6th International Conference on Industrial Engineering and Industrial Management.
XVI Congreso de Ingeniería de Organización. Vigo, July 18-20, 2012

Un algoritmo genético híbrido para el equilibrado de líneas de montaje con número fijo de estaciones

A hybrid genetic algorithm for the assembly line balancing problem with fixed number of workstations

Guerra JM[1], Pereira J[1], Vilà M[1]

Abstract (English) Assembly line balancing is a classical problem in industrial engineering. The objective is to assign the tasks in which the production has been divided to the workers of the assembly line, maximizing the total efficiency. The paper presents a new hybrid genetic algorithm to solve the version of the problem where the number of workstations is fixed. Preliminary results show that our proposal outperforms the best-known procedures found in the literature.

Resumen El equilibrado de líneas de montaje es un problema clásico de Ingeniería en Organización Industrial. El problema consiste en asignar las tareas en las que se divide el montaje de un producto a los operarios que las realizan, con el objetivo de maximizar la eficiencia de la línea. El presente trabajo muestra un nuevo algoritmo genético híbrido para resolver la versión del problema con un número fijo de estaciones. Los resultados preliminares muestran que el algoritmo mejora los procedimientos considerados como el estado del arte en la literatura.

Keywords: Assembly Lines, Manufacturing, Balancing, Dynamic Programming, Genetic Algorithms; **Palabras clave:**Líneas de montaje, producción, equilibrado, Programación Dinámica, Algoritmos Geneticos.

[1]José Miguel Guerra, Jordi Pereira (✉), Mariona Vilà
Escola Universitària d'Enginyeria Tècnica Industrial de Barcelona (EUETIB). Barcelona Tech, C/ Urgell, 187, 08036 Barcelona, Spain
e-mail: jose.miguel.guerra@upc.edu, jorge.pereira@upc.edu, mariona.vila.bonilla@upc.edu

1 Introducción

Equilibrar una línea de montaje consiste, simplificadamente, en asignar las tareas en que se divide el montaje de un producto entre las diferentes estaciones que componen la línea en que se ensambla. Dichas estaciones se consideran dispuestas en serie y el producto pasa de una estación a la siguiente cuando el tiempo concedido para elaborar las tareas asignadas a cada estación finaliza. Este tiempo, c, es idéntico para cada una de las estaciones y se conoce como tiempo de ciclo, siendo su inversa la tasa de producción de la línea de montaje.

Formalmente, un ejemplar del problema de equilibrado de líneas de montaje SALBP-2 queda definido por un conjunto de tareas V, en que se ha subdividido el montaje del producto. Cada tarea debe asignarse a una de las S_j, (j=1,…,m), estaciones disponibles. Las tareas tiene una duración conocida, d_i, (i=1,…,|V|). Adicionalmente, algunas tareas presentan relaciones de precedencia entre ellas, por ejemplo es necesario montar los asientos de un coche antes de instalar las puertas, que pueden representarse mediante un grafo acíclico G(V,A) donde existe un arco entre la tarea i, (i=1,…,|V|), y la tarea k, (k=1,…,|V|), si la tarea i debe realizarse con anterioridad a la tarea k, esto es si $i \in S_j$ y $k \in S_l$, se debe cumplir $j \leq l$. El objetivo consiste en encontrar una asignación de tareas a las estaciones tal que se cumplan con las relaciones de precedencia, use exactamente m estaciones, y que minimice el tiempo de ciclo $c=\text{MAX}_{\forall Sj} \{\Sigma_{\forall i \in Sj} d_i\}$.

La literatura incluye diversas propuestas para resolver el problema de equilibrado de líneas de montaje simple, véase el estado del arte de Scholl y Becker (2006). Los intentos de resolución del caso tratado pueden dividirse en dos tipologías principales según el enfoque adoptado: (1) Resolución iterativa del SALBP-2 a través de su relación con el SALBP-F, problema de factibilidad donde el tiempo de ciclo y el número de estaciones es fijo, o el SALBP-1, donde el tiempo de ciclo es fijo y se pretende minimizar el número de estaciones, Blum (2011), Goncalves y Almeida (2002); o (2) resolución directa de la instancia SALBP-2, Scholl y Voss (1997) o Ugurdag et al. (1997).

El primer enfoque es el que ha ofrecido mejores resultados, Blum (2011). El método parte de una solución factible, y busca heurística o exhaustivamente una solución factible a un problema con el mismo número de estaciones y menor tiempo de ciclo.

El presente trabajo expone un algoritmo genético híbrido, Talbi (2002), para resolver el problema. El algoritmo genético usa una representación indirecta de la solución a través de una lista de prioridad, parecida a la propuesta de Goncalves y Almeida (2002). La lista de prioridad se utilizará en el evaluador del algoritmo genético intentando mejorar la solución a través de un procedimiento de programación dinámica, Bautista y Pereira (2009).

El esquema del resto del trabajo es el siguiente: en la sección 2 se expone el método basado en programación dinámica y su adaptación para la resolución del problema SALBP-F, la sección 3 muestra el algoritmo genético utilizado, la sec-

ción 4 documenta los resultados de una experiencia computacional con las instancias de la literatura para las que no se conoce la solución óptima. Los resultados de la experiencia computacional muestran que el algoritmo es capaz de mejorar 6 mejores soluciones de las 14 instancias abiertas.

2 Programación Dinámica para el problema SALBP-F

La Programación Dinámica es un proceso de decisión polietápico basado en la resolución secuencial de suproblemas hasta que el problema total queda resuelto. En cada etapa, se define un conjunto de estados que describen las condiciones posibles del proceso de decisión en esa etapa. Los estados de una etapa m se transforman en estados de la etapa m+1 mediante una transición, que representa la decisión tomada para pasar de un estado a otro. Los estados y transiciones constituyen un grafo dirigido que puede ser utilizado en la búsqueda de una solución.

Jackson (1956) propone un procedimiento para resolver el problema basado en este enfoque. En su propuesta, cada etapa del grafo corresponde al número de estaciones asignadas, existiendo una etapa 0 constituida por un único estado en que aún no se ha asignado ninguna tarea, y las transiciones se asocian a la construcción de una nueva estación partiendo de la solución parcial representada por el estado original. La transición debe cumplir con las restricciones de precedencia y de tiempo disponible. En caso que exista un camino entre el estado con ninguna tarea asignada y un estado con todas las tareas asignadas con m transiciones, el problema SALBP-F es factible, no siéndolo en caso contrario.

Una aproximación directa al planteamiento anterior obliga a obtener una descripción completa de todos los estados que forman parte del grafo, y ésta puede contener un número prohibitivo de estados y transiciones. Es por ello que debe recurrirse a una metodología que reduzca el número de estados considerados. La descripción del algoritmo de programación dinámica implementado se divide en dos partes, la primera describe cómo obtener los estados, mientras que la segunda expone el método de programación dinámica acotada BDP, Bautista et al. (1996), que dará solución al problema anteriormente indicado.

2.1 Generación de estados

El procedimiento de enumeración de estados se basa en la heurística de Hoffmann (1963). El procedimiento original parte de una solución parcial y genera todas las asignaciones posibles para construir la estación siguiente, devolviendo únicamente la más prometedora, aquella con menor tiempo muerto en la estación construida. Para poder utilizar el procedimiento en la generación exhaustiva de estados, y me-

jorar su rendimiento computacional, se han realizado las siguientes modificaciones:

- Mantenimiento de varias asignaciones en paralelo. En caso de querer construir el programa dinámico completo sería necesario generar todas las transiciones, y por lo tanto en vez de guardar únicamente la mejor asignación, el algoritmo debería mantener todas las asignaciones posibles. En la implementación se limita el número de transiciones guardadas a un máximo fijado como un parámetro, denominado *transiciones*, descartando aquéllas con mayor tiempo muerto.
- Reordenación de tareas. Se sabe que las asignaciones propuestas por el procedimiento dependen del orden de las tareas en la instancia original, por tanto puede ser interesante usar órdenes diferentes al original de la instancia. El orden propuesto por Bautista y Pereira (2009), intenta minimizar el tiempo de ejecución y ofrece buenos resultados, aunque otras ordenaciones puedan producir soluciones de mayor calidad. En el presente trabajo el orden utilizado viene determinado por el orden codificado en los individuos de la población del algoritmo genético.
- Eliminación de simetrías forzando que las tareas se escojan en orden no decreciente al orden de las tareas.
- Descarte de asignaciones debido al dominio de las variables. Al conocer el tiempo de ciclo y el número de estaciones, es posible determinar el dominio de las variables, estación mínima y máxima a la que puede asignarse una tarea en la solución, Scholl (1999). En caso que la asignación construida incumpla el dominio de alguna tarea, la asignación no se almacenará.

2.2 Programación Dinámica Acotada

La Programación Dinámica Acotada, Bautista et al. (1996), es un procedimiento derivado de los conceptos de la programación dinámica. El procedimiento usa un enfoque basado en dos listas como el observado en Jackson (1956). La primera lista corresponde a una etapa ya generada del grafo y la segunda a la etapa siguiente que se encuentra en construcción. Cada etapa de la primera lista se utiliza para generar sus estados sucesores, mediante el procedimiento expuesto en el apartado 2.1, guardando los estados generados en la segunda lista. Cuando se han desarrollado todos los estados de la primera lista, un subconjunto de los estados de la segunda lista sustituyen a los de la primera lista. Este proceso se repite hasta que se construye un estado que representa una solución completa al problema y se reconstruye la solución mediante el camino formado por las transiciones tomadas. Es posible que en algún momento no pueda generarse ningún estado para la segunda lista. En tal caso no se ha podido demostrar la factibilidad de la instancia.

El procedimiento se inicia en el estado 0, que representa un estado sin tareas asignadas, y utiliza dos parámetros de control para definir su comportamiento. El

primero, *ancho_ventana*, determina el número máximo de estados que se mantienen de una etapa a la siguiente. El segundo, *transiciones*, corresponde al número máximo de estados que se permite desarrollar desde un estado y se utiliza en el algoritmo propuesto en el apartado 2.1.

Tras generar todos los estados de la etapa en construcción, se recurre a un procedimiento de reducción que selecciona un subconjunto de los estados para la siguiente etapa. Para ello, primero se ordenan los estados por tiempo muerto acumulado no decreciente y se seleccionan los *ancho_ventana* de menor tiempo muerto, o todos los existentes si este número es menor, que cumplan las siguientes condiciones:

1. El estado no está dominado por ningún otro estado ya seleccionado. Un estado está dominado por otro si las tareas asignadas en el primer estado es idéntico o es un subconjunto de las tareas asignadas en el segundo estado.
2. El estado supera un test de consistencia basado en el número mínimo de estaciones necesarias para asignar las tareas que todavía no forman parte de la solución parcial. El test consiste en comprobar las cotas inferiores del número de estaciones, utilizando las cotas derivadas de la relación del problema con el problema de Bin Packing, Scholl (1999).

3 Algoritmo Genético

Un algoritmo genético mantiene una población de individuos representados a través de coromosomas que deben codificar directa o indirectamente una solución del problema. La calidad de cada individuo, conocida como *fitness*, debe basarse en la función objetivo que tiene la solución representada. El *fitness* influye en las posibilidades reproductivas del individuo basándose en el concepto evolutivo de la supervivencia del más apto. Partiendo de dos individuos seleccionados según el criterio anterior, se genera un descendiente utilizando un operador de cruce que combina los cromosomas de los padres y un operador de mutación encargado de diversificar la búsqueda, y finalmente el nuevo individuo entra a formar parte de la población, reemplazando el lugar de otro miembro. Estos operadores se repiten hasta que se alcanza un criterio de finalización, en nuestro caso un límite de tiempo.

El algoritmo quedará representado por cada uno de sus elementos: representación e inicialización de los cromosomas, selección de padres, cruce, mutación y reemplazo. A continuación se estudia cada operador y algunos elementos adicionales del algoritmo implementado.

Representación e inicialización de los individuos. En el algoritmo, un individuo se representa mediante una permutación de |V| elementos, que marca la ordenación de las tareas. La permutación, a su vez, respeta las relaciones de precedencia, esto es, si i es precedente de j, entonces i aparece antes en la permutación que j.

Para inicializar un individuo, se construye su permutación siguiendo el siguiente procedimiento: (1) se crea una lista de tareas candidatas, aquellas sin predecesoras por incluir en la lista, (2) se selecciona equiprobablemente una de las tareas de la lista y se anexa a la permutación, (3) se actualiza la lista de tareas candidatas. Estos pasos se repiten |V| ocasiones hasta obtener un individuo.

Selección de progenitores y cruce. La selección de progenitores se basa en un operador de selección mediante un torneo con dos participantes. Para seleccionar cada progenitor, se escoge de forma equiprobable a dos miembros de la población y se selecciona como progenitor al que mejor valor de *fitness* presente. El operador de cruce genera un único hijo partiendo de dos progenitores. El operador selecciona un cromosoma al azar del primer progenitor y copia los cromosomas anteriores a éste en el hijo. El resto de elementos se crea siguiendo el orden marcado por el segundo progenitor. Este operador es conocido como OX, Golberg (1989).

Operador de mutación. Cada hijo generado recibe una mutación consistente en intercambiar la posición de dos elementos en la permutación. El operador comprueba que la nueva permutación cumple las condiciones indicadas en la inicialización de los individuos, y en caso de no cumplirlo busca una nueva pareja de elementos para intercambiar.

Evaluación de fitness. En el procedimiento la evaluación del *fitness* realiza dos funciones; (1) determinación de la calidad del individuo y (2) búsqueda de mejores soluciones a la instancia. La evaluación usa el procedimiento BDP del apartado 2, usando la permutación obtenida por el crossover como criterio de ordenación de las tareas, e intenta resolver problemas con tiempo de ciclo menor que la mejor solución conocida. Si durante una llamada a la BDP se obtiene una solución factible, ésta se guarda y se repite el procedimiento con un tiempo de ciclo menor, hasta que no se consiga mejorar la solución.

Si el procedimiento ha mejorado la solución, el *fitness* del individuo se iguala al tiempo de ciclo que ha aportado la mejora. En caso contrario, el valor de fitness será igual al valor de la mejor solución conocida más el triple de la diferencia entre el número de estaciones buscadas y la última etapa para la que se han podido construir estados durante la BDP. Con esto último se penaliza la imposibilidad de obtener soluciones factibles, de modo que no deba invertirse tiempo en calcular un primer tiempo de ciclo factible para el individuo generado, que no mejorarán la mejor solución conocida.

Reemplazo de soluciones. Una vez se ha construido un nuevo individuo, éste reemplaza a un elemento escogido al azar de la población original. Para aprovechar la arquitectura multiprocesador de los actuales ordenadores personales, el procedimiento trabaja con una única población, pero cada procesador genera, evalúa y reemplaza individuos sobre la misma población de forma autónoma.

Obtención de una primera solución para la instancia. Para evaluar el *fitness* de un individuo es necesario partir de un tiempo de ciclo conocido. Para ello se resuelve el problema con la heurística de Hoffmann original, Hoffmann (1963), partiendo de un tiempo de ciclo mínimo, obtenido dividiendo la duración de todas las

tareas por el número de estaciones, e incrementando este valor hasta encontrar un valor de tiempo de ciclo con el número de estaciones deseado.

Explotación de la propiedad de reversibilidad del problema. Una propiedad del problema es su reversibilidad, esto es, si el sentido de los arcos del grafo de precedencias se invierte, cualquier solución obtenida para la nueva instancia puede transformarse en una solución para la instancia original cambiando la ordenación de las estaciones obtenidas, $S_j \leftarrow S_{m+1-j}$, para todo S_j, (j=1,…,m). Esta propiedad se utiliza en la mayoría de trabajos de la literatura, ya que se conoce que el comportamiento de los procedimientos de resolución varía según se resuelva la instancia directa o inversa. Por tanto durante la fase de evaluación se resuelven tanto la instancia directa, usando la permutación codificada en el individuo, como la inversa, usando la permutación invertida.

4 Experiencia Computacional

Para comprobar la calidad del algoritmo propuesto, éste se ha implementado en C++ usando el compilador GCC versión 4.2.1. El ordenador utilizado para ejecutar la experiencia computacional ha sido un Macintosh Imac con un procesador 2.93 Ghz. Intel Core i7, con cuatro núcleos, 8 GB. de memoria RAM y sistema operativo MAC OS X 10.6.8.

Las instancias utilizadas en la experiencia computacional corresponden a la colección disponible en la página web www.assembly-line-balancing.de, y que son utilizadas en la literatura para comparar la calidad de las soluciones ofrecidas por los procedimientos propuestos. Esta colección cuenta con 302 instancias con un número de tareas entre 8 y 297, de las que se ha demostrado el valor de solución óptimo para 288 de ellas. La experiencia computacional se ha centrado únicamente en las 14 instancias para las que no se ha podido verificar la optimalidad de la mejor solución.

El algoritmo presentado se compara con la mejor solución conocida para cada instancia, disponibles en la misma página web, y la solución obtenida por la Programación Dinámica Acotada, obtenida mediante la resolución iterativa del problema con diferentes tiempos de ciclo partiendo del valor de cota inferior disponible en la página web hasta encontrar un tiempo de ciclo factible, usando anchos de ventana entre 50 y 1600, y número de transiciones entre 50 y 500 ambos en incrementos también de 50 unidades.

El algoritmo genético ha sido probado con un tiempo máximo de ejecución de una hora, tiempo límite utilizado para comprobar la calidad de los algoritmos exactos en problemas de equilibrado, y diversos anchos de ventana y número de transiciones. Finalmente se ofrecen los resultados con *ancho_ventana*=500 y *transiciones*=50 al ser los que reportan mejor relación entre el número de soluciones generadas (mayor exploración del espacio de soluciones) y la calidad de las soluciones obtenidas (mayor explotación de una permutación para generar soluciones).

Tabla 1 Resultados del algoritmo propuesto para las instancias de la literatura para las que no se conoce la solución óptima. Para cada instancia, definida por su grafo y número de estaciones, se reporta la mejor solución conocida con anterioridad, columna Mejor, la mejor solución encontrada utilizando BDP y la mejor solución encontrada por el algoritmo genético propuesto

Grafo(m)	Mejor	BDP	Genético	Grafo(m)	Mejor	BDP	Genético
Arcus2(18)	8377	8377	8377	Arcus2(25)	6101	6118	6096
Arcus2(19)	7922	7922	7922	Arcus2(26)	5855	5860	5850
Arcus2(20)	7524	7523	7523	Wee-Mag(18)	87	87	87
Arcus2(21)	7187	7186	7186	Wee-Mag(19)	85	85	85
Arcus2(22)	6856	6856	6855	Wee-Mag(23)	67	67	67
Arcus2(23)	6560	6560	6560	Wee-Mag(27)	65	65	65
Arcus2(24)	6282	6290	6280	Wee-Mag(28)	64	64	64

Como puede verse, el algoritmo resulta efectivo con las instancias más complejas del problema, obteniendo o mejorando la mejor solución en todos los casos. Los resultados se deben a la combinación de dos factores importantes: (1) un secuenciador basado en la BDP para el algoritmo SALBP-F y (2) el uso de diferentes ordenaciones para dicho secuenciador, mediante un algoritmo genético que explora el espacio de ordenaciones, lo que diversifica la búsqueda en vez de intensificarla mediante el incremento de los parámetros de la BDP.

5 Referencias

Bautista J, Companys R, Corominas A (1996) Heuristics and exact algorithms for solving the Monden problem, Eur J Oper Res 88:101-113

Bautista J, Pereira J (2009) A dynamic programming based heuristic for the assembly line balancing problem, Eur J Oper Res 194:787-794

Blum C (2011) Iterative Beam Search for Simple Assembly Line Balancing with a fixed number of work stations, Statistics and Operations Research Transactions 35:145-164

Goldberg DE (1989) Genetic Algorithms in Search, Optimization and Machine Learning, Kluwer, Boston MA.

Golcalves JF, Almedia JR (2002) A hybrid genetic algorithm for assembly line balancing, J Heur 8:629-642

Hackman ST, Magazine MJ, Wee TS (1989) Fast, effective algorithms for simple assembly line balancing problems, Oper Res 37:916-924

Hoffmann TR (1963) Assembly line balancing with a precedence matrix Manage Sci 9:551-562

Jackson JR (1956) A computing procedure for a line balancing problem Manage Sci 2:261-271

Scholl A (1999) Balancing and sequencing assembly lines. Physica, Heidelberg

Scholl A, Becker C (2006) State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. Eur J Oper Res168:666-693

Scholl A, Voss S (1996) Simple assembly balancing – Heuristic approaches, J Heuristics 2:217-244

Talbi E (2002) A Taxonomy of hybrid metaheuristics, J Heuristics 8:541-562

Ugurdag HF, Rachamadugu R, Papachristou CA (1997) Designing paced assembly lines with fixed number of workstations, Eur J Oper Res 102:488-501