Resolviendo un problema multi-objetivo de selección de requisitos mediante resolutores del problema SAT

Isabel del Águila[1], José del Sagrado[1],
Francisco Chicano[2], and Enrique Alba[2]

[1] Universidad de Almería, España
{imaguila, jsagrado}@ual.es
[2] Universidad de Málaga, España
{chicano, eat}@lcc.uma.es

Resumen El problema de selección de requisitos (o *Next Release Problem*, NRP) consiste en seleccionar el subconjunto de requisitos que se va a desarrollar en la siguiente versión de una aplicación software. Esta selección se debe hacer de tal forma que maximice la satisfacción de las partes interesadas a la vez que se minimiza el esfuerzo empleado en el desarrollo y se cumplen un conjunto de restricciones. Este es un problema de optimización combinatorio multi-objetivo para el que se han utilizado en el pasado técnicas heurísticas y metaheurísticas en su resolución, ya que es NP-difícil. En el presente trabajo proponemos la traducción de este problema a lógica proposicional y el uso de resolutores del problema SAT en una estrategia para encontrar el frente de Pareto de forma exacta.

Palabras clave Selección de requisitos, *next release problem*, optimización multi-objetivo, resolutores SAT

1. El problema de la siguiente versión

Los clientes demandan que se incorporen en el producto software una serie de nuevas características o requisitos, pero no todas las necesidades de los clientes pueden ser satisfechas en la siguiente versión, sobre todo cuando las expectativas de los clientes son altas, los plazos son cortos y los recursos son limitados. En estos casos se hace necesario seleccionar, de entre los requisitos candidatos, cuáles serán los siguientes a ser desarrollados. En NRP intervienen muchos factores [4]. Por un lado, cada requisito implica un coste en términos de esfuerzo que la empresa debe asumir, pero los recursos de la empresa son limitados y no es posible desarrollarlos todos. Por otro lado, ni todos los clientes son igualmente importantes para la empresa, ni los requisitos son igualmente importantes para los clientes. Los factores de mercado también pueden influir en este proceso de selección: la empresa puede estar interesada en satisfacer las necesidades de los clientes más nuevos o en garantizar que cada cliente ve cumplido al menos uno de sus requisitos propuestos.

Sea $\mathbf{R} = \{r_1, r_2, \ldots, r_n\}$ el conjunto de requisitos que aún no han sido desarrollados y que han sido propuestos por un conjunto de m clientes. Cada cliente i tiene un peso asociado $w_i \in \mathbb{R}$ que mide su importancia dentro del proyecto. Cada requisito $r_j \in \mathbf{R}$ tiene un coste c_j para la empresa si se desarrolla. El valor del requisito r_j para el cliente i se representa con $v_{ij} \in \mathbb{R}$. La satisfacción, s_j, o valor añadido por la inclusión de r_j en la siguiente versión del software, se puede calcular como la suma ponderada de los valores de importancia de los clientes, $s_j = \sum_{i=1}^{m} w_i * v_{ij}$. Los requisitos interaccionan entre ellos, imponiendo un orden de desarrollo determinado, lo que limita las alternativas para ser elegidos [2,5]. Las interacciones funcionales entre requisitos se clasifican en:

- *Implicación o precedencia.* $r_i \Rightarrow r_j$. Un requisito r_j no puede ser elegido, si previamente otro requisito r_i no ha sido implementado.
- *Combinación o acoplamiento.* $r_i \odot r_j$. Los requisitos r_i y r_j deben ser incluidos de forma conjunta en el software.
- *Exclusión.* $r_i \oplus r_j$. El requisito r_i no puede ser incluido junto con el r_j.

Si llamamos $X \subseteq R$ al conjunto de requisitos seleccionados, el coste y el valor de X vienen dados por las funciones:

$$coste(X) = \sum_{j, r_j \in X}^{n} c_j \quad \text{y} \quad valor(X) = \sum_{j, r_j \in X}^{n} s_j, \tag{1}$$

respectivamente. Consideraremos una versión multi-objetivo del problema que minimice el coste y maximice el valor del conjunto de requisitos seleccionados.

2. Transformación del problema

El problema SAT consiste en determinar si una fórmula de lógica proposicional es satisfacible (existe un modelo que la hace cierta) o no. A pesar de su dificultad (es NP-completo), los resolutores de SAT han evolucionado hasta ser capaces de resolver instancias de millones de variables en pocas horas. Este hecho, junto con algunos resultados previos [1] nos hace preguntarnos si podremos usar los avances de la comunidad SAT para resolver con mayor eficacia el problema NRP multi-objetivo que aquí nos planteamos.

Podemos transformar el problema de selección de requisitos en un programa lineal binario. Para ello, utilizaremos una variable binaria (puede tomar valores 0 y 1) por cada requisito seleccionable. Por simplicidad, aquí usaremos para dichas variables el mismo nombre que los requisitos asociados: r_1, r_2, etc. Para formar el programa lineal, es necesario transformar cada interacción funcional entre requisitos en una igualdad o desigualdad de expresiones lineales. Estas transformaciones se realizan de acuerdo al siguiente esquema:

- Implicación $r_i \Rightarrow r_j$: $r_j \leq r_i$.
- Combinación $r_i \odot r_j$: $r_i = r_j$.
- Exclusión $r_i \oplus r_j$: $r_i + r_j \leq 1$.

Los dos objetivos a optimizar son el coste (minimizar), $\sum_{j=1}^{n} c_j r_j$, y el valor (maximizar), $\sum_{j=1}^{n} s_j r_j$. Estos dos objetivos están contrapuestos, por lo que no existe una solución que satisfaga ambos. En su lugar, debemos proporcionar un conjunto de soluciones Pareto óptimas [3]. Es decir, un conjunto de soluciones que solo pueden mejorar en un objetivo si empeoran en otro. En nuestro caso particular, queremos todas las soluciones para las que minimizar el coste implique reducir el valor para los clientes.

Para resolver este problema usaremos `minisat+`[3], que es capaz de resolver problemas de programación lineal binaria usando como motor de búsqueda el resolutor `minisat`. No obstante, `minisat+` solo es capaz de resolver un objetivo cada vez. Para obtener el frente de Pareto completo tendremos que convertir uno de los objetivos en una restricción. Cuando tratemos de minimizar el coste, estableceremos una cota inferior, A, para el valor y resolveremos el problema:

$$\text{mín} : \sum_{i=1}^{n} c_j r_j \quad \text{sujeto a:} \quad \sum_{j=1}^{n} s_j r_j \geq A. \tag{2}$$

A la solución devuelta para este problema la llamaremos $minCoste(A)$. Si no existe tal solución supondremos que devuelve $\perp$. Cuando queramos maximizar el valor, estableceremos una cota superior para el coste, B, y resolveremos el problema:

$$\text{máx} : \sum_{j=1}^{n} s_j r_j \quad \text{sujeto a:} \quad \sum_{j=1}^{n} c_j r_j \leq B. \tag{3}$$

A la solución devuelta para este problema la llamaremos $maxValor(B)$. Si no existe tal solución supondremos que devuelve $\perp$. Intercalando apropiadamente estos dos problemas es posible obtener el frente de Pareto completo. El Algoritmo 1 muestra el pseudocódigo para obtenerlo.

Algorithm 1 Algoritmo para calcular el frente de Pareto

```
1: FP ← ∅ // Frente de Pareto
2: cmax ← Σ_{j=1}^{n} c_j
3: x ← maxValor(cmax)
4: while x ≠ ⊥ do
5:     x ← minCoste(valor(x))
6:     FP ← FP ∪{x}
7:     cmax ← coste(x) − 1
8:     x ← maxValor(cmax)
9: end while
```

Esta técnica se conoce con el nombre de *ε-constraint method* [6]. En un trabajo reciente publicado tras la escritura de este artículo, Veerapen *et al.* [8], usan el mismo enfoque para este mismo problema usando CPLEX como resolutor de los programas lineales.

[3] `http://minisat.se`

3. Caso de estudio

El caso de estudio considerado es el primero que aparece en [7], con 20 requisitos. Omitimos los detalles por falta de espacio, pero el lector interesado los puede encontrar en el artículo original. Al aplicar el Algoritmo 1 obtuvimos el frente de Pareto exacto con 38 soluciones en 1,4 segundos. La máquina utilizada tenía procesador Intel Core i5-2320 a 3 GHz y sistema operativo Linux (Ubuntu 14.04.2 LTS). En [7], se calculó una aproximación de parte del frente de Pareto (caracterizada por tener un coste menor o igual al 70 % de la suma del coste de todos los requisitos) usando tres algoritmos: GRASP, NSGA-II y ACS (*Ant Colony System*). Estos algoritmos tardaron entre 0,8 y 2,0 segundos en obtener dicha aproximación en una máquina con procesador Pentium 4 a 3,2 GHZ. Por tanto, para este caso concreto vemos que el tiempo de ejecución de la propuesta exacta es del mismo orden de magnitud que los algoritmos aproximados usados anteriormente y, además, garantiza la optimalidad de las soluciones.

4. Conclusiones

Este trabajo preliminar muestra cómo resolver el problema de la siguiente versión multi-objetivo empleando resolutores del problema SAT para encontrar el frente de Pareto de forma exacta. Para ello se trasforma el problema en un programa lineal binario y se define un algoritmo para obtener el frente de Pareto completo. En futuros trabajos, es necesario abordar la cuestión de la escalabilidad del enfoque cuando aumenta tanto el tamaño del problema como el número de objetivos. Asimismo, se debería comparar la efectividad de los resolutores SAT con las técnicas de programación lineal entera.

Referencias

1. Arito, F., Chicano, F., Alba, E.: On the application of SAT solvers to the test suite minimization problem. In: Proceedings of SSBSE. p. 45–59. Springer (2012)
2. Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., och Dag, J.N.: An industrial survey of requirements interdependencies in software product release planning. In: Proceedings of IEEE RE. pp. 84–93. IEEE Computer Society (2001)
3. Ehrgott, M.: Multicriteria optimization. Springer (2005)
4. IIBA, A.: guide to the business analysis body of knowledge (babok guide). International Institute of Business Analysis (IIBA) (2009)
5. Karlsson, J., Olsson, S., Ryan, K.: Improving practical support for large-scale requirement prioritising. Requirement Engineering 2(1), 51–60 (1997)
6. Mavrotas, G.: Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. Applied Mathematics and Computation 213(2), 455 – 465 (2009)
7. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Multi-objective ant colony optimization for requirements selection. Empirical Software Engineering pp. 1–34 (2013)
8. 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(0), 1 – 13 (2015)