. Sin embargo, el uso de vectores innecesarios y el tercio restante de alumnos hacen ver que la técnica resulta difícil de aprender y dominar

La implementación de dos algoritmos heurísticos, esbozados en el enunciado de la práctica 2, dio lugar al sorprendente resultado de que casi un cuarto de los alumnos (24’3%) tuvieron dificultades conceptuales, relacionadas con problema, algoritmo, función de selección y función objetivo.

Los resultados del apartado de razonamiento sobre estos algoritmos heurísticos son variados:

- El porcentaje de grupos que presentó un razonamiento completo (bien dos contraejemplos bien justificación más contraejemplo).fue alto en la primera entrega (56’8%). El resto quedó repartido entre los que presentaron un razonamiento incompleto (27%) o ninguno (16’2%). Aun así, incluso los grupos de la primera clase presentan diversas dificultades en sus aportaciones, reduciéndose a un porcentaje bajo (13’5%) los que aportaron contraejemplos cuidados.
- El porcentaje de grupos que presentó un informe completo en la segunda entrega, tras corregir el apartado siguiendo los comentarios del profesor, fue muy alto (75’7%), quedando muy bajo (5’4%) el porcentaje de los que no presentaron nada.
- Resulta difícil tener certeza de qué alumnos presentaron contraejemplos diseñados por ellos mismos o tomados de los datos generados por OptimEx. Aun así, es casi el doble el número de los primeros (54’3%% frente a 28’6%, sin contar los grupos que presentan ejemplos obtenidos de ambas fuentes).
- Casi todos los grupos (87’8%) aportan contraejemplos de tamaño pequeño, comprendido entre 2 y 5.
- Podemos concluir que el diseño de contraejemplos es una tarea fácil para los alumnos, al menos combinada con una experimentación con OptimEx. Sin embargo, convendría reforzar su docencia en varios aspectos: significado de contraejemplo, criterios de sencillez y “elegancia” de un contraejemplo, y obtención de un contraejemplo sencillo a partir de datos más grandes usados en un experimento.

Asimismo, los resultados de la práctica 3a son:

- Todos los alumnos diseñaron correctamente un árbol de búsqueda para resolver el problema planteado.
- Los alumnos tienen dificultades para identificar las comprobaciones de validez. Aunque la mitad de los alumnos las identificaron, la otra mitad, no. Estos alumnos se dividen entre los que no parecen haber entendido su significado (22’9%), los que las confunden con las comparaciones de valores entre cada nueva solución completa y la mejor solución encontrada hasta el momento (14’3%) y la comprobación de que la solución parcial ya está completa (11’4%).
- Los alumnos han desarrollado de forma mayoritaria algoritmos de vuelta atrás (80% en primera entrega) o de enumeración (11’4%).
- Los valores iniciales tomados para el proceso de búsqueda indican un cierto desconocimiento del valor neutro de la operación mín.
- Casi la mitad de los grupos (42’9%) han definido funciones de cota. Sin embargo, solamente el 25’7% de las definiciones proporcionan una cota

inferior (para un problema de minimización), mientras que el 17’1% restante dan lugar a cotas superiores.
- El 54’3% de los alumnos han confundido la definición y uso de cotas con la comprobación extra que puede hacerse (sólo en problemas de minimización) entre el valor de la solución parcial en construcción con el valor de la mejor solución encontrada hasta el momento.
- La dificultad de los alumnos para diseñar algoritmos de ramificación y poda también es evidente en los numerosos errores contenidos en sus soluciones, incluso en las segundas entregas.
- Según los comentarios abiertos de los alumnos, es convenientes usar programas auxiliares, mostrar más ejemplos y comparar el rendimiento de los distintos algoritmos de búsqueda.

7 Recomendaciones

A partir de los resultados de la sección anterior, podemos extraer varias conclusiones sobre las distintas técnicas de diseño.

En primer lugar, podemos extraer una conclusión general para la asignatura. Los alumnos tienen dificultades para identificar las distintas partes de la especificación de un problema de optimización y su correspondencia con las partes de los algoritmos desarrollados con diversas técnicas de diseño (al menos, algoritmos heurísticos y de búsqueda). Al igual que la experimentación con la optimidad, parece necesario que este aspecto se trate al comienzo de la asignatura y se trate recurrentemente con cada técnica de diseño.

Varios aspectos de las técnicas evaluadas necesitarían más atención, sobre todo en forma de más ejemplos o ejercicios para los alumnos, con más exposición es estos a situaciones problemáticas. Es el caso de la ordenación de candidatos en los algoritmos voraces mediante la presentación de resultados tras una ordenación directa o sus efectos laterales.

Algo similar puede decirse del diseño de contraejemplos, combinado con la experimentación usando OptimEx. En este caso, convendría reforzar varios aspectos: significado de contraejemplo, criterios de sencillez y “elegancia” de un contraejemplo, y obtención de un contraejemplo sencillo a partir de datos más grandes usados en un experimento.

En cuanto a las técnicas de búsqueda, podemos afirmar lo siguiente:

- Los alumnos no suelen encontrar dificultad en el diseño de los árboles de búsqueda.
- Los alumnos no tienen excesivas dificultades para programar algoritmos de vuelta atrás, pero hay que resaltar que se evite la técnica de enumeración a favor de la técnica de vuelta atrás.
- Los alumnos tienen dificultades para distinguir elementos que varían entre problemas de maximización y de minimización, al menos para el uso de valores iniciales de búsqueda y de cotas.

8 Conclusiones

Hemos presentado de forma detallada el análisis de tres prácticas, desarrolladas en otoño de 2015 y relacionadas con algoritmos de optimización. Las tareas analizadas son: determinación de las decisiones óptimas en algoritmos voraces con ordenación inicial de los candidatos, implementación de algoritmos heurísticos, razonamiento sobre la optimidad de algoritmos heurísticos, y diseño de algoritmos de vuelta atrás o de ramificación y poda. Se ha incluido el procedimiento y los enunciados de prácticas usados, los resultados detallados y comentados, así como una discusión de los mismos. Los resultados has sido útiles para valorar mejor las dificultades reales de los alumnos en estas cuestiones.

Agradecimientos. Este trabajo se ha financiado con los proyectos TIN2015-66731-C2-1 del Ministerio de Economía y Competitividad de España y S2013/ICE-2715 de la Comunidad Autónoma de Madrid.

Referencias

1. Velázquez Iturbide, J.Á.: An experimental method for the active learning of greedy algorithms. ACM Transactions on Computing Education 13, 4 (octubre 2013) artículo 18, DOI 10.1145/2534972
2. Velázquez Iturbide, J.Á.: Identification and removal of misconceptions on optimization concepts underlying greedy algorithms. Journal of Research and Practice in Information Technology 45, 3/4 (2013) 203-217
3. Velázquez Iturbide, J.Á.: Difficulties, attitudes and misconceptions on experimenting with optimization algorithms. En: Proceedings of the 2014 International Symposium on Computers in Education (SIIE'14), IEEE Xplore (2014) 17-22, DOI 10.1109/SIIE.2014.7017698
4. Velázquez Iturbide, J.Á.: Una segunda evaluación cualitativa de la comprensión de la optimalidad. En: Serie de Informes Técnicos DLSI1-URJC, no. 2015-02, Departamento de Lenguajes y Sistemas Informáticos I, Universidad Rey Juan Carlos, 2015
5. Velázquez Iturbide, J.Á., Debdi, O., Esteban Sánchez, N., Pizarro, C.: GreedEx: A visualization tool for experimentation and discovery learning of greedy algorithms. IEEE Transactions on Learning Technologies 6, 2 (abril-junio 2013) 130-143, DOI 10.1109/TLT.2013.8
6. Velázquez-Iturbide, J.Á.: Design and evaluation of OptimEx, an experimentation system for optimization algorithms. En: M.J. Marcelino, A.J. Mendes y C. Azevedo Gomes (eds.): ICT in Education - Multiple and Inclusive Perspectives. Springer, Nueva York (2016) 51-68, DOI 10.1007/978-3-319-22900-3_4
7. Velázquez-Iturbide, J.Á.: GreedEx and OptimEx: Two tools to experiment with optimization algorithms. International Journal of Engineering Education 32, 3A (2016) 1.097-1.106
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms, 3rd Ed. The MIT Press: Cambridge, MA, 2009
9. Kleinberg, J., Tardos, É.: Algorithm Design, Pearson Addison-Wesley, 2006
10. Velázquez Iturbide, J.Á.: Una tercera evaluación cualitativa de la comprensión de la optimalidad. En: Serie de Informes Técnicos DLSI1-URJC, no. 2016-01, Departamento de Lenguajes y Sistemas Informáticos I, Universidad Rey Juan Carlos, 2016

11. Velázquez Iturbide, J.Á., Hernán Losada, I.: Evaluación del aprendizaje de la programación dinámica usando bosques de recursión. En: Serie de Informes Técnicos DLSI1-URJC, no. 2015-06, Departamento de Lenguajes y Sistemas Informáticos I, Universidad Rey Juan Carlos, 2015.

Apéndice A: Enunciado de la Práctica 1

Grado en Ingeniería Informática
Asignatura *Algoritmos Avanzados*
Curso 2015/2016
Práctica nº 1

Objetivo
El objetivo de la práctica es profundizar en la implementación de los algoritmos voraces y aproximados.

Carácter
La práctica es voluntaria. Puede realizarse individualmente o en pareja.

Enunciado
Sea un conjunto A de n actividades $\{a_0, a_1, \ldots, a_{n-1}\}$ que necesitan utilizar un recurso común, p.ej. una sala de reuniones. El recurso sólo puede ser usado por una actividad en cada momento. Cada actividad tiene un instante de comienzo c_i y un instante de finalización f_i, donde $0 \leq c_i < f_i < \infty$. Si se selecciona la actividad a_i, se desarrolla en el intervalo semiabierto de tiempo $[c_i, f_i)$. Las actividades a_i y a_j son compatibles si sus intervalos $[c_i, f_i)$ y $[c_j, f_j)$ no se solapan, es decir, si $c_i \geq f_j$ o $c_j \geq f_i$.

El *problema de selección de actividades* consiste en determinar un subconjunto de actividades compatibles cuya cardinalidad sea máxima.

Por ejemplo, sea el siguiente conjunto de actividades:

i	0	1	2	3	4	5	6	7	8	9
c_i	11	24	7	0	5	12	23	2	16	15
f_i	21	29	8	3	11	25	24	18	20	24

Un subconjunto de actividades compatibles es $\{a_3, a_4, a_9, a_1\}$. Sin embargo, no es un subconjunto de cardinalidad máxima, como lo son $\{a_3, a_4, a_0, a_6, a_1\}$ y $\{a_3, a_2, a_8, a_6, a_1\}$.

Como puede comprobarse con GreedEx, hay dos funciones de selección de actividades óptimas: orden creciente de fin y orden decreciente de inicio. Suponiendo que las actividades ya están ordenadas según alguno de estos dos criterios, el algoritmo voraz puede codificarse así en Java y pseudocódigo:

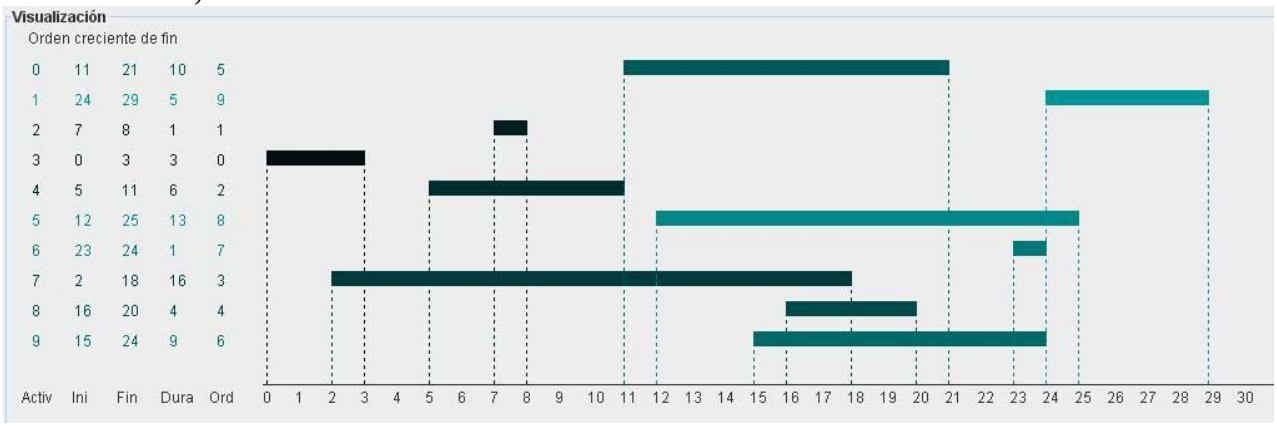
```
public static boolean[] selecActividades (int[] c, int[] f) {
   boolean[] s = new boolean[c.length];
   s[0] = true;
   int i = 0;
   for (int j=1; j<c.length; j++) {
      if (<<actividades i, j no se solapan>>) {
         s[j] = true;
         i = j;
      }
      else
         s[j] = false;
   }
   return s;
}
```

donde, dados dos vectores *c* y *f* con los instantes de comienzo y fin, el método devuelve un vector de booleanos que identifica las actividades seleccionadas.

Se desea:

1. Implementar el algoritmo anterior en Java usando una función de selección óptima.
2. Reimplementarlo de forma que pueda utilizarse en situaciones reales, es decir, sin suponer que los datos de entrada están ordenados y sin que los parámetros queden alterados tras ejecutar el algoritmo.

Por ejemplo, sean las actividades del enunciado anterior. Si se elige orden creciente de fin, las actividades se examinan en el orden indicado por los tonos de la figura (de oscuro a claro):



es decir, las actividades se irán examinando de izquierda a derecha en el siguiente orden:

i	3	2	4	7	8	0	9	6	5	1
c_i	0	7	5	2	16	11	15	23	12	24
f_i	3	8	11	18	20	21	24	24	25	29

quedando seleccionadas las 5 actividades {3,2,8,6,1} (las demás actividades se solapan con las seleccionadas). Por tanto, el algoritmo debe devolver un vector de booleanos con sus elementos 3, 2, 8, 6 y 1 a *true* y los demás elementos a *false*, es decir, el vector {*false*, *true*, *true*, *true*, *false*, *false*, *true*, *false*, *true*, *false*}

Entrega

El alumno debe entregar un informe elaborado siguiendo el índice detallado a continuación. El informe debe enviarse por medio del apartado de Evaluación del campus virtual. Si se tienen dificultades, puede enviarse por el correo del campus virtual con el asunto "Práctica 1". El plazo de entrega del informe es el domingo 27 de septiembre de 2015, incluido.

Informe

El alumno debe entregar un informe con la siguiente estructura:

1. **Algoritmo ejecutable.** El pseudocódigo dado antes para el algoritmo voraz debe implementarse como un algoritmo ejecutable. Se pide:
   a. Elegir una función de selección óptima.
   b. Explicar cómo quedará codificada en Java la condición del *if* del algoritmo para dicha función de selección, si es necesario con ayuda de un diagrama de tiempos como el anterior.
   c. Código en Java del algoritmo ejecutable, que supondrá que las actividades están ordenadas.
2. **Algoritmo final.** Desarrollar una nueva versión del algoritmo voraz que no dependa de que las actividades estén ordenadas y que no altere el contenido de los parámetros tras ser ejecutado.
3. **Conclusiones.** Se explican las conclusiones obtenidas tras realizar la práctica. Estas conclusiones pueden consistir en una valoración de la técnica voraz o cualquier comentario sobre la práctica. Por ejemplo, pueden describirse las incidencias que han dificultado la realización de la práctica, sus aspectos más atractivos o más difíciles, sugerencias sobre cómo mejorar la práctica, etc.

Evaluación

Se evaluará la calidad del algoritmo desarrollado y la claridad del informe.

Apéndice B: Enunciado de la Práctica 2

Grado en Ingeniería Informática
Asignatura *Algoritmos Avanzados*
Curso 2015/2016
Práctica nº 2

Objetivo
El objetivo de la práctica es que el alumno practique con los algoritmos heurísticos.

Carácter
La práctica es voluntaria, pero es un requisito para poder entregar la segunda parte de las prácticas 3 y 5. Puede realizarse individualmente o en pareja.

Enunciado
Sea un pequeño negocio con dos sedes, $s0$ y $s1$, en dos ciudades distintas. Según la clientela prevista en cada mes, le puede interesar realizar su actividad en una sede o en otra. Si durante el mes i está en la sede $s0$, sus operaciones le supondrán un coste $c_{0,i}$, pero si está en la sede $s1$, el coste será $c_{1,i}$ ($c_{s,i}>0$, para $s\in\{0,1\}$, $0\leq i\leq n-1$). El cambio de una sede a otra siempre conlleva un gasto fijo $f\geq 0$.

El *problema del plan de sedes de coste mínimo* consiste en decidir en qué sede debe el negocio concentrar su actividad durante cada mes de forma que se minimicen los costes totales de operación.

Por ejemplo, los costes de 4 meses pueden ser $c_0=\{1,3,20,30\}$ y $c_1=\{50,20,2,4\}$, con un coste de traslado $f=10$. Un plan con coste mínimo sería $\{s0,s0,s1,s1\}$, con un coste igual a 1+3+10+2+4=20. Puede comprobarse que cualquier otro plan tendrá un coste mayor.

Un algoritmo heurístico consiste en seleccionar en cada mes la sede que tiene menos coste. Una variación de este algoritmo tomaría la decisión teniendo también en cuenta el coste del traslado f.

El objetivo de la práctica es implementar ambos algoritmos y usar OptimEx para comparar su optimalidad, medida como el porcentaje de casos en los que cada algoritmo calcula una solución óptima.

Entrega
El alumno debe entregar un informe elaborado siguiendo el índice detallado a continuación. El informe debe enviarse por medio del apartado de Evaluación del campus virtual. Si se tienen dificultades, puede enviarse por el correo del campus virtual con el asunto "Práctica 2". El plazo de entrega del informe es el domingo 11 de octubre de 2015, incluido.

Informe
El alumno debe entregar un informe con la siguiente estructura:

3. **Implementación de los algoritmos.** Implementar ambos algoritmos en una sola clase de Java y con la siguiente cabecera (obviamente, con identificadores distintos):

   ```
   public static int sedes (int[] c0, int[] c1, int f)
   ```

   donde *c*0 y *c*1 son los vectores de costes mensuales de las sedes *s*0 y *s*1, respectivamente, y *f* es el coste fijo de cada cambio de sede. Para el ejemplo anterior, la llamada será *sedes*({1,3,20,30},{50,20,2,4},10). Cada algoritmo debe devolver el coste óptimo de un plan de sedes.

4. **Experimentación con la optimalidad de los algoritmos.** Se comparará la optimalidad de ambos algoritmos con OptimEx y se incluirá:
   a) Resultado. Se dirá si algún algoritmo es exacto.
   b) Evidencias. Deben aportarse los resultados recogidos en las tablas de resumen e histórica y explicar el significado de dichos resultados.

   Puede usarse OptimEx de la siguiente forma: se carga la clase con los dos algoritmos a comparar, se selecciona su signatura, se realiza una ejecución intensiva y se comparan los resultados obtenidos en la tabla resumida. Hay que tener cuidado con respetar las restricciones del enunciado al generar datos aleatoriamente: los costes mensuales son mayores que cero, y el coste fijo *f* es no negativo. La guía de uso de OptimEx identifica diversos resultados que, si se obtienen, son contradictorios con los conceptos presentados en clase; conviene leerla atentamente para, en su caso, identificar y corregir estas situaciones.

5. **(Opcional.) Razonamiento sobre la optimalidad de los algoritmos.** En el apartado anterior ha podido resultar que algún algoritmo, los dos o ninguno parezca ser óptimo. En este apartado se pide que se intente razonar sobre los algoritmos a partir de los resultados obtenidos. Si algún algoritmo ha dado siempre resultados óptimos, se pide una demostración (o boceto) de su optimalidad. Si algún algoritmo da a veces resultados subóptimos, se pide aportar un contraejemplo de su optimalidad que sea lo más sencillo posible (es decir, que implique pocos meses).

6. **Conclusiones.** Se explican las conclusiones obtenidas tras realizar la práctica. Estas conclusiones pueden consistir en una valoración de los algoritmos

heurísticos o cualquier comentario sobre la práctica. Por ejemplo, pueden describirse las incidencias que han dificultado la realización de la práctica, sus aspectos más atractivos o más difíciles, sugerencias sobre cómo mejorar la práctica, etc.

Evaluación

Se evaluará la calidad y claridad de todos los apartados del informe.

Apéndice C: Enunciado de la Práctica 3a

Grado en Ingeniería Informática
Asignatura *Algoritmos Avanzados*
Curso 2015/2016
Práctica nº 3.a

Objetivo
El objetivo de la práctica es que el alumno profundice en su conocimiento de las técnicas de búsqueda en espacios de estados.

Carácter
La práctica es voluntaria. Puede realizarse individualmente o en pareja.

Enunciado
En la práctica 2 se planteó el *problema del plan de sedes de coste mínimo*. Suponiendo que la planificación se realiza para 2 sedes ($s0$ y $s1$) y n meses, el objetivo consiste en decidir en qué sede debe el negocio concentrar su actividad durante cada mes de forma que se minimicen los costes de operación.

Por ejemplo, los costes de 4 meses pueden ser c_0={1,3,20,30} y c_1={50,20,2,4}, con un coste de traslado f=10. Un plan con coste mínimo sería {$s0$,$s0$,$s1$,$s1$}, con un coste igual a 1+3+10+2+4=20. Puede comprobarse que cualquier otro plan tendrá un coste mayor.

El objetivo de la práctica es desarrollar de forma sistemática algoritmos de búsqueda en espacios de estados que resuelvan el problema planteado. Los algoritmos deben desarrollarse siguiendo unos pasos que permitan completar el informe detallado a continuación.

Entrega
El alumno debe entregar un informe elaborado siguiendo el índice detallado a continuación. El informe debe enviarse por medio del apartado de Evaluación del campus virtual. Si se tienen dificultades, puede enviarse por el correo del campus virtual con el asunto "Práctica 3a". El plazo de entrega del informe es el jueves 29 de octubre de 2015 a las 15h.

Informe
El alumno debe entregar un informe con la siguiente estructura:

1. **Técnica de vuelta atrás.** Constará de varios apartados, que mostrarán el desarrollo sistemático de un algoritmo de vuelta atrás:
   a) Diseño de un árbol de búsqueda adecuado para resolver el problema. Debe incluirse una figura del mismo; si el árbol es muy grande, basta con mostrar una parte, siempre que sea fácil deducir la parte omitida. La figura debe acompañarse de una explicación breve y clara, en términos de los parámetros del problema, de: (1) número de niveles del árbol, y (2) candidatos en cada nodo del árbol.
   b) Comprobación de validez. Debe explicarse claramente: (a) comprobación de validez de la solución parcial correspondiente en cada nodo del árbol de búsqueda, y (b) implementación incremental.
   c) Código de un algoritmo basado en el esquema de vuelta atrás, que incorpora el diseño anterior. Al igual que en la práctica 2, la cabecera del método principal del algoritmo debe ser:

      ```
      public static int sedes (int[] c0, int[] c1, int f)
      ```

      donde *c*0 y *c*1 son los vectores de costes mensuales de las sedes *s*0 y *s*1, respectivamente, y *f* es el coste fijo de cada cambio de sede. Para el ejemplo anterior, la llamada del método principal será *sedes*({1,3,20,30},{50,20,2,4},10). Cada algoritmo debe devolver el coste óptimo de un plan de sedes y opcionalmente puede imprimir la solución óptima calculada.

2. **Técnica de ramificación y poda.** Partiendo del algoritmo de vuelta atrás anterior, se pide:
   a) Función de cota para este problema. Debe explicarse claramente: (1) definición de la función de cota, (2) valor inicial, y (3) actualización.
   b) Código de un algoritmo basado en el esquema de ramificación y poda, que incorpora la cota anterior. Debe tener la cabecera antes indicada.

3. **Conclusiones.** Se explican las conclusiones obtenidas tras realizar la práctica. Estas conclusiones pueden consistir en una valoración de las técnicas de búsqueda o cualquier comentario sobre la práctica. Por ejemplo, pueden describirse las incidencias que han dificultado la realización de la práctica, sus aspectos más atractivos o más difíciles, sugerencias sobre cómo mejorar la práctica, etc.

Evaluación
Se evaluará la calidad y claridad de todos los apartados del informe.