03-045

COEVOLUTIONARY ALGORITHMS FOR FACILITY LAYOUT DESIGN

Luque-Rodríguez, María[(1)]; Arauzo-Azofra, Antonio[(1)]; Gata-Moreno, Laura[(1)]; García-Hernández, Laura[(1)]; Salas-Morera, Lorenzo[(1)]

[(1)]Universidad de Córdoba

Coevolutionary Algorithms are a kind of evolutionary techniques used to solve complex problems, which can be split up. The unequal area facility layout problem (UA-FLP) is complex because it has many possible solutions whose number grows exponentially with the number of departments or zones to distribute. One widely studied layout structure for this problem is the flexible bay structure. In this structure there are two clear parts: the bay structure (how many and with how many departments in each of them) and the department order (which ones go in each bay and how are they ordered in the bay). We propose to approach this automatic design problem using these two parts to divide the problem and apply a coevolutionary cooperative algorithm to search one of the best solutions in an effective way. The tests show that the results are better with the coevolutionary algorithm than with the simple evolutionary version.

Keywords: *Facility Layout; UA-FLP; Coevolutionary algorithms*

ALGORITMOS COEVOLUTIVOS PARA EL DISEÑO DE DISTRIBUCIÓN EN PLANTA

Los algoritmos coevolutivos son una técnica de computación evolutiva que se utiliza para la resolución de problemas complejos cuando el problema o la estructura de la solución se pueden dividir en partes. El problema del diseño de distribución en planta con areas desiguales (UA-FLP) es complejo porque tiene un número de soluciones posibles que crece exponencialmente en relación al número de departamentos o zonas a distribuir. Uno de los tipos de distribución más estudiados es el de asignación por bahías flexibles. En este tipo de distribución, se pueden apreciar dos partes: la estructura de bahías (cuantas y con cuantos departamentos en cada una) y el orden de los departamentos (cuales van a cada bahía y en qué orden). Proponemos abordar el diseño utilizando estás dos partes para dividir el problema y aplicar algoritmos coevolutivos cooperativos para buscar una de las mejores soluciones de manera efectiva. Las pruebas realizadas demuestran que los resultados son mejores para el algoritmo coevolutivo frente a la versión evolutiva simple.

Palabras clave: *Distribución en planta; UA-FLP; Algoritmos coevolutivos*

Correspondencia: Antonio Araúzo Azofra arauzo@uco.es

©2018 by the authors. Licensee AEIPRO, Spain. This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (https://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introducción

El diseño de distribución en planta (Drira et al. 2007) es un problema ampliamente estudiado. Kusiak y Heragu (1987) resaltan la importancia de este diseño. Algunos de estos diseños pueden abordarse partiendo de la solución de un problema de optimización definido. Con este objetivo, se define el problema de la asignación de areas desiguales para distribución en planta UA-FLP (*Unequal Area - Facility Layout Problem*) (Armor y Buffa, 1963).

Este problema tiene un elevado número de soluciones posibles que crecen exponencialmente con el número de áreas a asignar. De esta forma, la complejidad computacional de encontrar la solución óptima es al menos de la clase no polinomial (NP). Se puede demostrar fácilmente reduciendo el problema QAP (*Quadratic Assignament Problem*) a UA-FLP y teniendo en cuenta que el problema QAP es NP-completo, según demuestran Sahni y Gonzalez (1976).

Por esta razón, UA-FLP es un problema de optimización abierto al que se siguen aplicando técnicas novedosas con el fin de mejorar las técnicas para encontrar un diseño más adecuado para plantas cada vez más grandes. Una de las técnicas más aplicadas son los algoritmos basados en metaheurísticas, como los conocidos algoritmos genéticos o los algoritmos evolutivos en general.

Los algoritmos evolutivos (AE) con una sola población a menudo tienen un rendimiento pobre cuando confrontan problemas que presentan una o más de las siguientes características (Peña-Reyes y Sipper, 2001): i) el espacio de búsqueda es complejo; ii) el problema o sus soluciones son claramente divisibles; iii) el genoma codifica diferentes tipos de valores; iv) existe una fuerte interdependencia entre los componentes de la solución. Estos algoritmos tienden a converger a óptimas locales o tienen un alto coste computacional. Sin embargo, los algoritmos coevolutivos abordan efectivamente estos problemas, ampliando el rango de aplicación de computación evolutiva.

Los algoritmos coevolutivos (CA) (Paredis, 1995) son un tipo de algoritmos evolutivos utilizados principalmente para resolver problemas complejos, que se pueden dividir. Cada parte del problema está representada por una especie diferente que evoluciona por separado (incluso con diferentes algoritmos, codificaciones y operadores). Sin embargo, todas las especies interactúan entre ellas para resolver el problema al usar una función de evaluación conjunta. Estos algoritmos descubren buenas soluciones para problemas con un espacio de búsqueda complejo.

Para descubrir una buena solución, las especies de los algoritmos coevolutivos pueden competir (por ejemplo, para obtener exclusividad en un recurso limitado) o cooperar (por ejemplo, para obtener acceso a algún recurso difícil de alcanzar). En los algoritmos competitivos (Rosin y Belew, 1996), la aptitud de un individuo se obtiene mediante la competencia con individuos de las otras especies. Mejorar la aptitud de una especie implica empeorar la aptitud de las otras especies. Cada especie busca estrategias que le permitan sobrevivir en la carrera por la mejor solución. Por otro lado, en los algoritmos cooperativos (Potter y De Jong, 2000), todas las especies cooperan para forma estructuras complejas, adecuadas para resolver un problema. La aptitud de un individuo depende de su capacidad para colaborar con individuos de las otras especies.

Los algoritmos coevolutivos cooperativos (COCA) han sido utilizado en diferentes campos para mejorar el rendimiento alcanzado por algoritmos previos. En el campo de la optimización, Liu et al. (2013) descomponen el espacio de búsqueda de las variables de decisión y usan un COCA para evolucionar mejores soluciones; Wang et al. (2013) coevoluciona una familia de preferencias de gestión junto con una población de soluciones

candidatas para optimización multi-objetivo; Deb et al. (2013) los aplica para resolver el problema dual de Lagrange asociado con problemas de optimización. Por otra parte, hay una conexión entre los juegos y coevolución (Samothrakis et al., 2013). Ballinger y Louis (2013a) y Ballinger y Louis (2013b) diseñan algoritmos coevolutivos para encontrar estrategias en juegos de tiempo real; Szubert et al. (2013) y Liskowski (2013) presentan sus algoritmos como un método para aprender a jugar el juego de Othello. Otros campos de aplicaciones son la biología (Komlen y Jakobovic, 2013), aeronáutica (Wang y Dong, 2013), programación (Su et al., 2013) o visión (Chaaraoui y Flórez-Revuelta, 2013; Rabil et al., 2013).

Los algoritmos coevolutivos también han sido aplicados al diseño de distribución en planta. Por ejemplo, Dunker et al. (2003) proponen un coevolutivo que trabaja con áreas de dimensiones dadas. Chang et al. (2002) abordan la versión dinámica de QAP. Furuholmen et al. (2010) utilizan los coevolutivos para el diseño de distribuciones en planta en tres dimensiones. Sin embargo, en nuestro mejor conocimiento, no se ha probado su aplicación en UA-FLP y eso es lo que proponemos en la presente contribución.

2. Problema de optimización

El problema de distribución en planta con áreas desiguales (UA-FLP) fue formulado por inicialmente por Armour y Buffa (1963). Aborda la distribución de departamentos (*d*) con diferentes áreas requeridas (A_d) en una planta rectangular con dimensiones fijas (*W* x *H*). Como los departamentos no deben superponerse, la suma de las áreas de departamento debe ser menor o igual que el área de la planta:

$$\sum_{d=1}^{n} A_d \leq W \times H$$

Como objetivo a minimizar se puede considerar el coste del flujo de materiales entre los diferentes departamentos:

$$\sum_{i \neq j}^{n} F_{i,j} \cdot D_{i,j}$$

siendo $F_{i,j}$ el coste del flujo entre los departamentos *i* y *j* por unidad de distancia y $D_{i,j}$ la distancia entre ambos departamentos en la distribución en planta a considerar. Según la instancia concreta del problema a abordar, se utilizan diferentes funciones de distancia y formas de medir el coste. También pueden utilizarse otro tipo de funciones objetivo como las que buscan optimizar la evaluación de la TRA (Tabla Relacional de Actividades).

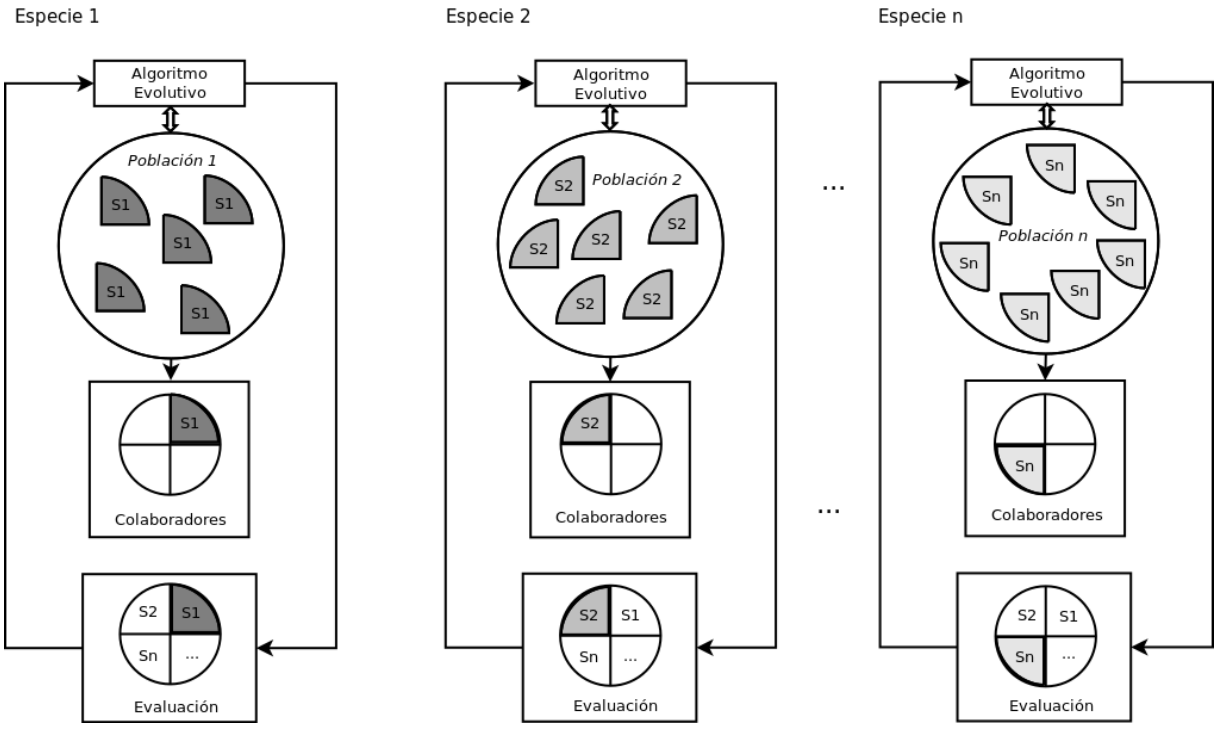
Trabajamos con departamentos rectangulares que tendrán un lado mayor L_d y un lado menor M_d. Se considera también una restricción de aspecto ($1 \leq R_d \leq +\infty$) para cada departamento de forma que no se diseñen departamentos demasiado estirados que den lugar a áreas de trabajo impracticables.

$$\forall d \in D : \frac{L_d}{M_d} \leq R_d$$

3. Algoritmos coevolutivos cooperativos

Los COCA se usan para resolver problemas que pueden descomponerse, el problema mismo o sus soluciones (Peña-Reyes y Sipper, 2001), en diferentes especies que cooperan para descubrir buenas soluciones. Cada especie evoluciona por separado utilizando su propio algoritmo, pero interactúa con los demás a través de una aptitud conjunta (Figura 1)

Figura 1: Cooperación coevolutiva



Para diseñar un algoritmo coevolutivo, es necesario especificar los algoritmos que cada especie utiliza para evolucionar (tipo, codificación, operadores) y, por otro lado, varios aspectos relacionados con la materia de la cooperación.

3.1 Algoritmo evolutivo individual

Cada especie puede usar un algoritmo independiente para evolucionar. Esto se puede elegir entre una amplia gama de algoritmos evolutivos, como por ejemplo:

- Algoritmos genéticos
- Programación genética
- Estrategias evolutivas
- Algoritmos meméticos

3.2 Interacción entre las especies

Se debe decidir la forma de elegir a los colaboradores de cada especie y cómo interactuarán entre ellos para evaluar a toda la población. Algunas características básicas de los algoritmos coevolutivos (Wiegand, 2003):

- **Número de cooperadores**. La cantidad de colaboradores de cada población que se usará para la evaluación de aptitud. Computacionalmente, la opción más obvia y más costosa es que un individuo de una población interactúe con todos los individuos de las otras poblaciones. En el otro extremo, la aptitud de un individuo solo depende de su interacción con un único individuo de cada especie. Entre ambos, hay una gran variedad de posibilidades, dependiendo del número de colaboradores.
- **Presión de selección**. El grado de parcialidad al elegir un colaborador en función de su evaluación. El conjunto de colaboradores de cada especie se puede elegir de varias

maneras. Cuando todos los colaboradores se eligen entre los mejores individuos de la generación anterior, tenemos una fuerte presión de selección. Sin embargo, si el conjunto de colaboradores se elige al azar, la presión de selección disminuye. Por supuesto, hay otras posibilidades que modifican la presión de selección. Los colaboradores se eligen usando varias combinaciones de tres mecanismos de selección: mejor, aleatorio y peor.

- **Asignación de crédito**. El método de evaluar aptitud a un individuo, cuando depende de múltiples interacciones con los colaboradores. Existen diferentes métodos para obtener un solo valor de aptitud cuando se llevan a cabo múltiples interacciones. Los métodos más comunes son:
    - Optimista: Este es el método más tradicional. Asigna, como crédito de aptitud individual, el valor obtenido con el mejor colaborador.
    - Promedio: El valor promedio de todas las colaboraciones.
    - Pesimista: El valor con el peor colaborador.

3.3 Tiempo de actualización

Otro aspecto importante en un algoritmo coevolutivo es la frecuencia de interacción entre especies. Vendrá dada por el tamaño de bloque, cada cuánto se actualizan las especies, y por cómo se actualizan las especies durante el proceso de coevolución. Podemos identificar dos tipos principales (Wiegand, 2003):

- **Secuencial**. Cada población se procesa en un orden secuencial, eligiendo al colaborador del estado actual de la otra población. Las poblaciones procesadas y actualizadas anteriormente pueden afectar el procesamiento de poblaciones procesadas posteriormente.
- **Paralelo**. Todas las poblaciones se procesan en paralelo. El conjunto de colaboradores se selecciona de la generación anterior de cada población.

Se puede consultar una descripción más detallada de las diferencias entre ambos enfoques en (Jansen and Wiegand, 2003).

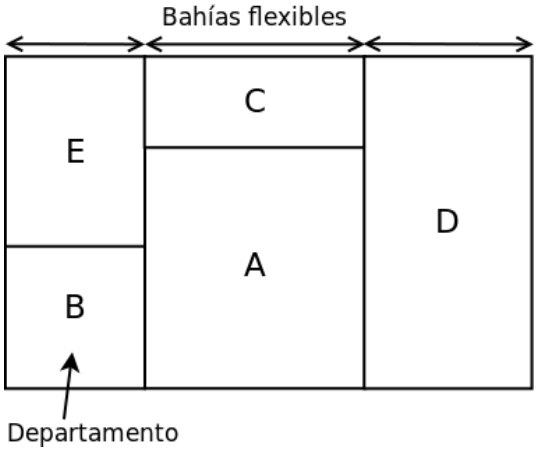
4. Propuesta coevolutiva para UA-FLP

El objetivo de esta contribución es el diseño de un COCA que nos ayude a encontrar soluciones adecuadas a los problemas de distribución en planta. Para ello utilizaremos la ampliamente conocida estructura de bahías flexibles, ya que este tipo de estructura proporciona buenos resultados para gran parte de los problemas UA-FLP.

La estructura de bahías flexibles utiliza una serie de columnas (denominadas bahías) para colocar los departamentos en orden. En esta estructura, el ancho de las columnas se adapta de forma flexible al área requerida por los departamentos que hay en ella. Esto se ilustra en la Figura 2.

A continuación, primero, definimos la división de nuestro problema en dos partes y establecemos los valores para algunas características relacionadas con la cooperación. Luego, los algoritmos evolutivos, que cada especie utiliza para evolucionar, se describen en detalle.

Figura 2: Estructura de bahías flexibles



4.1 Especies involucradas

El primer paso para aplicar COCA es dividir el problema de FL en varias partes que sean capaces de interactuar entre ellas. Hemos decidido dividir la solución del problema en dos partes independientes, por lo que nuestra COCA trabajará con dos especies:

- Una especie representa el orden en que se establecerán los departamentos, que se enumeran de izquierda a derecha y de abajo hacia arriba. Para la solución, que se muestra en la Figura 2, un individuo de esta especie almacenará la siguiente lista de departamentos: B, E, A, C, D.
- La otra especie define los puntos de corte que delimitan las bahías y la orientación de las mismas. Para el mismo ejemplo, un individuo almacenará que hay tres puntos de corte (entre los departamentos E y A, entre A y C, y entre C y D). Así como la orientación de las bahías, en este caso, vertical.

4.2 Características comunes

Ambas especies de nuestra propuesta usan el mismo tipo de algoritmo para evolucionar, un Algoritmo Genético clásico (GA) (Mitchell, 1998). El tipo de algoritmo no es la única característica común. Ambas especies comparten la función de aptitud y el operador de selección.

A continuación, definimos estas características comunes, mientras que en las dos subsecciones siguientes, las características específicas para cada especie se definen en detalle.

La función de evaluación a minimizar es la siguiente:

$$Evaluación = \sum_{i \neq j}^{n} F_{i,j} \cdot D_{i,j} + Penalización$$

donde se añade al coste indicado en la sección 2 la *Penalización*, que es el factor de penalización de Tate and Smith (1995).

El operador de selección usado en el GA es el de torneo binario (Miller, 1995). Se seleccionan aleatoriamente dos individuos de la población actual y el mejor de ellos se incorpora a la nueva población. Así hasta llenar ésta. Hemos elegido un sólo individuo como tamaño de torneo porque es el más utilizado y, además, los tamaños de torneo más grandes pueden provocar pérdida de diversidad (Blickle and Thiele, 1995; Whitley, 1989).

En la sección 3.2, se describen varios aspectos relacionados con la interacción de las especies. Nuestra propuesta utiliza actualización en paralelo y los aspectos son definidos por parámetros cuyos valores se fijarán en la experimentación.

4.3 Algoritmo evolutivo para la especie 1: orden de los departamentos

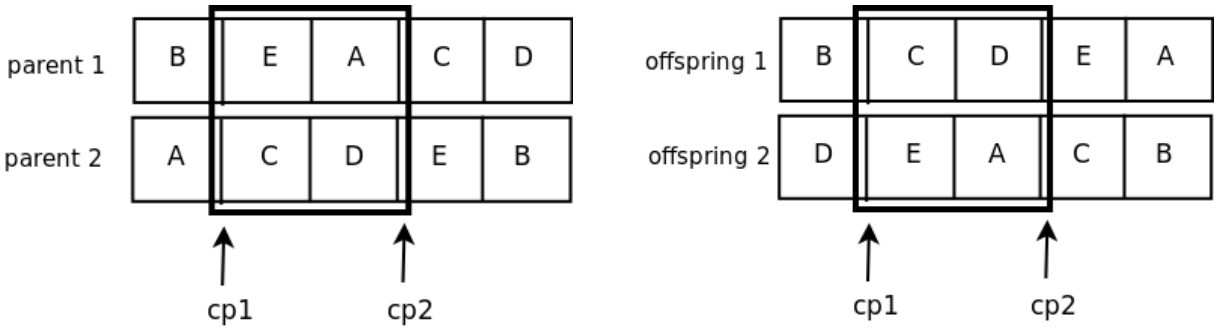
Cada cromosoma es un vector entero de tamaño igual al número de departamentos en el problema (*n*). El vector codifica el orden en que se configuran los departamentos, enumerados de izquierda a derecha y de abajo hacia arriba. Internamente, hay una relación entre el nombre de los departamentos y los enteros en el rango {0.. (n-1)}.

La primera población se inicializa aleatoriamente, verificando que no haya departamentos duplicados y que se generen en el rango correcto, es decir, el departamento existe en el problema.

Debido a la naturaleza del cromosoma, el operador de cruce elegido debe garantizar la ausencia de repeticiones de los departamentos. El resultado del cruce debe ser una permutación de la numeración de los departamentos. Por este motivo, hemos elegido el cruce PMX (Sivanandam, 2007).

En este operador, dos puntos de cruce se seleccionan al azar a lo largo de la longitud de los cromosomas. Ambos puntos proporcionan una selección coincidente, que se usa para afectar una operación de intercambio de posición por posición.

Figura 3: Operador de cruce PMX



La parte izquierda de la figura 3 muestra los dos padres a cruzar (cada uno de ellos es una permutación de los departamentos) y los puntos de cruce seleccionados al azar. Los genes entre ambos puntos se intercambian, es decir, el departamento E y el departamento C, el departamento A y el departamento D intercambian lugares. Después del PMX, la descendencia generada se muestra en la parte derecha de la figura.

La descendencia resultante del operador de mutación debe ser un cromosoma válido, una permutación de los departamentos. Por lo tanto, el operador de la mutación debe garantizar la ausencia de duplicados y valores fuera de rango. Para ello, hemos elegido una mutación basada en orden (Syswerda, 1991). Se seleccionan dos puntos aleatorios en el genotipo y se intercambian los departamentos en ellos.

4.4 Algoritmo evolutivo para la especie 2: Estructura de las bahías

Los individuos que definen los puntos de corte bay están codificados como un vector binario de *n* valores. Los primeros *n-1* bits se usan para definir puntos de corte, mientras que el último bit representa la orientación del diseño. En la primera parte del cromosoma, 0 significa que no hay una división entre departamentos, y ambos departamentos están ubicados en la misma bahía, y 1 significa que se crea una nueva bahía. En la segunda

parte, si aparece 1, las bahías se llenan en el sentido horizontal, mientras que 0 indica una orientación vertical.

Cada individuo de esta población se genera al azar, con valores binarios para cada gen.

Para esta población, no es necesario imponer ninguna restricción, por lo que el operador de cruce puede ser más simple. Concretamente, hemos utilizado el operador de cruce clásico propuesto por Holland (1975). Funciona como sigue. El corte de un punto se selecciona al azar, donde los cromosomas se dividen en partes separadas y luego las diferentes partes se recombinan. La mutación ha sido también el operador clásico flip binario (cambiar un valor de 0 a 1 o viceversa).

4.5 Múltiples poblaciones de cada especie

Las GA tradicionales evolucionan una sola población, que busca en todo el espacio de búsqueda. Esta búsqueda se puede mejorar usando más de una población (población múltiple). Cada población exploraría un área de búsqueda diferente, explotaría una zona prometedora o ambas tareas. Por lo tanto, en GA de poblaciones múltiples (Tsutsui, 1997), una población busca continuamente un nuevo óptimo, mientras que el resto de las poblaciones intenta explotar las otras áreas prometedoras previamente detectadas.

Normalmente, los COCA usan una sola población para cada especie que evoluciona por separado. Sin embargo, el concepto de población múltiple se puede incorporar a cada especie con el fin de mejorar la búsqueda de buenas soluciones en cada parte del problema. Nuestra propuesta utiliza más de una subpoblación para encontrar soluciones adecuadas para cada especie. Estas subpoblaciones evolucionan independientemente, pero utilizando el mismo algoritmo y proporcionando el mismo número de colaboradores al grupo. En la fase de evaluación, cada individuo se evalúa utilizando todos los colaboradores elegidos.

La idea es que cuanto más amplio sea el espacio de búsqueda, será necesario un mayor número de poblaciones para poder explorar todas las regiones. En nuestro problema, el espacio de búsqueda por orden de departamento (especie 1) es más amplio (*n! = factorial del número de departamentos*) que el espacio de cortes (especie 2), que es de tamaño 2^n, por lo que el número de poblaciones para la especie 1 será mayor que para la especie 2.

5. Estudio empírico

Los problemas utilizados en nuestra experimentación son públicos y han sido utilizados en diversos artículos. Entre estos, hemos utilizado los seleccionados por Komarudin y Wong (2010) junto con tres ejemplos con descripción real usados por García-Hernández et. al. (2015).

Realizamos la comparación del uso del algoritmo coevolutivo cooperativo propuesto con un algoritmo genético no coevolutivo con el mismo número de evaluaciones (para igualar el coste computacional de su ejecución). Para ello, al elegir los parámetros el número de generaciones se ha ajustado automáticamente para realizar un total de cien mil evaluaciones, según el tamaño de las poblaciones utilizadas en cada lanzamiento.

Se han probado diferentes valores de los parámetros de ambos algoritmos hasta obtener aquellos que dan mejores resultados. Los valores probados para el algoritmo genético no coevolutivo se muestran en la Tabla 1. Los valores que aparecen en negrita son los que mejores resultados han obtenido.

Tabla 1. Parametros del algoritmo genético no coevolutivo (valores elegidos en negrita)

Parámetros	Valores
Generaciones	100 … 2000
Tamaño de población	50, 100, 200, 500, **1000**
Probabilidad de cruce	0.3, 0.5, **0.7**, 0.9
Probabilidad de mutación	0.1, 0.2, **0.4**, 0.6

En la Tabla 2 se muestran el ajuste de parámetros realizado para el algoritmo coevolutivo. En este caso, dada la imposibilidad de probar todas las combinaciones, se han ido fijando en bloques, probando primero aquellos que por intuición creíamos que podían afectar más a los siguientes. Así, el orden seguido ha sido: probabilidades de cruce y mutación, tipo de agregación, tamaño de bloque, número de cooperadores, numero de poblaciones y sus tamaños.

Tabla 2. Parámetros evaluados del algoritmo coevolutivo (valores elegidos en negrita)

Parámetros	Valores
Generaciones	{166 … 2000}
Número de poblaciones de orden	1,2,**3**
Tamaño de poblaciones de orden	50, 75, 100, **150**, 200
Número de poblaciones de cortes	**1**,2,3
Tamaño de poblaciones de cortes	**50**, 75, 100, 150, 200
Probabilidad de cruce en poblaciones de orden	0.3, **0.5**, 0.7, 0.9
Probabilidad de mutación en poblaciones de orden	0.1, 0.2, **0.4**, 0.6
Probabilidad de cruce en poblaciones de cortes	0.3, 0.5, **0.7**, 0.9
Probabilidad de mutación en poblaciones de cortes	0.1, 0.2, **0.4**, 0.6
Número de cooperadores	1, **2**, 3, 4, 5, 6, 7
Asignación de crédito	mejor, **promedio**, peor
Tamaño de bloque	1, 2, 5, 10, **20**, 50

En la Tabla 3 se muestran los resultado obtenidos. Para evitar efectos aleatorios no deseados, se han repetido las ejecuciones 10 veces con una semilla diferente en el generador de números aleatorios. El resultado que se muestra es la media del mejor individuo encontrado en cada una de estas 10 ejecuciones.

Tabla 3. Comparativa del valor promedio del valor mínimo del coste del flujo de materiales obtenido en 10 ejecuciones de los algoritmos

Problema	Coevolutivo	No coevolutivo
Slaughterhouse	3357	5564
CartonPacks	48	75
ChoppedPlastic	23	33
O7	134	140
O8	245	261
O9	241	279
VC10	18818	27265
Ba12	8124	13052
Ba14	4732	10534
AB20	5363	-
SC30	3667	-
SC35	4036	-

6. Conclusiones

En esta contribución se ha presentado un nuevo algoritmo coevolutivo cooperativo basado en múltiples poblaciones para la resolución de UA-FLP. Las ventajas que aporta este algoritmo son su gran adaptabilidad al problema, centrando el esfuerzo de búsqueda en la parte del problema donde más se necesite (orden o cortes de bahías) y la posibilidad de su ejecución distribuida.

Los resultados obtenidos demuestran que el algoritmo coevolutivo propuesto supera las propuestas no coevolutivas y alcanza resultados similares a los de otras propuestas que no pueden ejecutarse de forma distribuida.

Creemos que esta propuesta abre el campo para futuros trabajos aplicando algoritmos coevolutivos a los muy diversos tipos de problemas que se definen en el diseño de distribución en planta.

Bibliografía

Armour, G.C., Buffa, E.S., 1963. A heuristic algorithm and simulation approach to relative location of facilities. Management Science 9, 294–309.

Ballinger, C., Louis, S., 2013a. Comparing coevolution, genetic algorithms, and hill-climbers for finding real-time strategy game plans, in: GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference Companion, pp. 47–48.

Ballinger, C., Louis, S., 2013b. Robustness of coevolved strategies in a real-time strategy game, in: 2013 IEEE Congress on Evolutionary Computation, CEC 2013, pp. 1379–1386.

Blickle, T., Thiele, L., 1995. A comparison of selection schemes used in genetic algorithms.

Chaaraoui, A.A., Flórez-Revuelta, F., 2013. Optimizing human action recognition based on a cooperative coevolutionary algorithm. Engineering Applications of Artificial Intelligence Article in Press.

Chang, M., Ohkura, K., Ueda, K., Sugiyama, M., 2002. A symbiotic evolutionary algorithm for dynamic facility layout problem, in: Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on, IEEE. pp. 1745–1750.

Chen, Z., Wang, L., 2008. Fuzzy association-rule extraction based on organizational coevolutionary. Journal of Computational Information Systems 4, 1501–1506.

Deb, K., Gupta, S., Dutta, J., Ranjan, B., 2013. Solving dual problems using a coevolutionary optimization algorithm. Journal of Global Optimization 57, 891–933.

Drira, A., Pierreval, H., Hajri-Gabouj, S., 2007. Facility layout problems: A survey. Annual Reviews in Control 31, 255–267.

Dunker, T., Radons, G., Westkämper, E., 2003. A coevolutionary algorithm for a facility layout problem. International Journal of Production Research 41, 3479–3500.

Furuholmen, M., Glette, K., Hovin, M., Torresen, J., 2010. A coevolutionary, hyper heuristic approach to the optimization of three-dimensional process plant layouts—a comparative study, in: Evolutionary Computation (CEC), 2010 IEEE Congress on, IEEE. pp. 1–8.

García-Hernández, L., Araúzo-Azofra, A., Pierreval, H., Salas-Morera, L., 2009. Encoding structures and operators used in facility layout problems with genetic algorithms, in: ISDA '09: Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, IEEE Computer Society, Washington, DC, USA. pp. 43–48.

Holland, J., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, USA.

Jansen, T., Wiegand, R.P., 2003. Sequential versus parallel cooperative coevolutionary (1+ 1) eas, in: Evolutionary Computation, 2003. CEC'03. The 2003 Congress on, IEEE. pp. 30–37.

Wong, K. Y. (2010). Solving facility layout problems using Flexible Bay Structure representation and Ant System algorithm. *Expert Systems with Applications*, *37* (7), 5523-5527.

Komlen, D., Jakobovic, D., 2013. Investigation of coevolutionary approach in gene regulatory network inference, in: 2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2013 - Proceedings, pp. 981–987.

Kusiak, A., Heragu, S.S., 1987. The facility layout problem. European Journal of Operational Research 29, 229–251.

Liskowski, P., 2013. Quantitative analysis of the hall of fame coevolutionary archives, in: GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference Companion, pp. 1683–1686.

Liu, R., Chen, Y., Ma, W., Mu, C., Jiao, L., 2013. A novel cooperative coevolutionary dynamic multi-objective optimization algorithm using a new predictive model. Soft Computing, 1–17Article in Press.

Meller, R., Gau, K., 1996. The facility layout problem: Recent and emerging trends and perspectives. Journal of Manufacturing Systems 15, 351–366.

Miller, B.L., Miller, B.L., Goldberg, D.E., Goldberg, D.E., 1995. Genetic algorithms, tournament selection, and the effects of noise. Complex Systems 9, 193–212.

Mitchell, M., 1998. An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA, USA.

Myszkowski, P.B., 2011. Rule induction based-on coevolutionary algorithms for image annotation. volume 6592 LNAI PART 2 of Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Cited By (since 1996):1.

Noraini, M.R., Geraghty, J., 2011. Genetic algorithm performance with different selection strategies in solving tsp, in: Proceedings of the World Congress on Engineering.

Paredis, J., 1995. Coevolutionary computation. Artificial life 2, 355–375.

Peña-Reyes, C.A., Sipper, M., 2001. Fuzzy coco: A cooperative-coevolutionary approach to fuzzy modeling. IEEE Transactions on Fuzzy Systems 9, 727–737. Cited By (since 1996):69.

Popovici, E., De Jong, K., 2006. The effects of interaction frequency on the optimization performance of cooperative coevolution, in: Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM. pp. 353–360.

Potter, M.A., De Jong, K.A., 2000. Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evolutionary computation 8, 1–29.

Rabil, B.S., Tliba, S., Granger, E., Sabourin, R., 2013. Securing high resolution grayscale facial captures using a blockwise coevolutionary ga. Expert Systems with Applications 40, 6693–6706. Cited By (since 1996):1.

Rosin, C., Belew, R., 1996. New methods for competitive coevolution. Evolutionary Computation 5, 1–29.

Sahni, S., Gonzalez, T., 1976. P-complete approximation problems. Journal of the ACM (JACM) 23, 555–565.

Samothrakis, S., Lucas, S., Runarsson, T.P., Robles, D., 2013. Coevolving game-playing agents: Measuring performance and intransitivities. IEEE Transactions on Evolutionary Computation 17, 213–226. Cited By (since 1996):1.

Sivanandam, S.N., Deepa, S.N., 2007. Introduction to Genetic Algorithms. 1st ed., Springer Publishing Company, Incorporated.

Su, S., Yu, H., Wu, Z., Tian, W., 2013. A distributed coevolutionary algorithm for multiobjective hybrid flowshop scheduling problems. International Journal of Advanced Manufacturing Technology , 1–18Article in Press.

Syswerda, G., 1991. Schedule optimization using genetic algorithms, in: Davis, L. (Ed.), Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, NY.

Szubert, M., Jaskowski, W., Krawiec, K., 2013. On scalability, generalization, and hybridization of coevolutionary learning: A case study for othello. IEEE Transactions on Computational Intelligence and AI in Games 5, 214–226.

Tan, K.C., Yu, Q., Ang, J.H., 2006. A coevolutionary algorithm for rules discovery in data mining. International Journal of Systems Science 37, 835–864. Cited By (since 1996):18.

Wang, F.B., Dong, C.H., 2013. Coevolutionary algorithm applied to skip reentry trajectory optimization design. volume 427-429 of Applied Mechanics and Materials.

Wang, R., Purshouse, R.C., Fleming, P.J., 2013. Preference-inspired coevolutionary algorithms for many-objective optimization. IEEE Transactions on Evolutionary Computation 17, 474–494.

Whitley, D., 1989. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best, in: Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann. pp. 116–121.

Wiegand, R.P., 2003. An analysis of cooperative coevolutionary algorithms. Ph.D. thesis. Citeseer.

Wiegand, R.P., Liles, W.C., De Jong, K.A., 2001. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 1235–1245.

Agradecimientos

Este trabajo ha sido financiado por un proyecto del Plan Propio de Investigación (2017) de la Universidad de Córdoba.