Un algoritmo híbrido para el problema NRP con interdependencias *

Francisco Palomo Lozano[1], Isabel M. del Águila[2] e Inmaculada Medina Bulo[1]

[1] Departamento de Ingeniería Informática, Universidad de Cádiz,
Avda. de la Universidad de Cádiz 10, 11519 Puerto Real, Spain
{francisco.palomo,inmaculada.medina}@uca.es
[2] Departamento de Informática, Universidad de Almería,
Ctra. de la Playa s/n, 04120 Almería, Spain
imaguila@ual.es

Resumen En este artículo presentamos un algoritmo híbrido para una variante del problema de la siguiente versión (NRP). En esta variante existe un conjunto de requisitos para los que se dispone de una estimación del esfuerzo necesario para su implementación y de la satisfacción percibida por los potenciales clientes con la inclusión de dichos requisitos. Entre estos requisitos existen relaciones de interdependencia, que establecen a ciertos requisitos como prerrequisitos de otros, o que obligan a implementar determinados requisitos simultáneamente en caso de incluirse alguno de ellos en la siguiente versión del producto a desarrollar. Dado un límite superior de esfuerzo prefijado, el objetivo es seleccionar un subconjunto de requisitos que cumpla todas las restricciones y maximice la satisfacción global de los clientes. La propuesta combina heurísticas con técnicas exactas. El rendimiento del algoritmo resultante en distintos escenarios realistas se compara con el de otras técnicas metaheurísticas previamente empleadas para el problema.

Keywords: Técnicas metaheurísticas, Ingeniería de requisitos, Problema de la siguiente versión, Interdependencias, NRP, SBSE.

1. Introducción

La selección de requisitos o características que debe poseer un producto entre las previamente definidas por los clientes es un proceso de gran importancia en proyectos de desarrollo de software que puede ser abordado con técnicas metaheurísticas y también con técnicas exactas [18]. En general, los requisitos presentan relaciones de interdependencia que recogen situaciones cotidianas en el desarrollo del software: requisitos que deben construirse en un orden concreto, que han de desarrollarse a la vez o que excluyen la posibilidad de desarrollar otros. El problema de la siguiente versión, o NRP, modela esta realidad.

* Parcialmente financiado por la red de excelencia TIN2015-71841-REDT (SEBASENet) y los proyectos TIN2014-60844-R (SAVANT) y TIN2015-65845-C3-3-R (DArDOS).

La principal dificultad estriba en que el problema computacional subyacente es complejo [16,15]. Sin embargo, dado su interés práctico para la industria del software, ha sido objeto de numerosos análisis desde este campo [14,3,11,2,4,20,21]. Un algoritmo híbrido ahorra tiempo de ejecución a costa de precisión, pero incluye una componente exacta que puede proporcionarle ventaja frente a otras técnicas aproximadas. Presentamos un algoritmo híbrido *determinista* para una variante del problema NRP basada en el modelo de van den Akker [2], en el que se considera un conjunto de requisitos para los que se dispone de una estimación del esfuerzo necesario para su implementación y de la satisfacción percibida por los potenciales clientes de incluirse dichos requisitos en la próxima versión del producto. Dicho modelo incluye también la posibilidad de que existan, como es común en la práctica, relaciones de interdependencia como las ya mencionadas.

El artículo se organiza como sigue. En la sección 2 se discuten los trabajos relacionados. A continuación, la sección 3 describe formalmente el problema y las distintas estrategias de resolución, con énfasis en las metaheurísticas. El algoritmo híbrido propuesto se discute en suficiente detalle en la sección 4. Una vez descrito, se evalúa su rendimiento en distintos escenarios y se comparan sus resultados con los obtenidos por las técnicas metaheurísticas de referencia en la sección 5. Para finalizar, en la sección 6 se resumen las conclusiones y se esbozan posibles líneas de trabajo futuro.

2. Trabajos relacionados

Es necesario aclarar que el problema NRP ha sido tratado profusamente en la literatura y que, no sólo la terminología, sino la propia definición del problema, pueden diferir de un autor a otro, por lo que existen numerosas variantes. Sin embargo, subyacen aspectos comunes en todas ellas y cabe decir que existen dos modelos principales, el de Bagnall y el de van den Akker, descritos a continuación, en los que se pueden incluir la mayoría.

Bagnall et al. [3] modeló el problema NRP como un problema de optimización. En su modelo se dispone de un conjunto de requisitos o mejoras que una compañía desea incluir en la siguiente versión de un producto de software, de los que se estima el coste de desarrollo, y de una serie de clientes que demandan la inclusión de ciertos requisitos en la siguiente versión del producto. Define variantes del problema, aunque nos referiremos a continuación a la más general.

Entre los requisitos pueden existir relaciones de dependencia; en particular, Bagnall considera que un requisito puede ser prerrequisito de otro, en cuyo caso la inclusión de este último exige la de aquel. En la estimación del coste de desarrollo de un requisito se presuponen satisfechos todos sus prerrequisitos, lo que permite estimar los costes por separado. En caso de que todos los requisitos exigidos por un cliente se incluyan, este aporta un beneficio a la compañía estimado según un peso o importancia que se asigna previamente al cliente. El objetivo es seleccionar un subconjunto de clientes a satisfacer, de forma que se maximice la suma de los pesos, o importancia total, y el coste total estimado de todos los requisitos a incluir en la siguiente versión no supere un presupuesto prefijado.

Van den Akker et al. [2] propuso un modelo alternativo en el que la importancia se asigna individualmente a cada requisito, con lo que se dispone de una forma de estimar su beneficio. El objetivo es seleccionar un subconjunto de requisitos a incluir en la siguiente versión que maximice el beneficio total y cuyo coste total estimado no exceda el presupuesto asignado. Al igual que en el modelo de Bagnall, en caso de existir interdependencias entre los requisitos, la solución propuesta debe respetarlas todas.

El modelo de van den Akker pone el foco en los requisitos, no en los clientes. No obstante, los clientes pueden incorporarse al modelo con la diferencia respecto al modelo de Bagnall de que, en este caso, se considera que pueden ser satisfechos parcialmente. En este sentido, cabe pensar que el modelo de van den Akker es más flexible. En presencia de clientes, puede obtenerse un modelo simple de estimación de beneficios para los requisitos si estos les asignan una importancia y se calcula para cada requisito su importancia ponderada con los pesos asignados a los clientes por la compañía.

Greer and Ruhe [11] trataron el problema de la planificación de versiones, importante en el contexto del desarrollo incremental de software. En este problema, se trata de distribuir los requisitos entre las distintas versiones planificadas teniendo en consideración las opiniones de los clientes, los recursos disponibles, las interdependencias y otros factores. El enfoque seguido consiste en resolver el problema a lo largo de una serie de iteraciones mediante un algoritmo genético.

Baker et al. [4] mostraron cómo los resultados que se obtienen mediante técnicas metaheurísticas relativos a la toma de decisiones en el contexto del problema NRP son superiores a los producidos por un experto.

Con algunas excepciones como [14,3,2,21], la mayor parte de la literatura se ha centrado en el desarrollo de técnicas metaheurísticas. En los trabajos de Sagrado et al. [20] y Pitangueira et al. [18] pueden encontrarse revisiones recientes de los trabajos relacionados que emplean dichas técnicas.

3. Formulación del problema y estrategias de resolución

Sea $R = \{r_1, \ldots, r_n\}$ un conjunto de requisitos propuestos por m clientes. Cada cliente i tiene distinta importancia para el proyecto, representada mediante un peso $w_i \in \mathbb{R}$. Todo $r_j \in R$ tiene asociado un esfuerzo de desarrollo e_j y un valor $v_{ij} \in \mathbb{R}$ para cada cliente i. La satisfacción, s_j, o valor añadido por la inclusión de r_j en la siguiente versión del software, se calcula como la suma ponderada $s_j = \sum_{i=1}^{m} w_i \cdot v_{ij}$. Se consideran una serie de interdependencias funcionales que pueden surgir entre los requisitos y que se definen como sigue:

1. *Implicación o precedencia* ($r_i \Rightarrow r_j$). El requisito r_j no puede ser incluido en el software sin incorporar r_i. Es decir, r_i es un prerrequisito de r_j.
2. *Combinación o acoplamiento* ($r_i \odot r_j$). Los requisitos r_i y r_j deben ser incluidos conjuntamente en el software.
3. *Exclusión* ($r_i \oplus r_j$). Los requisitos r_i y r_j no pueden ser incluidos conjuntamente en el software.

El objetivo del problema es encontrar un subconjunto de requisitos S que represente la mejor solución. Para ello es necesario construir S de forma que se maximice la satisfacción global de los clientes dentro de las limitaciones impuestas por los recursos asignados al proyecto y las interdependencias entre los requisitos. La satisfacción y el esfuerzo totales de S se definen como:

$$\text{sat}(S) = \sum_{r_j \in S} s_j \qquad \text{eff}(S) = \sum_{r_j \in S} e_j \tag{1}$$

y, formalmente, el problema consiste en:

$$\text{máx } \{\text{sat}(S) \mid \text{eff}(S) \leq B\} \tag{2}$$

siendo B el límite de esfuerzo.

3.1. Estrategias

A la hora de resolver un problema NRP que presenta interdependencias hay dos estrategias. La primera es tratarlas, bien dentro de los propios algoritmos, bien externamente. El tratamiento interno puede implicar, por ejemplo, adaptar la técnicas de búsqueda para que comprueben que no se producen conflictos [19]. El tratamiento externo puede realizarse, por ejemplo, traduciéndolas a lógica proposicional e incorporando resolutores SAT en el proceso [1]. La segunda consiste en transformar el problema original en otro que incluya únicamente requisitos independientes, tratando así de evitar las interdependencias. En el caso de las interdependencias de implicación, su eliminación fue tratada en el artículo original de Bagnall et al. [3], pero, en realidad, únicamente es efectiva para la versión más restringida del problema donde se debe satisfacer completamente al cliente.

En el trabajo de Sagrado et al. [20] se propone un método más general de eliminación de interdependencias para el problema NRP en el modelo de van der Akker. La idea es trasformar todas las interdependencias mediante la agrupación de requisitos en metarrequisitos, generando un nuevo escenario de metarrequisitos independientes. No obstante, existen ciertas limitaciones, pues en los grupos de requisitos que surgen al eliminar las interdependencias puede aparecer duplicados (un requisito puede formar parte de varios metarrequisitos).

Se describirán a continuación sucintamente las técnicas metaheurísticas que se compararán con el algoritmo híbrido propuesto en este trabajo. No consideraremos las dependencias de exclusión por las razones expuestas en [20], donde puede encontrarse una descripción más detallada de estas técnicas. Si bien cuando su número es pequeño puede resultar factible eliminarlas a cambio de resolver ejemplares extra, no se ha implementado aquí dicha posibilidad.

3.2. Metaheurísticas

A diferencia del algoritmo híbrido propuesto, las tres metaheurísticas que se presentan a continuación resuelven la versión biobjetivo del problema [20], pero en el contexto de este trabajo sus resultados se utilizan para maximizar un único objetivo, la satisfacción. Dado un conjunto de requisitos R, definimos:

1. La productividad:

$$p(R) = \sum_{r_i \in R} p(r_i) \quad \text{con} \quad p(r_i) = \frac{s_i}{e_i} \tag{3}$$

2. El conjunto de requisitos visibles, vis(R), que incluye todos los requisitos r_j que satisfacen las interdependencias y cumplen que $\text{eff}(R) + e_j \leq B$.

GRASP Greedy Randomized Adaptative Search Procedure (GRASP) [8] es una técnica iterativa que construye una solución voraz aleatoria y la mejora con búsqueda local. Se construye una lista de requisitos visibles ordenada por productividad. La búsqueda local es iterativa y trata de reemplazar la solución actual por una mejor de su vecindario. Es decir, se elimina de la solución el requisito con menor p y se selecciona otro de los visibles. Si la nueva solución mejora en términos de satisfacción de los clientes, sustituye a la anterior y se repite el proceso. GRASP termina cuando no hay una solución mejor en el vecindario.

Algoritmos genéticos Cada individuo se representa mediante un vector binario y define un conjunto de requisitos que satisface las restricciones del problema. Dado $[x_1, \ldots, x_n] \in \{0,1\}^n$, $x_i = 1$ si, y sólo si, el requisito $r_i \in S$. Cada generación representa la evolución de la población que, a través de las operaciones de cruce y mutación, puede producir nuevos individuos que poseen incluso mejores valores de satisfacción que los originales. Los operadores de cruce y mutación son operadores ciegos, en el sentido de que no garantizan ni que la solución satisfaga la restricción del límite de esfuerzo, ni las asociadas a las interdependencias. Por ello, se emplea un operador de reparación que elimina requisitos, en orden decreciente de productividad, hasta que se cumplen las restricciones [19].

Algoritmos de colonia de hormigas La optimización por colonia de hormigas emula el comportamiento de las hormigas cuando buscan el camino más corto desde su hormiguero hasta la comida [7]. Emplean una sustancia química, la feromona. Al elegir un camino, la hormiga adopta un comportamiento probabilístico. En cada encrucijada tiende a seguir el camino con más feromona, sustancia que segregan las hormigas conforme avanzan; si la cantidad de esta sustancia no es suficiente, elige al azar. La feromona se concentra en los caminos más transitados. El algoritmo utiliza un determinado número de hormigas, normalmente la mitad que requisitos. Cada una calcula una solución, comenzando en un lugar aleatorio y seleccionando los requisitos uno a uno. La hormiga localiza qué requisitos son visibles desde los que ya ha seleccionado y elige uno, combinando la información heurística (en este caso, la productividad) y la feromona. Finalmente, se actualizan los valores de la feromona de la mejor solución. Al construir la solución asociada a la hormiga k, a partir de un lugar i, se selecciona el próximo lugar, j, como sigue:

$$j = \begin{cases} \arg\,\text{máx}_{u \in N_i^k} [\tau_{ij}]^\alpha [\eta_{ij}]^\beta, & \text{si } q \leq q_0 \\ u, & \text{e.o.c.} \end{cases} \tag{4}$$

Así, si el último requisito seleccionado ha sido r_i, el siguiente será r_j. El valor q es un valor aleatorio entre $[0,1]$, y $q_0 \in [0,1]$ es un parámetro que controla el equilibrio entre intensificación y diversificación en el algoritmo, buscando la mejor cobertura del espacio de búsqueda. La probabilidad de que la hormiga k elija moverse de i a j es:

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{h \in N_i^k} [\tau_{ij}]^\alpha [\eta_{ij}]^\beta}, & \text{si } j \in N_i^k \\ 0, & \text{e.o.c.} \end{cases} \tag{5}$$

siendo la información heurística una media de productividad $\eta_{ij}^k = \mu \cdot p(r_j)$ con μ un valor de normalización y N_i^k el conjunto de requisitos visibles para la hormiga k. La mejor solución encontrada por las hormigas, S, actualiza el rastro de feromona τ_{ij} de acuerdo con:

$$(1-\rho)\tau_{ij} + \rho \Delta\tau_{ij} \qquad \Delta\tau_{ij} = \frac{\text{sat}(S)}{\text{sat}(R)} \tag{6}$$

siendo $\rho \in [0,1]$ la tasa de evaporación de la feromona.

4. Algoritmo híbrido para el problema NRP

El problema NRP es computacionalmente complejo y hereda su dificultad del problema de la mochila 0/1 (KP), que ha sido estudiado exhaustivamente por sus múltiples aplicaciones incluso desde antes de que existiera consciencia de este hecho [6]. La consecuencia principal es que, independientemente de la bondad de los algoritmos que se empleen, siempre exhibirán un bajo rendimiento para un número infinito de ejemplares del problema, en tanto en cuanto estos algoritmos sean exactos y $\mathcal{P} \neq \mathcal{NP}$ [10].

Por otro lado, en el caso particular de KP, existe un resultado negativo importante que nos impide encontrar algoritmos aproximados con rendimiento absoluto, esto es, que garanticen que el error absoluto en el que incurren esté acotado por una constante. Es decir, garantizar un error absoluto para KP es tan difícil computacionalmente como garantizar un resultado exacto.

Sin embargo, existen algoritmos eficientes para KP que garantizan que el error relativo que cometen está acotado por una constante. Así, existen algoritmos $(1-\epsilon)$-aproximados que producen soluciones cuyo valor es, al menos, $1-\epsilon$ veces el del máximo alcanzable. Esto es, el error relativo que poseen las soluciones aproximadas que producen respecto a las óptimas es, a lo sumo, ϵ. Lógicamente, su tiempo de ejecución depende de ϵ: cuanto menor sea ϵ, mayor será el tiempo de ejecución. Aunque para KP es posible disminuir ϵ aumentando el tiempo de ejecución, en el peor caso, en un polinomio en el tamaño de la entrada y ϵ^{-1} [13], estas garantías desaparecen cuando se introducen dependencias en el problema, como en el caso del problema NRP que nos ocupa. Considérese al respecto la noción de $\mathcal{NP}$-completitud en sentido estricto [9].

En tal caso, conforme se reduce el error relativo que se desea garantizar, la eficiencia de los algoritmos disponibles se degrada rápidamente, incluso aunque se mantenga el tamaño de los ejemplares. Existe pues, en general, un compromiso entre tiempo y precisión. Es posible sacrificar precisión, renunciando a obtener siempre una solución óptima, a cambio de disminuir el tiempo de ejecución, o invertir tiempo de ejecución en obtener mejores soluciones.

Un algoritmo híbrido, que combine un algoritmo exacto con heurísticas, representa un compromiso entre ambas opciones, pero posee el potencial de producir mejores soluciones que otros algoritmos puramente heurísticos o metaheurísticos a un coste razonable. Para que este enfoque sea factible, la parte exacta del algoritmo debe emplearse para resolver solo una parte concreta del problema o una versión simplificada del mismo, de forma que a partir del resultado se pueda recuperar una solución aproximada. El algoritmo híbrido propuesto en el presente trabajo consta de las siguientes etapas:

1. Transformación de los requisitos en metarrequisitos.
2. Resolución del problema NRP sobre los metarrequisitos.
3. Reparación de la solución para eliminar requisitos duplicados.
4. Resolución recursiva sobre los requisitos no incluidos en la solución.

El algoritmo ha sido cuidadosamente implementado en C++ estándar. A continuación se discute cada etapa y se presentan fragmentos de pseudocódigo, aunque un buen número de detalles y mejoras obvias se han omitido por claridad expositiva.

4.1. Transformación en metarrequisitos

La transformación de los requisitos en metarrequisitos se basa en [20] y tiene en consideración las interdependencias de precedencia y de combinación.

En primer lugar, el algoritmo 1 inicializa una matriz de adyacencia de $n \times n$ con la información proporcionada por las parejas de requisitos que aparecen en cada relación de interdependencia. Todas las entradas de la matriz booleana a reciben primero el valor falso ($\bot$) y cambiarán a verdadero ($\top$) según sean o no afectadas por alguna interdependencia. En el caso de las interdependencias de combinación, la relación es simétrica, lo que debe tenerse en cuenta al rellenar las entradas correspondientes.[3] En el caso de la relación de precedencia, se invierten las parejas. El resultado es la matriz de la relación $\odot \cup \Rightarrow^{-1} = \odot \cup \Leftarrow$, donde $\Leftarrow$ representa la inversa de la relación de precedencia, es decir, $r_i \Leftarrow r_j$ si r_j es prerrequisito de r_i. En lugar de trabajar directamente con los requisitos, el algoritmo emplea sus índices.

[3] En los conjuntos de datos disponibles para experimentación solo se especifica uno de los dos pares ordenados correspondientes a cada par de requisitos relacionados, lo que resulta conveniente por brevedad.

```
1  for i ← 1 to n do
2      for j ← 1 to n do
3          a[i, j] ← ⊥
4  ▷ Relación de combinación.
5  for all (i, j) ∈ ⊙ do
6      ▷ Simétrica.
7      a[i, j] ← ⊤
8      a[j, i] ← ⊤
9  ▷ Relación de precedencia.
10 for all (i, j) ∈ ⇒ do
11     ▷ Inversa.
12     a[j, i] ← ⊤
13 return a
```

Algoritmo 1: Inicialización.

A continuación, el algoritmo 2 proporciona el cierre reflexivo y transitivo de la relación resultante mediante una sencilla modificación del algoritmo de Warshall. Por consiguiente, el resultado es la matriz de la relación $(\odot \cup \Leftarrow)^*$. A partir de esta matriz puede determinarse en tiempo constante si un requisito depende de otro, y extraerse fácilmente todos los requisitos de los que depende uno dado.

```
1  ▷ Cierre transitivo.
2  for k ← 1 to n do
3      ▷ Cierre reflexivo.
4      a[k, k] ← ⊤
5      for i ← 1 to n do
6          ▷ Mejora.
7          if a[i, k] then
8              for j ← 1 to n do
9                  a[i, j] ← a[i, j] ∨ a[k, j]
10 return a
```

Algoritmo 2: Cierre reflexivo y transitivo.

Esta modificación incorpora una ligera, pero efectiva, mejora. La matriz a se representa internamente de forma muy compacta (1 bit por entrada), lo que permitiría realizar otras mejoras, como procesar cada fila mediante operaciones de bits muy eficientes.

Finalmente, el proceso de obtención del conjunto de metarrequisitos se ilustra en el algoritmo 3. Internamente, cada metarrequisito contiene el índice del requisito a partir del que se genera, así como el esfuerzo y satisfacción totales de todos los requisitos que lo integran. Cada requisito produce un metarrequisito. Un requisito sin dependencias da lugar a un metarrequisito unitario, que solo lo contiene a él mismo. Si un metarrequisito posee idéntica composición que otro previamente generado, se elimina, aunque para facilitar la comprensión esto no se ha reflejado en el algoritmo.

```
1  M ← ∅
2  for i ← 1 to n do
3      ▷ Requisitos de los que depende r_i.
4      T ← ∅
5      for k ← 1 to n do
6          if a[i, k] then
7              T ← T ∪ {r[k]}
8      ▷ Incluye el nuevo metarrequisito.
9      M ← M ∪ {⟨i, eff(T), sat(T)⟩}
10 return M
```

Algoritmo 3: Metarrequisitos.

4.2. Resolución sobre los metarrequisitos

La técnica de programación dinámica permite obtener una familia importante de *algoritmos pseudopolinómicos* para KP. Por ejemplo, sea S un conjunto de objetos, $c(x)$ y $b(x)$ el coste y beneficio de un objeto $x \in S$, y B el presupuesto disponible. La siguiente ecuación de Bellman puede resolverse mediante programación dinámica por costes, proporcionando una solución óptima para KP a un coste razonable cuando el número de objetos y el presupuesto son moderados:

$$\begin{aligned} &z(\varnothing, B) = 0 \\ &z(S, B) = z(S \setminus \{x\}, B) && \text{si } c(x) > B \\ &z(S, B) = \text{máx}\{z(S \setminus \{x\}, B), z(S \setminus \{x\}, B - c(x)) + b(x)\} && \text{si } c(x) \leq B \end{aligned} \quad (7)$$

Esto puede traducirse fácilmente al dominio del problema NRP (si se eliminan las interdependencias), cambiando objetos por requisitos, coste por esfuerzo, beneficio por satisfacción y presupuesto por límite de esfuerzo. Así, en la ecuación 7, el conjunto S representaría una colección finita de requisitos, x un requisito arbitrario de S con esfuerzo $c(x)$ y satisfacción $b(x)$, B el límite de esfuerzo y z la máxima satisfacción que se puede alcanzar con cualquier subconjunto de S sin exceder el límite de esfuerzo.

Para la resolución del problema NRP sin interdependencias se ha desarrollado una versión muy eficiente del algoritmo de Nemhauser-Ullmann, que emplea programación dinámica por costes [17]. Este algoritmo es exacto, por lo que siempre produce soluciones óptimas y, a diferencia de otros, no impone restricciones adicionales sobre su entrada. Esta versión se presenta en el algoritmo 4. Su buen rendimiento radica, en parte, en el empleo de *funciones escalera*. Una función escalera no es más que una función creciente con un número finito de peldaños o rellanos. Estas funciones se pueden representar eficientemente como listas de pares, que contienen las coordenadas de cada peldaño, con cuatro operaciones: *cero*, *desplazamiento*, *máximo* y *aplica*. Su funcionamiento se explica en más detalle en [12].

```
1  ▷ Calcula las satisfacciones máximas.
2  m[0] ← cero()
3  for k ← 1 to n do
4      f ← desplazamiento(m[k − 1], eff(r[k]), sat(r[k]))
5      m[k] ← máximo(m[k − 1], f)
6  ▷ Recupera la solución óptima calculada.
7  S ← ∅
8  for k ← n to 1 do
9      if aplica(m[k], B) ≠ aplica(m[k − 1], B) then
10         S ← S ∪ {k}
11         B ← B − eff(r[k])
12 return S
```

Algoritmo 4: Nemhauser-Ullmann para NRP sin interdependencias.

Desde la publicación de este algoritmo en 1969, varios autores habían constatado que parecía comportarse bien en la práctica cuando se resolvían ejemplares aleatorios del problema KP, a pesar de tratarse de un problema $\mathcal{NP}$-difícil, sin existir una justificación teórica que soportara estas observaciones. Sin embargo, en 2004, Beier and Vöcking proporcionaron una explicación del comportamiento del algoritmo, bajo hipótesis muy generales, mediante técnicas de *análisis suave*. Queda fuera del ámbito del presente trabajo discutir los aspectos, muy técnicos, de estos resultados [5], pero pueden resumirse diciendo que bajo suposiciones razonables sobre las distribuciones de coste y valor de los objetos implicados, el tiempo de ejecución y el espacio consumido esperados son polinómicos en el número de objetos.[4]

4.3. Reparación de la solución

Debido al proceso de transformación, un requisito puede formar parte de varios metarrequisitos. Por tanto, la solución S puede estar formada por metarrequisitos en los que puede aparecer varias veces un mismo requisito. La presencia de repeticiones indica que la solución obtenida en la etapa anterior puede ser subóptima. De ahí que aunque el algoritmo empleado sea exacto para NRP sin interdependencias, su combinación con la etapa previa de transformación que permite resolver el problema NRP original con interdependencias sea heurística.

Es necesario sustituir los metarrequisitos por sus requisitos para obtener una solución al problema original, pero sin que estos aparezcan repetidos. Para ello se construye un vector f con la frecuencia de aparición de cada requisito en los metarrequisitos. Este vector permite expandir eficientemente los metarrequisitos contenidos en S evitando las repeticiones. Al final de la etapa de reparación, la solución S contendrá la unión de todos los requisitos previamente contenidos en sus metarrequisitos.

[4] El algoritmo es pseudopolinómico en el peor caso, como se desprende de un análisis ordinario, y, bajo las hipótesis de suavidad correspondientes, polinómico.

4.4. Resolución recursiva

Una vez que disponemos del conjunto de requisitos solución S, el conjunto $R' = R \setminus S$ contiene aquellos requisitos que no se han incluido en ella. Debido a la necesidad de reparar la solución en la etapa anterior, es posible que el límite de esfuerzo sobrante $B' = B - \text{eff}(S)$ pueda emplearse para incluir requisitos de R' en la solución. No obstante, es necesario asegurarse de que estos requisitos siguen satisfaciendo las restricciones de interdependencia.

El algoritmo 5 permite reducir las interdependencias obviando aquellos pares que los requisitos de S satisfacen. Para ello basta emplear el vector de frecuencias generado en la etapa anterior. Únicamente pervivirán los pares en los que ninguna componente corresponda a un requisito de S, es decir, aquellos en los que la frecuencia sea 0 en ambos extremos.

```
1  ▷ Nueva relación de combinación.
2  C ← ∅
3  for all (i, j) ∈ ⊙ do
4      if f[i] = 0 ∧ f[j] = 0 then
5          C ← C ∪ {(i, j)}
6  ▷ Nueva relación de precedencia.
7  P ← ∅
8  for all (i, j) ∈ ⇒ do
9      if f[i] = 0 ∧ f[j] = 0 then
10         P ← P ∪ {(i, j)}
11 return ⟨C, P⟩
```

Algoritmo 5: Reducción.

Finalmente, únicamente queda resolver recursivamente el problema para R', con límite de esfuerzo B' e interdependencias reducidas C y P. La solución al problema reducido se une a S, salvo que dicha solución sea $\varnothing$, en cuyo caso el proceso recursivo termina y S es la solución final.

5. Resultados experimentales

Los conjuntos de datos empleados siguen el modelo de van den Akker [2] para el problema NRP con interdependencias. El primero [11] consta de 20 requisitos propuestos por 5 clientes. El segundo [19] ha sido generado aleatoriamente y cuenta con 100 requisitos propuestos por 5 clientes. Los valores de esfuerzo se fijan entre 1 y 20 días de trabajo y los pesos de clientes y valores asignados por estos a los requisitos pueden entenderse como los siguientes niveles linguísticos: sin importancia (1), poca importancia (2), importante (3), muy importante (4) y extremadamente importante (5).

Los escenarios de experimentación para cada conjunto de datos se han seleccionado estableciendo unos límites de esfuerzo relativos al esfuerzo total necesario para implementar todos los requisitos. Para cada conjunto de datos, los valores

limites son el 30 %, el 50 % y el 70 % del esfuerzo total y se recogen, junto al número de requisitos y de interdependencias de cada tipo, en la tabla 1.

Tabla 1. Características de los conjuntos de datos y escenarios.

	1.er conjunto			2.o conjunto		
B	30 %	50 %	70 %	30 %	50 %	70 %
	25	43	60	312	519	726
$\|R\|$	20			100		
$\|\Rightarrow\|$	8			38		
$\|\odot\|$	2			4		

Para cada escenario, las tablas 2 y 3 contienen las satisfacciones máximas, los esfuerzos con los que se alcanzan y los tiempos de ejecución de las correspondientes técnicas. Respecto a las metaheurísticas, para cada escenario se han realizado 100 ejecuciones consecutivas con distintas configuraciones de los parámetros propios de cada una, como pueden ser las probabilidades de cruce o mutación en NSGA II o el grado de voracidad en GRASP. Los datos mostrados en las tablas son los valores de la media de los experimentos con las mejores configuraciones establecidas en trabajos previos [20]. Los parámetros utilizados en los algoritmos se han seleccionado para conseguir en torno a 10 000 ejecuciones de las funciones de evaluación. En cambio, el algoritmo híbrido es determinista y, aunque solo es necesaria una ejecución por cada escenario, se han realizado varias al efecto de estimar mejor su tiempo de ejecución.

Tabla 2. Satisfacción máxima alcanzada y esfuerzo empleado para ello.

	1.er conjunto						2.o conjunto					
	30 %		50 %		70 %		30 %		50 %		70 %	
	sat	eff	sat	eff	sat	eff	sat	eff	sat	eff	sat	eff
ACS	516	25	626	40	689	60	1239	307	1635	513	2034	726
NSGA II	461	21	567	39	607	48	1139	295	1666	509	2100	718
GRASP	508	25	602	43	689	60	1157	311	1524	519	1984	720
Híbrido	504	24	684	43	783	57	1275	311	1805	519	2247	725

Tabla 3. Tiempo (ms) medio empleado para alcanzar la solución.

	1.er conjunto			2.o conjunto		
	30 %	50 %	70 %	30 %	50 %	70 %
ACS	639,10	787,62	836,76	616873,55	770221,22	881950,91
NSGA II	1891,55	1980,93	2034,25	28127,77	35045,76	38295,95
GRASP	362,80	1208,25	839,84	28915,30	118336,00	324604,00
Híbrido	4,00	6,00	13,00	37,00	40,00	28,00

El algoritmo híbrido es superior en la práctica totalidad de los escenarios y es capaz de encontrar soluciones más satisfactorias, invirtiendo el esfuerzo sobrante en ello. Las técnicas metaheurísticas, en cambio, a menudo no son capaces de encontrar estas soluciones en los extremos del frente de Pareto óptimo. GRASP obtiene sus mejores resultados con el mayor grado de aleatoriedad, 0,9 en el valor de parámetro que controla la aleatoriedad frente a la voracidad, favoreciendo la exploración del espacio de búsqueda. NSGA II utiliza una probabilidad de mutación $1/n$ para n requisitos. Sus mejores resultados los obtuvo con los mayores tamaños de población, siempre manteniendo en 10 000 el número de ejecuciones de la función de evaluación. ACS utiliza $n/2$ hormigas, que en el inicio se colocan aleatoriamente en requisitos visibles distintos. Sus mejores resultados se obtuvieron teniendo en cuenta tanto la información heurística como la feromona, pero dando mayor importancia a la heurística.

Respecto a los tiempos consignados en la tabla 3 conviene advertir que los de las metaheurísticas se calcularon con un Intel Pentium 4 a 3,20 GHz y los del algoritmo híbrido con un Intel Core i5 5200U a 2,20 GHz. En ambos casos se limitó la memoria a 1 GiB y la ejecución a un solo núcleo. No obstante, aunque las máquinas son diferentes, es evidente que los tiempos normalizados son muy inferiores en todos los escenarios.

6. Conclusiones y trabajo futuro

Se ha presentado un nuevo algoritmo para la resolución del problema NRP con interdependencias funcionales. Este algoritmo es híbrido, pues incorpora un algoritmo exacto para resolver un problema transformado sin interdependencias y una heurística que permite reducir el problema original al transformado. Posteriormente a la resolución del problema transformado, el algoritmo produce una solución al problema original, quizás tras resolver una serie de problemas subsidiarios más sencillos.

Se ha evaluado el algoritmo propuesto con conjuntos de datos empleados previamente en la literatura de referencia y comparado con tres técnicas heurísticas. Los resultados preliminares son favorables en la práctica totalidad de los escenarios, exhibiendo no únicamente mejoras en las soluciones obtenidas mediante las otras técnicas, sino además una importante reducción del tiempo de ejecución necesario. El algoritmo híbrido es, en esencia, determinista, por lo que no requiere ajuste de parámetros ni ejecuciones adicionales.

Entre las posibles líneas de trabajo futuro cabe destacar la realización de un estudio estadístico detallado, la ampliación del número de experimentos, la consideración de otras interdependencias aparte de las de precedencia y combinación, y la comparación del algoritmo híbrido con otras técnicas, tanto heurísticas como exactas.

Referencias

1. del Águila, I.M., del Sagrado, J., Chicano, F., Alba, E.: Resolviendo un problema multi-objetivo de selección de requisitos mediante resolutores del problema sat. In:

XX Jornadas de Ingeniería del Software y Bases de Datos. SISTEDES (2015)
2. van den Akker, J.M., Brinkkemper, S., Diepen, G., Versendaal, J.: Determination of the next release of a software product: an approach using integer linear programming. In: Proceeding of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ'05. pp. 247–262 (2005)
3. Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.: The next release problem. Information & Software Technology 43(14), 883–890 (2001)
4. Baker, P., Harman, M., Steinhöfel, K., Skaliotis, A.: Search based approaches to component selection and prioritization for the next release problem. In: 22nd IEEE International Conference on Software Maintenance (ICSM 2006), 24-27 September 2006, Philadelphia, Pennsylvania, USA. pp. 176–185 (2006)
5. Beier, R., Vöcking, B.: Random knapsack in expected polynomial time. Journal of Computer and System Sciences 69(3), 306–329 (2004)
6. Dantzig, G.B.: Discrete variable extremum problems. Operations Research 5, 266–277 (1957)
7. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 26(1), 29–41 (1996)
8. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. Journal of global optimization 6(2), 109–133 (1995)
9. Garey, M.R., Johnson, D.S.: Strong NP-completeness results: Motivation, examples, and implications. Journal of ACM 25, 499–508 (1978)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W. H. Freeman (1979)
11. Greer, D., Ruhe, G.: Software release planning: an evolutionary and iterative approach. Information and Software Technology 46, 243–253 (2004)
12. Harman, M., Krinke, J., Medina-Bulo, I., Palomo-Lozano, F., Ren, J., Yoo, S.: Exact scalable sensitivity analysis for the next release problem. ACM Transaction on Software Engineering Methodology 23(2), 19:1–19:31 (2014)
13. Hochbaum, D.S. (ed.): Approximation Algorithms for NP-hard Problems, chap. 9, pp. 346–398. PWS Publishing Company (1997)
14. Jung, H.W.: Optimizing value and cost in requirements analysis. IEEE Software 15(4), 74–78 (1998)
15. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
16. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Willey & Sons (1990)
17. Nemhauser, G.L., Ullmann, Z.: Discrete dynamic programming and capital allocation. Management Science 15, 494–505 (1969)
18. Pitangueira, A.M., Maciel, R.S.P., de Oliveira Barros, M.: Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. Journal of Systems and Software 103, 267–280 (May 2015)
19. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Requirements interaction in the next release problem. In: Proceedings of the 13th annual conference companion on genetic and evolutionary computation. pp. 241–242. ACM (2011)
20. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Multi-objective ant colony optimization for requirements selection. Empirical Software Engineering 20(3), 577–610 (2015)
21. Veerapen, N., Ochoa, G., Harman, M., Burke, E.K.: An integer linear programming approach to the single and bi-objective next release problem. Information and Software Technology 65, 1–13 (2015)