XVIII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2018)

CAEPIA 4:
Optimización y Computación de Altas Prestaciones

La importancia de un aplicativo web como ayuda para la conformación de grupos de trabajo colaborativo

Franco Esteban Córdoba Pérez
Universidad de Nariño
San Juan de Pasto, Colombia
franco12594@udenar.edu.co

Oscar Revelo Sánchez
Universidad de Nariño
San Juan de Pasto, Colombia
orevelo@udenar.edu.co

Alexander Barón Salazar
Universidad de Nariño
San Juan de Pasto, Colombia
abaron_98@udenar.edu.co

Resumen– En este artículo se describe la importancia que tiene una herramienta web como ayuda en la conformación de grupos de trabajo colaborativo donde se busca que estos sean de igual tamaño y equitativos respecto a más de un atributo, siendo este un paso importante y complejo para diseñar actividades eficaces de trabajo colaborativo. Dando paso a una solución sistematizada debido a que la búsqueda exhaustiva no siempre será conveniente debido a la explosión combinatoria que puede presentarse, se propone una herramienta basada en algoritmos genéticos donde intervienen los diferentes tipos de operadores genéticos para el proceso de selección hasta dar con una solución satisfactoria.

Palabras clave-- Herramienta web, agrupamiento, trabajo colaborativo, algoritmos genéticos.

I. INTRODUCCIÓN

La calidad y la mejora continua son los pilares fundamentales en el progreso de diferentes áreas, como lo pueden ser las áreas laborales o de educación. Debido a esto se deben buscar nuevas estrategias para la mejora, una de estas es el trabajo colaborativo que se define como una estrategia de cooperación y aprendizaje en grupos de trabajo, en donde se organizan pequeños grupos en los que cada miembro tiene objetivos en común que han sido establecidos previamente y sobre los cuales se realizará el trabajo, en oposición al trabajo individual y competitivo de partes separadas de miembros.

La formación de grupos es un paso importante y complejo para diseñar actividades eficaces de trabajo colaborativo. A través de la selección adecuada de los individuos a un grupo, es posible crear ambientes que favorezcan la aparición de interacciones significativas, y, por lo tanto, aumentar el aprendizaje sólido, el crecimiento intelectual y la correcta elaboración de una labor.

Debido a que el trabajo colaborativo puede ser usado en diferentes ámbitos existen diferentes tipos y cantidad de características que pueden ser evaluadas para la conformación de grupos como lo pueden ser habilidades laborales, estilos de aprendizaje, calificaciones, rasgos de personalidad, y así sucesivamente, debido a esto existirán muchas combinaciones para formar grupos dando lugar a espacios de búsqueda muy grandes por lo que no se podría formar grupos mediante técnicas determinísticas. Para la solución de este tipo de problemas destacan los Algoritmos Evolutivos que han demostrado ser especialmente adecuados para problemas combinatorios y dentro de estos los Algoritmos Genéticos los cuales simulan en un computador el proceso de selección del "más apto" obteniendo así la población más óptima según los criterios de selección.

En el presente artículo se presenta una investigación que busca mostrar la importancia de una herramienta de agrupamiento para la conformación de grupos de trabajo colaborativo, basándose en diferentes características de los integrantes para poder conformar grupos heterogéneos utilizando como técnica de agrupamiento y de optimización los algoritmos genéticos. El resto del artículo está dividido de la siguiente forma: En la siguiente sección se aborda el marco teórico donde se hablara del trabajo colaborativo, algoritmos genéticos y antecedentes que tienen relación con la investigación. En la sección posterior se describe el modelo propuesto para la conformación de grupos y la herramienta de agrupamiento desarrollada aplicando este modelo. Luego se da a conocer los resultados obtenidos mediantes varias pruebas. Finalmente se presentan las conclusiones.

II. MARCO TEÓRICO

A. *Trabajo Colaborativo*

El trabajo colaborativo se define como aquellos procesos intencionales de un grupo para alcanzar objetivos específicos, siendo una metodología de enseñanza y de realización de la actividad laboral basada en la creencia que el aprendizaje y la actividad laboral se incrementa cuando se desarrollan destrezas cooperativas para aprender y solucionar los problemas y acciones en las cuales los individuos se ven inmersos. [1] Este combina el trabajo individual dentro de un grupo de trabajo para lograr un objetivo común, en condiciones en las que todos los miembros deben cooperar en la realización de una tarea, con lo cual cada individuo y miembro es responsable por el resultado absoluto beneficiando a todos los miembros del grupo. Este permite alcanzar metas gracias al trabajo en conjunto de todos sus integrantes reduciendo así la carga del trabajo individual y mejorando diferentes habilidades grupales e individuales.

El trabajo colaborativo se caracteriza por la igualdad que debe tener cada individuo en el proceso de aprendizaje, la conexión, profundidad y bidireccionalidad de la experiencia, siendo ésta una variable en función del nivel de competitividad

existente, la distribución de responsabilidades, la planificación conjunta y el intercambio de roles. [2]

La principal importancia del trabajo colaborativo se describe como: "cada participante asume su propio ritmo y potencialidades, impregnando la actividad de autonomía, pero cada uno comprende la necesidad de aportar lo mejor de sí al grupo para lograr un resultado sinérgico, al que ninguno accedería por sus propios medios; se logra así una relación de interdependencia que favorece los procesos individuales de crecimiento y desarrollo, las relaciones interpersonales y la productividad". [3]

El trabajo colaborativo implica una cooperación que es muy útil para sus integrantes según el constructivismo social, el cual afirma que las personas activamente construyen conocimiento mientras interactúan con su ambiente ya que trabajan conjuntamente para agilizar la formación de algún tema en específico, generando una comunidad de aprendizaje dado el hecho de que se está realizando una tarea en compañía, facilitado por la interacción social, la interacción entre pares, la cooperación y la evaluación. También se debe tener en cuenta que en el trabajo colaborativo se constituyen grupos según criterios de heterogeneidad respecto tanto a características personales como de habilidades y competencias de sus miembros, lo cual propicia la complementariedad. [4]

En muchas organizaciones, la razón de su éxito es el trabajo en grupo efectivo pero generalmente los grupos que forman sin consideraciones cuidadosas (es decir, aleatoriamente) a menudo esto causa problemas tales como la participación desproporcionada de individuos, desmotivación y resistencia al trabajo en grupo en las actividades futuras.

Se debe tener en cuenta que al formar varios grupos de trabajo estos deben ser totalmente equitativos ya que de lo contrario se generara desigualdad en cuanto a rendimiento de ciertos grupos frente a otros, de esta manera afectando su avance en el tema que se esté tratando y la desigualdad de oportunidades. En otras palabras el objetivo del trabajo colaborativo es crear grupos que sean lo más similares entre sí, pero que al interior de cada uno de ellos se potencie las diferencias individuales de los integrantes que los conforman. Permitiendo así obtener logros globales que son similares entre sí.

A través del proceso de selección de las personas a participar en un grupo, se puede analizar y combinar características tales como antecedentes culturales, conocimientos, habilidades, estilos de aprendizaje, las funciones, objetivos, intereses, las calificaciones, la disponibilidad para reuniones, y así sucesivamente.

B. Algoritmos Genéticos

La computación evolutiva reúne a todos aquellos métodos de optimización inspirados en la teoría darwiniana de la evolución de las especies. Existe una gran variedad de modelos de computación evolutiva, los cuales han sido propuestos y estudiados en las últimas décadas. Desde el punto de vista de la algoritmia, los algoritmos evolutivos son algoritmos probabilistas, es decir, toman decisiones basándose en una distribución de probabilidad y, por tanto, el mismo algoritmo puede obtener distintos resultados para ejecuciones diferentes sobre los mismos datos. [5]

Los algoritmos evolutivos permiten encontrar buenas soluciones en tiempos razonables. La computación evolutiva se inspira en la teoría de la evolución de los seres vivos, retomando los conceptos de selección natural y genética. Estos algoritmos permiten abordar problemas complejos que surgen en las ingenierías y los campos científicos: problemas de planificación de tareas, horarios, tráfico aéreo y ferroviario, búsqueda de caminos óptimos, optimización de funciones, etc. [6]

Una debilidad de estos es que algunas veces se puede encontrar la solución de manera muy rápida en otros casos se tardan mucho en encontrarse o inclusivo nunca converger en un resultado. De todos modos, tiene otro gran número de ventajas como la diversidad en su aplicación, poder trabajar sobre múltiples soluciones, la eficiencia en el cálculo de una solución objetivo y su fácil ejecución. [7]

Dentro de los algoritmos evolutivos encontramos los algoritmos genéticos los cuales tienen un esquema general con las siguientes propiedades:

- Procesan simultáneamente, trabajan representaciones de posibles soluciones al problema, denominadas individuos.
- La estructura de la población cambia a medida que pasan las iteraciones del algoritmo, denominadas generaciones.
- Para cada generación se realiza un proceso de selección según una función de aptitud, evaluando la probabilidad de que cada individuo permanezca en la población y participe en las operaciones de reproducción (cruce y mutación).

C. Antecedentes

Expertos en el tema del trabajo colaborativo consideran que la conformación de grupos se debe hacer de forma independiente por los miembros de los mismos, lo cual no siempre produce buenos resultados. Debido a esto los estudios se centran principalmente en la interacción o correcto funcionamiento del trabajo colaborativo pero se deja a un lado la conformación de grupos para este. Sin embargo existen trabajos relacionados con el tema.

Por ejemplo FROG (FORMING REASONABLY OPTIMAL GROUPS) que es un trabajo de Investigación realizado en la Universidad de Toronto (Canadá) en el cual se plantea que existen herramientas para facilitar el proceso de conformación de grupos definiendo un modelo matemático para la formación de estos. Se muestra el proceso de implementación de un optimizador que utiliza un algoritmo evolutivo para crear grupos de acuerdo con los criterios del instructor. [8]

También se encuentra el trabajo de investigación "A Group Formation Tool for e-Learning Environments" realizado en la Universidad del Pireo (Grecia) donde se presenta una herramienta para formación de grupos basada en la web, apoyando al instructor para crear grupos homogéneos y heterogéneos basándose hasta en tres criterios para gestionar el agrupamiento. [9]

III. PROCESO DE DESARROLLO

A. *Modelo Propuesto*

El agrupamiento de elementos es un problema combinatorio general que consiste en la repartición de un total de elementos entre un número definido de grupos, generalmente del mismo tamaño, de tal manera que se satisfaga una cierta condición. Aunque a primera vista parezca simple, la complejidad de este problema se focaliza principalmente en dos aspectos. El primero se refiere a la condición que debe ser satisfecha, la cual en el caso más común se trata de obtener grupos "equitativos" u homogéneos considerando una cierta medida de valor para cada elemento. [10]

En muchas organizaciones, los principales proyectos o trabajos se llevan a cabo en grupos de trabajo. Desafortunadamente, no existe un método claro que pueda abordar los pasos para elegir el grupo adecuado. El problema está, en este caso, en la formación óptima de grupos es la explosión combinatoria que se generaría debido al número total de individuos y los grupos a formar, siendo este el segundo aspecto a tener en cuenta. De una manera general el número posible de combinaciones que se podría obtener al desear formar q grupos, con un número total de individuos p ($q<=p$) considerando relevante el ordenamiento de los grupos está dado por la fórmula de combinatoria (1).

$$\binom{p}{q} = \frac{p!}{(p-q)!\,q!} \tag{1}$$

Por ejemplo al repartir 50 individuos en grupos de 5, el valor será de 2.118.760 posibles combinaciones diferentes de grupos, resaltando así que la búsqueda exhaustiva no es la mejor solución en muchos casos. En estos casos los algoritmos meta heurísticos son una buena alternativa debido a que al utilizarlos se podría llegar a una solución satisfactoria aunque no se puede garantizar el hallar una solución óptima, empleando para ello un esfuerzo de cómputo mucho menor. Entre estos algoritmos se pueden encontrar: algoritmo de recocido simulado, búsqueda local, búsqueda tabú, algoritmo de la colonia de hormigas, algoritmos genéticos, siendo este último el cual se ha seleccionado como objeto de estudio para el presente trabajo.

En el modelo propuesto como ya se ha mencionado se busca encontrar grupos similares entre sí, pero respetando la heterogeneidad de la totalidad de los miembros contrastando diferentes características como pueden ser rasgos de personalidad, calificaciones, edad, experiencia, etc.

Una vez obtenidos los datos, si existe un caso donde las características estén definidas de una forma categórica se debe discretizar numéricamente. Por ejemplo los valores "alto", "medio" y "bajo" se podrían cambiar por 1, 2 y 3 respectivamente. Luego se deben estandarizar los valores medidos en diferentes escalas a una escala en común para que no se presenten complicaciones en el cálculo de la función objetivo. Para facilitar esto se aplica la normalización basada en la unidad mostrado en la formula (2).

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{2}$$

Donde X_{max} y X_{min} son el valor máximo y mínimo de la característica correspondiente. Al obtener y organizar las características deseadas para formar los grupos se procede a conformar lo grupos mediante algoritmos genéticos, primero obteniendo el promedio total para cada característica de los m miembros siendo este el objetivo a lograr, buscando que cada uno de los grupos formados se asemejen lo más posible a este promedio. Es decir, si el promedio de una característica es 0,54 entonces cada uno de los grupos formados debería acercarse a ese promedio. En la Figura 1 se muestra un ejemplo donde se tienen 6 miembros, 5 características medidas y se desean formar 3 grupos, se puede evidenciar que cada grupo generado es relativamente similar al promedio.

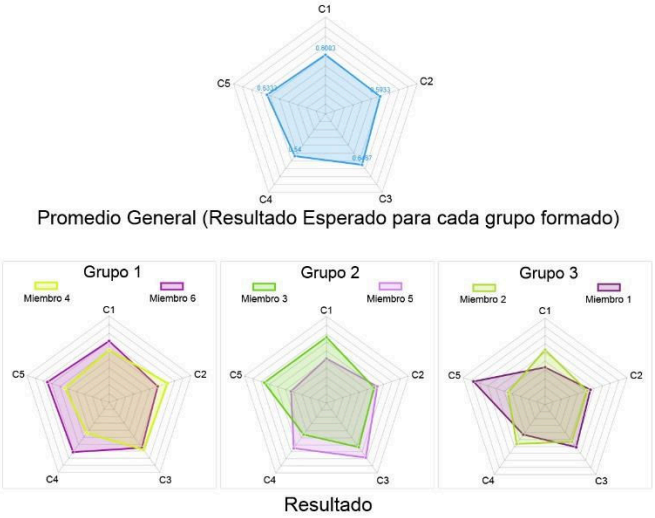

Figura 1. Resultado esperado por el algoritmo propuesto.

Lo siguiente es generar aleatoriamente una población inicial de individuos, entendiendo individuos como una posible solución al problema, teniendo en cuenta que el número de grupos a formar está dado por el número de individuos n y el número de integrantes que se desea tener en cada grupo g, así cada grupo tendrá n/g miembros sin repetición alguna. En dado caso que falten miembros para formar un grupo, se generaran miembros falsos, adoptando el promedio total de cada una de las características de la población total para que estos afecten en un bajo grado el proceso. Así por ejemplo si se desea formar grupos donde cada uno tenga 3 miembros de un total de 30 miembros, se formaran 10 grupos.

Una vez se genera la población inicial se procede a calcular el promedio de cada característica C de cada grupo g de cada individuo i, posteriormente se calcula la sumatoria de las diferencias al cuadrado del promedio de cada característica de cada grupo del individuo y el promedio calculado anteriormente para cada individuo, usando la ecuación (3).

$$D^i = \sum_{g=1}^{G} \left[\left(\overline{C_1} - \overline{X_{g,1}^l}\right)^2 + \left(\overline{C_2} - \overline{X_{g,2}^l}\right)^2 + \cdots + \left(\overline{C_m} - \overline{X_{g,m}^l}\right)^2 \right] \tag{3}$$

Entre menor sea este valor (mínimo 0) más similar serán cada uno de los grupos del individuo con respecto al promedio del total de miembros, siendo este valor de la media de aptitud.

Una vez realizado esto se puede llevar a cabo cada uno de los operados genéticos de un algoritmo genético selección, cruce y mutación, hasta que se alcance la media de aptitud deseada. En el primero se seleccionan lo individuos "más aptos", es decir con mejor media de aptitud para que luego sean clonados (bajo el principio de supervivencia del más fuerte), el segundo consiste en generar "hijos" a partir de los individuos seleccionados. Combinando los genes (miembros) de dos padres diferentes de alguna manera, teniendo en cuenta en no repetir el mismo miembro en diferentes grupos. El último consiste en cambiar de manera aleatoria de uno o más cromosomas de los individuos dando paso así a una nueva generación (bajo el principio que dichos cambios pueden ser favorables). Los individuos a mutar, al igual que los cromosomas que mutan son seleccionados de manera probabilística.

El proceso continúa así hasta alcanzar la media de aptitud deseada o que pasen un número de generaciones deseadas esperando que cada vez se encuentren individuos mejores. Generalmente en trabajos donde se utilizan algoritmos genéticos, la estructura de datos que se emplea para representar un individuo es un vector donde cada posición corresponde un gen de la posible solución. En el modelo propuesto se plantea utilizar una matriz, donde el número de filas corresponde al número de grupos deseado g y el número de columnas corresponde al tamaño máximo de cada grupo n/g. Así cada gen que compone el cromosoma contiene el identificador de un miembro, y su posición dentro de la matriz define el grupo al que pertenecería. Por ejemplo, si se tiene un total de 20 miembros y se desea formar 4 grupos, cada uno tendría exactamente 5 miembros. En este caso un posible individuo, si los miembros son numerados consecutivamente, podría ser como el que se presenta en la Tabla 1.

Tabla 1. Representación de un individuo.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

B. *Herramienta Desarrollada*

Teniendo en cuenta lo expuesto anteriormente se realizó el desarrollo de un sistema de agrupamiento integrado en un aplicativo web modular de código abierto, esta herramienta puede ser útil para personas interesadas en desarrollar actividades colaborativas en sus espacios académicos o laborales utilizando el sistema de agrupamiento para el apoyo a la conformación de grupos de trabajo colaborativo.

Dicha aplicación ha sido nombrada como "CW-TEAMS: Software para la conformación de grupos de trabajo colaborativo basado en algoritmos genéticos" la cual se presenta como una solución informática para la administración de grupos de trabajo colaborativo creando estos a partir de ciertas características de cada integrante, utilizando como técnica de agrupamiento y de optimización los algoritmos genéticos.

Primero se debe subir un archivo de texto con los datos necesarios para formar los grupos con un formato dado en la herramienta (Figura 2). Luego se podrá cambiar algunos parámetros como el título y descripción, así como también escoger que miembros serán los utilizados para conformar los grupos.

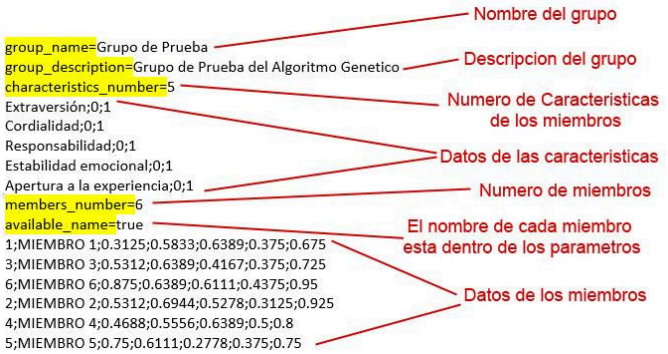

Figura 2. Formato de archivo de texto para subir a la herramienta.

Seguidamente se asignaran los parámetros para la ejecución del algoritmo genético como lo son el número de miembros por grupo, número de generaciones, media de aptitud, porcentaje de selección, y las probabilidades de mutación de individuos y genes (Figura 3). Luego de verificar los datos el aplicativo arrojara los resultados con un formato similar al mostrado en la Figura 1, también permitiendo visualizar el tiempo de generación y parámetros, volver a generar los resultados o descargar estos.

Figura 3. Ventana de asignación de parámetros de la herramienta.

El código fuente de la herramienta desarrollada se encuentra en un repositorio de GitHub, donde también se encuentra un manual de usuario (https://github.com/francoecp/cwteams).

IV. RESULTADOS

Para probar la eficiencia de la herramienta desarrollada y por consiguiente del algoritmo planteado se realizaron algunas pruebas con datos simulados y reales, 3 de las cuales se

presentaran a continuación, estas fueron realizadas con datos reales de estudiantes de la Universidad de donde proceden los autores, en tres diferentes asignaturas, cada estudiante realizó un test de personalidad denominado Big Five [11] el cual mide lo que muchos psicólogos consideran ser las cinco dimensiones fundamentales de personalidad para así obtener las características necesarias para la conformación de grupos, para cada uno de las asignaturas se procedió a crear el archivo necesario para cargarlo a la herramienta.

Para cada prueba se utilizó una asignatura diferente, por consiguiente y favorablemente el total de estudiantes de cada asignatura difiere, con antelación se hicieron diferentes ejecuciones del algoritmo variando sus parámetros, una vez obtenidos resultados estables estos se tomaron para su asignación en las pruebas. Las pruebas fueron realizadas en un computador con un procesador Intel Core i5 de 2,2 GHz y 8 GB de RAM. Los datos, asignaciones y resultados se muestran en la Figura 4.

No. Prueba	Asignatura	Total de estudiantes	Numero deseado de estudiantes por grupo	Total de posibles combinaciones	Número de individuos iniciales	Numero de generaciones	Media de aptitud deseada	Porcentaje de selección (%)	Probabilidad de Mutación de cada individuo (%)	Probabilidad de mutación para los genes de cada cromosoma (grupo) (%)	Tiempo de ejecución del algoritmo (segundos)	Media de aptitud obtenida por el mejor individuo
1	Software Gráfico	24	3	2024	100	800	0,02	30	15	15	0,9704	0,0249
2	Métodos Numéricos	48	3	17296	80	900	0,01	20	10	10	1,1343	0,0432
3	Ingeniería de Software	22	4	7315	90	1000	0,03	40	20	20	0,6652	0,0041

Figura 4. Resultados de las pruebas.

Analizando los datos se infiere que el proceso es muy rápido y los valores de la media de aptitud son muy cercanos al deseado. Esto demuestra la efectividad y viabilidad de la herramienta desarrollada para crear grupos equitativos, la cual utiliza un método heurístico de búsqueda que no garantiza el valor optimo pero si un valor muy cercano a pesar de su baja demanda de recursos y tiempo de generación.

IV. CONCLUSIONES Y TRABAJOS FUTUROS

Considerando el problema principal, donde se desea obtener grupos homogéneos a partir de un conjunto de elementos con varios atributos, es difícil de resolver por métodos analíticos o de búsqueda exhaustiva debido a la explosión combinatoria que puede llegar a presentarse dependiendo del número de elementos y de grupos, se demuestra que una herramienta computacional es de vital importancia para la formación de estos grupos de manera rápida y eficiente.

Con los resultados que se obtuvieron gracias a las pruebas realizadas se pudo comprobar la utilidad del método implementado en la herramienta web ya que logra obtener grupos bastante homogéneos (considerando la medida de aptitud que se escoja), incluso cuando el número de combinaciones posibles es muy elevado, sin que esto implique un elevado tiempo de cómputo.

Por otra parte, a pesar de que la formación del grupo se dice que jugar un papel crítico en términos de mejorar el éxito del trabajo colaborativo y por lo tanto aumentar el progreso de aprendizaje o de la realización de una labor, se observa que hay poca investigación que se ocupa de la formación de grupos de una manera heterogénea. [9-10] Los métodos basados en computación para ayudar en el proceso de formación de grupo no se han explorado completamente, a pesar de la popularidad de las herramientas basadas en la web para apoyar el trabajo colaborativo, los desarrolladores se centran principalmente en la interacción de colaboración para hacer frente a las técnicas de intercambio de información y recursos entre los miembros.

En el futuro, el resultado del algoritmo implementado se compara con otros algoritmos y métodos de optimización. Finalmente, se plantea incorporar en la herramienta un módulo para la gestión de cuestionarios donde primeramente se presentaran diferentes test de personalidad para obtener los datos necesarios (características) que son de utilidad para la conformación de grupos.

REFERENCIAS

[1] E. E. Silva Beltrán y I. Morales Hernández, "Autonomía Y Trabajo Colaborativo", *XII Congreso Internacional De Teoría De La Educación*, 2011.

[2] M. E. Calzadilla, Aprendizaje colaborativo y tecnologías de la información y la comunicación, *Revista Iberoamericana de Educación*, vol. 29, n. º 1, pp. 1-10, ene. 2002.

[3] B. C. Castro, La interrelación cognitiva entre alumno y docente, *Memorias de la Semana de Divulgación y Video Científico UJAT,* pp. 11-16, 2006.

[4] C. L. Fraile, Hacia una comprensión del aprendizaje cooperativo. *Revista de Psicodidáctica*, pp. 59-76, 1997.

[5] I. R. Fernandez, Aprendizaje Evolutivo De Reglas Para Agrupamiento Jerárquico De Datos En Robótica Móvil, *PhD thesis*, *Universidad De Santiago De Compostela,* 2016.

[6] C. C. Lourdes Araujo, Algoritmos Evolutivos Un Enfoque Práctico, *RA-MA Editorial*, 2009.

[7] S. G. J. M. Andrés F. Deleon, Uso de tests de aptitud y algoritmos genéticos para la conformación de grupos en ambientes colaborativos de aprendizaje, *Avances en Sistemas e Informática,* pp. 165-172, 2009.

[8] D. H. F. P. Michelle Craig. Forming Reasonably Optimal Groups (FROG), *Proceedings of the 16th ACM international conference on Supporting group work*, pp. 141-150, 2010.

[9] K. A. P. Christos E. Christodoulopoulos. A Group Formation Tool for e-Learning Environments, *Proceedings of IEEE Conference on Tools with Artificial Intelligence*, pp. 117–123, 2010.

[10] J. C. R. Y. F. C. Julián Moreno, Agrupamiento Homogéneo De Elementos Con Múltiples Atributos Mediante Algoritmos Genéticos. *DYNA*, 2009.

[11] O. E. J. Verónica Benet-Martínez. Los Cinco Grandes Across Cultures and Ethnic Groups: Multitrait Multimethod Analyses of the Big Five in Spanish and English. Journal of Personality and Social Psychology, pp. 729-750, 1998.

Uso de CMSA para resolver el problema de selección de requisitos

Miguel Ángel Domínguez-Ríos
Universidad de Málaga
miguel.angel.dominguez.rios@uma.es

Francisco Chicano
Universidad de Málaga
chicano@lcc.uma.es

Enrique Alba
Universidad de Málaga
eat@lcc.uma.es

***Resumen*—El problema de selección de requisitos ha sido abordado en multitud de ocasiones en la literatura, tanto en su versión monoobjetivo, como biobjetivo. En su formulación usual, consiste en seleccionar un subconjunto de requisitos que se van a desarrollar en la siguiente versión de una aplicación software. Cada requisito tiene un coste y está asociado a uno o varios clientes. Para satisfacer a un cliente es necesario implementar todos los requisitos que desea. Cada cliente tiene asociado un valor para la empresa. Al resolver el problema se pretende maximizar la suma del valor de los clientes satisfechos sin superar el presupuesto de desarrollo de la siguiente versión. Existen restricciones adicionales asociadas a los requisitos que hacen el problema más complejo. Para un número elevado de variables y restricciones, los algoritmos exactos pueden resultar inviables y ser necesario recurrir a métodos heurísticos. En este trabajo proponemos el uso de la heurística *Construct, Merge, Solve and Adapt* para resolver el problema de selección de requisitos, y comparamos los resultados obtenidos con tres métodos, un algoritmo exacto y dos heurísticos, para instancias con un gran número de requisitos, clientes y dependencias.**

I. Introducción

El problema de la siguiente versión (*Next Release Problem, NRP*) tiene como objetivo satisfacer las necesidades de los clientes, minimizando el esfuerzo de desarrollo de la siguiente versión de una aplicación software. Fue inicialmente propuesto por Bagnall *et al*. [2]. La idea consiste en encontrar un subconjunto de requisitos o un subconjunto de clientes que maximice una cierta propiedad, como la satisfacción de los clientes, mientras se mantiene una restricción de cota superior sobre otra propiedad, que generalmente es el coste. Hay que tener en cuenta que el NRP solo tiene en cuenta la siguiente versión software. La planificación sobre varias versiones es más compleja y se estudia dentro del *Release Planning Problem* [8], que puede considerarse como una generalización del NRP. En [13] se pueden ver aproximaciones híbridas del *Release Planning Problem*.

Recientemente, Veerapen *et al*. [14], resolvieron el problema con técnicas ILP tomando como resolutor el software de IBM, CPLEX, usando unas instancias proporcionadas por Xuan, *et al*. [16]. Éstas fueron resueltas en pocos segundos cuando años atrás era impensable, obviamente debido a las capacidades de cómputo de la actual generación de computadores y los avances en ILP. NRP es NP-duro porque el problema de la mochila se puede reducir a NRP.

Existen dos enfoques para modelar el actual NRP monoobjetivo. Por un lado, se puede exigir que todas las demandas de los clientes sean satisfechas para poder incluir un requisito en la siguiente versión software [1], o que la satisfación del cliente sea parcialmente aceptable [7], es decir, que se permitan satisfacer solo ciertas demandas de cierto cliente, y esto suponga también un aumento de la función objetivo. En este trabajo consideramos el primer caso, es decir, será necesario satisfacer todas las demandas del cliente, de acuerdo con la formulación general dada por Veerapen et. al. [14].

Blum, *et al*. [3] publicaron un trabajo genérico sobre el uso de una metaheurística híbrida que resolvía problemas de optimización combinatoria. Concretamente, su trabajo es una instanciación específica dentro de un marco conocido en la literatura como *Generate-and-Solve* [9], y se conoce como *Construct, Merge, Solve & Adapt* (CMSA). La idea que proponen para resolver el problema consiste en generar una subinstancia del problema original, donde la solución óptima sea también una solución factible del problema original. La subinstancia se genera mediante la selección de ciertos componentes a tener en cuenta. Después se obtiene, mediante un resolutor exacto, una solución óptima para el subproblema. Esta solución obtenida podría estar muy alejada del verdadero óptimo. Sin embargo, los componentes que forman parte de la solución óptima del subproblema tienen un peso mayor para formar parte de la siguiente subinstancia, al menos durante un tiempo determinado. Así, se irán resolviendo subproblemas y guardando la mejor solución obtenida hasta el momento, hasta que se alcance un tiempo límite de ejecución previamente establecido. La descripción de este proceso general tiene también una fase de adaptación donde aquellos componentes que forman parte del subproblema y no han aparecido como parte de la solución óptima dentro de un número establecido de iteraciones, son desechados, lo cual no implica que puedan ser seleccionados posteriormente para formar parte de la solución. Este método híbrido, donde una heurística se mezcla con el uso de un resolutor exacto, tiene la ventaja de obtener soluciones de alta calidad (debido al uso de algoritmos exactos) a la vez que resultan rápidas, por resolver de manera exacta solo subproblemas de un tamaño reducido. La idea de resolución de instancias de un problema reducido ha sido ya explorada en varios trabajos, como por ejemplo en [4] y [10]. El método CMSA ha sido aplicado a algunos problemas, como el *Minimum Common String Partition* (MCSP) [12] o el *Minimum Covering Arborescence* (MCA) [15], entre otros, pero, hasta donde sabemos, no ha sido aplicado a NRP.

II. Formulación del NRP

Siguiendo la formulación general del NRP dada por [14], estamos interesados en maximizar el beneficio de los clientes estableciendo además un límite total de coste debido a los requisitos. Supongamos que disponemos de n posibles requisitos y m clientes. Sean $x_1, x_2, \ldots, x_n$ las variables binarias que indican la inclusión o no de cada requisito en la siguiente versión. Sean $y_1, y_2, \ldots, y_m$ las variables binarias que indican si la demanda de cada cliente ha sido completamente satisfecha o no. Sean $c_1, c_2, \ldots, c_n$ los costes asociados a los requisitos, y sean $w_1, w_2, \ldots, w_m$ los valores de beneficio asociados a los clientes. Llamemos b al límite de presupuesto para el desarrollo de la siguiente versión del software. Es posible que algunos requisitos tengan dependencias. En [17] se definieron tres posibles tipos de relaciones entre requisitos, que nosotros hemos adaptado dando nombre a cada una de ellas.

- Combinación ($r_i \bigodot r_j$): los requisitos r_i y r_j deben aparecer juntos en cualquier selección.
- Exclusión ($r_i \bigoplus r_j$): los requisitos r_i y r_j no pueden aparecer juntos en una selección.
- Implicación ($r_i \Rightarrow r_j$): El requisito r_i requiere que el requisito r_j se incorpore en la misma selección.

El primer conjunto de dependencias indica que existen requisitos que deben aparecer (o no) conjuntamente. Cuando hay en el problema dependencias de este tipo se puede modelar el problema de manera que todas las variables con dependencia mutua sean substituidas por una nueva variable. De esta forma, la nueva variable formará parte de la solución si y solo si el conjunto de variables que dependen mutuamente forma parte de la solución. Por ello, no vamos a tener en cuenta esta situación, y consideraremos que cada variable representa a un único requisito, o directamente a un conjunto de requisitos que debe aparecer conjuntamente. El segundo conjunto de dependencias establece que pueden existir situaciones en las que dos requisitos determinados no pueden ser desarrollados conjuntamente en la siguiente versión. En este trabajo, de acuerdo con las instancias usadas por [17], tampoco vamos a considerar este tipo de restricciones. El tercer conjunto de dependencias implica que ciertos requisitos necesitan de la inclusión de otros prerrequisitos previos. Un requisito puede necesitar previamente de la inclusión de otro, u otros, y éstos, a su vez, de otros requisitos. En este trabajo sí vamos a considerar este tipo de dependencias.

En relación a los clientes, cada uno de ellos demanda una serie concreta de requisitos, que deben ser completamente satisfechos para poder considerar como beneficio el peso que proporciona el cliente. No se admite satisfacción parcial de un cliente en este trabajo.

Sea P el conjunto de pares (i, j) donde el requisito i es un prerrequisito para el requisito j. Sea Q el conjunto de pares (i, k) donde el requisito i es demandado por el cliente k. Con todas estas consideraciones podemos modelar el NRP como un problema ILP de la siguiente manera:

$$\text{máx} \sum_{i=1}^{m} w_i y_i \tag{1}$$

sujeto a:

$$\sum_{i=1}^{n} c_i x_i \leq b \tag{2}$$

$$x_i \geq x_j \quad \forall (i,j) \in P \tag{3}$$

$$x_i \geq y_k \quad \forall (i,k) \in Q \tag{4}$$

$$x_i, y_k \in \{0,1\} \quad \forall i = 1, \ldots, n, \forall k = 1, \ldots, m \tag{5}$$

III. Método CMSA

Christian Blum *et al.* [3] establecieron un algoritmo genérico híbrido que fue usado para resolver diversos problemas combinatorios. Exponemos en el Algoritmo 1 el pseudocódigo de CMSA, que también será la base de nuestros algoritmos heurísticos, y explicamos brevemente su significado a continuación.

Algoritmo 1 *Construct, Merge, Solve and Adapt (CMSA)*

1: **input**: *instancia I, valores de los parámetros* n_a *y* age_{max}
2: S_{bsf} = `NULL`; $C' = \emptyset$
3: $age[c] = 0$ para todo $c \in C$
4: **mientras** *tiempo de CPU no excedido* **hacer**
5: **para** $i = 1$ hasta n_a **hacer**
6: *S=ProbabilisticSolutionGenerator(C)*
7: **para todo** $c \in S$ **y** $c \notin C'$ **hacer**
8: $age[c] = 0$; $C' = C' \cup c$
9: **fin para**
10: **fin para**
11: S'_{opt}=*ApplyExactSolver*(C')
12: **si** S'_{opt} es mejor que S_{bsf} **entonces** $S_{bsf} = S'_{opt}$
13: *Adapt* (C', S'_{opt}, age_{max})
14: **fin mientras**
15: **output**: S_{bsf}

El algoritmo parte de la suposición de que cada solución válida para una instancia I de un problema P dado, se puede representar mediante un subconjunto de un conjunto C de componentes. Este concepto de componente puede referirse a un conjunto de variables del problema, o representar algo distinto. Lo importante es que si tenemos a nuestra disposición todos los componentes para formar una solución y escogemos la mejor de ellas mediante un algoritmo exacto, resolvemos el problema original. Si solo disponemos de un subconjunto de C para formar soluciones, entonces estaremos resolviendo una subinstancia de la instancia original. El conjunto $S \subset C$ representa una solución, no necesariamente óptima, al problema original. Por otro lado, el subconjunto C' es un contenedor que va guardando todas las componentes seleccionadas y que serán utilizadas para resolver una subinstancia de *I*. Por tanto, también es $C' \subset C$. El bucle principal se ejecuta mientras no se alcance el máximo tiempo de ejecución permitido al algoritmo. Cada iteración del algoritmo se puede dividir en cuatro partes:

1. *Construct*: se generan probabilísticamente n_a soluciones (línea 6 del Algoritmo 1).
2. *Merge*: las componentes de dichas soluciones son añadidas al contenedor $C^{'}$ y, a cada componente c que ha sido nuevamente añadido al contenedor, se le inicializa su edad a cero.
3. *Solve*: se resuelve la subinstancia formada por las componentes del contenedor (línea 11). Si la solución encontrada es mejor que la obtenida hasta el momento, se actualiza.
4. *Adapt*: el contenedor se adapta en base a la solución obtenida al aplicar el resolutor exacto. Cada componente de la solución óptima del subproblema reinicializa su edad a cero, y el resto de componentes aumentan su edad en una unidad. Si se supera un límite máximo de edad preestablecido, age_{max} el componente es eliminado del contenedor, aunque podrá ser incluido nuevamente en una posterior iteración.

IV. CMSA para NRP

En esta sección detallamos nuestra propuesta de adaptación de CMSA para resolver el NRP. La estructura base del programa ya está expuesta en el apartado anterior y lo único que resta por explicar es el contenido de las funciones *ProbabilisticSolutionGenerator* y *ApplyExactSolver*. En primer lugar, es necesario definir el concepto de componente para resolver una instancia del NRP. De manera natural surgen dos posibilidades: que los componentes sean los requisitos o que los componentes sean los clientes. Si los componentes son los requisitos, se seleccionará aleatoriamente un subconjunto de ellos, que no viole la restricción de coste total. Si los componentes son los clientes, se seleccionan al azar también un número determinado de ellos, pero teniendo en cuenta que las demandas de todos ellos no pueden superar el coste máximo total permitido. El procedimiento *ProbabilisticSolutionGenerator* debe generar soluciones aleatorias y de gran calidad para servir de base al resolutor exacto. Para conseguir soluciones de calidad sería deseable obtener óptimos locales, para una cierta definición de vecindario. Debemos tener en cuenta que cuando se añade un requisito a la solución parcial, es necesario añadir también todos sus prerrequisitos. Explicamos a continuación como gestionar el uso de la función *ProbabilisticSolutionGenerator* para cada modalidad sobre el concepto de componente y después explicamos también la gestión del uso de prerrequisitos.

Si consideramos que un componente está formado por un requisito, estamos interesados en generar un vector binario $(x_1, \ldots, x_n)$ que sea un máximo local. Para ello, y teniendo en cuenta de que después trataremos de maximizar el beneficio de los clientes, hemos optado por ir seleccionando requisitos que vayan saturando las demandas de los clientes, ya que una selección totalmente aleatoria de requisitos podría implicar una baja satisfacción de los clientes, y por tanto, un bajo valor objetivo. Así, iremos seleccionando aleatoriamente clientes, y añadiendo todas sus demandas mientras el coste total de los requisitos y los prerrequisitos asociados a estos requisitos no sobrepasen el límite establecido. Si la selección de un cliente no satisface la condición de coste, se descarta y se pasa a otro, y este proceso se repite hasta que todos los clientes han sido analizados. En el siguiente paso, tras analizar a todos los clientes, si todavía no se ha alcanzado la cota máxima para el coste, se seleccionan de manera aleatoria más requisitos a formar parte de la solución, siempre que sea posible.

Si consideramos que un componente está formado por un cliente, estamos interesados en generar un vector binario $(y_1, \ldots, y_m)$ que sea un máximo local. Para ello procedemos de la misma forma que en el caso anterior, pero teniendo en cuenta que vamos seleccionando clientes, y no requisitos, y que no se seleccionan los requisitos finales para tratar de saturar la restricción de coste total.

Para calcular los prerrequisitos asociados a un requisito usamos inicialmente una estructura de datos consistente en un vector *Padre* $= (p_1, \ldots, p_n)$ donde $p_i = i$ si el requisito no tiene ningún prerrequisito asociado, y $p_i = j$ si el requisito i tiene como prerrequisito a j. De esta forma, resulta sencillo gestionar de manera recursiva el cálculo de todos los prerrequisitos asociados a un requisito. Sin embargo, es posible que un requisito tenga varios prerrequisitos asociados, por lo que el vector *Padre* pasa a convertirse en un vector donde cada elemento es una lista de elementos.

Algoritmo 2 *add-components(type)*

```
1:  aux = ∅; X = (0,...,0); S = (0,...,0)
2:  mientras queden clientes por seleccionar hacer
3:    i ← elegir cliente aleatoriamente y seleccionarlo
4:    pcost = 0
5:    para cualquier requisito r no incluido previamente
6:    y demandado por i hacer
7:      X[index(r)] = 1
8:      actualizar pcost ; aux = aux ∪ {index(r)}
9:      para todo prerrequisito r' de r no seleccionado
10:     previamente hacer
11:       X[index(r')] = 1
12:       actualizar pcost ; aux = aux ∪ {index(r')}
13:     fin para
14:   fin para
15:   si pcost ≤ b entonces
16:     totalcost = pcost ; S[i] = 1
17:   si no
18:     para todo r ∈ aux hacer X[r] = 0
19:   fin si
20: fin mientras
21: si type = requisitos entonces return (X)
22: si type = clientes entonces return (S)
```

En el Algoritmo 2 se muestra como se añaden los componentes a la solución parcial dependiendo de si son requisitos o clientes, y en el Algoritmo 3 se muestra la función *ProbabilisticSolutionGenerator*. El vector X es un vector binario de tamaño n que toma el valor 1 si y solo si el requisito correspondiente ha sido seleccionado. El vector *S* es un vector binario de tamaño *m* que toma el valor 1 si y solo si el cliente correspondiente es seleccionado. La variable *pcost* almacena

Algoritmo 3 *ProbabilisticSolutionGenerator(type)*

```
1: S = add-components(type)
2: si type = requisitos entonces
3:    marcar como requisito no seleccionado cualquier r tal
4:    que S[index(r)] = 0
5:    mientras queden requisitos por seleccionar hacer
6:       r ← elegir requisito aleatoriamente y seleccionarlo
7:       r' ← seleccionar prerrequisitos de r no seleccionados
8:       previamente
9:       si el coste añadido no supera el límite entonces
10:         S[r] = 1
11:         S[r*] = 1 para todo r* ∈ r'
12:         actualizar totalcost
13:      fin si
14:   fin mientras
15: fin si
16: return (S)
```

el coste parcial que será añadido, en su caso, a la variable *totalcost* que almacena el coste total acumulado y que nunca debe ser mayor que *b*. El conjunto aux guarda los índices de los requisitos y prerrequisitos seleccionados para formar parte de la solución. Si se viola la restricción de coste máximo, gracias al conjunto aux es posible restaurar el vector *X*.

A continuación explicamos la función *ApplyExactSolver*. Suponemos que disponemos de un resolutor que resuelve problemas ILP. De una manera muy sencilla podemos adaptar el contenido de esta función sin más que añadir una restricción al modelo, aquella que no tiene en cuenta los requisitos que no están en el contenedor o los clientes que no están en el contenedor, según el caso. El pseudocódigo de esta función puede verse en el Algoritmo 4.

Algoritmo 4 *ApplyExactSolver* $(C^{'})$

```
1: añadir restriccion Σ_i x_i = 0 para todo x ∉ C' al
   problema original
2: S'_opt ← obtener solución de la subinstancia
3: borrar la restricción añadida previamente
4: return (S'_opt)
```

V. Resultados computacionales

En esta sección mostramos los resultados computacionales para ciertas instancias del NRP que han sido generadas aleatoriamente. Dado que las instancias dadas por Xuan *et al*. [16] ya han sido resueltas de manera exacta en pocos segundos, no tiene sentido aplicar la heurística CMSA en este caso. Para ello, hemos creado instancias aleatorias con un número muy elevado de requisitos, clientes y dependencias, para poder así comprobar los resultados en relación a lo que proporciona la heurística CMSA en sus dos variantes (siendo los componentes requisitos o clientes) y la que proporciona el propio resolutor cuando resolvemos el problema exacto durante un tiempo prefijado.

Las instancias aleatorias han sido creadas teniendo en cuenta que no se produzcan bucles en la lectura de los prerrequisitos, lo cual se ha logrado usando estructuras *Union-Find* [5, cap. 21]. Los parámetros de las instancias generadas aleatoriamente son los siguientes: Se determinan n requisitos, m clientes y k dependencias entre requisitos. Los costes de los requisitos son números aleatorios enteros entre 1 y 10. Los pesos de los clientes son números aleatorios enteros entre 1 y 100. Cada cliente demanda un número aleatorio de requisitos que varía entre 1 y 8. Se han generado seis grupos de instancias, con los prefijos desde *a* hasta *e*. Para nombrar una instancia, se indica el nombre del grupo y los valores n, m, y k, separados por guiones entre ellos. Cada grupo representa una relación distinta entre los parámetros variables n, m y k. Así, se han considerado casos en que n puede ser mayor que m (grupos a y c), mucho mayor (grupos b y d) o similar (grupos e y f), al igual que m puede ser mayor que k (grupos a, b ó e) o similar (grupos (c, d ó f). Como límite total de coste para los requisitos se suele utilizar un coeficiente reductor sobre la suma total de los costes de todos los requisitos. En nuestro caso hemos optado por usar dos valores para este coeficiente: 0.3 y 0.7. El tiempo de ejecución máximo ha sido fijado en 60 segundos. Como resolutor hemos usado CPLEX 12.6.2.

Los experimentos han sido ejecutados bajo entorno Linux (Ubuntu 16.04 LTS) en un máquina HP con Intel Core 2 Quad (Q9400), velocidad de procesador 2.7 GHz y 4 GB de RAM, usando un máximo de 2GB de RAM y un único núcleo.

Los tres métodos usados, cuyos resultados serán comparados entre ellos son:

1. Algoritmo exacto usando el problema original y estableciendo tiempo límite de ejecución (*exact*)
2. Algoritmo CMSA utilizando los requisitos como componentes (*cmsa_r*) y parámetros $n_a = 5$, $age_{max} = 2$
3. Algoritmo CMSA utilizando los clientes como componentes (*cmsa_s*) y parámetros $n_a = 5$, $age_{max} = 2$

En el caso del algoritmo exacto se ha ejecutado 10 veces cada instancia, ya que la calidad de los resultados de CPLEX pueden variar en distintas ejecuciones debido a la diferente carga de la máquina. Las variantes de CMSA se han ejecutado 30 veces para cada instancia, un número considerablemente mayor dado el componente aleatorio de la heurística. En todos los casos se utiliza el valor promedio dado por las funciones objetivo. Los resultados para los distintos coeficientes reductores pueden verse en los Cuadros I y II. Las instancias marcadas con asterisco en la primera columna de los Cuadros I y II, son aquellas en las que el algoritmo exacto obtiene el óptimo global.

El análisis de los resultados para el coeficiente 0.3 del cuadro I muestra una variedad en cuanto a los resultados. Se ha marcado en negrita el mejor resultado para cada una de las instancias, teniendo en cuenta los valores promedio. Puede observarse que para aquellas instancias en las que el algoritmo exacto es mejor, el valor objetivo proporcionado por las heurísticas no es muy distante. Sin embargo, en aquellas instancias en las que la heurística es mejor, la diferencia en relación a la solución obtenida por el algoritmo exacto es,

Cuadro I
Valores objetivo promedio para los tres métodos propuestos, usando instancias generadas aleatoriamente durante 60 segundos y con presupuesto $b = 0.3 \sum_{i=1}^{n} c_i$.

instancia	*exact*	*cmsa_r*	*cmsa_s*
a20000-15000-4000	146018.0	**263387.1**	253287.2
a40000-30000-8000	287108.0	337520.6	**479641.2**
a80000-60000-16000	**562858.0**	506890.8	558241.5
a120000-90000-24000	**866539.0**	777432.0	852102.8
b20000-15000-15000	89272.0	103588.9	**178089.7**
b40000-30000-30000	166559.0	134943.8	**323032.7**
b80000-60000-60000	**340646.0**	276249.3	291939.1
b120000-90000-90000	360537.8	418888.4	**443369.2**
c20000-5000-4000*	**168572.0**	**168572.0**	**168572.0**
c40000-10000-8000*	**329473.0**	326309.9	325006.1
c80000-20000-16000	290985.0	301780.0	**371700.9**
c120000-30000-24000	449524.0	449683.2	**458293.3**
d20000-5000-5000*	**161697.0**	161696.7	**161697.0**
d40000-10000-10000*	**322819.0**	315514.8	313263.4
d80000-20000-20000	269804.0	281128.0	**295590.2**
d120000-30000-30000	391402.0	392500.8	**398782.4**
e20000-18000-4000	175321.0	**290948.8**	269240.6
e40000-36000-8000	355931.0	494322.9	**525250.4**
e80000-76000-16000	**770482.0**	654382.5	707105.5
e120000-108000-24000	**1078305.9**	926170.8	1004343.5
f20000-18000-18000	87259.0	107617.3	**199510.7**
f40000-36000-36000	177267.0	146402.7	**319233.9**
f80000-75000-75000	**361914.0**	295269.3	315160.4
f120000-108000-108000	-	435019.0	**459930.6**

Cuadro II
Valores objetivo promedio para los tres métodos propuestos, usando instancias generadas aleatoriamente durante 60 segundos y con presupuesto $b = 0.7 \sum_{i=1}^{n} c_i$.

instancia	*exact*	*cmsa_r*	*cmsa_s*
a20000-15000-4000	420936.0	417669.1	**438157.3**
a40000-30000-8000	856336.0	867279.3	**888971.7**
a80000-60000-16000	562858.0	557780.3	**564892.4**
a120000-90000-24000	866539.0	858235.1	**869302.4**
b20000-15000-15000	325142.0	330771.0	**343229.2**
b40000-30000-30000	650947.5	664219.9	**677113.9**
b80000-60000-60000	**1237348.0**	1164320.8	1220072.4
b120000-90000-90000	463548.6	438897.5	**733116.3**
c20000-5000-4000*	**249727.0**	**249727.0**	**249727.0**
c40000-10000-8000*	**492957.0**	**492957.0**	**492957.0**
c80000-20000-16000*	**993343.0**	**993343.0**	**993343.0**
c120000-30000-24000*	**1484724.0**	**1484724.0**	**1484724.0**
d20000-5000-5000*	**244459.0**	**244459.0**	**244459.0**
d40000-10000-10000*	**493255.0**	**493255.0**	**493255.0**
d80000-20000-20000*	**993806.0**	**993806.0**	**993806.0**
d120000-30000-30000*	**1485762.0**	**1485762.0**	**1485762.0**
e20000-18000-4000	433501.0	438125.7	**458748.1**
e40000-36000-8000	877810.0	887865.9	**924538.8**
e80000-76000-16000	770482.0	760625.6	**773035.3**
e120000-108000-24000	1077713.5	888231.8	**1082415.2**
f20000-18000-18000	87259.0	**352584.9**	319687.6
f40000-36000-36000	177267.0	704608.2	**719070.1**
f80000-75000-75000	**1383684.0**	1283686.5	1099710.2
f120000-108000-108000	-	-	**158342.4**

en ocasiones, muy elevada, como por ejemplo en la instancia *b120000-90000-90000*, donde *cmsa_s* proporciona un objetivo en torno al 67 % mejor que el algoritmo exacto. Esto justifica la utilidad del uso de estos nuevos algoritmos. También aparecen instancias en las que los resultados obtenidos por los tres métodos son iguales, como en el caso de *c20000-5000-4000*. Obsérvese también que en la instancia *f120000-108000-108000*, al tener gran cantidad de requisitos, clientes y dependencias, CPLEX es incapaz de encontrar ninguna solución durante el primer minuto de ejecución. Se conjetura que en torno a esos valores elevados de requisitos y clientes empieza a observarse un umbral a partir del cual parece ser más productivo el uso de la heurística CMSA en favor del uso de algoritmos exactos. Si comparamos *cmsa_r* con *cmsa_s*, podemos comprobar que en la mayoría de los casos *cmsa_s* proporciona unos resultados mejores, lo que nos permite deducir la importancia de elegir una buena definición de componente en el uso de la heurística CMSA, ya que los resultados para una misma instancia pueden variar considerablemente si cambiamos dicha definición.

Analizando ahora el Cuadro II para las mismas instancias de antes, pero aumentado el coeficiente reductor a 0.7, realizando el mismo número de ejecuciones que en el caso anterior, y calculando promedios, se observa como los resultados varían considerablemente. Para los grupos de instancias *c* y *d* los objetivos obtenidos son iguales para los tres casos (la solución óptima), por lo que no se detecta ninguna diferencia en el uso de los tres algoritmos. Sin embargo, para la mayoría de las restantes instancias, es claramente *cmsa_s* el método con el que se obtienen mejores resultados. Si suprimimos los grupos en el que los resultados son iguales, el método exacto solo es mejor en dos casos, mientras que *cmsa_r* solo lo es en un caso. Además, para la instancia *f120000-108000-108000*, ni el algoritmo exacto, ni *cmsa_r* consiguen encontrar una solución factible dentro del primer minuto de ejecución. Este hecho resalta nuevamente la importancia de una buena elección de componentes, que en este caso parece ser favorable a *cmsa_s*.

VI. Conclusiones y líneas futuras de investigación

Tras los resultados obtenidos con el uso de CMSA para resolver el NRP en instancias aleatorias con gran número de requisitos, clientes y dependencias, se puede conjeturar que el uso de la heurística *cmsa_s* es la más adecuada para resolver instancias grandes de NRP. Todos los experimentos se han realizado fijando los parámetros $n_a = 5$ y $age_{max} = 2$, aunque podrían haberse obtenido resultados diferentes (posiblemente mejores) para otra configuración de estos parámetros. Tras realizar el test de Wilcoxon con datos apareados a cada pareja de algoritmos y en cada una de las modalidades, los resultados de los cuadros III y IV muestran que existen diferencias significativas en cuanto al comportamiento del algoritmo *cmsa_s* respecto a los otros dos (al nivel de significancia del 5 %). Realizando nuevamente los tests pero estableciendo como hipótesis alternativa que *cmsa_s* es mejor, se obtienen los resultados que se muestran en el cuadro V. Acorde a [6]

el número de algoritmos usados para la comparación, que es tan solo de tres, no recomienda el uso del test de Friedman. En conclusión, podemos garantizar estadísticamente que, con estas instancias, el algoritmo *cmsa_s* es el mejor de los tres algoritmos analizados.

Una línea futura de investigación sobre este trabajo podría ser aplicar el paquete *iRace* [11], que permitiría obtener una configuración óptima de dichos parámetros dentro de un rango preestablecido. También se podría estudiar más a fondo el problema, con un rango más amplio de requisitos, clientes y dependencias, y estimar a partir de qué valores de estos parámetros se obtienen mejores resultados para un método u otro. Esto se podría conseguir también usando el paquete *iRace* y se matizaría en un algoritmo mediante combinación de los anteriores que sirviese como método híbrido que determina cuál es el mejor método a utilizar en función de los parámetros de entrada.

Cuadro III
TEST DE WILCOXON (DATOS APAREADOS) PARA CONTRASTAR SI HAY DIFERENCIA ENTRE LOS ALGORITMOS PROPUESTOS, $b = 0{,}3\sum_{i=1}^{n} c_i$

	exact	*cmsa_r*
cmsa_r	0.89110	-
cmsa_s	0.01629	0.00040

Cuadro IV
TEST DE WILCOXON (DATOS APAREADOS) PARA CONTRASTAR SI HAY DIFERENCIA ENTRE LOS ALGORITMOS PROPUESTOS, $b = 0{,}7\sum_{i=1}^{n} c_i$

	exact	*cmsa_r*
cmsa_r	0.97730	-
cmsa_s	0.01620	0.02449

Cuadro V
CONTRASTES UNILATERALES CON HIPÓTESIS ALTERNATIVA $H_1 : cmsa\_s > col$, SIENDO $col \in \{exact, cmsa\_r\}$

	exact	*cmsa_r*
$b = 0{,}3\sum_{i=1}^{n} c_i$	0.00814	0.00020
$b = 0{,}7\sum_{i=1}^{n} c_i$	0.00810	0.01225

También es posible generalizar el método CMSA para el caso multiobjetivo y aplicarlo inicialmente al NRP biobjetivo. De hecho, hemos trabajado en esta posibilidad, pero no hemos obtenido unos resultados iniciales satisfactorios, debido principalmente a dos problemas. El primero es que la elección de componentes como aquellos entes más válidos a formar parte de una solución, no viene a ser una tarea fácil para un problema biobjetivo, donde el espacio de soluciones es bidimensional y el comportamiento de las variables de decisión puede variar considerablemente para distintas zonas del espacio objetivo. El segundo problema encontrado es la forma de aplicar la función *ExactSolver* para el caso biobjetivo, ya que un barrido completo por el espacio objetivo en cada iteración, no solo consume demasiado tiempo de ejecución, sino que además las soluciones obtenidas suelen estar muy lejos de los valores óptimos, por lo que el prototipo de CMSA biobjetivo resulta poco eficaz. Hemos pensado en la posibilidad de subdividir el espacio objetivo en varias zonas y aplicar CMSA a cada una de ellas. Seguimos trabajando en este sentido.

VII. Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad (proyectos TIN2014-57341-R y TIN2017-88213-R) y por la Universidad de Málaga.

Referencias

[1] Philip Achimugu, Ali Selamat, Roliana Ibrahim, and Mohd Naz'ri Mahrin, *A systematic literature review of software requirements prioritization research*, Inform. and software technology **56** (2014), no. 6, 568–585.

[2] Anthony J. Bagnall, Victor J. Rayward-Smith, and Ian M Whittley, *The next release problem*, Information and software technology **43** (2001), no. 14, 883–890.

[3] Christian Blum, Pedro Pinacho, Manuel López-Ibáñez, and José A Lozano, *Construct, merge, solve & adapt a new general algorithm for combinatorial optimization*, Computers & Operations Research **68** (2016), 75–88.

[4] W. Cook and P. Seymour, *Tour merging via branch-decomposition*, INFORMS Journal on Computing **15** (2003), no. 3, 233–248.

[5] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein, *Introduction to algorithms*, MIT press, 2009.

[6] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera, *A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms*, Swarm and Evolutionary Computation **1** (2011), no. 1, 3–18.

[7] J.J. Durillo, Y. Zhang, E. Alba, and Antonio J Nebro, *A study of the multi-objective next release problem*, Search Based Software Engineering, 2009 1st International Symposium on, IEEE, 2009, pp. 49–58.

[8] Des Greer and Guenther Ruhe, *Software release planning: an evolutionary and iterative approach*, Information and software technology **46** (2004), no. 4, 243–253.

[9] Janusz Kacprzyk, *Studies in computational intelligence, volume 153*, (2008).

[10] Gunnar W Klau, Ivana Ljubić, Andreas Moser, Petra Mutzel, Philipp Neuner, Ulrich Pferschy, Günther Raidl, and René Weiskircher, *Combining a memetic algorithm with integer programming to solve the prize-collecting steiner tree problem*, Genetic and Evolutionary Computation Conference, Springer, 2004, pp. 1304–1315.

[11] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle, *The irace package: Iterated racing for automatic algorithm configuration*, Operations Research Perspectives **3** (2016), 43–58.

[12] Claudio N Meneses, Carlos AS Oliveira, and Panos M Pardalos, *Optimization techniques for string selection and comparison problems in genomics*, IEEE Engineering in Medicine and Biology Magazine **24** (2005), no. 3, 81–87.

[13] Günther Ruhe and Moshood Omolade Saliu, *The art and science of software release planning*, IEEE Software **22** (2005), 47–53.

[14] Nadarajen Veerapen, Gabriela Ochoa, Mark Harman, and Edmund K Burke, *An integer linear programming approach to the single and bi-objective next release problem*, Information and Software Technology **65** (2015), 1–13.

[15] V Venkata Rao, R Sridharan, et al., *The minimum weight rooted arborescence problem: weights on arcs case*, Tech. report, Indian Institute of Management Ahmedabad, Research and Publication Department, 1992.

[16] Jifeng Xuan, He Jiang, Zhilei Ren, and Zhongxuan Luo, *Solving the large scale next release problem with a backbone-based multilevel algorithm*, IEEE Transactions on Software Engineering **38** (2012), no. 5, 1195–1212.

[17] Yuanyuan Zhang, Mark Harman, and S Afshin Mansouri, *The multi-objective next release problem*, Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM, 2007, pp. 1129–1137.

A First Step to Accelerating Fingerprint Matching based on Deformable Minutiae Clustering*

*Note: The full contents of this paper have been published in the volume *Lecture Notes in Artificial Intelligence 11160* (LNAI 11160)

A.J. Sanchez
Dpt. of Computer Architecture
University of Málaga
Málaga, Spain
ajsanchez@ac.uma.es

L.F. Romero
Dpt. of Computer Architecture
University of Málaga
Málaga, Spain
felipe@uma.es

S. Tabik
Dpt. Computer Science and Artificial Intelligence
University of Granada
Granada, Spain
siham@ugr.es

M.A. Medina-Pérez
Tecnológico de Monterrey
México
migue@itesm.mx

F. Herrera
Dpt. Computer Science and Artificial Intelligence
University of Granada
Granada, Spain
herrera@decsai.ugr.es

***Abstract*—Fingerprint recognition is one of the most used biometric methods for authentication. The identification of a query fingerprint requires matching its minutiae against every minutiae of all the fingerprints of the database. The state-of-the-art matching algorithms are costly, from a computational point of view, and inefficient on large datasets. In this work, we include faster methods to accelerating DMC (the most accurate fingerprint matching algorithm based only on minutiae). In particular, we translate into C++ the functions of the algorithm which represent the most costly tasks of the code; we create a library with the new code and we link the library to the original C# code using a CLR Class Library project by means of a C++/CLI Wrapper. Our solution re-implements critical functions, e.g., the bit population count including a fast C++ PopCount library and the use of the squared Euclidean distance for calculating the minutiae neighborhood. The experimental results show a significant reduction of the execution time in the optimized functions of the matching algorithm. Finally, a novel approach to improve the matching algorithm, considering cache memory blocking and parallel data processing, is presented as future work.**

***Index Terms*—Fingerprint Recognition, Cache Optimization, Language Interoperability**

Running Genetic Algorithms in the Edge: A First Analysis*

*Note: The full contents of this paper have been published in the volume *Lecture Notes in Artificial Intelligence 11160* (LNAI 11160)

José Á. Morell, Enrique Alba
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
Málaga, Spain
{jamorell, eat}@lcc.uma.es

***Abstract*—Nowadays, the volume of data produced by different kinds of devices is continuously growing, making even more difficult to solve the many optimization problems that impact directly on our living quality. For instance, Cisco projected that by 2019 the volume of data will reach 507.5 zettabytes per year, and the cloud traffic will quadruple. This is not sustainable in the long term, so it is a need to move part of the intelligence from the cloud to a highly decentralized computing model. Considering this, we propose a ubiquitous intelligent system which is composed by different kinds of endpoint devices such as smartphones, tablets, routers, wearables, and any other CPU powered device. We want to use this to solve tasks useful for smart cities. In this paper, we analyze if these devices are suitable for this purpose and how we have to adapt the optimization algorithms to be efficient using heterogeneous hardware. To do this, we perform a set of experiments in which we measure the speed, memory usage, and battery consumption of these devices for a set of binary and combinatorial problems. Our conclusions reveal the strong and weak features of each device to run future algorihms in the border of the cyber-physical system.**

***Index Terms*—Edge Computing, Fog Computing, Evolutionary Algorithms, Genetic Algorithms, Metaheuristics, Smartphone, Tablet, Ubiquitous AI**